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 primal 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 strengthens 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]. 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.

Revised Manuscript Received on September 28, 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

DOI: 10.5940/ijitee.1391.119119
Detecting Complex Control-Flow Constructs for Choosing Process Discovery Techniques

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).

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.

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.

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: \(\geq \), \(\Delta E\), \(\varnothing E\), \(\rightarrow\), \(\perp\). Relation \(\geq\) represents two tasks that can be executed successively. In a case where the same task is executed two or multiple times successively, relation \(\geq\) refers to length-1 loop structure (e.g., \(aa\)). Notation \(\Delta E\) illustrates length-2 loop structure (e.g., \(aba\)). However, this relation also used to differentiate between length-2-loop and parallel branches. Relation \(\varnothing E\) expresses two-way loop-length, referring to two tasks having the \(\Delta E\) relations among each other (e.g., \(aba\), \(bab\)). Relation \(\rightarrow\) depicts the direct causal relation between two tasks. Relation \(\perp\) represents the tasks that are executed in parallel (e.g., \(ab\), \(ba\)). Finally, relation \(\perp E\) 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 \(a\) and \(a^+\) algorithms for ordering are as follows:

- \(a \geq b \Leftrightarrow \exists \mu = v_1v_2...v_n \in E, i \in 1,..., n-1 : v_i = a \wedge v_{i+1} = b\).
- \(a \Delta E b \Leftrightarrow \exists \mu = v_1v_2...v_n \in E, i \in 1,..., n-2 : v_i = v_{i+2} = a \wedge v_{i+1} = b\).
- \(a \varnothing b \Leftrightarrow (a \Delta E b) \vee (b \Delta E a)\).
- \(a \rightarrow b \Leftrightarrow ((a \geq E b) \wedge (b \notin E a)) \vee (a \varnothing b)\).
- \(a \perp b \Leftrightarrow (a \geq E b) \wedge (b \notin E a)\).
- \(a \not\sim E b \Leftrightarrow (a \geq E b) \wedge (b \notin E a)\).

B. Advanced Relations for Ordering

In the \(\alpha\) extensions algorithms, the following relations were defined: \(\gg \), \(\vartriangle\), \(\vartriangledown\), \(\vartriangledown E\), \(\ll\). Between-Set, as \(\gg E\) and \(\gg E\). Relation \(\gg E\) represents the indirect reachable dependency between two activities where \(\vartriangle E\) expresses the case where there is a direct or indirect reachable relationship between two activities. \(\vartriangledown E\) refers to XOR-Split, \(\vartriangledown E\) represents XOR-Join and \(\ll E\) 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 \(\ll\) and \(\gg\) are defined as \(\ll E\) and \(\gg E\) respectively to remove the interference of concurrent constructs. We define \(\ll E\) and \(\gg E\) to represent invisible tasks of type short skip and short redo, \(\ll E\) and \(\gg E\) 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 \(\alpha^+, \alpha^+\), \(\alpha^+\) algorithms for ordering are as follows:

- \(a \gg b \Leftrightarrow \exists \mu = v_1v_2...v_n \in E, i \in 1,..., n-1 : v_i = a \wedge v_{i+1} = b \forall k \in [i+1,..., j-1] : v_k \neq a \land v_k \neq b\).
- \(a \gg R b \Leftrightarrow (a \rightarrow b) \vee (a \gg b)\).
- \(a \ll b \Leftrightarrow (a \gg b) \wedge \exists c \in O : (c \rightarrow a) \wedge (c \rightarrow E b)\).
- \(a \rightarrow b \Leftrightarrow (a \gg b) \wedge \exists c \in O : (a \rightarrow c) \wedge (b \rightarrow E c)\).
- \(a \ll \Leftrightarrow \exists x, y \in O : (a \rightarrow E x) \wedge (a \rightarrow E y) \wedge (x \ll E y)\).
- \(a \ll \Leftrightarrow (\mu \geq E b) \wedge (b \gg E a)\).
- Between (\(\mu, a, b\)) = \(\{\mu_k | 1 \leq k \leq m \} \setminus \{\mu_k | 1 \leq k \leq m \} \wedge (\mu \gg E b)\).
- Between (\(\mu, a, b\)) = \(\{\mu_k | 1 \leq k \leq m \} \setminus \{\mu_k | 1 \leq k \leq m \} \wedge (\mu \gg E b)\).
- Between (\(E, a, b\)) = \(\cup_{\mu E E} (\mu \gg E b) \setminus (\mu \gg E a)\).
- Between (\(E, a, b\)) = \(\cup_{\mu E E} (\mu \gg E b) \setminus (\mu \gg E a)\).

IV. CONTROL-FLOW CONTRACTS 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**
  \(a \gg a \Leftrightarrow \exists \mu = v_1v_2...v_n \in E, i \in 1,..., n-1 : v_i = a \wedge v_{i+1} = a\).
- **Short skip in sequence**
  \(a \gg E b \Leftrightarrow (a \rightarrow b) \wedge (x \rightarrow E b) \wedge (\exists x \in E : (a \rightarrow E x) \wedge (x \rightarrow E b) \wedge (a \notin E b)\).
- **Long skip in sequence**
  \(a \gg E b \Leftrightarrow (a \rightarrow E b) \wedge (x \rightarrow E x) \wedge (\exists x \notin E E : (a \rightarrow E x) \wedge (x \rightarrow E b) \wedge (a \notin E b)\).
- **Short redo in sequence**
  \(a \gg E b \Leftrightarrow (a \rightarrow E b) \wedge (x \rightarrow E y) \wedge (\exists x \notin E E : (a \rightarrow E x) \wedge (x \rightarrow E b) \wedge (a \notin E b)\).
- **Long redo in sequence**
  \(a \gg E b \Leftrightarrow (a \rightarrow E b) \wedge (x \rightarrow E y) \wedge (\exists x \notin E E : (a \rightarrow E x) \wedge (x \rightarrow E b) \wedge (a \notin E b)\).
y \rightarrow_{E}^{lrp} x \iff \exists a, b \in O: (a \rightarrow_{E} x) \land (y \rightarrow_{E} b) \land (x \underset{E}{\equiv} b) \land (y \underset{E}{\equiv} a) \land (b \rightarrow_{E} a).

- **Non-free Choice Constructs**
  \[ a \rightarrow_{E} b \iff (a \rightarrow_{E} b) \land \exists c \in O: (a \rightarrow_{E} c) \land (c \rightarrow_{E} b) \land (a \rightarrow_{E} a) \lor (u \parallel a) \land (u \parallel b) \land (a \rightarrow_{E} b) \land (a \rightarrow_{E} u) \land (u \parallel_{E} b) \land (u \parallel_{E} u), \]
  \[ a \rightarrow_{E} b \iff (a \rightarrow_{E} b) \land (a \rightarrow_{E} x) \land (a \rightarrow_{E} y). \]

- **Invisible constructs associated with a non-free choice**
  \[ x \rightarrow_{E} \theta \iff \exists (a \land v \land E \land b) \land (a \rightarrow_{E} b) \land (x \rightarrow_{E} b) \land (b \rightarrow_{E} \theta) \land (b \rightarrow_{E} \theta) \land (b \rightarrow_{E} \theta) \land (a \rightarrow_{E} \theta) \land \land (a \rightarrow_{E} \theta). \]

- **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 ($L_{1p}$), length-two-loop structure ($L_{2p}$), hidden tasks of types sequence short-skop ($1vT_{SR_{seq}}$), sequence long-skop ($1vT_{LS_{seq}}$), sequence short-redo ($1vT_{SR_{seq}}$), sequence long-redo ($1vT_{LR_{seq}}$), concurrent short-skop ($1vT_{SS_{par}}$), concurrent long-skop ($1vT_{LS_{par}}$), concurrent short-redo ($1vT_{SR_{par}}$), concurrent long-redo ($1vT_{LR_{par}}$), non-free choice structure ($NFC$), and hidden tasks associated with non-free choice structure ($1vT_{NS_{seq}}$). 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.

B. Evaluation using artificial event data

To evaluate the framework detecting the complex constructs of a workflow, 30 artificial event logs ($W_1$) and 30 corresponding reference models ($R_1$) have been used. The complex constructs mentioned above are generated randomly in the 30 artificial event logs ($W_1$). 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 |
|------------|---------------------|-----------------|-------------------|
| $W_1$      | lVT_{SS_{par}}      | $R_1$           | lVT_{SS_{par}}    |
| $W_2$      | lVT_{LS_{par}}      | $R_2$           | lVT_{LS_{par}}    |
| $W_3$      | lVT_{SS_{par}}      | $R_3$           | lVT_{SS_{par}}    |
| $W_4$      | lVT_{LR_{par}}      | $R_4$           | lVT_{LR_{par}}    |
| $W_5$      | lVT_{SS_{par}}      | $R_5$           | lVT_{SS_{par}}    |
| $W_6$      | lVT_{LS_{par}}      | $R_6$           | lVT_{LS_{par}}    |
| $W_7$      | lVT_{SS_{par}}      | $R_7$           | lVT_{SS_{par}}    |
| $W_8$      | lVT_{SS_{par}}      | $R_8$           | lVT_{SS_{par}}    |
| $W_9$      | lVT_{LS_{par}}      | $R_9$           | lVT_{LS_{par}}    |
| $W_{10}$   | lVT_{SS_{par}}      | $R_{10}$        | lVT_{SS_{par}}    |
| $W_{11}$   | lVT_{SS_{par}}      | $R_{11}$        | lVT_{SS_{par}}    |
| $W_{12}$   | lVT_{SS_{par}}      | $R_{12}$        | lVT_{SS_{par}}    |
| $W_{13}$   | lVT_{SS_{par}}      | $R_{13}$        | lVT_{SS_{par}}    |
### 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’bigiui 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, LNBP, vol. 171, Cham/Springer, pp. 66–78, 2014.
5. H. R’bigiui, 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’bigiui and C. Cho, “Customer Oder Fulfillment Process Analysis with Process Mining: An Industrial Application in a Heavy Manufacturing Company,” Proc. of the ACM Int. Conf 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. Workshop on Algorithm & Theory for the Analysis of Event Data (Torut, Poland); CEUR-WS.org, pp. 16-30, 2016.
8. H. R’bigiui 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, L. 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’bigiui and C. Cho, “Complex Control-Flow Constructs Detection from Process Related Data,” 2019 21st Int. Conf. on Advanced Communication and Technology (ICACT), PyeongChang Kwangwono_Do, Korea (South), 2019, pp. 579-582.