International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-1, November 2019

Detecting Complex Control-Flow Constructs for Choosing Process Discovery Techniques

Hind R'bigui, Mohammed Abdulhakim Al-Absi, Chiwoon Cho

Abstract*— Process models are the analytical illustration of an organization's activity. They are very primordial to map out the current business process of an organization, build a baseline of process enhancement and construct future processes where the enhancements are incorporated. To achieve this, in the field of process mining, algorithms have been proposed to build process models using the information recorded in the event logs. However, for complex process configurations, these algorithms cannot correctly build complex process structures. These structures are invisible tasks, non-free choice constructs, and short loops. The ability of each discovery algorithm in discovering the process constructs is different. In this work, we propose a framework responsible of detecting from event logs the complex constructs existing in the data. By identifying the existing constructs, one can choose the process discovery techniques suitable for the event data in question. The proposed framework has been implemented in ProM as a plugin. The evaluation results demonstrate that the constructs can correctly be identified.

Keywords*— Process models, process discovery, event log, process structure, recommendation.

I. INTRODUCTION

Process mining is the latest technologies which fall between machine learning, data mining, and analysis. Process mining provides a strong bridge among Business Process Management, process modelling, and Business Intelligence by combining both event data and process models forming a new form of process-driven analytics. Moreover, Process mining enables and strengthen the Business Process Improvement approaches such Six-sigma, CPI, TQM…, where processes are investigated to explore possible improvements [1]. Current information systems such as workflow management systems (WFM), supply chain management (SCM) systems, enterprise resource planning (ERP) systems, and Manufacturing Operation Management (MOM) systems store business-related events in so-called event logs [2].

Revised Manuscript Received on November 30, 2019.
***** Correspondence Author
Hind R'bigui, Digital Enterprise Department, Nsoft, CO., LTD, Ulsan, 44776, Republic of Korea. Email: hind.rbigui@gmail.com
Mohammed Abdulhakim Al-Absi, Department of Computer Engineering, Graduate School, Dongseo University, 47 Jurye-ro, Sasang-gu, Busan 47011, Republic of Korea. Email: mohammed.a.absi@gmail.com
Chiwoon Cho*, School of Industrial Engineering, University of Ulsan, Ulsan 680-749, Republic of Korea. Email: chiwoon6@mail.ulsan.ac.kr

Usually these event logs contain information about the operations or tasks that have been executed at the organization, machines, the time of the execution of tasks, the people, or systems handling those activities and other data. Process mining automatically Builds a representation of the behaviour of business processes of the organization and extracts knowledge from these events logs. The main purpose is to find deviations, bottlenecks and other issues that prevent the enterprise from achieving its strategic goal. These event logs allow three major tasks of process mining techniques to be performed for different aims: process discovery, conformance checking, and performance analysis [2]. In process mining, process discovery is the most important technique. This techniques task takes an event log as input and automatically creates a process model which shows how the business process is behaving. Conformance checking techniques compare an existing process model with the reference model of an enterprise or the discovered model with the corresponding process event data. The aim of conformance checking techniques is to investigate that what is happening in the organization conforms to the a-priori process model. Process enhancement techniques enable the enhancement of the existing process model using the information obtained based on the information in the log or from the constructed model. Figure 1 shows an overview of mining process. Additionally, based on the information provided in the event data, three aspects of the process discovery class can be retrieved. If the log contains information about the activities handling a particular case that have been executed, the control-flow perspective can be discovered. If the event log provides data on persons machines or systems involved in handling the activities, the organisational perspective can be retrieved, and if the event log provides other data related to tasks, the case perspective can be discovered.

The control-flow perspective of the process discovery category is the main focus of this work. Even if process mining is considered as a recent set of techniques, several process discovery algorithms have been developed today [3-5] and successfully applied to various domains [6-8]. However, there are complex control-flow constructs that current process discovery techniques are incapable of correctly discovering them in process models based on event logs. These constructs are non-free choice constructs, invisible tasks, and short loops. Currently, no algorithm is capable of handling all these structures in a restricted time when they exist all together [5]. This study introduces a framework for identifying these complex control-flow constructs based only on event logs. However, framework can be used to choose the process discovery algorithm for a particular event log before applying a specific discovery technique.

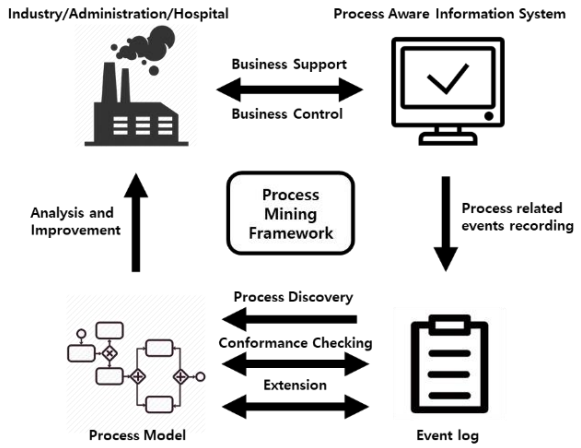

Fig. 1 Process mining framework

II. PROCESS CONSTRUCTS

A process model can be composed of standard structures like free-choice, sequence, concurrency and of complex constructs like loops, invisible tasks, duplicate tasks, etc. All today's process discovery techniques can discover the standard constructs. However, not all of them are capable of retrieving complex structures of a process model. In this study, we are not interested with the standard constructs. The main focus is the complex constructs.

A. Short Loops

There are two types of short loops, loops of length one as well as loop of length two. Loops of length one allow the one task to be executed multiple times (Fig. 2, Task C) while loops of length two allow two tasks following each other to be executed several times (Fig. 3, Tasks B and C).

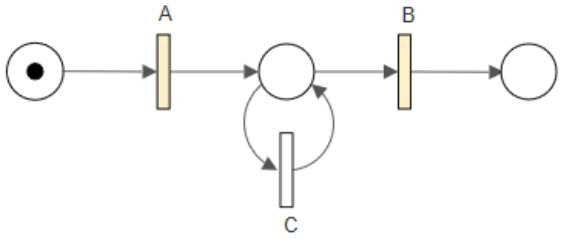

Fig. 2 An example of length one loop with a process model

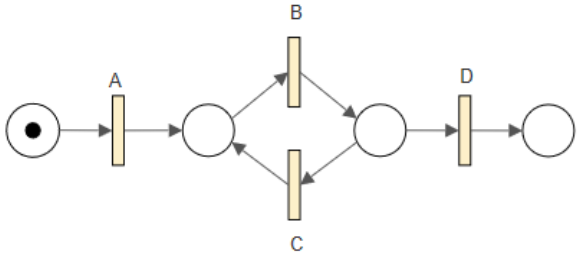

Fig. 3 An example of length two loop with a process model

B. Invisible Tasks

An invisible task is a task that appears in a process model. However, it is not available in the corresponding event data. For instance, skipping the execution of specific tasks is represented in a process model by a transition called invisible task. The transition is playing the role of skip. Hence, it is not listed in the event data [8]. Invisible tasks are classified into sequence short and long skip types, concurrent short and long skip types for execution skipping purpose, and into sequence short and long redo types, concurrent short and long redo types for execution repeating purpose [5]. An example of an invisible task is depicted in Fig. 4.

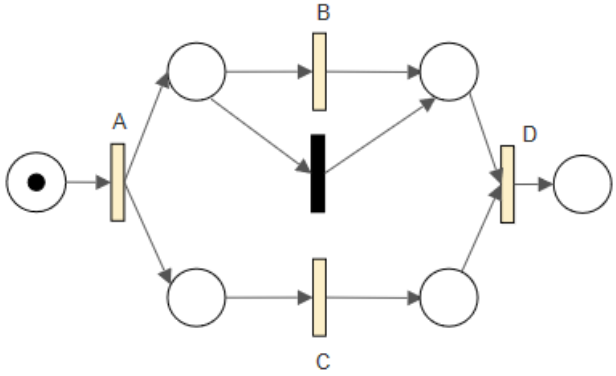

Fig. 4 An invisible task (task with black color) of type short skip in sequence.

C. Non-Free Choice Structures

A non-free choice structure is a combination of synchronization and choice. Synchronization and choice are associated with each other which can generate implicit dependencies. Existing mining techniques have no problem in mining explicit dependencies, however, fail in detecting implicit dependencies. Implicit dependencies are indirect causal relationships between tasks. Fig. 5 illustrates a non-free choice construct with a process model. If task B is executed tasks D will be executed and E will not be.

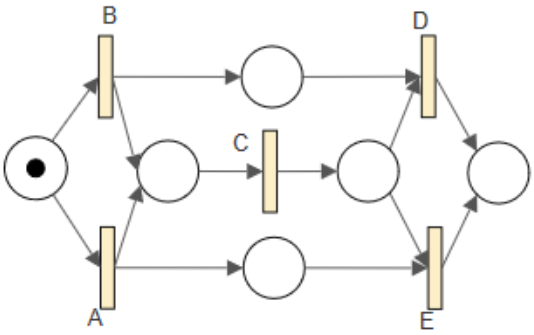

Fig. 5 An example of a non-free choice construct with a process model

III. COMPLEX CONSTRUCTS DETECTION OF A PROCESS MODEL

This section provides the equations defined to detect the complex structures introduced in the previous section. The equations are based on the extensions (i.e., alpha^{++} [9], alpha$^{\$}$ [10], alpha$^{\#}$ [11] of alpha algorithm [3]. The ordering relations introduced in alpha miner extensions to mine a model are improved to detect the complex constructs in an event log without discovering a process model [12].

A. Basic Relations for Ordering

In α algorithm and its extensions, the following relations were defined: $>_E$, Δ_E, $\lozenge_E$, $\rightarrow_E$, $\|_E$, $\neq_E$. Relation $>_E$ represents two tasks that can be executed successively. In a case where the same task is executed two or multiple times successively, relation $>_E$ refers to length-1 loop structure (e.g., *aa*). Notation Δ_E illustrates lenght-2-loop structure (e.g., *aba*). However, this relation used also to differentiate between length-2-loop and parallel branches. Relation $\lozenge_E$ expresses two-way loop-2-length, referring to two tasks having the Δ_E relations among each other (e.g. *aba*, *bab*). Relation $\rightarrow_E$ depicts the direct causal relation between two tasks. Relation $\|_E$ represents the tasks that are executed in parallel (e.g., ab, ba). Finally, relation $\neq_E$ denotes two tasks never following each other directly.

Definition (*Basic Relations for Ordering) [5]*

Let O be a set of tasks, E be a log over O, a and b be two tasks from O. The relations introduced by α and α^+ algorithms for ordering are as follows:

- $a >_E b \Leftrightarrow \exists \mu = v_1 v_2 \dots v_n \in L, i \in 1, \dots, n-1 : v_i = a \wedge v_{i+1} = b$,
- $a \Delta_E b \Leftrightarrow \exists \mu = v_1 v_2 \dots v_n \in L, i \in 1, \dots, n-2 : v_i = v_{i+2} = a \wedge v_{i+1} = b$,
- $a \lozenge_E b \Leftrightarrow (a \Delta_E b) \vee (b \Delta_E a)$,
- $a \rightarrow_E b \Leftrightarrow ((a >_E b) \wedge (b \not>_E a)) \vee (a \lozenge_E b)$,
- $a \|_E b \Leftrightarrow (a >_E b) \wedge (b >_E a) \wedge (b \not\lozenge_E a)$,
- $a \neq_E b \Leftrightarrow \big((a \not>_E b) \wedge (b \not>_E a)\big)$,

B. Advanced Relations for Ordering

In the α −extensions algorithms, the following relations were defined: $\gg_E$, $\succ_E$, $\lhd_E$, $\rhd_E$, $\lhd\!|_E$, Between-Set, as $\rightrightarrows_E$ and $\geq_E$. Relation $\gg_E$ represents the indirect reachable dependency between two activities whereas $\succ_E$ expresses the case where there is a direct or indirect reachable relationship between two activities. $\lhd_E$ refers to XOR_Split, $\rhd_E$ represents XOR_Join and $\lhd\!|_E$ denotes the AND-split. Between-Set($Between(W, a, b)$) depicts the tasks that occur between two tasks. In addition, Between-Set is a set of the tasks in the parallel routing, if two tasks are endpoints of parallel branches. In a concurrent workflow, the relations $\rightarrow_W$ and $>_E$ are defined as $\rightrightarrows_E$ and $\geq_E$ respectively to remove the interference of concurrent constructs. We define $\rightrightarrows_E^{sh}$ and $\geq_E^{sh}$ to represent invisible tasks of type *short skip and short redo*, $\rightrightarrows_E^{sh}$ and $\geq_E^{sh}$ to represent invisible tasks of type *long skip and long redo*).

Definition (*Advanced Relations for Ordering) [9-10, 12]*

Let O be a set of tasks, E be a log over O, a and b be two tasks from O. The relations introduced by α^{++}, $\alpha^{\#}$, $\alpha^{\$}$ algorithms for ordering are as follows:

- $a \gg_E b \Leftrightarrow \exists \mu = v_1 v_2 \dots v_n \wedge i, j \in 1, \dots, n : i < j \wedge v_i = a \wedge v_j = b \wedge \forall k \in [i+1, \dots, j-1 : vk \neq a \wedge vk \neq b$,
- $a \succ_E b \Leftrightarrow (a \rightarrow_E b) \vee (a \gg_E b)$,
- $a \lhd_E b \Leftrightarrow (a \neq_E b) \wedge \exists c \in O : (c \rightarrow_E a) \wedge (c \rightarrow_E b)$,
- $a \rhd_E b \Leftrightarrow (a \neq_E b) \wedge \exists c \in O : (a \rightarrow_E c) \wedge (b \rightarrow_E c)$,
- $a \lhd\!|_E \Leftrightarrow \exists x, y \in O : (a \rightarrow_E x) \wedge (a \rightarrow_E y) \wedge (x \|_E y)$,
- $Between(\mu, a, b) = \{\mu_k | \exists_{1 \leq i < j \leq m} (\mu_i = a \wedge \mu_j = b \wedge i < k < j \wedge \nexists i < l < j \mu l = a \vee \mu l = b)\}$,
- $\lceil Between(\mu, a, b) = \{\mu_k | 1 \leq k \leq m\} \setminus Between(\mu, a, b)$,
- $Between(E, a, b) = \cup_{\mu \in E} Between(\mu, a, b) \setminus \cup_{\mu \in E} \lceil Between(\mu, a, b)$,
- $a \geq_E^{sh} b \Leftrightarrow \exists x \in Between(E, a, b) \wedge \forall m \in (Between(E, a, b) \setminus \{x\}) : (m \|_W x)$
- $\wedge \exists_{\mu \in E} Between(\mu, a, b) \subseteq (Between(W, a, b) \setminus x$,
- $a \rightrightarrows_E^{sh} b \Leftrightarrow \Big((a \geq_E^{sh} b) \wedge (b \not\geq_E^{sh} a)\Big) \vee (a \lozenge_E b)$,
- $a \geq_E^{lg} b \Leftrightarrow \exists_{x,y \in O} Between(E, x, y) \subset Between(E, a, b) \wedge \forall m \in \big(Between(E, a, b) \setminus (Between(E, x, y) \cup x, y \forall n \in Between\ E, x, y : m \|E\ n \wedge \exists \mu \in E\ Between(\mu, a, b) \subseteq (Between(E, a, b) \setminus \{x, y\})$.

IV. CONTROL-FLOW CONTRUCTS DETECTION

This section defines the equations used to detect the following complex control-flow constructs: short loop of length one, invisible tasks of types short skip, short redo, long skip and long redo in the sequence and in the parallel workflow, and hidden tasks associated with non-free choice structures [12].

- *Loop of length one*

  $a >_E a \Leftrightarrow \exists \mu = v_1 v_2 \dots v_n \in E, i \in 1, \dots, n-1 : v_i = a \wedge v_{i+1} = a$.
- *Short skip in sequence*

  $a \dashrightarrow_E^{sss} b \Leftrightarrow (a \rightarrow_E b) \wedge \exists x \in O : (a \rightarrow_E x) \wedge (x \rightarrow_E b) \wedge (x \nparallel_E b) \wedge (a \nparallel_E x) \wedge (x \not\succ_E x)$.
- *Long skip in sequence*

  $a \dashrightarrow_E^{lss} b \Leftrightarrow (a \rightarrow_E b) \wedge \exists x, y \in O : (a \rightarrow_E x) \wedge (y \rightarrow_E b) \wedge (y \not\succ_E x) \wedge (x \nparallel_E b) \wedge (a \nparallel_E y) \wedge (x \not\succ_E y)$.
- *Short redo in sequence*

  $y \dashrightarrow_E^{srs} x \Leftrightarrow (y \not\succ_E x) \wedge \exists a \in O : (a \rightarrow_E x) \wedge (y \rightarrow_E a) \wedge (x \nparallel_E a) \wedge (a \nparallel_E y) \wedge (a >_E a)$.
- *Long redo in sequence*

  $y \dashrightarrow_E^{lrs} x \Leftrightarrow \exists a, b \in O : (y \rightarrow_E b) \wedge (a \rightarrow_E x) \wedge (x \nparallel_E b) \wedge (y \nparallel_E a) \wedge (b \not\succ_E a) \wedge (a >_E b)$.
- *Short skip in the parallel*

  $a \dashrightarrow_E^{ssp} b \Leftrightarrow \exists x, y \in O : (a \rightarrow_E x) \wedge (y \rightarrow_E b) \wedge (x \nparallel_E b) \wedge (y \nparallel_E a) \wedge (a \rightrightarrows_E^{sh} b)$.
- *Long skip in the parallel*

  $a \dashrightarrow_E^{lsp} b \Leftrightarrow \exists x, y \in O : (a \rightarrow_E x) \wedge (y \rightarrow_E b) \wedge (x \nparallel_E b) \wedge (y \nparallel_E a) \wedge (a \rightrightarrows_E^{lg} b) \wedge (y \not\geq_E x) \wedge (x \not\succ_E y)$.
- *Short redo in the parallel*

  $y \dashrightarrow_E^{srp} x \Leftrightarrow \exists a \in O : (y \rightarrow_E a) \wedge (a \rightarrow_E x) \wedge (a \nparallel_E x) \wedge (y \nparallel_E a) \wedge (y \rightrightarrows_E^{sh} x) \wedge (a >_E a)$.

- *Long redo in the parallel*

$y \dashrightarrow_E^{lrp} x \Leftrightarrow \exists\, a,b \in O: (a \rightarrow_E x) \wedge (y \rightarrow_E b) \wedge (x \nparallel_E b) \wedge (y \nparallel_E a) \wedge \left(y \rightrightarrows_E^{lg} x\right) \wedge (b \not\succeq_E a) \wedge (b \succ_E a)$.

- *Non-free Choice Constructs*

$a \rightarrow_{E^1} b \Leftrightarrow (a \not\succ_E b) \wedge \exists c \in O: (a >_E c) \wedge (c \lhd_E b) \wedge \nexists u \in O: (u >_E a) \wedge \big((u >_E a) \vee (u \parallel_E a)\big)$,

$a \rightarrow_{E^{21}} b \Leftrightarrow (a \gg_E b) \wedge a \lhd\!\!\cdot_E \wedge \exists b' \in O: (b \lhd_E b') \wedge \nexists u \in O: (a >_E u) \wedge (u \succ_E b \text{ or } u \parallel_E b) \wedge \exists u' \in O: (a >_E u') \wedge (u' \succ_E b \vee u' \parallel_E b)$,

$a \rightarrow_{E^{22}} b \Leftrightarrow (a \gg_E b) \wedge b \rhd\!\!\cdot_E \wedge \exists u' \in O: (a >_E u') \wedge (u' \succ_E b \vee u' \parallel_E b)$,

$a \rightarrow_{E^2} b \Leftrightarrow a \rightarrow_{E^{21}} b \vee a \rightarrow_{E^{22}} b$,

$a \rightarrow_{E^3} b \Leftrightarrow (a \lhd_E a') \wedge (b \rhd_E b') \wedge (a \gg_E b) \wedge (a \neg\gg_E b')$.

- *Invisible tasks associated with a non-free choice*

$x \succ_W m \Leftrightarrow \exists_{(a=\perp \vee a \in E) \wedge b \in E} (a \rightarrow_E x) \wedge (x \rightarrow_E b) \wedge (b \succ_{E,pred=a} m) \wedge \nexists_{n \in \mu} (b \succ_{E,pred=a} n)$,

$m \succ_E x \Leftrightarrow \exists_{a \in E \wedge (b \in E \vee b=\top)} (a \rightarrow_E x) \wedge (x \rightarrow_E b) \wedge (m \succ_{E,pred=b} a) \wedge \nexists_{n \in \mu} (m \succ_{E,pred=n} a)$,

$x \succ_E y \Leftrightarrow \exists_{(a_1=\perp \vee a_1 \in E) \wedge b_1 \in E \wedge a_2 \in E \wedge (b_2 \in E \vee b_2=\top)} (a_1 \rightarrow_E x) \wedge (x \rightarrow_E b_1) \wedge (a_2 \rightarrow_E y) \wedge (y \rightarrow_E b_2) \wedge (b_1 \succ_{E,pred=a_1,post=b_2} a_2) \wedge \nexists_{n \in E} (b_1 \succ_{E,pred=a_1,post=b_n} a_2)$

$a >_E a$ is the dependency which detects length-one-loop. $a \dashrightarrow_E^{sss} b$, $a \dashrightarrow_E^{lss} b$, $x \dashrightarrow_E^{srs} y$, $x \dashrightarrow_E^{lrs} y$, $a \dashrightarrow_E^{ssp} b$, $a \dashrightarrow_E^{lsp} b$, $y \dashrightarrow_E^{srp} x$, $y \dashrightarrow_E^{lrp} x$ are responsible for detecting respectively invisible tasks of types sequence short skip, sequence long skip, sequence short redo, sequence long redo, concurrent short skip, concurrent long skip, concurrent short redo, concurrent long redo. $\rightarrow_{E^1}$, $\rightarrow_{E^2}$, and $\rightarrow_{E^3}$ reflect the implicit dependencies that are responsible of detecting non-free choice structures. Finally, $x \succ_E m$, $m \succ_E x$, $x \succ_E y$ are in charge of detecting invisible tasks associated with non-free choice structure.

V. FRAMEWORK EVALUATION

A. Implementation

This work implemented as a plugin the framework identifying the complex structures of a workflow from the process event data in the Open source ProM. This later provides of plenty of plugins for mining process models, verifying the conformance and analyzing the process models. ProM is available in http://www.processmining.org.

To detect the complex control-flow constructs, we import the event log to be analyzed into ProM then we select and run the plugin "Control Flow Constructs Detection". The plugin investigates the status of the complex constructs in the event log and report the results in a table. The constructs that are investigated are length-one-loop structure (L_{1p} for short), length-two-loop structure (L_{2p} for short), hidden tasks of types sequence short-skip ($IvT_{SS_{seq}}$), sequence long-skip ($IvT_{LS_{seq}}$), sequence short-redo ($IvT_{SR_{seq}}$), sequence long-redo ($IvT_{LR_{seq}}$), concurrent short-skip ($IvT_{SS_{par}}$), concurrent long-skip ($IvT_{LS_{par}}$), concurrent short-redo ($IvT_{SR_{par}}$), concurrent long-redo ($IvT_{LR_{par}}$), non-free choice structure (Nfc), and hidden tasks associated with non-free choice structure (IvT_{Nfc}). If a complex construct is detected to be existing, "Yes" is shown in the table, otherwise it is shown "No". Fig. 6 showed the example of the output.

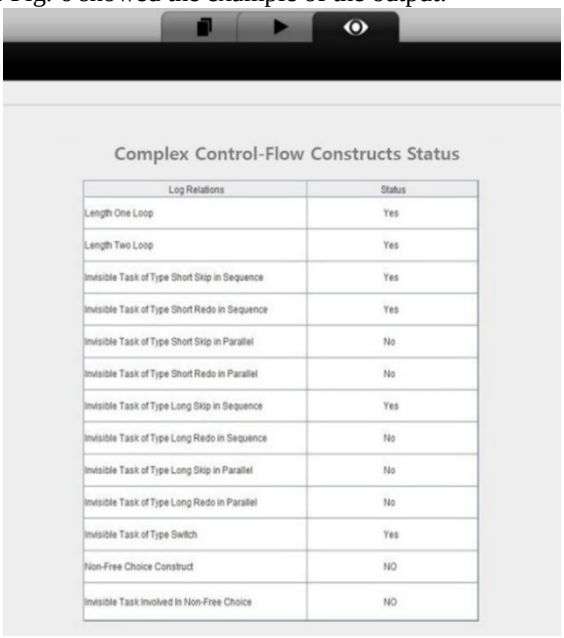

Fig. 6 A screenshot of the an example of the output

B. Evaluation using artificial event data

To evaluate the framework detecting the complex constructs of a workflow, 30 artificial event logs ($\boldsymbol{W_i}$) and 30 corresponding reference models ($\boldsymbol{R_i}$) have been used. The complex constructs mentioned above are generated randomly in the 30 artificial event logs ($\boldsymbol{W_i}$). In the 30 corresponding reference models, the maximum number of tasks is less than 20 tasks and the number of cases is less than 40 in one event data. In the evaluation, detected structures are compared with the structures of the corresponding a-priori models. Table 1 depicts the comparison results. According to the Table 1, structures detected using the proposed methodology match those in the reference models. This indicates that we can use the proposed framework to choose a suitable process discovery algorithm based on structures identified in the event data.

Table-I: Comparison between the detected constructs and the constructs existing in the reference models

Event logs	Detected constructs	Reference models	Existing constructs
$\boldsymbol{W_1}$	$IvT_{SS_{par}}$	$\boldsymbol{R_1}$	$IvT_{SS_{par}}$
$\boldsymbol{W_2}$	$IvT_{LR_{par}}$	$\boldsymbol{R_2}$	$IvT_{LR_{par}}$
$\boldsymbol{W_3}$	$IvT_{SS_{seq}}$	$\boldsymbol{R_3}$	$IvT_{SS_{seq}}$
$\boldsymbol{W_4}$	$IvT_{LS_{seq}}$	$\boldsymbol{R_4}$	$IvT_{LS_{seq}}$
$\boldsymbol{W_5}$	L_{2p}, $IvT_{LR_{seq}}$	$\boldsymbol{R_5}$	L_{2p}, $IvT_{LR_{seq}}$
$\boldsymbol{W_6}$	L_{1p}, $IvT_{LR_{par}}$	$\boldsymbol{R_6}$	L_{1p}, $IvT_{LR_{par}}$
$\boldsymbol{W_7}$	IvT_{NFC}	$\boldsymbol{R_7}$	IvT_{NFC}
$\boldsymbol{W_8}$	IvT_{NFC}, $IvT_{SS_{seq}}$	$\boldsymbol{R_8}$	IvT_{NFC}, $IvT_{SS_{seq}}$
$\boldsymbol{W_9}$	L_{1p}, $IvT_{SS_{seq}}$, $IvT_{LR_{seq}}$	$\boldsymbol{R_9}$	L_{1p}, $IvT_{SS_{seq}}$, $IvT_{LR_{seq}}$
$\boldsymbol{W_{10}}$	L_{1p}, L_{2p}, IvT_{NFC}	$\boldsymbol{R_{10}}$	L_{1p}, L_{2p}, IvT_{NFC}
$\boldsymbol{W_{11}}$	NFC, $IvT_{SS_{seq}}$	$\boldsymbol{R_{11}}$	NFC, $IvT_{SS_{seq}}$
$\boldsymbol{W_{12}}$	$IvT_{SR_{seq}}$, L_{2p}	$\boldsymbol{R_{12}}$	$IvT_{SR_{seq}}$, L_{2p}
$\boldsymbol{W_{13}}$	L_{2p}, $IvT_{SS_{par}}$	$\boldsymbol{R_{r13}}$	L_{2p}, $IvT_{SS_{par}}$

W_{14}	$L_{1p}, IvT_{LR_{seq}}, IvT_{NFC}$	R_{14}	$L_{1p}, IvT_{LR_{seq}}, IvT_{NFC}$
W_{15}	$IvT_{SS_{seq}}, IvT_{LS_{seq}}$	R_{15}	$IvT_{SS_{seq}}, IvT_{LS_{seq}}$
W_{16}	$L_{2p}, IvT_{LS_{seq}}, IvT_{SS_{seq}}$	R_{16}	$L_{2p}, IvT_{LS_{seq}}, IvT_{SS_{seq}}$
W_{17}	$IvT_{LS_{seq}}, IvT_{NFC}$	R_{17}	$IvT_{LS_{seq}}, IvT_{NFC}$
W_{18}	$IvT_{SS_{seq}}, IvT_{LS_{seq}}, L_{2p}, IvT_{NFC}$	R_{18}	$IvT_{SS_{seq}}, IvT_{LS_{seq}}, L_{2p}, IvT_{NFC}$
W_{19}	L_{2p}, IvT_{NFC}	R_{19}	L_{2p}, IvT_{NFC}
W_{20}	L_{1p}, IvT_{NFC}	R_{20}	L_{1p}, IvT_{NFC}
W_{21}	$IvT_{SS_{par}}$, $IvT_{LR_{par}}$, IvT_{NFC}	R_{21}	$IvT_{SS_{par}}$, $IvT_{LR_{par}}$, IvT_{NFC}
W_{22}	$IvT_{LR_{seq}}, IvT_{NFC}$	R_{22}	$IvT_{LR_{seq}}, IvT_{NFC}$
W_{23}	$IvT_{SS_{par}}, IvT_{LR_{par}}$	R_{23}	$IvT_{SS_{par}}, IvT_{LR_{par}}$
W_{24}	$L_{1p}, L_{2p}, IvT_{SS_{par}}$	R_{24}	$L_{1p}, L_{2p}, IvT_{SS_{par}}$
W_{25}	$IvT_{SR_{seq}}, IvT_{LS_{seq}}$, L_{2p}, IvT_{NFC}	R_{25}	$IvT_{SR_{seq}}, IvT_{LS_{seq}}$, L_{2p}, IvT_{NFC}
W_{26}	$IvT_{SS_{seq}}, IvT_{LS_{seq}}$, IvT_{NFC}	R_{26}	$IvT_{SS_{seq}}, IvT_{LS_{seq}}$, IvT_{NFC}
W_{27}	$L_{1p}, L_{2p}, IvT_{LS_{seq}}$	R_{27}	$L_{1p}, L_{2p}, IvT_{LS_{seq}}$
W_{28}	$L_{2p}, IvT_{LR_{par}}$	R_{28}	$L_{2p}, IvT_{LR_{par}}$
W_{29}	$L_{1p}, IvT_{LR_{seq}}$	R_{29}	$L_{1p}, IvT_{LR_{seq}}$
W_{30}	$IvT_{SS_{par}}$, $IvT_{SS_{seq}}$, $IvT_{LR_{par}}, IvT_{NFC}$	R_{30}	$IvT_{SS_{par}}$, $IvT_{SS_{seq}}$, $IvT_{LR_{par}}, IvT_{NFC}$

VI. CONCLUSION

Today, a tremendous number of process discovery techniques have been proposed. However, the capability of each technique in mining complex constructs of a process is different which makes deciding which algorithm to use difficult. In this paper, we have presented a proposition for detecting the complex constructs of a process model from its corresponding event log. By detecting the existing complex constructs, we can recommend the algorithms capable of discovering such constructs. This work has implemented the proposition in ProM as a plugin. The evaluation is conducted using reference models and the corresponding event data. Accordingly, structures detected with the framework from the event logs have been detected to be the same as the structures in the a-priori models. This shows that we can utilize the proposed methodology to choose or recommend an appropriate discovery technique for a given event data. This proposition is recommended to be utilized as a pre-processing phase before discovering the process model form an event log.

ACKNOWLEDGMENT

This work is sponsored by the University of Ulsan in 2019.

REFERENCES

1. H. R'bigui and C. Cho, "An Industrial Application of Process Mining for Analyzing a Real Order Fulfillment Process of a Shipbuilding industry," *Proc. Of the Korean Institute of Industrial Engineers*, vol.2018, no. 11, 2018, pp. 368-375.
2. W.M.P. Van der Aalst, "Process Mining: Discovery, Conformance and Enhancement of Business Processes", 1st ed., Berlin Heidelberg: Springer-Verlag, 2011.
3. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs, " *IEEE T. Knowl. Data En.*, vol.16, no.9, pp. 1128-1142, 2004.
4. S. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs containing infrequent behavior," *Int. Conf. BPM, LNBIP*, vol. 171, Cham/Springer, pp. 66–78, 2014.
5. H. R'bigui, and C. Cho, "Heuristic Rule-based Process Discovery Approach from Events Data," *Int. J. Technology Policy and Management, Special Issue on: Knowledge Management. System*, vol.19, no. 4, in press, 2019.
6. H. R'bigui and C. Cho, "Customer Oder Fulfillment Process Analysis with Process Mining: An Industrial Application in a Heavy Manufacturing Company, " *Proc. of the ACM Int. Con on Computer Science and Artificial Intelligence*, pp. 247-252, Jakarta, Indonesia, Dec 05-07, 2017.
7. D. Vazquez Barreiros, M. Chapela, M. Mucientes, Lama, and D. Berea, "Process Mining in IT Service Management: A Case Study," *Int. Worshop on Algorithm & Theory for the Analysis of Event Data* (Toruń, Poland), CEUR-WS.org, pp. 16-30, 2016.
8. H. R'bigui and C. Cho, "Purchasing Process Analysis with Process Mining of a Heavy Manufacturing Industry," *IEEE Proc. of the 9th Int. Conf. on Information and Communication Technology*, pp. 495-498, Jeju Island, Korea, Oct 17-19, 2018.
9. L. Wen, J. Wang and J. Sun, "Mining Invisible Tasks from Event Logs, " *LNCS 4505*, Springer, Berlin, Heidelberg, pp. 358-365, 2007a.
10. Q. Guo, L. Wen, J. Wang, Z. Yan and P S. Yu, "Mining invisible tasks in non-free-choice constructs, " *LNCS 9253*, Motahari-Nezhad ., Recker J, Weidlich M (eds.), Springer, Cham, pp. 109–125, 2015.
11. L. Wen, W M P. Van der Aalst, J. Wang, and J. Sun, "Mining process models with non-free-choice constructs, " *Data Mining and Knowledge Discovery*, vol.15 no.2., pp. 145-180, 2007b.
12. H. R'bigui and C. Cho, "Complex Control-Flow Constructs Detection from Process Related Data," *2019 21st Int. Conf. on Advanced Communication and Technology (ICACT)*, PyeongChang Kwangwoon_Do, Korea (South), 2019, pp. 579-582.

AUTHORS PROFILE

Hind R'bigui received the State Engineer degree in Industrial Engineering from the National School of Applied Sciences of Fes, Morocco in 2013, and her Ph.D. degree in industrial engineering from the University of Ulsan, South Korea in 2019. Currently, she is working as a Senior Research Engineer at the Digital Enterprise Department of Nsoft Co., Ltd, Siemens Industry Software Partner, South Korea.

Mohammed Abdulhakim Alabsi received his BS degree in Computer Application from Bangalore University in India. He earned his (MS) degree at Dongseo University-South Korea in 2018. Currently, he is a Ph.D. student in the Department of Information and Communication Engineering at Dongseo University, Korea.

Chiwoon Cho is a full Professor of Industrial Engineering at the University of Ulsan in South Korea. He has a lot of industry experiences including Hyundai Heavy Industries, LG CNS, and Samsung SDS.