Mining Invisible Tasks in Non-free-choice Constructs

Qinlong Guo[1], Lijie Wen[1(✉)], Jianmin Wang[1], Zhiqiang Yan[2], and Philip S. Yu[3,4]

[1] School of Software, Tsinghua University, Beijing, China
guoqinlong@gmail.com, {wenlj,jimwang}@tsinghua.edu.cn
[2] Information School, Capital University of Economics and Business, Beijing, China
zhiqiang.yan.1983@gmail.com
[3] Department of Computer Science, University of Illinois at Chicago, Chicago, USA
[4] Institue for Data Science, Tsinghua University, Beijing, China
psyu@cs.uic.edu

Abstract. The discovery of process models from event logs (i.e. process mining) has emerged as one of the crucial challenges for enabling the continuous support in the life-cycle of a process-aware information system. However, in a decade of process discovery research, the relevant algorithms are known to have strong limitations in several dimensions. *Invisible task* and *non-free-choice construct* are two important special structures in a process model. Mining invisible tasks involved in non-free-choice constructs is still one significant challenge. In this paper, we propose an algorithm named $\alpha^{\$}$. By introducing new ordering relations between tasks, $\alpha^{\$}$ is able to solve this problem. $\alpha^{\$}$ has been implemented as a plug-in of ProM. The experimental results show that it indeed significantly improves existing process mining techniques.

Keywords: Process mining · Non-free-choice constructs · Invisible tasks

1 Introduction

Process mining is an essential discipline for addressing challenges related to Business Process Management and "Big Data" [15]. Nowadays, more and more organizations are applying workflow technology to their information systems, in order to manage their business processes. The information systems are logging events that are stored in so-called "event log". Informally, process mining algorithms are meant to extract meaningful knowledge from event logs, and use this knowledge for supporting or improving the process perspective.

Of the three process mining scenarios (i.e., discovery, conformance checking, and enhancement), discovery of a process model from an event log is the most important. In this paper we focus on the scenario of discovering Workflow nets [8] from event logs. However, the techniques presented in this paper may be adapted for the discovery of other process formalisms.

In many cases, the benefit of process mining depends on the exactness of the mined models [2]. The mined models should preserve all the tasks and the

H.R. Motahari-Nezhad et al. (Eds.): BPM 2015, LNCS 9253, pp. 109–125, 2015.
DOI: 10.1007/978-3-319-23063-4_7

dependencies between them that are present in the logs. Although much research is done in this area, there are still some significant and challenging problems to be solved [2][5]. In this paper, we focus on mining invisible tasks involved in non-free-choice constructs (IT-in-NFC for short). *Invisible tasks* (IT) [4] are such tasks that appear in a process model, while not observable in the corresponding event log. In a Workflow net, places cannot be linked with each other directly, so invisible tasks can be used for bridges between places. Besides, invisible tasks can also be used for expressing routing information. AND construct is that several places are connected to a transition, which has a meaning of parallel construct; XOR construct is that several transitions are connected to a place, which means a choice construct. *Non-free-choice construct* (NFC) [3] is a special kind of choice construct, whether to choose some task is dependent on what have been executed in the process model before (i.e. this choice is not "free"). In other words, NFC means the input places set of two transitions share some common places while they are not same. IT-in-NFC means that a process model contains both IT and NFC. As mentioned in [3][4], both of them are difficult to be discovered from event logs. Moreover, the combination of them (namely, some tasks in NFCs are invisible) even increases the difficulty of mining.

α [8] is a pioneering process mining algorithm which mines the Workflow net by considering relations between tasks in the event log. In order to mine NFCs, α^{++} [3] takes a new relation called *implicit dependency* (i.e. the indirect casual relation between tasks) into consideration based on α. Similarly, $\alpha^{\#}$ [4] is able to mine invisible tasks by considering *mendacious dependency*, which is the improper casual relations caused by invisible tasks. However, none of $\alpha^{\#}$ and α^{++} can properly mine IT-in-NFC. There are several other state-of-the-art mainstream process mining algorithms, such as *Genetic* [6], *Heuristic* [11], *ILP* [12], and *Region* [1]. Each of these algorithms has its own advantages. However, none of them is able to mine IT-in-NFC correctly.

In this paper, $\alpha^{\$}$ algorithm takes both *mendacious dependency* and *implicit dependency* into consideration. However, simple combination of considering these two dependencies cannot guarantee a correct mining result. There will be two significant challenging issues encountered:

1. The *reachable dependency*, which means one task can be executed after the execution of another task directly or indirectly, is a required relation to obtain the *implicit dependency*. However, reachable dependencies are detected by scanning the event log, but invisible tasks are unobservable here. Thus, *reachable dependencies* involved with these invisible tasks cannot be detected without complementation. The details can be found in Subsection 4.3.
2. Non-free-choice constructs, which are discovered after invisible tasks, bring more dependencies (e.g. the *implicit dependency*). These newly added dependencies may make invisible tasks unstructured. Thus, the affected invisible tasks should be split or combined to make the mined model sound. The details can be found in subsection 4.5.

Besides dealing with these issues, $\alpha^{\$}$ algorithm also addresses two drawbacks of *implicit dependencies* and *mendacious dependencies* respectively:

1. *Mendacious dependencies* cannot deal with some invisible tasks spanning one whole branch in a parallel construct.
2. *Implicit dependencies* cannot deal with the Length-1-loop (L1L for short) involved in NFCs.

The remainder of this paper is organized as follows. Section 2 shows a motivating example. Section 3 gives some preliminaries about process mining. In Section 4, we propose $\alpha^{\$}$ algorithm. Experimental results are given in Section 5. Section 6 concludes the paper and sketches the future work.

2 A Motivating Example

Figure 1 shows a real-life process model in SY company that is the largest construction machinery manufacturer in China. This model depicts the roadheader repairing process in the mine. A roadheader is a piece of excavating equipment consisting of a boom-mounted cutting head, a loading device usually involving a conveyor, and a crawler traveling track to move the entire machine forward into the rock face. For repair, there are three options. One option is that the roadheader is directly repaired underground. This option usually applies to simple repairs. In the other two options, the roadheader has to be repaired on the ground. The difference between the later two options is whether the roadheader should be decomposed before moved up to the ground through the tunnel, and then should be assembled after taken down to the mine.

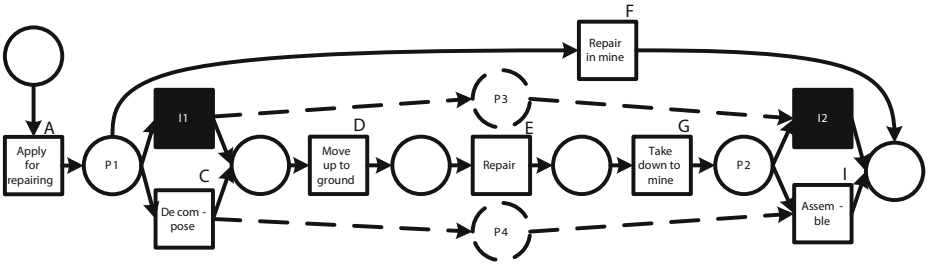


Fig. 1. A process model in SY company

The process model in Figure 1 is modelled in Petri net [8]. There are two invisible tasks (i.e., *I1* and *I2*), which are designed to skip the task *Decompose* and *Assemble* respectively. *I2*, together with *Assemble*, *P2*, *P3*, and *P4* composes a NFC. After *Take down to the mine*, whether *Assemble* is executed is dependent on whether *Decompose* has been executed before.

For convenience, we use the label in the right top of each task as abbreviation, for example *A* is short for *Apply for repairing*. Then [<A,F>,<A,D,E,G>, <A,C,D,E,G,I>] is an example event log corresponding to this model. Despite its apparent simplicity, this model and its corresponding log represent a hard case

for all existing techniques. Table 2(c) is an example model similar to this one. $\alpha^{\#}$ mines a similar model, while the NFC has not been discovered. The models mined by *Genetic* and *Heuristic* are identical, with the ITs not discovered. The models mined by α^{++} and ILP are not Workflow nets at all.

3 Preliminaries

Firstly, we discuss event log in detail and give an example. Then we give WF-net and its relevant concepts.

3.1 Event Log

The *event*, which represents a real-life action, is the basic unit of event logs. As defined in Definition 1, each event has several attributes, such as the time (i.e., when the event happens), activity [1] (i.e., what task this event corresponds to).

Definition 1. *(Event, Attribute) Let* $\mathscr{E}$ *be* event universe*, i.e., the set of all possible event identifiers. Let* AN *be a set of attribute names. For any event* $e \in \mathscr{E}$ *and a name* $n \in AN$*:*$\#_n(e)$ *is the value of an attribute* n *for event* e*.*

For convenience we assume the following standard attributes: $\#_{activity}(e)$ is the task associated to event e, $\#_{time}(e)$ is the timestamp associated to event e.

Though *time* is an important attribute for an event, we only consider the time order between events in this paper. Namely, the exact start time, end time or duration of an event are not taken in to consideration. The time order between events is enough for $\alpha^{\$}$ to mine a process model.

An event log consists of cases. Each case consists of ordered events, which are represented in the form of a *trace*, i.e., a sequence of unique events. Moreover, cases, like events, can also have attributes. Each case or event has an attribute of *case id* or *event id* respectively as the unique identifier.

Definition 2. *(Case, Trace, Event log, Simple Event Log) Let* $\mathscr{C}$ *be the case universe, i.e., the set of all possible case identifiers. For any case* $c \in \mathscr{C}$ *and name* $n \in AN$*:*$\#_n(c)$ *is the value of an attribute* n *for case* c*. Each case has a special mandatory attribute trace :* $\#_{trace}(c) \in \mathscr{E}^*$*.* $c = \#_{trace}(c)$ *is a shorthand for the trace of case* c*.*

A trace *is a finite sequence of events* $\sigma \in \mathscr{E}^*$ *such that each event appears only once, where* $|\sigma|$ *means the number of events contained in trace* σ*. The order of events in a trace is according to their timestamp, namely* $\forall_{1 \le i < j \le |\sigma|} \#_{time}(\sigma_i) \le \#_{time}(\sigma_j)$*. An* event log *is a set of cases* $L \subset \mathscr{E}^*$*.*

Let $\mathscr{A}$ *be the set of activity names, A simple trace* σ *is a sequence of activities, i.e.,* $\sigma \in \mathscr{A}^*$*. A simple even log* L *is a multi-set of traces over* $\mathscr{A}$*.*

[1] In this paper, we use activity and task alternatively.

Since information of time order and activity name of an event are sufficient for $\alpha^{\$}$, the simple event log and trace are used in the rest part of the paper.

Given a set of tasks (say T), an *event log W over T* means the task associated to any event in W is contained in T, i.e. $\forall e \in W : \#_{activity}(e) \in T$.

Table 1 is an example simple event log of the process model in Figure 1. This log contains four cases. For example, for case 1 and case 4, A and F are executed successively. The event log in Table 1 can be shortened as $[< A, F >^2, < A, D, E, G >, < A, C, D, E, G, I >]$, where 2 means $< A, F >$ occurs twice.

As mentioned in [5], dealing with noises in event logs is a challenging issue in process mining domain. However, many log filtering plugins have been implemented in Prom 6 [9]. The plugin *Filter Log with Simple Heuristic* is used for abating noise. By this way, we assume that the event log has no noise.

Table 1. An example simple event log of the process model in Figure 1

Case Id	Event Id	Activity	Case Id	Event Id	Activity	Case Id	Event Id	Activity
1	35654422	A		35654485	G		35654583	G
	35654423	F	3	35654579	A		35654584	I
2	35654481	A		35654580	C	4	35655442	A
	35654483	D		35654581	D		35655443	F
	35654484	E		35654582	E			

3.2 Workflow Net

In this paper, *Workflow net* [8] is used as the process modelling language. Workflow nets as defined in Definition 5, are a subset of labeled Petri nets.

Definition 3. *(Petri net) A Petri net is a triplet $N = (P, T, F)$ where P is a finite set of places, T is a finite set of transitions such that $P \cap T = \emptyset \wedge P \cup T \neq \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the flow relation.*

Definition 4. *(Labeled Petri Net) A labeled Petri net is a tuple $L\Sigma=(P, T, F, A, l)$, where (P, T, F) is a Petri net, A is a finite set of task names, and l is a surjective mapping from T to $A \cup \{\tau\}$ (τ is a symbol unobservable to the outside world).*

Definition 5. *(Workflow net). Let $N = (P, T, F, A, l)$ be a labeled Petri net and $\bar{t}$ be a fresh identifier not in $P \cup T$. N is a workflow net (WF-net) if and only if (a) P contains a unique source place i such that $\cdot i = \emptyset$, (b) P contains a unique sink place o such that $o\cdot = \emptyset$, and (c) $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\}, A \cup \{\tau\}, l \cup \{(\bar{t}, \tau)\})$ is strongly connected.*

Figure 1 gives an example of a process modelled in WF-net. This model has two IT-in-NFCs. The transitions (drawn as rectangles) $A, C, ..., I$ represent tasks, where hollow rectangles represent visible tasks, and the solid one (i.e. *I1* or *I2*) represents an invisible task. The places (drawn as circles) $P1, P2, ..., P4$ represent conditions. We adopt the formal definitions, properties, and firing rules of WF-net from [7] [8]. For mining purpose, we demand that each visible task (i.e., transition) has a unique name in one process model.

4 The New Mining Algorithm $\alpha^{\$}$

$\alpha^{\$}$ is composed of five steps: *Detect Invisible Tasks*, *Complement Reachable Dependencies*, *Detect Non-free-choice Constructs*, *Adjust Invisible Tasks*, and *Construct Workflow Net*. By applying *improved mendacious dependencies*, the first step of $\alpha^{\$}$ finds invisible tasks from the given event log. In the second step, $\alpha^{\$}$ complements reachable dependencies related to the found invisible tasks. Then, it discovers NFCs by using *implicit dependencies*. Next, $\alpha^{\$}$ adjusts the invisible tasks in order to ensure the mined model's soundness. Finally, $\alpha^{\$}$ constructs the process model based on the relations found in previous steps. Since the last step is identical to other *α-series* algorithms [3][4][8], it would not be elaborated. Initially, basic relations are introduced, then the following subsections elaborate the first four steps and the whole algorithm respectively.

4.1 Basic Relations

α algorithm, which is the forerunner of the α-series algorithms, defined six relations, $>_W$,$\triangle_W$, $\lozenge_W$, $\rightarrow_W$, $||_W$, and $\#_W$. $>_W$ expresses two tasks can be executed successively. $\triangle_W$ means a possible length-2-loop structure (i.e. a loop with length 2). $\lozenge_W$ shows two tasks have the $\triangle_W$ relations between each other. $\rightarrow_W$ is referred as the (direct) casual relation. $||_W$ suggests the concurrent behavior, namely two activity can be executed in any order. Relation $\#_W$ reflects that two tasks never follow each other directly. For example, in the event log in Table 1, $A >_W F$, $A >_W C$, $A \rightarrow_W F$, $G \rightarrow_W I$ hold.

Definition 6. *(Relations defined in α [8], Mendacious dependency [4], Reachable dependency [3]) Let T be a set of tasks, W be an event log over T, a and b be two tasks in T, the relations defined in α algorithm, mendacious dependency, and reachable dependency are defined as follows:*

- $a >_W b \iff \exists \sigma = t_1 t_2 ... t_n \in W, i \in 1, ..., n-1 : t_i = a \wedge t_{i+1} = b$,
- $a \triangle_W b \iff \exists \sigma = t_1 t_2 ... t_n \in W, i \in 1, ..., n-2 : t_i = t_{i+2} = a \wedge t_{i+1} = b$,
- $a \lozenge_W b \iff a \triangle_W b \wedge b \triangle_W a$,
- $a \rightarrow_W b \iff (a >_W b \wedge b \not>_W a) \vee a \lozenge_W b$,
- $a ||_W b \iff a >_W b \wedge b >_W a \wedge a \not\lozenge_W b$,
- $a \#_W b \iff a \not>_W b \wedge b \not>_W a$,
- $a \rightsquigarrow_W b \iff a \rightarrow_W b \wedge \exists x, y \in T : a \rightarrow_W x \wedge y \rightarrow_W b \wedge y \not>_W x \wedge x \nparallel_W b \wedge a \nparallel_W y$,
- $a \gg_W b \iff \exists \sigma = t_1 t_2 ... t_n \wedge i, j \in 1, ..., n : i < j \wedge t_i = a \wedge t_j = b \wedge \forall k \in [i+1, ..., j-1] : t_k \neq a \wedge t_k \neq b$, *and*
- $a \succ_W b \iff a \rightarrow_W b \vee a \gg_W b$.

In $\alpha^{\#}$ algorithm, *mendacious dependency* $\rightsquigarrow_W$ is proposed to describe the relations reflecting invisible tasks. It is based on six pre-conditions as defined in Definition 6. $A \rightsquigarrow_W D$ holds in the event log in Table 1, because $A \rightarrow_W D$, $A \rightarrow_W C$, $C \rightarrow_W D$, $C \nparallel_W D$, $C \nparallel_W A$, and $C \not>_W C$, which means there should be an invisible task between A and D (i.e., the invisible task *I1* in Figure 1).

Reachable dependency is used to depict the indirect dependency between activities. In α^{++}, reachable dependency is a necessary condition for discovering NFCs. For example, in Table 1, $E \gg_W G$, $C \gg_W I$, and $D \succ_W G$ hold.

4.2 Detecting Invisible Tasks by Improved Mendacious Dependency

The aim of this step is to discover invisible tasks from the given event log. Most invisible tasks can be detected by applying *mendacious dependency* proposed in [4]. However, mendacious dependency cannot deal with some invisible tasks involved in one whole branch of a parallel construct. Thus, we propose an *improved mendacious dependency* to resolve this issue.

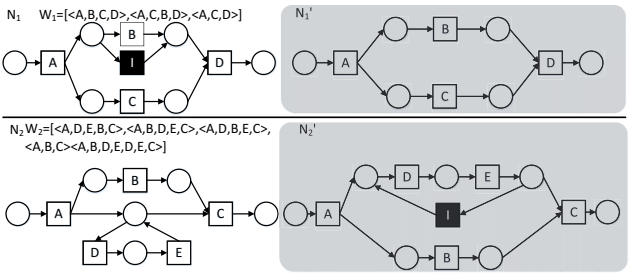


Fig. 2. Two examples for the defect of mendacious dependency

Two examples in Figure 2 show the defect of mendacious dependency. N_1 and N_2 are the original models, N_1' and N_2' are the models mined by $\alpha^{\#}$ algorithm. In N_1', $\alpha^{\#}$ didn't discover the invisible task I. This is because that $A \rightarrow_W D$, which is a requirement for discovering task I, does not hold due to the interference from task C in another branch. As for N_2', the disturbance from task B makes $A \not\succ_W C$ hold. This leads to the invisible task I improperly discovered in N_2'. Due to task I, N_2' has less behavior than N_2, e.g., trace $< A, B, C >$ cannot be replayed on N_2'. In order to the overcome this defect, we introduce *Between-Set* and define *improved mendacious dependency*.

Between-Set, which is defined in Definition 7, is to depict the tasks that occur between two tasks. When the two tasks are the endpoints of a parallel construct, the Between-Set is the set of tasks in the parallel branches. For examples in Figure 2, $Between(W_1, A, D) = \{B, C\}$, $Between(W_2, A, C) = \{B, D, E\}$.

Definition 7. *(Between-Set) Let T be a set of tasks, W be an event log over T, a and b be two tasks in T, σ be a trace of W with length n, namely $\sigma \in W$, the* Between-Set *of a,b (i.e. Between(W,a,b)) can be defined as follows:*

- $Between(\sigma, a, b) = \{\sigma_k | \exists_{1 \le i < j \le n} (\sigma_i = a \wedge \sigma_j = b \wedge i < k < j \wedge \nexists_{i<l<j} (\sigma_l = a \vee \sigma_l = b))\}$,
- $\neg Between(\sigma, a, b) = \{\sigma_k | 1 \le k \le n\} \backslash Between(\sigma, a, b)$, *and*
- $Between(W, a, b) = \cup_{\sigma \in W} Between(\sigma, a, b) \backslash \cup_{\sigma \in W} \neg Between(\sigma, a, b)$

The *improved mendacious dependency* is defined in Definition 8. We redefine $\rightarrow_W$ and $>_W$ in [4] as $\rightrightarrows_W$ and $\gg_W$ respectively. Compared with the old ones, $\rightrightarrows_W$ and $\gg_W$ are able to eliminate the interference of parallel constructs. For instance, in N_1, $A \rightarrow_W D$ does not hold. However, $A \rightrightarrows_W D$ holds. In N_2, $A \not\succ_W C$ 'improperly' holds. Nevertheless, $A \gg_W C$ holds.

Definition 8. *(Improved mendacious dependency) Let T be a set of tasks, W be an event log over T, a,b be two tasks from T, the* improved mendacious dependency *$a \hookrightarrow_W b$ is defined as follows:*

- $a \geqslant_W b \iff \exists_{x,y \in T}(Between(W,x,y) \subset Between(W,a,b) \wedge \forall_{m \in (Between(W,a,b) \backslash (Between(W,x,y) \cup \{x,y\}))} \forall_{n \in Between(W,x,y)} m \parallel_W n \wedge \exists_{\sigma \in W} Between(\sigma,a,b) \subseteq (Between(W,a,b) \backslash (Between(W,x,y) \cup \{x,y\})))$,
- $a \rightrightarrows_W b \iff (a \geqslant_W b \wedge b \not\geqslant_W a) \vee a \Diamond_W b$, *and*
- $a \hookrightarrow_W b \iff \exists_{x,y \in T}(a \rightarrow_W x \wedge y \rightarrow_W b \wedge x \nparallel_W b \wedge y \nparallel_W a \wedge a \rightrightarrows_W b \wedge y \not\geqslant_W x)$.

4.3 Complementing Reachable Dependencies

Since invisible tasks do not appear in any event log, $>>_W$ and $\succ_W$ related to invisible tasks are missing for discovering NFSs. For example, in Figure 1, the part with solid lines is the model mined by $\alpha^{\$}$ without this complementing step. NFC (i.e. the dotted edge line part) was not discovered due to incomplete reachable dependencies regarding to invisible tasks.

In order to deal with this incompleteness, we first introduce the definition of *conditional reachable dependency* (CRD for short). Symbol $a>>_\sigma b$ expresses that task a is indirectly followed by task b in trace σ. We artificially add a *starting task* (i.e. $\perp$) and an *ending task* (i.e. $\top$) to each trace in the event log. Namely, for a trace σ with length n, $\#_{activity}(\sigma_0) = \perp$ and $\#_{activity}(\sigma_{n+1}) = \top$.

In Definition 9, there are three kinds of CRDs: *pre-CRD* (i.e. $\succ_{W,Pre=x}$), *post-CRD* (i.e. $\succ_{W,Post=y}$), and *both-CRD* (i.e. $\succ_{W,Pre=x,Post=y}$). $a \succ_{W,Pre=x,Post=y} b$ means there is a trace σ where $a>>_\sigma b$ holds, and x occurs directly before a, y occurs directly after b. For example, in Table 1, $C \succ_{W,Pre=A,Post=\top} I$ holds. The pre-CRD and post-CRD are special cases of both-CRD.

Definition 9. *(Conditional reachable dependency) Let T be a set of tasks, W be an event log over T, a,b be two tasks from T, x,y be two tasks from* $T \cup \{\perp\} \cup \{\top\}$. Conditional reachable dependencies *are defined as follows:*

- $a \succ_{W,Pre=x} b \iff a \rightarrow_W b \vee (\exists_{\sigma \in W \wedge 1 \leq i \leq |\sigma|} \sigma_i = a \wedge \sigma_{i-1} = x \wedge a>>_\sigma b)$,
- $a \succ_{W,Post=y} b \iff a \rightarrow_W b \vee (\exists_{\sigma \in W \wedge 1 \leq j \leq |\sigma|} \sigma_j = b \wedge \sigma_{j+1} = y \wedge a>>_\sigma b)$,
- $a \succ_{W,Pre=x,Post=y} b \iff a \rightarrow_W b \vee (\exists_{\sigma \in W \wedge 1 \leq i,j \leq |\sigma|} \sigma_i = a \wedge \sigma_j = b \wedge \sigma_{j+1} = y \wedge \sigma_{i-1} = x \wedge a>>_\sigma b)$.

Based on CRDs, the *Reachable dependency related to invisible task* is defined in Definition 10. For two invisible tasks x and y, $x \succ_W y$ holds if there are four tasks a_1, a_2, b_1, and b_2 satisfying $a_1 \rightarrow_W x$, $x \rightarrow_W b_1$, $a_2 \rightarrow_W y$, $y \rightarrow_W b_2$, and $b_1 \succ_{W,Pre=a_1,Post=b_2} a_2$ holds. For an invisible task x, and a task m, $x \succ_W m$ holds if there are two tasks a and b satisfying $a \rightarrow_W x$, $x \rightarrow_W b$, and $b \succ_{W,Pre=a} m$ holds. For instance, the event log in Table 1, $I1 \succ_W E$ holds, because $A \rightarrow_W I1$, and $D \succ_{W,Pre=A} E$ and $I1 \rightarrow_W D$. $m \succ_W x$ is similar to $x \succ_W m$.

Definition 10. *(Reachable dependency related to invisible task) Let T be a set of tasks, W be an event log over T, m be a task from T, x,y be two invisible tasks, the* reachable dependency related to invisible task *is defined as follows:*

- $x \succ_W m \iff \exists_{(a=\bot \vee a \in W) \wedge b \in W} a \rightarrow_W x \wedge x \rightarrow_W b \wedge b \succ_{W,Pre=a} m$,
- $m \succ_W x \iff \exists_{a \in W \wedge (b \in W \vee b=\top)} a \rightarrow_W x \wedge x \rightarrow_W b \wedge m \succ_{W,Post=b} a$,
- $x \succ_W y \iff \exists_{(a_1=\bot \vee a_1 \in W) \wedge b_1 \in W \wedge a_2 \in W \wedge (b_2 \in W \vee b_2=\top)}$
  $a_1 \rightarrow_W x \wedge x \rightarrow_W b_1 \wedge a_2 \rightarrow_W y \wedge y \rightarrow_W b_2 \wedge b_1 \succ_{W,Pre=a_1,Post=b_2} a_2$.

4.4 Detecting Non-free-choice Constructs

After making the reachable dependencies complete, the aim of this step is to discover NFCs. α^{++} algorithm can mine NFCs in most cases. However, α^{++} is not able to mine the length-1-loop construct (L1L for short) involved in NFCs. L1L set, as defined in Definition 11, is a set of tasks, where each task appears at least twice continuously in an given event log. α^{++} excludes all tasks in L1L set when considering implicit dependencies, which impedes discovering the L1L involved in NFCs. For example, Figure 3 shows the defect of α^{++} on dealing with such issue. N_3 and N_4 are the original models which contain NFCs combined with L1L, N_3' and N_4' are models mined by α^{++}. The NFC is not detected in N_3', which makes N_3' have more behavior than N_3: trace ¡*A,C,G,F,D,B,E*¿ can be replayed on N_3' but cannot be replayed on N_3. Besides, α^{++} does not discover the arcs related to task D, which results in N_4' not sound at all [8].

Definition 11. *(length-1-loop set) Let T be a set of tasks, W be an event log over T, the L1L set is defined as follows:*

- $L1L = \{t \in T | \exists_{\sigma=t_1 t_2 \ldots t_n \in W; i \in 2,3,\ldots,n} t = t_{i-1} \wedge t = t_i\}$.

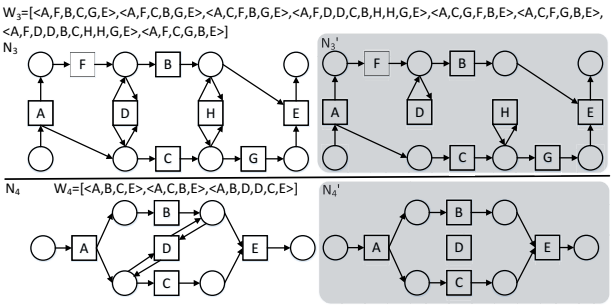


Fig. 3. Defect of α^{++} on mining L1L involved in NFCs

Thus, a set of tasks (called L1L-Free set) is determined before applying α^{++}, in which the tasks should not be excluded when detecting NFCs. L1L-Free set is summarized in Definition 12. Namely, for each such task x of L1L, there exists a pair of tasks a, b parallel with each other. Besides, two *sequence relations* $a \rightarrow_W x$ and $x \rightarrow_W b$ hold. For example, L1L-Free set of N_3 is {D,H}. For task D, $F \rightarrow_W D$, $D \rightarrow_W C$, and $F ||_W C$ hold.

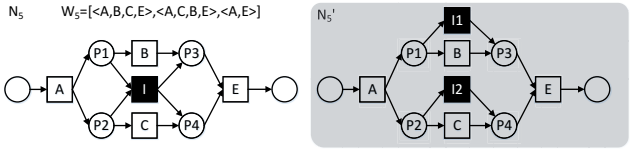
Definition 12. *(L1L-Free set) Let T be a set of tasks, W be an event log over T, and L1L be the set of length-1-loop. The* L1L-Free set *is defined as follows:*

- $L1L\text{-}Free = \{x \in L1L | \exists_{a \in T \wedge b \in T} (a \rightarrow_W x \wedge x \rightarrow_W b \wedge a \parallel_W b)\}$.

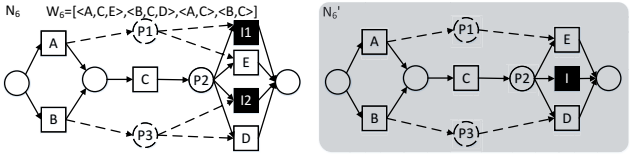
4.5 Adjusting Invisible Tasks

In order to construct a sound and accurate process model, this step adjusts the invisible tasks by combining or splitting. Before introducing the details of invisible task adjustment, some auxiliary functions are given.

- *MD* means the set of improved mendacious dependencies,
- *ID* means the set of implicit dependencies,
- *MD(t)* is the set of improved mendacious dependencies related to an invisible task t, and
- $\sigma \uparrow X$ is the projection of σ onto some task set $X \subset T$.



(a) Mined model without combining



(b) Mined model without spliting

Fig. 4. Example of mined process models without combining or splitting

Combining Invisible Tasks. When there are invisible tasks in different branches of a parallel construct, there is a possibility that invisible tasks should be combined together. However, $\alpha^{\#}$ does not take this situation into consideration. For example, N_5 in Figure 4(a) is a process model with an invisible task I combined with the parallel construct (B,C,P1,P2,P3,P4). N_5' is the process model mined by $\alpha^{\#}$, where there is one invisible task *I1* or *I2* in each parallel branch. N_5' has more behavior than N_5, such as traces $< A, C, E >$ and $< A, B, E >$. Definition 13 is used to discover the pairs of combinable invisible tasks.

Definition 13. *(Combinable invisible tasks) Let T be a set of tasks, W be an event log over T, T_I be the set of invisible tasks discovered from W, and trace $\sigma \in W$. The* Combinable invisible tasks *is defined as follows:*

- $R(t) = \{z | (a, x, y, b) \in MD(t) \wedge (z = x \vee z = y)\}$,
- $P(\sigma, a, b) = \sigma \uparrow (R(a) \cup R(b))$,
- $a \bigotimes_{\sigma} b \iff |P(\sigma, a, b)| = 0 \vee (|P(\sigma, a, b)| \% 2 = 0 \wedge \forall_{1 \leq k < |P|/2}((P(\sigma, a, b)_{2k+1} \in R(a) \wedge P(\sigma, a, b)_{2k+2} \in R(b)) \vee (P(\sigma, a, b)_{2k+1} \in R(b) \wedge P(\sigma, a, b)_{2k+2} \in R(a)))$,

- $a \bigotimes_W b \iff \forall_{\sigma \in W} a \bigotimes_\sigma b$, *and*
- *Combinable_Set(W,*T_I*)* $= \{(a,b) | a \parallel_W b \wedge a \in T_I \wedge b \in T_I \wedge a \bigotimes_W b\}$.

$R(t)$ is the set of tasks that are related to invisible task t. $P(\sigma, a, b)$ is the projection of trace σ on the task set $R(a) \cup R(b)$. $a \bigotimes_\sigma b$ holds only if either the tasks from R(a) and R(b) alternately occurs in $P(\sigma, a, b)$, or none of tasks in R(a) and R(b) occurs. It means that in one trace, either invisible tasks a and b occur together or none of them occurs. If for any trace $\sigma \in W$, $a \bigotimes_\sigma$ b holds (i.e. $a \bigotimes_W b$), invisible tasks a and b should be combined.

Splitting Invisible Tasks. NFCs are detected after the discovery of invisible tasks, and they would bring extra dependency relations between tasks. This would lead to deadlock when they are involved with invisible tasks in some case. N_6 and N_6' in Figure 4 present an example of this situation. N_6 is the original sound process model, while N_6' is the process model mined without splitting. In N_6', task I is found earlier than the NFC (P1, P2, P3, E, D). Place P1, P3 and the dotted edges are added for constructing the NFC. The new-added dependencies (A,E) and (B,D) make the net not sound. For example, after the execution of $< A, C, I >$ or $< B, C, I >$, there would be a remaining token in P1 or P3. Thus, we should check each invisible task, and split them if necessary.

Definition 14. *(Splittable invisible tasks) Let T be a set of tasks, W be an event log over T. The* Splittable invisible tasks *is defined as follows:*

- $IMD(t) = \{(a, x, y, b) \in MD(t) | \exists (m, n) \in ID : (n = x \vee m = y)\}$, *and*
- *Splittable_Set(W,*T_I*)*$=\{t \in T_I || IMD(t)| > 1\}$.

For each invisible task t, $IMD(t)$ is the set of mendacious dependencies involved in implicit dependencies (we call this mix dependencies for short). By analyzing the workflow net, we discover that each mix dependency should be expressed as one unique invisible task. If two mix dependencies are expressed in one invisible task, the mined model would not be sound, like the invisible task I in N_6'. Thus, the invisible task t should be split if $|IMD(t)| > 1$.

4.6 The $\alpha^\$$ Algorithm

The $\alpha^\$$ algorithm is defined in Definition 15. According to the previous sections, the algorithm is luminous. And thus, there is no further explanation.

Definition 15. *Let T be a set of tasks, W be an event log over T. The* $\alpha^\$(W)$ *algorithm is defined as follows:*

1. $T_{log} = \{t \in T | \exists_{\sigma \in W} t \in \sigma\}$,
2. $L1L_{raw} = \{t \in T_{log} | \exists_{\sigma = t_1 t_2 t_3 \ldots t_n \in W; i \in 1,2,\ldots,n} t = t_{i-1} \wedge t = t_i\}$,
3. $L1L_{free} = getL1LFree()$,
4. $L1L = L1L_{raw} - L1L_{free}$,
5. $W^{-L1L} = removeL1L(W, L1L)$,

6. $(P_{W^{-L1L}}, T_{W^{-L1L}}, T^I_{W^{-L1L}}, F_{W^{-L1L}}) = \alpha^{\sharp}_{improved}(W^{-L1L})$,
7. $R_{W^{-L1L}} = getReachableDependency(T^I_{W^{-L1L}}, W^{-L1L})$,
8. $ID_{W^{-L1L}} = \alpha^{++}(P_{W^{-L1L}}, T_{W^{-L1L}}, T^I_{W^{-L1L}}, F_{W^{-L1L}}, R_{W^{-L1L}})$,
9. $(F'_{W^{-L1L}}, P'_{W^{-L1L}}) = addImplicitDependency(F_{W^{-L1L}}, P_{W^{-L1L}}, ID_{W^{-L1L}})$,
10. $(P_W, T_W, F_W) = addL1L(L1L, F'_{W^{-L1L}}, P'_{W^{-L1L}}, T_{W^{-L1L}})$,
11. $(P_W, T_W, F_W) = adjustInvisibleTask(T^I_{W^{-L1L}}, F_W)$, *and*
12. $\alpha^{\$}(W) = (P_W, T_W, F_W)$.

5 Experimental Evaluation

The $\alpha^{\$}$ algorithm has been implemented as a plug-in in ProM [9]. The ProM 5.2-based implementation of $\alpha^{\$}$, is publicly accessible from GitHub (https://github.com/ guoqinlong/Alpha-Dollar-Process-Mining-Algorithm).

In the evaluation, we compared the performance of $\alpha^{\$}$ with other process mining algorithms: α^{++}, $\alpha^{\#}$, *Genetic*, *Heuristic*, *ILP*, and *Region*. Note that the default mining results of Genetic and Heuristic are Heuristic nets. With the transformation package in ProM, a transformation to Petri net has been implemented. Besides, all algorithms were executed using their default settings.

5.1 Evaluation Based on Artificial Logs

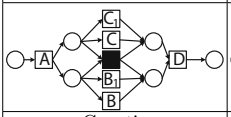
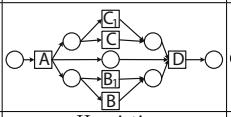
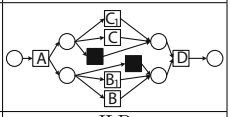
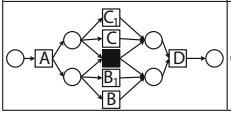
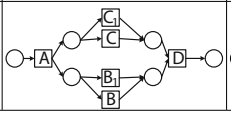
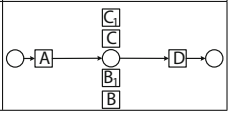
We first focus on artificial examples which demonstrate that $\alpha^{\$}$ significantly improves existing approaches. To illustrate the capabilities of $\alpha^{\$}$, we first show some concrete experimental results in Table 2.

Table 2(a) is an example of an invisible task involved in a parallel construct. In the reference model, two parallel branches share the same invisible task. $\alpha^{\$}$ and *Genetic* managed to mine the proper process model. α^{++} and Heuristic failed to discover the invisible task. $\alpha^{\#}$ managed to mine two invisible tasks, but failed to combine them into one. *ILP* cannot mine a sound process model. Besides, *ILP* also cannot mine proper process models for other examples in Table 2, which will not be elaborated then. The reference model in Table 2(b) has two invisible tasks combined with NFCs. $\alpha^{\$}$ managed to rediscover it. The models mined by $\alpha^{\#}$, *Heuristic* and *Genetic* failed to detect all NFCs, and these three mined models have more behavior than the reference model. α^{++} failed to discover the invisible task. Table 2 is an example of IT-in-NFC. α^{++}, *Genetic* and *Heuristic* failed to mine the invisible task. Two invisible tasks were detected by $\alpha^{\#}$, while the NFC was not discovered. The reference model in Table 2(d) has one invisible task. α^{++}, $\alpha^{\#}$ and *Heuristic* failed to discover the invisible task. Besides, the process model discovered by *Genetic* is not sound.

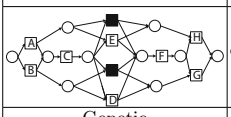
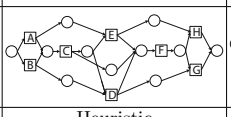
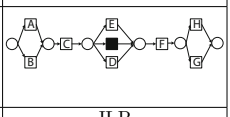
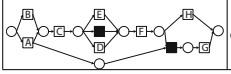
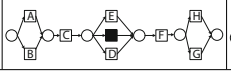
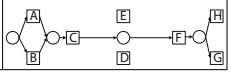
There are 40 process models in the artificial data set, 30 of which are from the *general significant reference model set* proposed in [13]. Besides, five artificial models (i.e., Artif-1) with IT-in-NFC are supplied in the experimental data set to demonstrate the capabilities of $\alpha^{\$}$ algorithm. Another five artificial models (i.e., Artif-2) are generated by tool *Process Log Generator*(PLG) [17].

Table 2. Four example process models

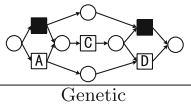
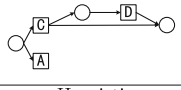
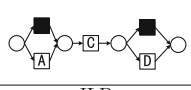
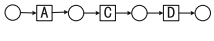
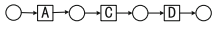
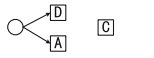
(a)

Log	$[< A,B,C,D >^{7}, < A,C,B,D >^{9}, < A,B_1,C,D >^{4}, < A,B,C_1,D >^{3}, < A,C_1,B,D >^{6}, < A,C,B_1,D >^{5}, < A,C_1,B_1,D >^{3}, < A,B_1,C_1,D >^{4}, < A,D >^{11}]$	
Reference Model & $\alpha^{\$}$	α^{++}	$\alpha^{\#}$
		
Genetic	Heuristic	ILP
		

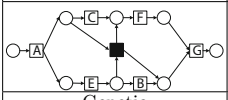
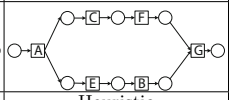
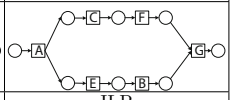
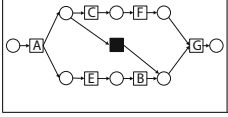
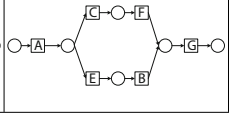
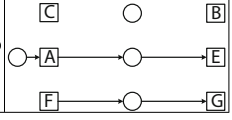
(b)

Log	$[< B,C,D,F,G >^{22}, < A,C,E,F,H >^{17}, < B,C,F,G >^{32}, < A,C,F,H >^{18}]$	
Reference Model & $\alpha^{\$}$	α^{++}	$\alpha^{\#}$
		
Genetic	Heuristic	ILP
		

(c)

Log	$[< A,C,D >^{22}, < C >^{23}]$	
Reference Model & $\alpha^{\$}$	α^{++}	$\alpha^{\#}$
		
Genetic	Heuristic	ILP
		

(d)

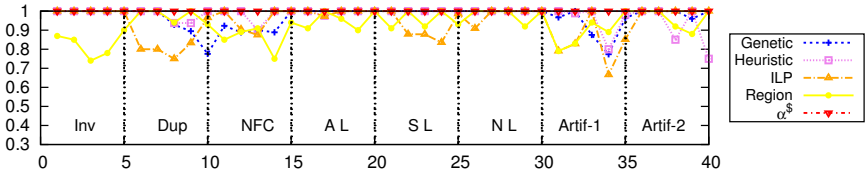
Log	$[< A,E,F,G >, < A,E,B,C,F,G >^{6}, < A,E,C,B,F,G >, < A,E,C,F,B,G >^{10}, < A,C,E,F,B,G >^{8}, < A,C,F,E,B,G >^{6}, < A,C,E,B,F,G >^{4}]$	
Reference Model & $\alpha^{\$}$	α^{++}	$\alpha^{\#}$
		
Genetic	Heuristic	ILP
		

For evaluation of the performance on artificial dataset, two criteria are applied : *Fitness* [10] tests the conformance between the mined model and the given log. The fitness is determined by replaying the log on the model, i.e., for each trace the "token game" is played. If $fitness = 1$ then the log can be parsed by the model without any error; *Model rediscoverability* [13] tests the conformance between the mined model and the original model. Model rediscoverability of a process mining algorithm is measured by the similarity degree between the original model and the model mined by the process mining algorithm.

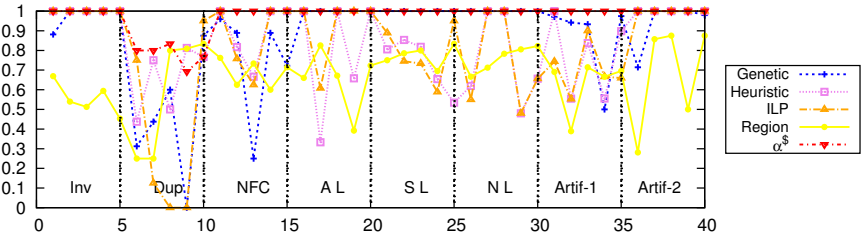
Table 3. Time expense on experiments in seconds

Mining algorithm	$\alpha^{\$}$	α^{++}	$\alpha\#$	Genetic	Heuristic	ILP	Region
Artificial logs	14.63	10.33	12.06	122.04	14.95	23.95	47.13

Table 3 shows the time expense on evaluation. As for the artificial logs, the cost of $\alpha^{\$}$ are more than that of either $\alpha^{\#}$ or α^{++}, yet less than the sum of them. Compared with *Genetic*, the time cost of $\alpha^{\$}$ is much smaller.



(a) Fitness



(b) Model rediscoverability

Fig. 5. Experiment result on artificial logs

Figure 5(a) shows the fitness result of different process mining algorithms. Since $\alpha^{\$}$ is based on $\alpha^{\#}$ and α^{++}, $\alpha^{\$}$ outperforms both of them. For the clearness of the figure, the results of $\alpha^{\#}$ and α^{++} are not listed in Figure 5. In the fitness measurement, 2.5% models mined by $\alpha^{\#}$ and 10.0% models mined by α^{++} do not have a value of 1, but $\alpha^{\$}$ has a fitness of 1 in all the models. $\alpha^{\$}$ has a better fitness result than all the other algorithms. Model rediscoverability result is shown in Figure 5(b). Except the process models with duplicate tasks, $\alpha^{\$}$ is able to mine

each process model identical with the original one in the experiment data set, while the other algorithms cannot mine them successfully. $\alpha^{\$}$ has a better result in model rediscoverability than $\alpha^{\#}$ and $\alpha^{\$}$. 47.5% of models mined by $\alpha^{\#}$ and 32.5% of models mined by α^{++} have a less model rediscoverability than $\alpha^{\$}$.

5.2 Evaluation Based on Real-Life Log

In this section, we use the event log used in paper [6]. This real-life log shows the event traces (process instances) for four different applications to get a license to ride motorbikes or drive.

Since there is no groundtruth model of the event log, *model rediscoverability* cannot be applied as the evaluation criteria. In this section, two additional quality measures in ProM are applied: precision - a measure how closely the behavior in the log is represented by the Petri net, simplicity - all places, transitions and arcs of the discovered Petri nets are counted and accumulated to a simplicity measure. Additionally, whether the mined model is a WF-net and the time cost of mining are evaluated.

Table 4. Evaluation results of different process discovery algorithms on a real-life log

	$\alpha^{\$}$	α^{++}	$\alpha^{\#}$	Genetic	Heuristic	ILP	Region
Fitness	***1.00***	0.95	***1.00***	***1.00***	***1.00***	***1.00***	0.90
Precision	***1.00***	***1.00***	0.84	***1.00***	0.84	0.56	0.91
Simplicity	22	20	22	26	20	***17***	26
Workflow Net	***True***	***True***	***True***	***True***	***True***	False	False
Time(ms)	224	175	185	34301	***102***	470	2786

Table 4 is the evaluation result on the log. Due to strongly unconnected nodes, models mined by ILP and Region are not WF-nets. Only $\alpha^{\$}$ and Genetic mined process model with both Precision = 1 and Fitness = 1. Other algorithm such as α^{++}, $\alpha^{\#}$ and Heuristic cannot mine invisible tasks and NFCs simultaneously, which leads either Precision or Fitness less than 1. Though $\alpha^{\$}$ and Genetic mine the same model, $\alpha^{\$}$ shows about 2 orders of magnitudes improvement in time costs, and there is no parameter setting needed for $\alpha^{\$}$.

6 Conclusion and Future Work

A novel process mining algorithm named $\alpha^{\$}$ is proposed. Using the *improved mendacious dependency* and *implicit dependency*, $\alpha^{\$}$ is the first algorithm which can adequately mine IT-in-NFC. Experiments show that $\alpha^{\$}$ can outperform the state-of-the-art mainstream process mining algorithms. The efficiency of $\alpha^{\$}$ is comparable to the fastest process mining algorithms by far.

Our future work would mainly focus on the following two aspects. One is to enhance mining capability of $\alpha^{\$}$ on process models with duplicated tasks. The

other one is to design a parallel and distributed process mining algorithm based on $\alpha^{\$}$ to handle huge event logs.

Acknowledgments. This work is supported by National Natural Science Foundation of China (No. 61472207, 61325008). The authors would like to thank Boudewijn van Dongen for his detailed and constructive suggestions.

References

1. van der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Software & Systems Modeling **9**(1), 87–111 (2010)
2. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.: Workflow mining: A survey of issues and approaches. Data & knowledge engineering **47**(2), 237–267 (2003)
3. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Mining and Knowledge Discovery **15**(2), 145–180 (2007)
4. Wen, L., Wang, J., van der Aalst, W.M.P., Huang, B., Sun, J.: Mining process models with prime invisible tasks. Data & Knowledge Engineering **69**(10), 999–1021 (2010)
5. van der Aalst, W.M.P., Weijters, A.J.M.M.: Process mining: a research agenda. Computers in industry **53**(3), 231–244 (2004)
6. Medeiros, A.K., Weijters, A.J., Aalst, W.M.: Genetic process mining: an experimental evaluation. Data Mining and Knowledge Discovery **14**(2), 245–304 (2007)
7. van der Aalst, W.M.P.: The application of Petri nets to workflow management. Journal of circuits, systems, and computers **8**(01), 21–66 (1998)
8. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering **16**(9), 1128–1142 (2004)
9. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W.E., Weijters, A.J.M.M.T., van der Aalst, W.M.P.: The ProM framework: a new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005)
10. Rozinat, A., van der Aalst, W.M.P.: Conformance testing: measuring the fit and appropriateness of event logs and process models. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 163–176. Springer, Heidelberg (2006)
11. Weijters, A.J.M.M., van der Aalst, W.M.P., De Medeiros, A.A.: Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP, 166, pp. 1–34 (2006)
12. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: van Hee, K.M., Valk, R. (eds.) PETRI NETS 2008. LNCS, vol. 5062, pp. 368–387. Springer, Heidelberg (2008)
13. Guo, Q., Wen, L., Wang, J., Ding, Z., Lv, C.: A universal significant reference model set for process mining evaluation framework. In: Ouyang, C., Jung, J.-Y. (eds.) AP-BPM 2014. LNBIP, vol. 181, pp. 16–30. Springer, Heidelberg (2014)

14. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007)
15. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Heidelberg (2011)
16. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **2**(2), 182–192 (2012)
17. Burattin, A., Sperduti, A.: PLG: a framework for the generation of business process models and their execution logs. In: Muehlen, M., Su, J. (eds.) BPM 2010 Workshops. LNBIP, vol. 66, pp. 214–219. Springer, Heidelberg (2011)