Mining process models with invisible tasks involved in non-free-constructs

ABSTRACT

The discovery of process models (i.e. process mining) from event logs 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 algorithms and tools that have appeared are known to have strong limitations in several dimensions. Mining invisible tasks involved in non-free-choice constructs is still one significant challenge nowadays. In this paper, we propose an algorithm named $\alpha^{\$}$. By introducing new ordering relations between tasks in the event log, $\alpha^{\$}$ is able to solve this problem. $\alpha^{\$}$ has been implemented as plug-ins in Business Process Management Tools BeehiveZ 3.5 and ProM 6. The experimental results show that it indeed significantly improves existing process mining techniques.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.2.8 [**Database Management**]: Data Mining; I.1.2 [**Symbolic And Algebraic Manipulation**]: Algorithms

General Terms

Data mining

Keywords

Process Mining, Non-free-choice Constructs, Missing Events

1. INTRODUCTION

Process mining is an essential discipline for addressing challenges related to Business Process Management (BPM) and "Big Data"[10]. 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 stored by information systems, and use this knowledge for supporting or improving the process perspective.

Process mining supports three scenarios relating event logs and process models: discovery of a process model from an event log, conformance checking given a process model and a log, and enhancement of a process model with the information obtained from a log. In this paper we focus on the scenario of discovery of Workflow-Net [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 [7]. The mined models should preserve all the tasks and the 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 [11] [12] [1]. In this paper, we focus on mining invisible tasks involved in non-free-choice constructs.

Invisible tasks are such tasks that appear in a process model, while not in its event log, which increase the difficulty of mining the proper process model. However, they are very common in real-life process models: the tasks in process models that are used for routing purpose only. The executions of such routing tasks are not recorded. There are also real tasks which have been executed but they are systematically lost in event traces. There are quite a number of real-life business processes containing invisible tasks. In Xiamen King Long United Automotive Industry Co., all of the deployed process models contain quite a few invisible tasks. [17].

The term *non-free-choice construct* originates from the Petri net domain. A Petri net with non-free-choice constructs is such one where there exists at least a pair of tasks sharing one input place, but their input places are not the same. Two tasks and the shared input place compose a choice structure. However, since the input places of these two tasks are not same, the choice of which task would execute is dependent on all the other input places. This situation is called the non-free-choice construct. Non-free-choice construct is difficult to discover because the choice is non-local, and the mining algorithm has to 'remember' all the earlier events. As mentioned in [16], non-free-choice construct is common but hard to deal with in process mining domain.

Invisible tasks involved in non-free-choice constructs means that a process model contains both invisible tasks and non-free-choice constructs. As mentioned earlier, both invisible tasks and non-free-choice constructs are difficult to discover

from event logs. Moreover, the combination of invisible tasks and non-free-choice constructs (namely, some tasks in non-free-choice construct are invisible ones) even increase the difficulty of mining.

α algorithm [8] is a pioneering process mining algorithm which mines the workflow net by considering relations between tasks in the event log. However, the relations considered in α are inadequate, invisible tasks and non-free-choice constructs cannot be properly mined by α. Based on α algorithm, α^{++} algorithm [16] adds a new relation called *implicit dependency*, which makes it able to mine the non-free-choice constructs. Similarly, $\alpha^{\#}$ algorithm is able to mine invisible tasks by considering a relation called *mendacious dependency*. However, none of these algorithms can properly mine invisible tasks involved in non-free-choice constructs.

Besides the aforementioned α-series algorithms, there are several other state-of-the-art mainstream process mining algorithms, such as *Genetic* [2], $Heuristic$ [15], ILP [13], and $Fuzzy$ [3]. Each of these algorithm has its own advantages and disadvantages. However, none of these algorithms is able to mine the invisible tasks involved in non-free-choice constructs.

In this paper, in order to mine invisible tasks involved in non-free-choice constructs, $\alpha^{\$}$ algorithm takes both *mendacious dependency* and *implicit dependency* into consideration. First, $\alpha^{\$}$ algorithm discovers invisible tasks by taking advantage of mendacious dependencies. Then, it complements necessary dependencies related to the found invisible tasks. Finally, it discovers the non-free-choice constructs and constructs the process model. Moreover, in this paper, we improve the mendacious dependency and implicit dependency, and $\alpha^{\$}$ algorithm repairs flaw of $\alpha^{\#}$ and α^{++} respectively.

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

2. A MOTIVATING EXAMPLE

Figure 1 is a real-life process model in SN company, which is the largest construction machinery manufacturer in China. This process model is about underground equipment repair process. According to this process, after *apply for repairing*, there are three choices for underground equipment to be repaired. One choice is that the equipment is repaired in the well(i.e., Task *Repair in well*). The equipment has to be repaired on the ground in other two options. In one option, the equipment should be decomposed (task *Decompose*) if it is too big to rise in the roadway to ground, while the equipment should not be decomposed in the other option. After the machine is repaired and moved back to the well, there is another choice(i.e., *P2*, *I2*, *Assemble*): whether assemble the machine or not. Unlike the previous choice, this choice is a non-free-choice. This is because that whether to assemble or not is dependent on whether the the machine has been decomposed before. This is implemented by P3 and P4, namely Assemble should be executed after Decompose is executed, I2 is executed after I1 is executed. In all, invisible task I2, together with task Assemble, place P2, P3, and P4 construct a non-free-choice construct.

For convenience, we use the label in the right top of each task as the abbreviation, for example A is short for *Apply for repairing*. Then {AF,ABDEGH,ACDEGI} is an example event log corresponding log of this model.

In spite of its apparent simplicity, this model and its corresponding log represents a hard case for most of the existing techniques. Table 2(c) is an example model similar to this one. None of the models mined by aforementioned mainstream can discover the process model properly. The $\alpha^{\#}$ algorithm mine a similar model, while the non-free-choice constructs are not discovered. The models mined by *Genetic* and $Heuristic$ are identical, with the invisible task not discovered. The model mined by α^{++} and ILP are not correct.

3. PRELIMINARIES

In this section, we give some definitions used throughout this paper. Firstly, we discuss the event log in detail and give an example. Then we give the WF-net and its relevant concepts. Finally, we introduce the notations of process mining.

3.1 Event log

The goal of process mining is to extract information about processes from event logs. The *event*, which represents a real-life event, is the basic unit of event logs. As defined in 3.1, each event has several attributes, such as the timestamp(i.e., when the event happens), activity (i.e., what task this event corresponds to), and so on.

Definition 1. (Event, Attribute) Let ϵ be *event universe*, i.e., the set of all possible event identifiers. Events may be characterized by various *attriutes*, e.g, an event may have a timestamp, correspond to an activity, br executed by a particular person, have associated costs, etc. Let AN be a set of attribute names. For any event $e \in \epsilon$ and name $n \in AN$:$\#_n(e)$ is the value of attribute n for event e. If event e does not have an attribute named n, then $\#_n(e) = \perp$ (null value).

For convenience we assume the following standard attributes:

- $\#_{activity}(e)$ is the task associated to event e.
- $\#_{time}(e)$ is the time associated to event e.

An event log consists of cases. Each case means a possible event sequence. 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 have attributes.

Definition 2. (Case, Trace, Event log) Let ξ be the case universe, i.e., the set of all possible case identifiers. Cases, like events, have attributes. For any case $c \in \xi$ and name $n \in AN$:$\#_n(e)$ is the value of attribute n for case c ($\#_n(c) = \perp$ if case c has no attribute named n) Each case has a special

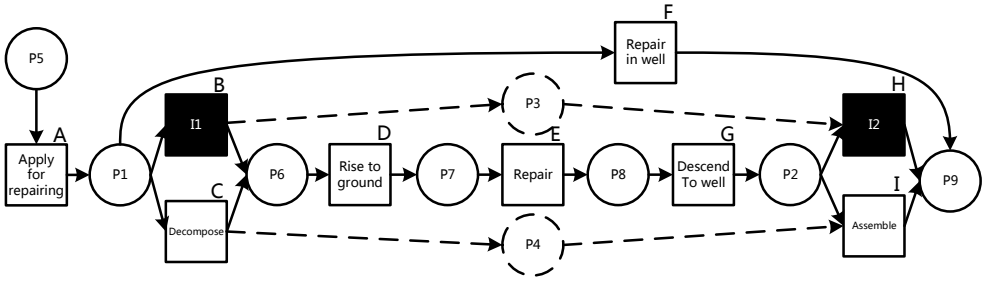


Figure 1: A process model in SY company

mandatory attribute $trace : \#_{trace}(c) \in \epsilon^*$. $c = \#_{trace}(c)$is a shorthand for referring to the trace of a case.

A *trace* is a finite sequence of events $\sigma \in \epsilon^*$ such that each event appears only once, i.e., for any $1 \leq i < j \leq |\sigma| : \sigma(i) \neq \sigma(j)$, where $|\sigma|$ means the number of events contained in trace σ.

An *event log* is a set of cases $L \subset \xi$ such that each event appears at most once in the entire log, i.e., for any $c_1, c_2 \in L$ such that $\nexists_{e \in L} e \in c_1 \wedge e \in c_2$.

If an event log contains timestamps, then the ordering in a trace should respect these timestamps, i.e., for any $c \in L$, i and j such that $1 \leq i < j \leq |c| : \#_{time}(c(i)) \leq \#_{time}(c(j))$.

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

Table 1 gives an example of an event log of Figure 1. This log contains information of four traces, the events in each trace are not identical with each other(i.e., no two events share the same *event id*) In this paper, for purpose of process mining, we only consider the *activity* attribute and the time order between events. Namely, for trace 1 and trace 4, activity A, and F are executed. For trace 2, activity A, B, D, E, G, and H are executed. For trace 3, activity A, C, D, E, G, and I are executed. The event log in Table 1 can be shortened as $\{AF^2, ABDEGH, ACDEGI\}$, where 2 means the trace A,F occurs twice. In fact, no matter how many traces appear in the event log, there are only three distinct activity traces, i.e., AF, ABDEGH and ACDEGI. Thus for the process model shown in Figure 1, this event log is a complete one.

As mentioned in [12], 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 [14] to solve it. The plugin *Filter Log with Simple Heuristic* is used for abating noise. Thus, in the subsequent part of this paper, we assume that the event log has no noise.

Table 1: An example event log of Figure 1

Case Id	Event Id	properties	
		timestamp	activiy
1	35654422	30-12-2010:11.02	A
	35654423	31-12-2010:13.32	F
2	35654481	01-01-2011:01.02	A
	35654483	01-01-2011:13.24	D
	35654484	02-01-2011:03.22	E
	35654485	02-01-2011:21.22	G
3	35654579	03-02-2011:01.12	A
	35654580	03-02-2011:03.23	C
	35654581	03-02-2011:04.53	D
	35654582	03-02-2011:01.23	E
	35654583	03-02-2011:03.04	G
	35654584	03-02-2011:03.04	I
4	3565442	30-12-2010:11.02	A
	3565443	31-12-2010:13.32	F

3.2 Workflow net

In this paper, *Workflow net* [9] is used as the process modelling language. Workflow nets as defined in Definition 5, are a as subset of labeled Petri nets. Each WF-net has a unique source place and a unique sink place and every other node is on the path from the source place to the sink place.

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 $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the flow relation. A marked Petri net is a pair (N, M), where $N = (P, T, F)$ is a Petri net and $M \in \mathcal{B}(P)$ is a multi-set over P denoting the marking of the net. The set of all marked Petri nets is denoted $\mathcal{N}$.

Definition 4. (Labeled Petri Net) A labeled Petri net is a five-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 mapping from T to $A \cup \{\tau\}$ (τ represents a transition not visible to the outside world.)

Definition 5. (Workflow net). Let $N = (P, T, F, A, l)$ be a labeled Petri net and t be a fresh identifier not in $P \cup T$. N is a workflow net(WF-net) if and only if (a) P contains an input place i (also called source place) such that $\cdot i = \emptyset$, (b) P contains an output place o (also called sink place) such that $i \cdot = \emptyset$, and (c) $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\}, A \cup \{\tau\}, l \cup \{(t, \tau)\}$ is strongly connected, i.e., there is a directed path between any pair of nodes in $\bar{N}$.

Figure 1 gives an example of a process modelled in WF-net. This model has an invisible task involved in a non-free-choice construct. The transitions (drawn as rectangles) $A, B, C, ..., I$ represent tasks, where hollow rectangles represent visible tasks, and the solid one (i.e. B or H) represents an invisible task. The places (drawn as circles) $P0, P1, P2, ..., P8$ represent conditions. The arcs (drawn as directed edges) between transitions and places represent flow relations. As mentioned before, there is a non-free-choice construct in this model, i.e., the sub-construct composed of $P3$, $P4$, $P2$, H, I, and relevant edges.

We adopt the formal definitions, properties, and firing rules of WF-net from [9] [8]. For mining purpose, we demand that each task (i.e., transition) has a unique name in one process model. Of course, each task can appear multiple times in one case for the presence of the iterative routings.

3.3 Process Mining

The problem of process mining requires the computation of a workflow model M that adequately represents a given event log L. Namely, process mining is to construct a process model like the one shown in Figure 1, based on an event log like the one shown in Table 1.

4. THE NEW MINING ALGORITHM $\alpha^{\$}$

Taking an event log as the input, the mining algorithm $\alpha^{\$}$ constructs a workflow net as the output. It is composed of five basic 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 the reachable dependencies related to the found invisible tasks. Then, it discovers the non-free-choice constructs 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[8] [17] [16], it would not be elaborated. Initially, some basic relations are introduced, then the following subsections elaborate the first four basic steps respectively.

4.1 Basic relations

α algorithm, which epoches the α-series algorithms, defined four relations, $>_W$, $\rightarrow_W$, $||_W$, and $\#_W$. These four relation lays the foundation for constructing process model. The original definition of these relations in α cannot distinguish short-loop between parallel constructs. $\alpha^{\#}$ algorithm solve this issue by introducing two new relations $\triangle_W$ and $\diamond_W$, as Defined in Definition 6. $>_W$, and $\triangle_W$ are the fundamental ordering relations. The other relations are derived from them. $>_W$ reflects that two tasks can be executed successively. $\triangle_W$ means a loop structure. $\rightarrow_W$ is referred as the (direct) casual relation derived from event log. Relation $||_W$ suggests the concurrent behaviour, namely two activity can be executed at the same time. Relation $\#_W$ gives pairs of tasks that never follows each other directly. Relation $\diamond_W$ shows two tasks have the $\diamond_W$ relations between each other.

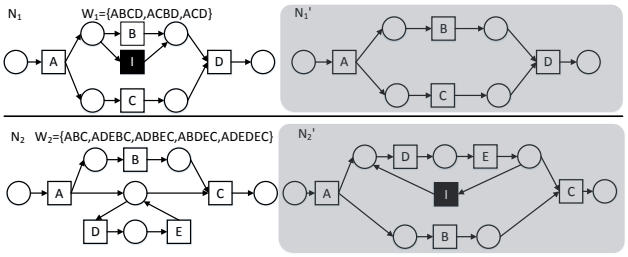


Figure 2: Two examples for the defect of mendacious dependency

For example, in the event log in Table 1, $A > F$, $A > C$, $A \rightarrow F$,$G \rightarrow I$ holds.

Definition 6. (Relation defined in α algorithm, Mendacious dependency, Reachable dependency) Let T be a set of task, W be an event log over T, a and b are tasks in T, the relation defined in α algorithm, mendacious dependency, and reachable dependency are defined as follows:

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

In $\alpha^{\#}$ algorithm, *mendacious dependency* $\rightsquigarrow$ is proposed to describe the relations reflecting invisible tasks. The mendacious dependency is based on six pre-conditions as defined. $A \rightsquigarrow D$ holds in the event log in Table 1, because $A \rightarrow C$, $C \rightarrow D$, $C \not> D$, , $C \nparallel D$, $A \rightarrow C$ it means there should be an invisible task between A and D. (i.e., the invisible task B in Figure 1.

reachable dependency is used to depict the indirect dependency between activities. In α^{++}, reachable dependency is a necessary condition for discovering non-free-choice constructs. For example, in process model in Figure 1, $E \gg G$, $C \gg I$, and $D \succ G$, and so on.

4.2 Detecting invisible tasks with improved mendacious dependency

The aim of this step is to discover the invisible tasks from the given event log. Most invisible tasks can be detected by applying the *mendacious dependency* proposed in [17]. However, the mendacious dependency cannot deal with the invisible tasks involved in one branch of a parallel construct. Thus, we propose an *improved mendacious dependency* to resolve this issue.

Two models in Figure 2 show the defect of the mendacious dependency in dealing with invisible tasks involved in the parallel construct. N_1 and N_2 are the original models, N_1' and N_2' are the models mined by the $\alpha^{\#}$ algorithm. In N_1', $\alpha^{\#}$ didn't discover the invisible task I. This is because that $A \rightarrow D$, 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 $A \not\succ C$ 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., the trace ABC cannot be replayed on N_2'.

In order to the overcome this defect, we introduce the definition of *Between-Set*. Then we define the *improved mendacious dependency*.

Given a set of set T, an event log W over T, and two task a, b from T, *Between-Set*, which is defined in Definition 7, is to depict the tasks that occur between task a and task b. When the two task are the endpoints of a parallel construct, the Between-Set is the set of tasks in the parallel branches. For example, $Between(W_1, A, D) = \{B, C\}$, $Between(W_2, A, C) = \{B, D, E\}$.

Definition 7. (Between-Set) Let T be a set of task, W be an event log over T, a and b are tasks in T, σ is 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:

- $Between(\sigma, a, b) = \{\sigma_k | \forall_{1 \leq i \leq n} \forall_{i<j \leq n}((i < k < j) \wedge \nexists_{i<l<j}(\sigma_l = a \vee \sigma_l = b))\}$
- $\neg Between(\sigma, a, b) = \{\sigma_k | 1 \leq k \leq n\} - Between(\sigma, a, b)$
- $Between(W, a, b) = \cup_{\sigma \in W} Between(\sigma, a, b) - \cup_{\sigma \in W} \neg Between(\sigma, a, b)$

The *improved mendacious dependency* is defined in Definition 8. By using the notation of Between-Set, we redefine $\rightarrow$ and $>$ in [17] as $\rightrightarrows$ and $\geqslant$ respectively. Compared with the old ones, $\rightrightarrows$ and $\geqslant$ are able to eliminate the interference of parallel constructs. For instance, in N_1, $A \rightarrow D$ does not holds. However, $A \rightrightarrows_{W1,B,C} D$ holds. In N_2, $A \not\succ C$ 'improperly' holds. Nevertheless, $A \geqslant C$ holds. In all, by using the improved mendacious dependency, the invisible tasks involved in parallel constructs can be properly mined.

Definition 8. (Improved Mendacious Dependency) Let T be a set of task, W be an event log over T, a,x,y,b are four tasks from T, the *improved mendacious dependency* $\rightsquigarrow$ is defined as follows:

- $a \geqslant_{W,x,y} b \iff Between(W, x, y) \subset Between(W, a, b) \wedge \forall_{m \in (Between(W,a,b) - (Between(W,x,y) \cup \{x,y\}))} \forall_{n \in Between(W,x,y)} m \parallel_W n \wedge \exists_{\sigma \in W} Between(\sigma, a, b) \subseteq (Between(W, a, b) - (Between(W, x, y) \cup \{x, y\}))$
- $a \rightrightarrows_{W,x,y} b \iff a \geqslant_{W,x,y} b \wedge (b \not\geqslant_{W,x,y} a \vee a \diamond_W b)$
- $a \rightsquigarrow_{W,x,y} b \iff a \rightarrow_W x \wedge y \rightarrow_W b \wedge x \nparallel_W b \wedge y \nparallel_W a \wedge a \rightrightarrows_{W,x,y} b \wedge y \not\geqslant_{W,b,a} x$

4.3 Complementing reachable dependencies

Reachable dependency [16] which is labeled as $\succ$, is such a relation that one task can be directly or indirectly followed by another task. Complete reachable dependencies over the given log is required for discovering non-free-choice constructs. The *long-distance dependency* ($>>$) [16], which expresses indirect following relation, is one requirement on discovering reachable dependencies. A long-distance dependency between two tasks holds, if there is a trace where a event of one task occurs indirectly after the other one. However, since invisible tasks do not appear in the given event log, long-distance dependencies and reachable dependencies related to invisible tasks are missing for discovering non-free-choice constructs. For example, in Figure 1, the part with solid lines are the model mined by $\alpha^{\$}$ without this complementing step. Non-free-choice construct (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* (we called it CRD for short). Symbol $a >>_\sigma b$ to express that task a is indirectly followed by task b in the 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 $\#_{activity}(\sigma_{n+1}) = \top$.

As defined 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. $\succ_{W,Pre=x,Post=y}$). As for both-CRD, $a \succ_{W,Pre=x,Post=y} b$ means that there is a trace where $a >>_\sigma b$ holds, and x occurs directly before a, y occurs directly after b. For example, in Figure 1, $T3 \succ_{W,Pre=\perp,Post=\top} T4$ holds. The pre-CRD and post-CRD are special cases of both-CRD, where the pre or the post can be any events, $\perp$, or $\top$.

Definition 9. (Conditional Reachable dependency) Let T be a set of task, W be an event log over T, a,b be two tasks from W, x,y be two tasks from W, $\perp$, or $\top$. the *Conditional Reachable dependency* $\succ_{W,Pre=x}$, $\succ_{W,Post=y}$, and $\succ_{W,Pre=x,Post=y}$ 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 $a_1 \rightarrow x$, $x \rightarrow b_1$, $a_2 \rightarrow y$, $y \rightarrow b_2$, 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 $a \rightarrow x$, $x \rightarrow b$, and $b \succ_{W,Pre=a} m$ holds. For instance, the process model in Figure 1, $T1 \succ_W T4$ holds, because $\perp \rightarrow T1$, and $T3 \succ_{W,Pre=\perp} T4$ and $T1 \rightarrow T3$. $m \succ_W x$ is similar to $x \succ_W m$, and it is not necessary to go into details.

Definition 10. (Reachable dependency related to Invisible task) Let T be a set of task, W be an event log over T, m

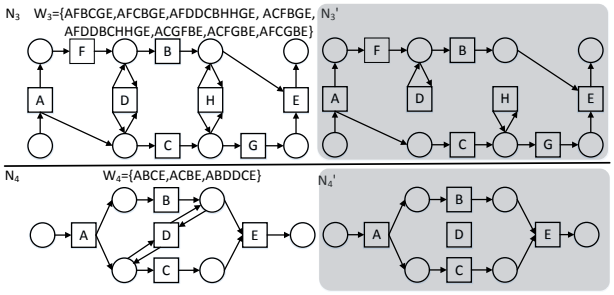


Figure 3: Defect of α^{++} on mining L1L involved in non-free-choice constructs

is a task from T, x,y are two invisible tasks, the *Reachable dependency related to Invisible task* is defined as follows:

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

4.4 Detecting non-free-choice constructs

After making the reachable dependencies complete in the previous step, the aim of this step is to discover non-free-choice constructs. The α^{++} algorithm can mine non-free-choice constructs in most cases.

However, α^{++} is not able to mine the length-1-loop construct (L1L for short) involved in non-free-choice constructs. L1L set, as defined in Definition 11, is a set of tasks, where each task appears in at least continuous events in an given event log. α^{++} excludes all tasks in L1L set when considering implicit dependencies, which impedes discovering the L1L involved in non-free-choice constructs. 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 non-free-choice constructs combined with L1L, N_3' and N_4' are models mined by α^{++}. The non-free-choice construct is not detected in N_3', which makes N_3' have more behavior than N_3: trace $ACGFDBE$ 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 leads N_4' not sound at all.

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:

- $L1L = \{t \in T | \exists_{\sigma = t_1 t_2 \ldots t_n \in W; i \in 1,2,\ldots,n} t = \#_{activity}(t_{i-1}) \wedge t = \#_{activity}(t_i)\}$

Thus, in this step, a set of tasks (called the L1L-Free set) is determined before applying the α^{++} algorithm, in which the tasks should not be excluded when detecting non-free-choice constructs.

The L1L-Free set is summarized in Definition 12. Namely, for each task x of L1L, there exists a pair of tasks a, b parallel with each other. Besides, two *sequence relations* $a \rightarrow x$ and $x \rightarrow b$ hold. Then task x should not be excluded when detecting the non-free-choice constructs. For example, the L1L-Free set of N_3 is {D,H}. For task D, $F \rightarrow D$, $D \rightarrow C$, and $F||C$ hold. Likewise, task H of N_3 and task D of N_4 should be in their corresponding L1L-Free set.

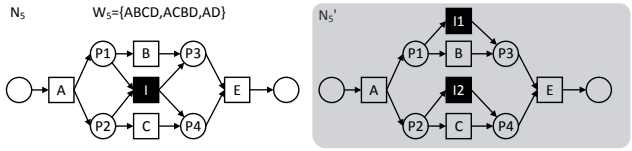


Figure 4: An example of a mined process model without combining

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

- L1L-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 the sound and accurate process model, this step adjusts the invisible tasks by combining or splitting.

Before introducing the details of invisible task adjustment, some basics are given. There are four functions in the basics on adjusting invisible task:

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

4.5.1 Combining invisible tasks

When there are invisible tasks in different branches of a parallel construct, there is a possibility that the invisible tasks should be combined together. However, the $\alpha^{\sharp}$ algorithm does not take this situation into consideration. For example, N_5 in Figure 4 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 behaviour than N_5, such as trace ACE and ABE. Thus, in order to make the mined model more concise, this step is to combine necessary invisible tasks into one. We use the method defined in Definition 13 to discover the pairs of combinable invisible tasks.

Definition 13. (Combinable invisible tasks) Let T is a set of Tasks, W is 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:

- $R(t) = \{z | (a, x, y, b) \in MD(t) \wedge (z = x \vee z = y)\}$
- $P(\sigma, a, b) = \sigma \uparrow (R(a) \cup R(b))$

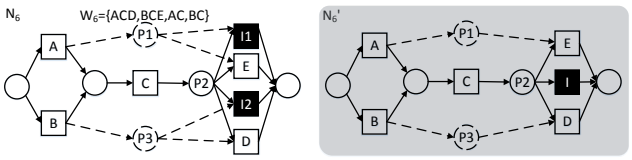


Figure 5: An example of a mined process model without splitting

- $a \bigotimes_\sigma b \iff |P(\sigma,a,b)| = 0 \vee (|P(\sigma,a,b)|\%2 = 0 \wedge \forall_{1\le k<|P|/2}((P(\sigma,a,b)_{2k+1} \in R(a) \wedge P(\sigma,a,b)_{2k+2} \in R(b)) \vee (P(\sigma,a,b)_{2k+1} \in R(b) \wedge P(\sigma,a,b)_{2k+2} \in R(a)))$
- $a \bigotimes_W b \iff \forall_{\sigma\in W} a \bigotimes_\sigma b$
- Combinable_Set(W,T_I) = $\{(a,b)|a \parallel b \wedge a \in T_I \wedge b \in T_I \wedge a \bigotimes_W b\}$

$R(t)$ 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.

4.5.2 Splitting invisible tasks

Non-free-choice constructs are detected after the discovering 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 5 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', the invisible task I is found earlier than the non-free-choice construct (P1, P2, P3, E, D). Place P1, P3 and the dotted edges are added for constructing the non-free-choice construct. The new-added dependencies (A,E) and (B,D) make the net not sound, for example, after the execution of ACI or BCI, there would be a remaining token in P1 or P2. 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.

- $IMD(t) = \{(a,x,y,b) \in MD(t) | \exists (m,n) \in ID : (n = x \vee m = y)\}$
- 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 splited if $|IMD(t)| > 1$.

5. EXPERIMENTAL EVALUATION

The $\alpha^\$$ algorithm has been implemented as plug-ins in both ProM [14] and BeehiveZ [5], which can be download from their websites respectively [1] [2].

5.1 Evaluation criteria

Although the visual inspection of the mined process model and the original process model is an intelligible method to see whether the mining result is correct, it works well only for a small amount of process models with simple structures. Therefore, the following two metrics are introduced, which are able to evaluate mining algorithms for large quantity of complex process models.

- The metric *fitness* (f for short) [6] 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 a řtoken gameą́s is played as suggested by the log. In other words, fitness reflects the extent to which the log traces can be associated with execution paths specified by the process model. Thus if $f = 1$ then the log can be parsed by the model without any error.
- The metric *model rediscoverability* [4] tests the conformance between the mined model and the original model. Given a process model and its corresponding event log, 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.

5.2 Evaluation based on smaller logs

Before showing large real-life models, we first focus on smaller artificial examples that demonstrate the fact that $\alpha^\$$ significantly improves existing approaches including α^{++}, $\alpha^\#$, *Genetic*, *Heuristic*, and ILP. To illustrate the capabilities of the $\alpha^\$$ algorithm, we first show some concrete experimental results.

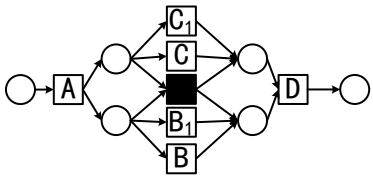
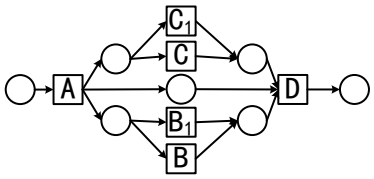
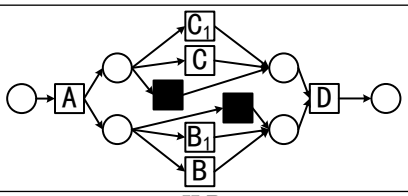
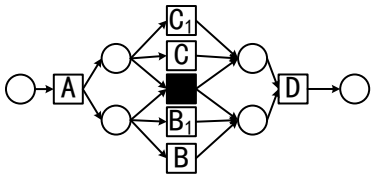
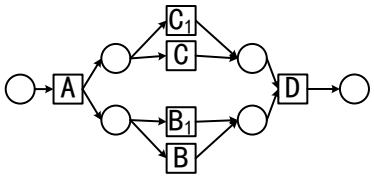
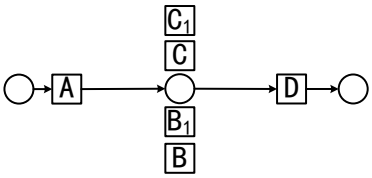
Table 2(a) is an example of an invisible task involved in a parallel construct. In the ground truth model, two parallel branches share the same invisible task. The $\alpha^\$$ and *Genetic* algorithms managed to mine the proper process model. The α^{++} and Heuristic algorithms failed to discover the invisible task, which makes the trace AD cannot be replayed on the mined models. The $\alpha^\#$ algorithm managed to mine two invisible tasks, but failed to combine these two invisible tasks into one. Thus, the process model mined by $\alpha^\#$ has more behavior than the ground truth, for example traces ACD, AB_1D can be replayed on the model mined by $\alpha^\#$ while cannot be replayed on the ground truth. The ILP algorithm cannot mine a sound process model, there are 4 tasks without flows (i.e. C_1, C, B, B_1) in the mined model. Besides, ILP also cannot mine proper process models for other examples in Table 2, which will not be elaborated then.

The ground truth model in Table 2(b) has two invisible tasks combined with non-free-choice constructs. The $\alpha^\$$ algorithm managed to discover the identical process model with

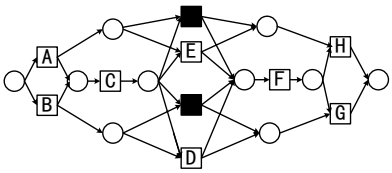
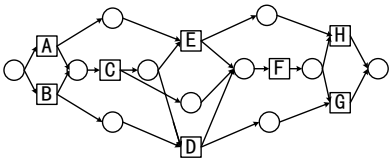
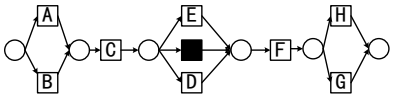
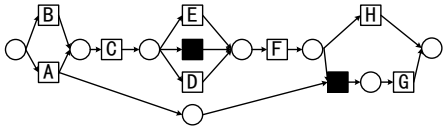
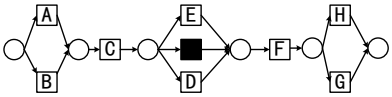
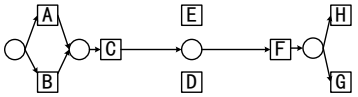
[1] http://www.promtools.org/prom6/prom63.html
[2] https://code.google.com/p/beehivez

Table 2: Four example process models

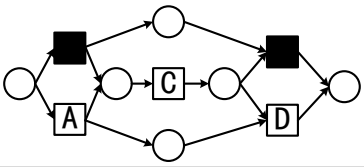
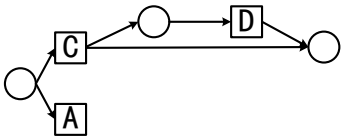
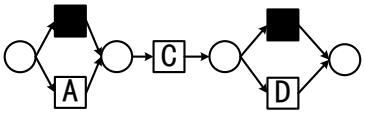
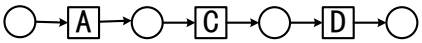
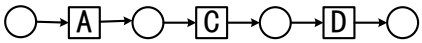
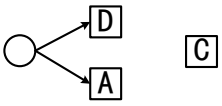
(a)

Log	{ABCD, ACBD, AB_1CD, ABC_1D, AC_1BD, ACB_1D, AC_1B_1D, AB_1C_1D, AD}	
GroundTruth & $\alpha^{\$}$	α^{++}	$\alpha^{\#}$
		
Genetic	Heuristic	ILP
		

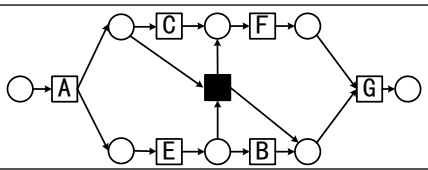
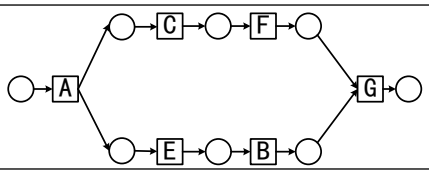
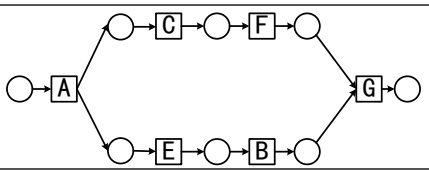
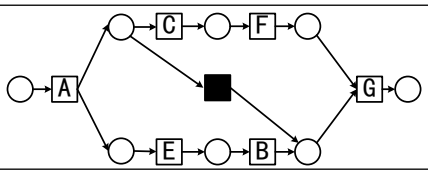
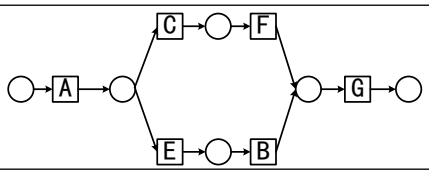
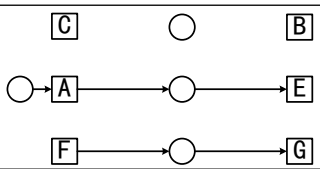
(b)

Log	{BCDFG, ACEFH, BCFG, ACFH}	
GroundTruth & $\alpha^{\$}$	α^{++}	$\alpha^{\#}$
		
Genetic	Heuristic	ILP
		

(c)

Log	{ACD, C}	
GroundTruth & $\alpha^{\$}$	α^{++}	$\alpha^{\#}$
		
Genetic	Heuristic	ILP
		

(d)

Log	{AEFG, AEBCFG, AECBFG, AECFBG, ACEFBG, ACFEBG, ACEBFG}	
GroundTruth & $\alpha^{\$}$	α^{++}	$\alpha^{\#}$
		
Genetic	Heuristic	ILP
		

the ground truth. The models mined by $\alpha^{\#}$, *Heuristic* and *Genetic* failed to detect all non-free-choice constructs, and these three mined models have more behavior than the ground truth. The α^{++} algorithm failed to discover the invisible task, and the traces *BCFH*, *ACFG* cannot be replayed on the mined model.

Table 2(c) is an example of invisible tasks involved in non-free-choice constructs. The α^{++}, *Genetic* and *Heuristic* algorithms failed to mine the invisible task. Two invisible tasks are detected by the $\alpha^{\#}$ algorithm, while the non-free-choice construct is not discovered. This leads to the model mined by $\alpha^{\#}$ has more behavior (such as trace *ACD*) than the ground truth.

The ground truth model in Table 2(d) has one invisible task. The α^{++}, $\alpha^{\#}$ and *Heuristic* algorithms failed to discover the invisible task. Besides, the process model discovered by the *Genetic* algorithm is not sound.

There are 70 process models in the smaller data set, 60 of which are from the *general significant reference model set* proposed in [4]. Besides, since the general significant reference model set does not contain process models with non-free-choice constructs combined with invisible tasks, ten artificial models with such feature are supplied in the experimental data set to demonstrate the capabilities of the $\alpha^{\$}$ algorithm.

Table 3 shows the time expense on evaluation. As for the smaller logs, $\alpha^{\$}$ costs more than the cost of either $\alpha^{\#}$ or α^{++}, yet less than the sum of $\alpha^{\#}$ and α^{++}. Compared with *Genetic*, the time cost of $\alpha^{\$}$ is much smaller. Time cost for mining algorithms on evaluating real-life logs is similar with the one on smaller logs. Although the real-life logs are more complicated than the smaller ones, the number of the real-life logs is smaller than that of the smaller logs.

Figure 6(a) shows the fitness result of different process mining algorithms. Since $\alpha^{\$}$ is based on$\alpha^{\#}$ and α^{++}, $\alpha^{\$}$ outperforms the previous two algorithms. For the clearness of the figure, the result of $\alpha^{\#}$ and α^{++} are not listed. For the same reason, the result of α^{++} and $\alpha^{\#}$ are not listed in the Figure 6(b), Figure 8(a), and Figure 8(b). The $\alpha^{\$}$ algorithm has a better fitness result than all the other algorithms. All the models mined by $\alpha^{\$}$ have a fitness of 1 except the model *Dup2*. As shown in Figure 7, there are two pairs of duplicate tasks in *Dup2*, which obstructs $\alpha^{\$}$ from discovering the proper model. Still, the fitness of model *Dup2* mined by $\alpha^{\$}$ is higher than the ones mined by other mining algorithms.

Model rediscoverabilty result is shown in Figure 6(b). Except the process models with duplicate tasks, the $\alpha^{\$}$ algorithm 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. Besides, as for the models with duplicate tasks, only the model *Dup2* made $\alpha^{\$}$ perform less model rediscoverability than other algorithms.

5.3 Evaluation based on real-life logs

The 40 real-life models are composed from *DG Boiler Group Co., Ltd. (DG for short)* and *TS Railway Vehicle Co., Ltd. (TRC for short)*. This real-life model set is public accessible[3]. *DG* is a major supplier of thermal power equipment, nuclear power equipment, power plant auxiliaries, chemical vessels, environmental protection equipment and coal gasification equipment in China (with more than 33% market share). TRC is the oldest enterprise in China's railway transportation equipments manufacturing industry.

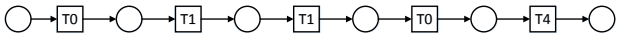


Figure 7: Model Dup2 with two pairs of invisible tasks

Figure 8(a) and 8(b) respectively show the fitness and model rediscoverability results on the real-life process models. The results in Figure 8(a) show that the $\alpha^{\$}$ algorithm is performing very well on these real-life examples, (i.e. all mined models have a fitness of 1). Except the $\alpha^{\#}$ algorithm, the fitness of models mined by other process mining algorithms are not all 1.

Figure 8(b) shows that $\alpha^{\$}$ has a higher model rediscoverability result than all the other algorithms.

5.4 Discussion

From the experiments done in this section, we can see the $\alpha^{\$}$ algorithm is practical on mining non-free-choice constructs from event logs with missing events.

One drawback of the $\alpha^{\$}$ algorithm is its inadequate behaviour on mining duplicate tasks. From the experiments, we can see that several models with duplicate tasks cannot be mined properly by $\alpha^{\$}$. Even so, it is obvious that $\alpha^{\$}$ is a good process mining algorithm, and outperforms the state-of-the-art mainstream process mining algorithms.

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 non-free-choice constructs from event logs with missing events. Experiments using the artificial and the real-life models show that $\alpha^{\$}$ can outperform the state-of-the-art mainstream process mining algorithms. The efficiency of $\alpha^{\$}$ is quite close to the fastest process mining algorithms by far.

Our future work would mainly focus on the following two aspects. One is to enhance $\alpha^{\$}$'s mining capability 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.

7. REFERENCES

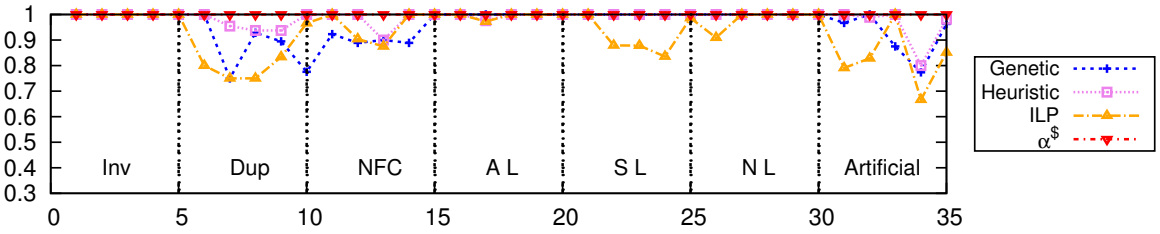
[1] A. K. A. de Medeiros, W. M. van der Aalst, and A. Weijters. Workflow mining: Current status and future directions. In *On the move to meaningful internet systems 2003: Coopis, doa, and odbase*, pages 389–406. Springer, 2003.

[2] A. K. A. de Medeiros, A. J. Weijters, and W. M. van der Aalst. Genetic process mining: an

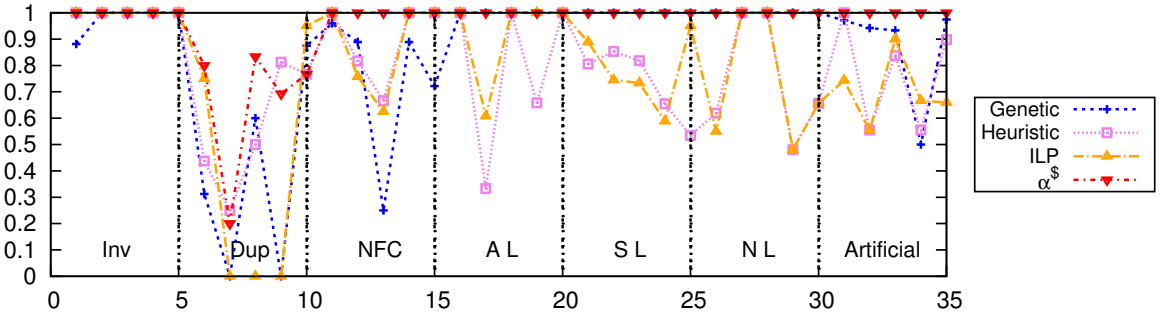
[3] http://laudms.thss.tsinghua.edu.cn/trac/Test/attachment/wiki/chengguo/real_data.rar

Table 3: Time expense on experiments in seconds

Mining algorithm	$\alpha^{\$}$	α^{++}	$\alpha\#$	Genetic	Heuristic	ILP
smaller logs	34.42	19.06	24.84	344.968	22.20	49.96
Real-life logs	35.21	23.32	28.83	290.91	23.00	32.42

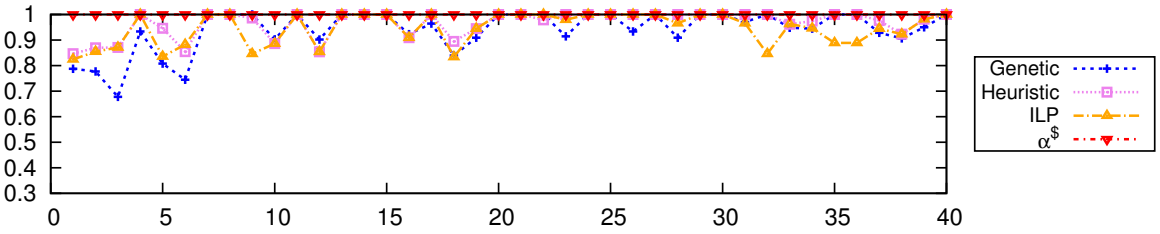


(a) Fitness

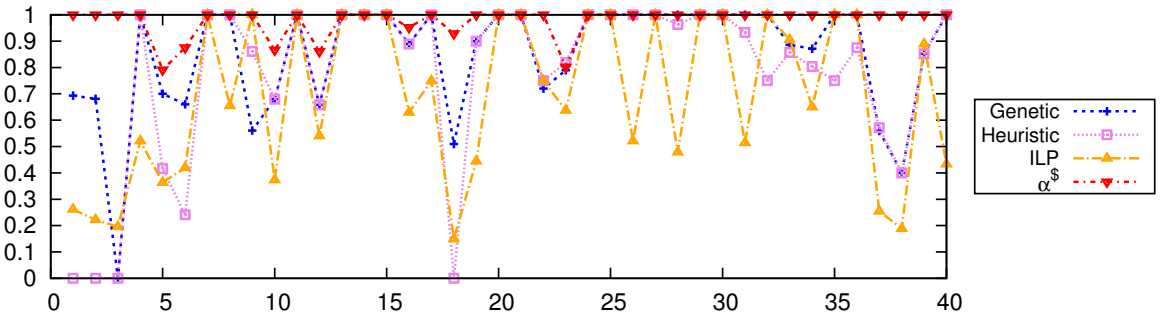


(b) Model rediscoverability

Figure 6: Experiment result on smaller logs



(a) Fitness result



(b) Model rediscoverabilty

Figure 8: Experiment result on real-life logs

experimental evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.

[3] C. W. Günther and W. M. Van Der Aalst. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *Business Process Management*, pages 328–343. Springer, 2007.

[4] Q. Guo, L. Wen, J. Wang, Z. Ding, and C. Lv. A universal significant reference model set for process mining evaluation framework. In *2nd AP-BPM*, 2014.

[5] T. Jin, J. Wang, and L. Wen. Efficiently querying business process models with beehivez. In *BPM (Demos)*, 2011.

[6] A. Rozinat and W. M. van der Aalst. Conformance testing: Measuring the fit and appropriateness of event logs and process models. In *Business Process Management Workshops*, pages 163–176. Springer, 2006.

[7] G. Schimm. Mining exact models of concurrent workflows. *Computers in Industry*, 53(3):265–281, 2004.

[8] W. Van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *Knowledge and Data Engineering, IEEE Transactions on*, 16(9):1128–1142, 2004.

[9] W. M. van der Aalst. The application of petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66, 1998.

[10] W. M. Van der Aalst. *Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[11] W. M. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters. Workflow mining: A survey of issues and approaches. *Data & knowledge engineering*, 47(2):237–267, 2003.

[12] W. M. Van der Aalst and A. Weijters. Process mining: a research agenda. *Computers in industry*, 53(3):231–244, 2004.

[13] J. M. E. van der Werf, B. F. van Dongen, C. A. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. In *Applications and Theory of Petri Nets*, pages 368–387. Springer, 2008.

[14] B. F. van Dongen, A. K. A. de Medeiros, H. Verbeek, A. Weijters, and W. M. Van Der Aalst. The prom framework: A new era in process mining tool support. In *Applications and Theory of Petri Nets 2005*, pages 444–454. Springer, 2005.

[15] A. Weijters, W. M. van der Aalst, and A. A. De Medeiros. Process mining with the heuristics miner-algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP*, 166, 2006.

[16] L. Wen, W. M. van der Aalst, J. Wang, and J. Sun. Mining process models with non-free-choice constructs. *Data Mining and Knowledge Discovery*, 15(2):145–180, 2007.

[17] L. Wen, J. Wang, W. M. van der Aalst, B. Huang, and J. Sun. Mining process models with prime invisible tasks. *Data & Knowledge Engineering*, 69(10):999–1021, 2010.