International Journal of Innovative
Computing, Information and Control
Volume **14**, Number **3**, June **2018**

 ISSN 1349-4198
pp. 971–995

A MORE EFFICIENT DETERMINISTIC ALGORITHM IN PROCESS MODEL DISCOVERY

Hermawan[1,2] and Riyanarto Sarno[2]

[1]Departemen Informatika
Fakultas Teknik
Universitas Trunojoyo Madura
Po. Box 2 Kamal, Bangkalan 69162, Indonesia
hermawan@trunojoyo.ac.id

[2]Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Jalan Raya ITS, Sukolilo, Surabaya 60111, Indonesia
riyanarto@if.its.ac.id

Received June 2017; revised December 2017

Abstract. *Alpha is a basic deterministic discovery algorithm that has been enhanced by Alpha*, Alpha++ and Alpha#. Alpha does an analysis of place, transition, and firing locally on each trace in the event log, which causes the time complexity of Alpha to be high for large event logs. In this paper, the Alpha-Tree (Alpha-T) algorithm is proposed to enhance Alpha's time complexity performance and quality of discovery. Based on generalized tuple pattern recognition (GTPR) inside the adjacency list tree (ALT) data structure, Alpha-T is able to simplify the tuple pattern analysis, resulting in a more efficient time complexity* $\left(O\left(t^3\right)\right)$ *compared to Alpha* $\left(O\left(t^4\right)\right)$*. Alpha-T reduces the time complexity by localizing the effect of the event log size to the preprocessing stage, which diminishes the number of steps in the discovery processing stage. Then within processing, Alpha-T does execution pattern of logic directly and induction in the places gateway, and make it have more dynamically pattern that produces more completeness and correctness model than other algorithms. Finally, in the post-processing stage, Alpha-T has a single graph structure, which reduces the complexity and memory space needed for workflow firing between place and transition.*

Keywords: Process mining, Deterministic discovery algorithm, Alpha algorithm, Alpha-T algorithm, Generalized tuple pattern recognition (GTPR), Adjacency list tree (ALT)

1. **Introduction.** Today, the process mining of transaction data is increasingly required by various organizations [1]. Process mining is widely implemented to analyze business process activities using process aware information systems (PAIS) [2]. PAIS are used to improve competitiveness, performance, and organizational policy [3] in various fields, such as manufacture [4], insurance [1], finance [5], healthcare [6], information technology [7], and education [8].

The main activity in mining is the discovery process [9], which examines the event log [10] to produce a model that reflects the workflow of the business process [11]. The workflow model should be accurate in describing the real operational activities from the business transactions that generate the event log.

There are many standardized workflow models that are used for business processes, such as *Petri nets*, *business process model notation* (BPMN), *event-driven process chain* (EPC), and *yet another workflow language* (YAWL) [12].

In this study, the YAWL-model was chosen for the workflow design because YAWL has the best workflow pattern analysis, which covers all workflow perspectives and has control-flow patterns, data patterns, resource patterns, change patterns, and also exception patterns [12]. YAWL has been adopted by common discovery tools such as ProM, which is a qualified framework and tool used in the process mining.

Several algorithms have been developed to do the mining in the discovery process: a deterministic algorithm, Alpha, including Alpha* [13], Alpha++ [14], and Alpha# [15], a fuzzy algorithm [16], a heuristic algorithm [17], a genetic algorithm [18], machine learning [19], discriminative patterns [20] and also based on association rule approaches [21,22].

From a performance comparison of the discovery algorithms, it was found that the best accuracy was achieved by the genetic algorithm and the best time complexity was achieved by the heuristic algorithm when compared with Alpha [23]. Although the heuristic algorithm produces an optimal solution with minimal side effects, it is unsuitable for complex mining containing invisible tasks [2]. Other than that, it produces a directed graph of a standard causality network (C-net) that has bias representational model [24] and only supports structured relations, not in behavioral relations. Meanwhile, the computation complexity of the genetic algorithm is worse and unfavorable for big event logs.

For these reasons, this study aimed at enhancing the deterministic algorithm based on the Alpha method [25], whose performance is not optimal in the discovery on big event logs and has low accuracy in complex mining that contains invisible task [15]. The enhanced method that is proposed here to solve the shortcomings of Alpha is called Alpha-T.

Through the Alpha-T algorithm, the mining analysis on tuple relationships can be simplified by using prefix generalizations in ALT structure, a process that is called GTPR. During general retrieval, using ALT is more efficient in terms of time and space than using a general matrix structure. Also, using GTPR will shorten the discovery step because it executes logic pattern recognition directly on the tuple relation.

To test several algorithms that are commonly used, the *model driven analysis* (MDA) strategy was applied to measuring conformance of workflow model that shows quality of discovery. Using MDA, a comparison of the reference model and the discovery model can be aligned as driven by qualified tools such as YAWL and ProM.

For benchmarking the computation performance, time complexity analysis was done with focus between Alpha-T and Alpha. Also, for quality discovery, it was done by conformance checking of completeness and correctness model. *Alpha-T is expected having higher performance and better discovery quality than others as the main contribution and novelty of this study.*

2. **Problem Statement and Preliminaries.** In this section, we describe the Alpha stages to be compared with the proposed method to show the difference between them. Alpha as the basis of the deterministic discovery algorithm does the tuple mining by tracing the task sequence locally in the event log. The tuple sequence patterns describe the workflow that is built from the event log. Alpha consists of 4 ordering relation patterns, i.e.:

1) Follows ($>$), if (A, B) are 2 related tasks
2) Causal ($\rightarrow$), if tasks A $>$ B and B! $>$ A
3) Parallel ($||$), if tasks A $>$ B and B $>$ A
4) Unrelated (#), if tasks A! $>$ B and B! $>$ A

The relation sequence patterns are based on the causality logic from the Alpha discovery algorithm. All Alpha steps are described in the following Lemma 1.

(1) $T_w = \{t \in T | \exists_{\sigma \in w} t \in \sigma\}$
(2) $T_I = \{t \in T | \exists_{\sigma \in w} t \in \text{first}(\sigma)\}$

$$
\begin{aligned}
&(3)\ T_O = \{t \in T | \exists_{\sigma \in w} t \in \text{last}(\sigma)\}\\
&(4)\ X_w = \{(P_I, P_O) | P_I \subseteq T_w \wedge P_O \subseteq T_w \wedge \forall_{a \in P_I} \forall_{b \in P_O} \rightarrow_w b \wedge\\
&\qquad \forall_{a_1, a_2 \in P_I} a_1 \neq_w a_2 \wedge \forall_{b_1, b_2 \in P_O} b_1 \neq_w b_2\}\\
&(5)\ Y_w = \left\{(P_I, P_O) \in X_w | \forall_{(P'_I \in P'_O) \in X_w} P_I \subseteq P'_I \wedge P_O \subseteq P'_O \Rightarrow (P_I, P_O) = (P'_I, P'_O)\right\}\\
&(6)\ P_w = \left\{p_{(P_I, P_O)} | (P_I, B) \in Y_w\right\} \cup \{I_w, O_w\}\\
&(7)\ F_w = \left\{a, p_{(P_I, P_O)} | (P_I, P_O) \in Y_w \wedge a \in P_I\right\} \cup \left\{p_{(A,B)}, b | (P_I, P_O) \in Y_w \wedge P_O\right\}\\
&\qquad \cup \{(I_w, t) | t \in T_I\} \cup \{(t, O_w) | t \in T_O\}\\
&(8)\ \alpha_w = \{P_w, T_w, F_w\}
\end{aligned}
\tag{1}
$$

Performance evaluation of Alpha and other discovery algorithms can be done through conformance checking by trace matching of the sequence in the event log and the discovery process model [25]. To gain better conformance checking, model-driven analysis was used, using a top-down and a bottom-up strategy respectively [26].

In the top-down strategy, a YAWL reference model was designed using the YAWL editor to create a business process workflow model, as shown in Figure 1. In step-by-step task simulation, the workflow is executed by a YAWL server, as shown in Figure 2.

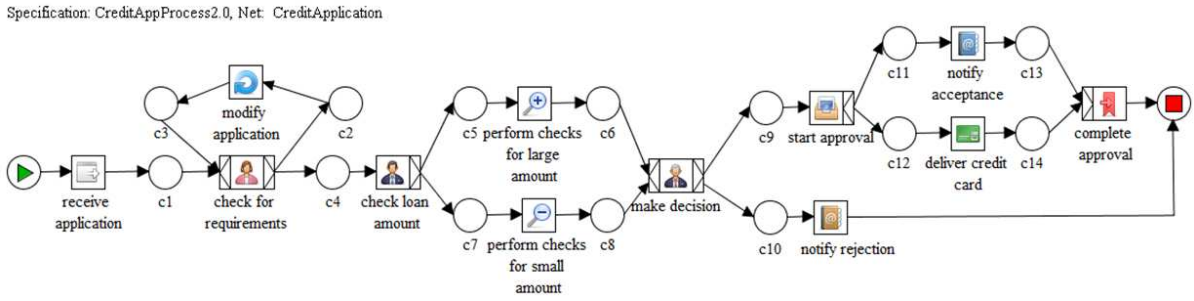


FIGURE 1. YAWL model to simulate a credit application business process [22]

FIGURE 2. YAWL server engine workflow task executor

The YAWL server also automatically generates an application form according to the data model that is determined for every task by the YAWL model, as alternative other technology supported this simulation such as *business process model server* (BPMS) that generates *business process execution language* (BPEL) [26]. This simulation model, also produces event log data that reflect the transactions in the overall task workflow.

By using the bottom-up strategy, the result of the event log can be applied to discovering a functional workflow model [27]. Conformance checking between the workflow produced by the discovery model and the reference model will prove the correctness of the discovery algorithm.

The list of tasks from the business process workflow is shown in Table 1, while L_1 is the log set inside the event log that is generated by each transaction goal from the simulation. The simulated workflow describes transaction automation for credit application submission in a banking business process.

TABLE 1. Follow footprint matrix for L_1

T_{in}/T_{out}	A	B	C	D	E	F	G	H	I	J	K	L
A		>										
B			>	>								
C		>										
D					>	>						
E						>	>					
F					>		>					
G								>	>			
H										>	>	
I												
J												>
K												>
L												

This business process scenario begins when a customer proposes a loan to the bank. For the first task, through a credit staff, bank will receive file submission "*receive application* (A)", and then he checks the fulfillment of requirements required "*check for requirements* (B)". If a problem is found, it is necessary for customer to repair "*modify application* (C)", and he will check the application again until it meets the requirements.

When the credit application meets the requirements, a "*check loan amount* (D)" is estimated. These are two loan categories, first "performs checks for large amount (E)" and second "checks for small amount (F)". A credit analyst will assess credit appropriateness and "*make decision* (G)".

If the application is approved, the bank offers a "*start approval* (H)", then requests the customer to sign a "*notify acceptance* (J)" and he accepted a credit card "*deliver credit card* (K)" which indicates credit approval has been agreed "*complete approval* (L)". However, if application is not approved, then the "*notify rejection* (I)" is delivered to the customer.

Considering a simulation of the YAWL model in Figure 1, the business process scenario is done by the bank to check the consumer's credit application when applying for a loan. Through all the different data transaction forms that are executed by the operators and credit analyzers, the credit application in Figure 2 will generate the following event log:

L_1: ABDEFGI, ABDFEGI, ABDEFGHJL, ABDEFGHKL, ABCBDEFGI, ABCBDFEGI, ABCBDEFGHJL, ABCBDEFGHKL, ABDFEGHJL, ABDFEGHKL, ABCBDFEGHJL, ABCBDFEGHKL.

For discovery of an event log, the discovery process in Alpha [14] starts with event log extraction, i.e., steps (1) to (3) in Lemma 1. These steps check the tuple relation between each task and the task precedence, generating the follow relation matrix, as shown in Table 1. In order to construct a workflow network (WF-net), the follow relation matrix is converted to a footprint matrix based on causality logic, i.e., causality ($\rightarrow$), parallel ($||$) and unrelated ($\#$). The causal relations inside the footprint matrix are shown in Table 2.

TABLE 2. Causal footprint matrix based on Alpha base logic

T_{in}/T_{out}	A	B	C	D	E	F	G	H	I	J	K	L
A	$\#$	$\rightarrow$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$
B	$\leftarrow$	$\#$	$\rightarrow$	$\rightarrow$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$
C	$\#$	$\rightarrow$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$
D	$\#$	$\#$	$\#$	$\#$	$\rightarrow$	$\rightarrow$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$
E	$\#$	$\#$	$\#$	$\leftarrow$	$\#$	$\|\|$	$\rightarrow$	$\#$	$\#$	$\#$	$\#$	$\#$
F	$\#$	$\#$	$\#$	$\leftarrow$	$\|\|$	$\#$	$\rightarrow$	$\#$	$\#$	$\#$	$\#$	$\#$
G	$\#$	$\#$	$\#$	$\#$	$\leftarrow$	$\leftarrow$	$\#$	$\rightarrow$	$\rightarrow$	$\#$	$\#$	$\#$
H	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\leftarrow$	$\#$	$\#$	$\rightarrow$	$\rightarrow$	$\#$
I	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\leftarrow$	$\#$	$\#$	$\#$	$\#$	$\#$
J	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\leftarrow$	$\#$	$\#$	$\rightarrow$
K	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\leftarrow$	$\#$	$\#$	$\rightarrow$
L	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\leftarrow$	$\leftarrow$	$\#$

Alpha must build a WF-net structure by determining the candidate places (X_w) based on the transition tasks (T_w) from the footprint matrix generated by steps (4) and (5). X_w contains the pair set of input transition places (P_I) and output transition places (P_O) with the possibility of single or multi-independent causality without being followed by the same transition. For example, task A is the input for task B. Because A has single causality, it will produce place (A, B). Moreover, task (A, C) is the set of inputs for task B and has multi-independent causality so it will produce place $p(\{A, C\}, B)$.

The footprint analysis result expresses the place sets of causal task pairs, so that X_w = set of single causality $\cup$ set of multi causality from $(P_I \in T_w, P_O \in T_w) = (A, B)$, $(B, C), (B, D), (C, B), (D, E), (D, F), (E, G), (F, G), (G, H), (G, I), (H, J), (H, K), (J, L)$, (K, L), $(\{A, C\}, B)$, $(\{E, F\}, G)$, $(B, \{C, D\})$, $(D, \{E, F\})$, $(\{J, K\}, L)$, $(G, \{H, I\})$, $(H, \{J, K\})$.

Because of the avoidance of redundant places, the place set contains only the maximal path from combined set X_w as the complete final place set (Y_w) by removing the single causality relation that is contained within the multi-independent causality collection, so that $Y_w = (\{A, C\}, B), (\{E, F\}, G), (\{J, K\}, L), (B, \{C, D\}), (D, \{E, F\}), (G, \{H, I\})$, $(H, \{J, K\})$.

Finally, the place set is P_w, which contains place set Y_w added with the input start place set $p(I_w)$ and the output finish place set $p(O_w)$, so that P_w has the following Lemma 2:

$$\begin{aligned} P_w &= p(Y_w) \cup p(I_w, O_w) = p(Y_w) \cup p(t \in T_I, t \in T_O) \\ &= (Start, A), (\{A, C\}, B), (\{E, F\}, G), (\{J, K\}, L), (B, \{C, D\}), \\ &\quad (D, \{E, F\}), (G, \{H, I\}), (H, \{J, K\}), (\{I, L\}, End). \end{aligned} \tag{2}$$

According to the gateway behavior in the *structured workflow net* (SWF-net), P_w has five logical possibilities: CAUSAL, XOR-split, XOR-join, AND-split and AND-join. In this case a CAUSAL relation is found only at the start place because the other places have multi-independent causality. XOR-split is built from the combined set of unrelated

tasks (#) in the output destination, $p(B, \{C, D\})$. Then XOR-join is created from the combined set of # from the input destination, $p(\{A, C\}, B)$. Moreover, the AND gate is built from the combined set of the parallel input and output (||), i.e., $p(D, \{E, F\})$ for AND-split and $p(\{E, F\}, G)$ for AND-join.

The complete WF.net graph will be generated by the firing map (F_w), which is produced by the combined sets of input task, place and output task from the P_w set as follows:

$$\begin{aligned} F_w = &(Start, A), (A, p(\{A, C\})), (C, p(\{A, C\})), (p(\{A, C\}, B)), (B, p(B, \{C, D\})), \\ &(p(B, \{C, D\}), C), (p(B, \{C, D\}), C), (p(B, \{C, D\}), D), (D, p(D, \{E, F\})), \\ &(p(D, \{E, F\}), E), (p(D, \{E, F\}), F), (E, p(\{E, F\}, G)), (F, p(\{E, F\}, G)), \\ &(p(\{E, F\}, G), G), (G, p(G, \{H, I\})), (p(G, \{H, I\}, H)), (p(G, \{H, I\}, I)), \\ &(H, p(H, \{J, K\})), (p(H, \{J, K\}, J), (p(H, \{J, K\}, J), K), (J, p(\{J, K\}, L)), \\ &(K, p(\{J, K\}, L)), (J, p(\{J, K\}, L)), (p(\{I, L\}), End). \end{aligned}$$

The final Alpha discovery result has a WF-net structure that is determined by $\alpha_w = \{P_w, T_w, F_w\}$, as shown in Figure 3. Alpha establishes the correctness of the discovery result with the ProM tool and it also fully conforms to the YAWL reference model. Compared to the heuristic algorithm, Alpha's discovery model has more correctness than heuristic, as shown in Figure 4.

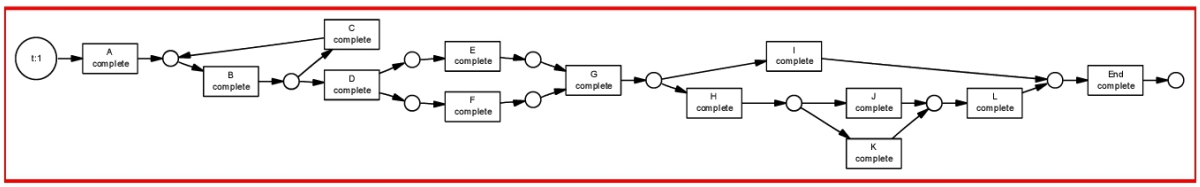


FIGURE 3. Alpha discovery result in ProM

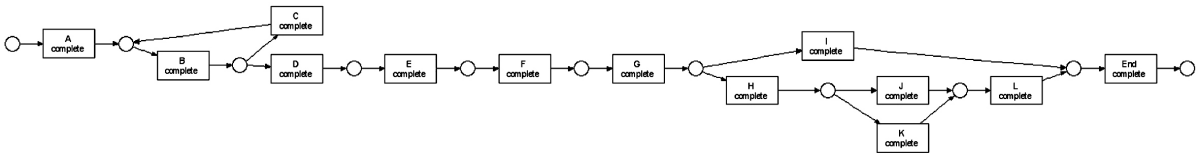


FIGURE 4. Heuristic discovery result in ProM

For the enhancement of Alpha, the existing Alpha* [12] and Alpha++ [13] were used. They contain the following two additional logical patterns:

5) One-loop (Δ), if redundant tasks are found (A, A), $t = t_1 = A$

6) Two-loop ($\diamondsuit$), if a loop structure is found (A, B, A), $t = t_2 = A$ and $t_1 = B$

With these two additional logic patterns, the Alpha++ algorithm has an advanced tuple loop pattern based on Δ and $\diamondsuit$ in every log trace, so that Alpha++ has better discovery accuracy than Alpha in loop mining. However, the addition of loop base patterns significantly increases the time complexity.

Alpha and Alpha++ have become standard plug-ins in ProM tools. Moreover, they have been proven to improve the discovery results according to the WF-net workflow model [14].

The next enhancement of Alpha++ for Alpha is Alpha#. Alpha# improves some logic patterns that are not solved accurately during the discovery process by Alpha++,

i.e., *redo*, *skip*, and *switch* based on logic inspection of the follow (>) and two-loop (Δ) patterns [15].

Alpha and its enhancements perform every step of the process of determining places and firings locally on the whole event log, which greatly increases the time complexity, possibly leading to overfitting of the numbers of places and firings.

3. **Main Result.** The main result of this study is a novel algorithm within the deterministic discovery mining process, which adopts several patterns from Alpha's tuple analysis. The final discovery result from our proposed method, the Alpha-T algorithm, is shown in Figure 5.

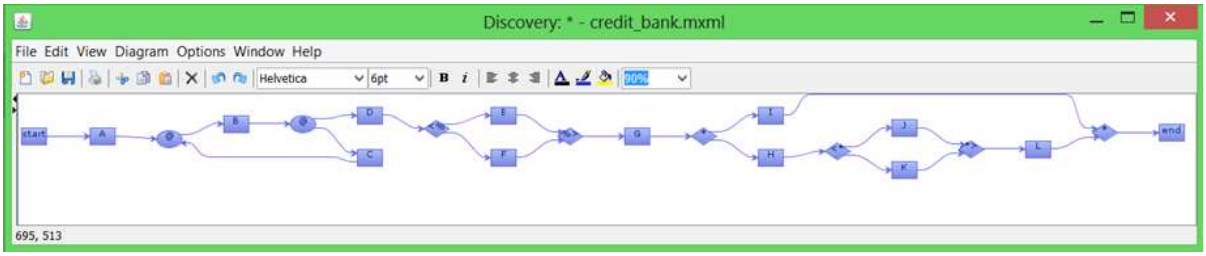


FIGURE 5. Graph visualization of Alpha-T discovery result

The graph visualization of Alpha-T was created using JAVA and *JGraph* graph library support. From this result it can be found that the Alpha-T result had the same correctness level as the Alpha discovery result standardized by BPMN model. Moreover, the result ensures the soundness of the SWF-net on the firing connections, so that it is proven that the workflow model has correctness in the structural and behavioral perspectives.

4. **Control Design.** In this section, we describe Alpha-T algorithm to discover the logic pattern contained within an event log. It aims to construct workflow model which is expected to produce behavior model in BPMN and achieve SWF-net structure.

4.1. **Alpha-T algorithm.** The Alpha-T algorithm is presented as a proposed method to improve time complexity performance and simplify concurrent processes to discover the event log.

The Alpha-T algorithm has 3 stages, i.e., preprocessing, processing and post-processing. These are 4 sub-processes in the preprocessing stage:

- structuring the ALT object
- extracting the event log to the ALT forest (steps (1)-(4))
- weighting on the ALT forest node (coincide step (1))
- sorting on the ALT forest node (step (5))

The processing stage has 3 sub-processes, i.e.:

- Boolean logic matching on the concurrent looping (step (6))
- filtering on the node transition (step (7))

Furthermore, in the post-processing stage a direct follow graph is built and a graph visualization of the firings from the discovery is produced (step (8)).

The process of the Alpha-T algorithm is as Lemma 3 followed:

(1) $\forall N = \left\{t_{\{\rho,\partial\}} \in \forall N | \exists_{\sigma=t_1 t_2 \ldots t_n \in w; i \in \{1,2,\ldots,n\}} t \in \sigma \rightarrow [\rho = t_i > t; \partial = t > t_i]\right\}$
(2) $L_0 L = \left\{t \in \forall N | \exists_{\sigma=t_1 t_2 \ldots t_n \in w; i \in \{1,2,\ldots,n\}} [t = t_{i-1} > t_i; t_{i-1} = t_i]\right\}$
(3) $T_I = \{t \in \forall N | \exists_{\sigma \in w} t \in \text{first}(\sigma)\}$
(4) $T_O = \{t \in \forall N | \exists_{\sigma \in w} t \in \text{last}(\sigma)\}$

$$
\begin{aligned}
&(5)\ \forall N = \text{Sort asc}\{N_{\{\rho,\partial\}} \in \forall N\} \text{ by } W_t \\
&(6)\ \forall N = P\{N_{\{\rho,\partial\}} \in \forall N | \text{as Lemma 13}\} \\
&(7)\ \forall N = F\{N_{\{\partial\}} \in \forall N | \text{as Lemma 8}\} \\
&(8)\ \textit{Alpha-T} = \textit{Graph}\left(N_{\text{vertex}\rightarrow\text{edge}(\partial)} \in \forall N\right)
\end{aligned} \tag{3}
$$

4.2. **Construct adjacency list tree structure.** To obtain a more efficient and dynamic data structure, the abstract of data type (ADT) used by Alpha-T is the adjacency list tree (ALT). ALT is an alternative ADT for the graph representation, where ALT is more efficient in terms of memory space than ADT.

Alpha-T creates a prefix tuple relationship based on the global event log in step (1), called GTPR, to directly analyze the pattern of logic between a task and its precedent, as illustrated in Figure 6.

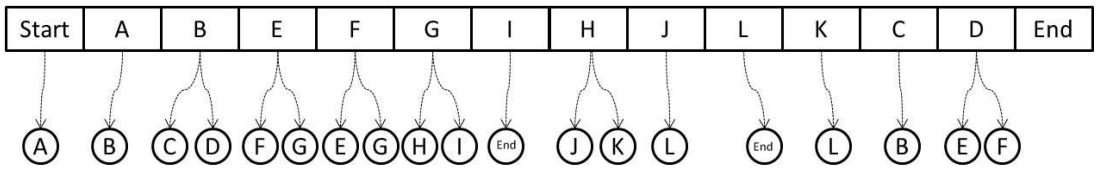


FIGURE 6. Adjacency list tree from event log L_1

In accordance with event log L_1, ALT shows the relationship between activity list $T(A \ldots Z)$ in the reference forest tree relationship and the determined children (∂), which is a successor tuple from the sequence order event log in accordance with step (1). The object definition of the ALT task node is expressed as Definition 4 followed:

$$
\begin{aligned}
&\textit{Class Node} \\
&\quad \textit{String data}; \\
&\quad \textit{String type}; \\
&\quad \textit{ConcurrentLinkedDeque (Node) parents} \\
&\quad \textit{ConcurrentLinkedDeque (Node) children} \\
&\quad \textit{Node startPoint}; \\
&\quad \textit{Node endpoint}; \\
&\quad \textit{Boolean inner-gate}; \\
&\quad \textit{Boolean enable}; \\
&\quad \textit{Boolean depth};
\end{aligned} \tag{4}
$$

From the definition of the node class, each node object has an input set as parents $< \rho >$ and an input set as children $< \partial >$, with the conditions as given in Lemma 5 followed:

$$
\begin{aligned}
&T \in T_\rho | \rho \text{ are parents of } T,\ \rho = \{A, B, \ldots, Z\} \\
&T \in T_\partial | \partial \text{ are children of } T,\ \partial = \{A, B, \ldots, Z\}
\end{aligned} \tag{5}
$$

As the bag of task nodes, the forest tree object is used, which is defined as Definition 6 followed:

$$
\begin{aligned}
&\textit{Class Forest} \\
&\quad \textit{ConcurrentLinkedDeque(Node) adjacencyList}; \\
&\quad \textit{Node Start}; \\
&\quad \textit{Node End};
\end{aligned} \tag{6}
$$

The ALT graph grows with additional members from the place gateway, which changes the task and gateway relationship as determined by Lemma 7 followed:

$$\forall N = T_w \cup G_w \tag{7}$$

while,

$\forall N$: ALT node set

G_w: gateway set, $G_w = \{G_<, G_>\}$

$G_<$: split gateway

$G_>$: join gateway

$G_<$ and $G_>$ are produced by Boolean logic pattern analysis when the relationship between task and gateway for valid firings (F_w) is as Lemma 8 followed:

$$\begin{aligned} F_w &\leftarrow N \in T_\partial | N : \{G_<, G_>\},\ |N| = 1 \\ F_w &\leftarrow N \in G_{<_\partial} | N : \{T, Slack\},\ |G_<| = 1 \\ F_w &\leftarrow N \in G_{>_\partial} | N : \{G_<, T\} \end{aligned} \tag{8}$$

The determination of a simple parent-child relationship pattern between task and gateway produces a workflow model with a closed boundary in the behavioral gateways $G_<$ and $G_>$, also allowing an open boundary inside singleton and free-choice patterns that have only $G_\rightarrow$. This rule intends to fulfill the rules of a WF-net based on Petri-net firing between the place and transition sets $\{P_w, T_w, F_w\}$, as shown in Figure 7.

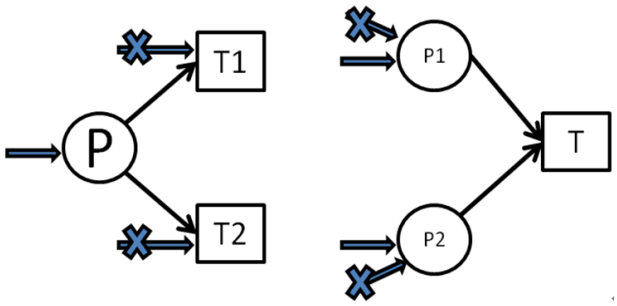


FIGURE 7. WF-net *soundness* rules [15]

4.3. **Weighting ALT node.** After the process of extraction from the event log, Alpha-T performs depth weighting of the tree nodes, which is useful for establishing the sequencing order of each task node and also their children. The weight updating W_T of the depth of each node inside $\forall N$ is expressed in the following Expression 9:

$$\begin{aligned} &for\ \exists T \in \forall N \rightarrow \\ &\quad iff\ (T\ is\ First) \\ &\quad\quad W_T = 1 \\ &\quad else\ iff\ (T' \in \forall N\ AND\ T'_\rho \nexists T_\rho) \\ &\quad\quad\quad W'_T = (W'_T + W_\rho)/2 \\ &\quad\quad\quad iff\ (T'_\vartheta \nexists T_\rho) \\ &\quad\quad\quad\quad W'_T = W_\rho + 1 \\ &\quad\quad\quad else \\ &\quad\quad\quad\quad W'_\rho = (W'_T + W_\rho)/2 \end{aligned} \tag{9}$$

By Expression 9, the task T traversal is done inside $\forall N$. For the first time, when T as the starting node $\forall N \leftarrow T$, that has weight as $W_T = 1$. Otherwise, for the next input if

T has existed inside ALT ($T' \in \forall N$) and the predecessor of T that is T_ρ has not existed inside ALT task parent T'_ρ, the weight of existing task is updated as average of the weight, $W'_T = (W'_T + W_\rho)/2$.

Furthermore, to avoid overlapping a weight in concurrency relation, if T' does not have parallel relation within its parent T_ρ ($T'_\vartheta \nparallel T_\rho$), T' weight is updated with a weight $W'_T = W_\rho + 1$. Other than that, if they have parallel relation, existing parent weight updated with $W'_\rho = (W'_T + W_\rho)/2$. This expression has purpose to close the weight deviation of parent and child task on the parallel relation and to far on the direct causality relation.

After the task depth weighting process, we get the following number of results as shown in Table 3.

TABLE 3. Task weight node inside ALT

Start	A	B	C	D	E	F	G	H	I	J	K	L	End
0	1	2.5	2.75	3	4.5	4.75	6	7	7	8	8	9	14

From Table 3, the depth weighting results obtained a clear task weight deviation as a reference for the sequence order from the *starting* task to *finish* that is very useful to avoid an overlapping of tasks order that often occur due to the concurrency causality.

Furthermore, the result of the weighted task is also used to analyze the implications of the logic possibility on the tuples either between the parent-child relationship $N(\alpha, \beta)$ or also between each child $(\alpha_\partial, \beta_\partial)$, and then will be combined with sequential causality analysis to improve logical analysis accuracy.

This method has advantages on computational complexity when compared with the using of advanced complex pattern causality matching as applied to Alpha# where too many patterns are analyzed in various invisible patterns [15]. The logical implications that can be produced from the weighting assessment are as Lemma 10.

In the parent to children relationship,

$$W_N < W_{(\alpha,\beta)} \rightarrow \mathit{Follow}$$
$$W_N > W_{(\alpha,\beta)} \rightarrow \mathit{Reply}$$

Also, in the child to child relationship,

$$W_N < W_{(\alpha,\beta)},\ W_\alpha = W_\beta \rightarrow XOR$$
$$W_N < W_{(\alpha,\beta)},\ 0 < W_\alpha - W_\beta < 1 \rightarrow \mathit{AND}, \mathit{TwoLoop}, \mathit{Skip} \tag{10}$$

Meanwhile, the depth weights of gateways $G_<$ and $G_>$ for the gateway pair are the mean values of the task members' weight, as in the following Lemma 11:

$$W(G_S, G_J) = \sum_{c=0}^{n} W\left(G_{S_{\partial(c)}}\right) \Big/ n | G_{S_\partial} : T \tag{11}$$

where,

n: number of children

c: child weight

4.4. **Sorting ALT nodes.** By using tree-map hashing, sorting of the task nodes is done based on the depth weight value as expressed in Lemma 12. Sorting of their children is also done, based on the task node sorting. The result of ALT after sorting is as illustrated in Figure 8.

$$Map(k, v) \rightarrow \mathit{treeMap}(N, W(N))$$
$$\forall N \leftarrow k \tag{12}$$

where,
Map: bag set of the generic data structure map
treeMap: hashing/sorting function on the map
k: node key
v: node value

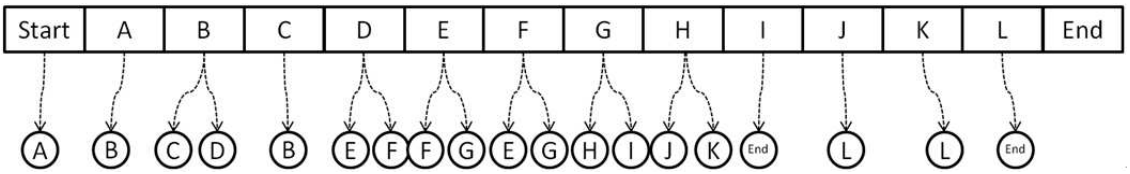


FIGURE 8. Sorting result of ALT from L_1

4.5. **Alpha-T logic pattern.** The GTPR pattern in Alpha-T is analyzed by the place of node N as a parent reference related to second-tuple child node α and sibling β as in the following Lemma 13:

$$P_{w(N,\alpha,\beta)} = \sum_{i=0}^{m} N_{(i)} \ni \sum_{j=0}^{n} \alpha_{\partial(j)} \wedge \beta_{\partial(j+1)} \tag{13}$$

where,
P_w: place of workflow gateway
N: parent node $|N : \{T, G_>\}, T \neq < T >, G_> \neq \diamondsuit$
α: first-tuple node $|\alpha : T$
β: second-tuple node $|\beta : T$
M: numbers of $\forall N$
N: number of children
i, j: counter index

GTPR is executed in a concurrent loop inside $\forall N$ when N is limited only for a non-inner-gate task $< T >$ and join gateway nodes. Whereas join gateway is not a *two-loop* ($\diamondsuit$). We define 11 logical patterns that are used in Alpha-T that fulfill behavior and structural logic of SWF-net, which are:

1) Serial direct
2) Reply
3) Free-Choice
4) Free-Join
5) Non-Free Choice
6) One-loop
7) Two-loop
8) XOR
9) Skip
10) AND
11) OR

1) *Serial direct* ($^{\rightarrow}$)

$^{\rightarrow}$ is a singleton relation that occurs when parent nodes have only one child, as defined in the following Lemma 14:

$$\textit{Iff } (N_\partial \ni (\alpha) || N_\partial| = 1) \rightarrow (N_\partial \leftarrow {}^{\rightarrow}; {}^{\rightarrow}{}_\partial \leftarrow \alpha) \tag{14}$$

Example 4.1. *ABC. Node A has only one child B, so that serial gateway* $^{\rightarrow}$ *is created and also B is transferred as* $^{\rightarrow}$ *child. The same goes for task B, which has only one child, C. These configurations are shown in Figure 9(a) as the parent-child relationship inside ALT, while the discovery result is shown in Figure 9(b).*

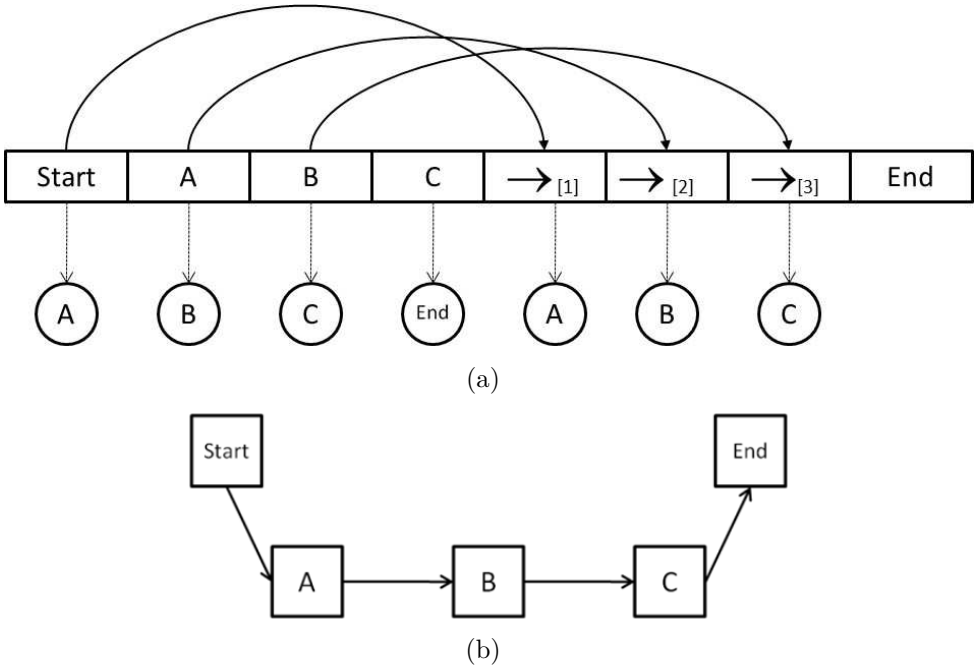


FIGURE 9. (a) SD pattern in ALT, (b) SWF-net for SD

2) *Reply* ($^{\leftarrow}$)

$^{\leftarrow}$ occurs when the parent node has a smaller weight than the child node and has vice versa relation, so it will produce *XOR* loop that is also called *redo* pattern. This pattern is applied for as defined by the following Lemma 15:

$$\begin{aligned} &\textit{Iff } (N_{\partial} \ni (\alpha, \beta) | W(N) > W(\alpha)) \rightarrow \\ &(N_{\partial} \leftarrow {}^{<*}; {}^{<*}{}_{\partial} \leftarrow ({}^{*>}, \beta); {}^{*>}{}_{\partial} \leftarrow \alpha; \alpha_{\partial} \leftarrow {}^{\rightarrow}; {}^{\rightarrow}{}_{\partial} \leftarrow N; \alpha_{\rho} \leftarrow {}^{*>}) \end{aligned} \tag{15}$$

Example 4.2. *ABCD, ABCBCD. Within the tuple relationship between* $C(B, D)$*, C has a B child node that has a smaller weight, so that C has an* $^{\leftarrow}$ *to B that is implemented as* $({}^{<*}, {}^{*>})$ *loop as shown in Figures 10(a) and 10(b).*

3) *Free-Choice* ($^{\rightarrow *}$)

$^{\rightarrow *}$ has only an inclusive *XOR* gateway-split from *multi-serial* that constructs places that each other has tight different link [11] (See Figure 11). This pattern is applied for as defined by the following Lemma 16:

$$\textit{Iff } (N_{\partial} \ni (\alpha, \beta) | \alpha \notin \beta_{\partial}; \beta \notin \alpha_{\partial}; \alpha_{\partial} \notin \beta_{\partial}) \rightarrow (N_{\partial} \leftarrow {}^{\rightarrow *}; {}^{\rightarrow *}{}_{\partial} \leftarrow (\alpha, \beta)) \tag{16}$$

Example 4.3. *ABDF, ACEF. Within the tuple relationship between* $A(B, C)$*, B and C have different child dependency on each other.*

4) *Free-Join* ($^{*\rightarrow}$)

Opposite to *free-choice*, $^{*\rightarrow}$ has only an inclusive *XOR* gateway-join that constructs termination from many places that each other has a common termination. This pattern has purpose to guarantee all output places have correctness in generalization termination.

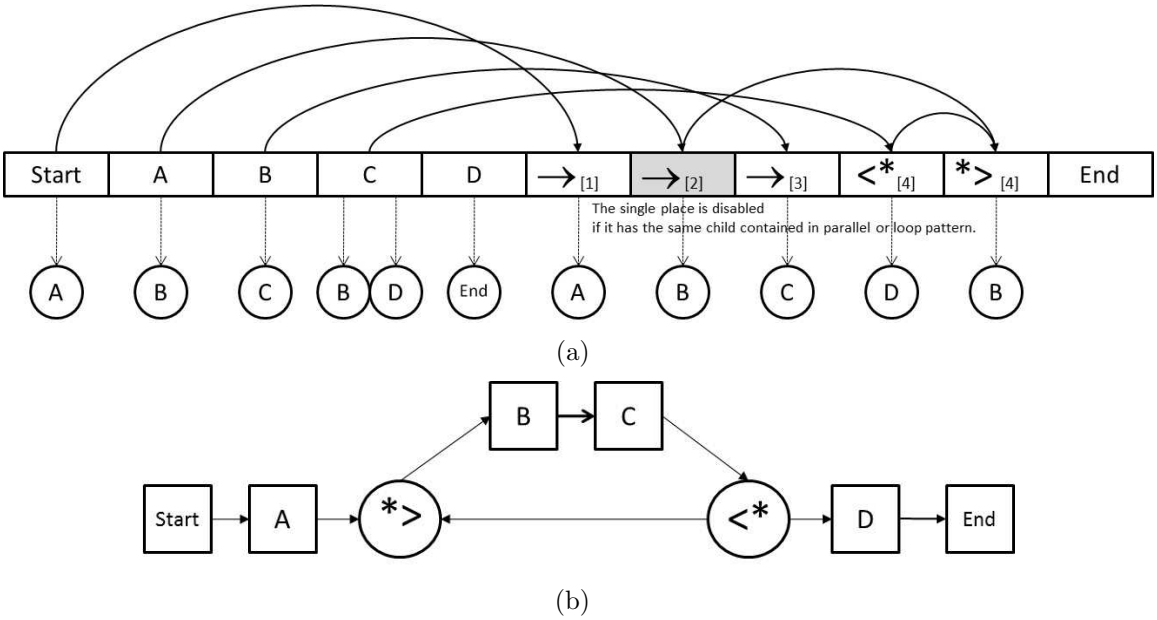


FIGURE 10. (a) Reply pattern in ALT, (b) SWF-net for *reply* pattern

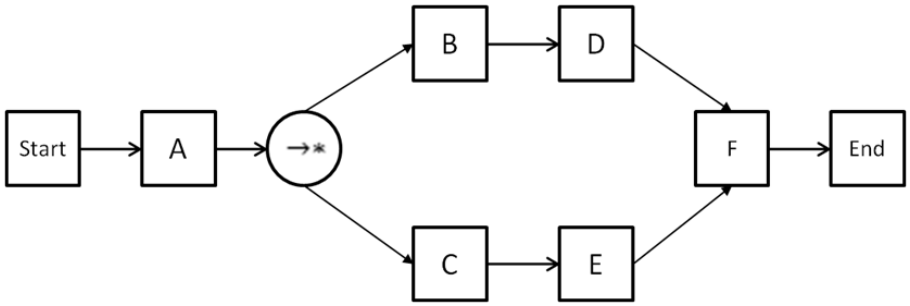


FIGURE 11. SWF-net for *free-choice* pattern

Other than that, $^{*}\!\rightarrow$ is also used to solve termination for the event finish. This pattern refers to configure P_W output place Lemma 2 in Alpha. This pattern is applied for as defined by the following Expression 17.

$$\begin{aligned} &for\ \exists T \in \forall N \rightarrow \\ &\quad iff\ (T\ is\ non\_inner\_gate) \\ &\quad iff\ (|T_\rho| > 1) \\ &\qquad T_{\rho\leftarrow\partial} \leftarrow {}^{*}\!\rightarrow \\ &\qquad {}^{*}\!\rightarrow_{\partial} \leftarrow T \end{aligned} \tag{17}$$

Different from all other patterns that are generated by the children causality, $^{*}\!\rightarrow$ is generated only by *the parent causality*. As Expression 17, $^{*}\!\rightarrow$ is executed on the *post-processing* after filtering step. $^{*}\!\rightarrow$ has a logical pattern (see Figure 12) to reconfigure multi-serial termination that has a *soundness* SWF-net considered with Example 4.3.

5) *Non-Free-Choice* (*NFC*)

NFC has an exclusive XOR gateway that constructs implicit places whenever each other transition as the input for a place has a different dependency [28]. NFC has a

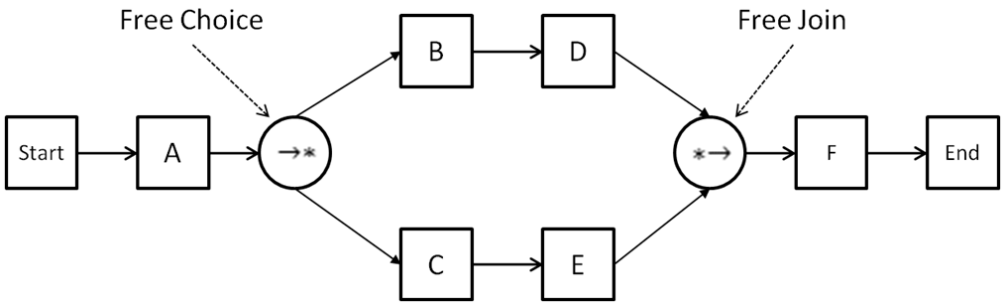


FIGURE 12. SWF-net for *reply* pattern

logical pattern as defined in the following Lemma 18:

$$Iff\ (N_\partial \ni (\alpha,\beta)|\alpha \notin \beta_\partial; \beta \notin \alpha_\partial; \alpha_\partial \cap \beta_\partial; W(\alpha) \neq W(\beta)) \rightarrow XOR \tag{18}$$

Example 4.4. *ABDB, ACDE. Within the tuple relationship between $A(B,C)$, B and C have the same child D, so an exclusive XOR is created. Other than that, on tuple $D(B,E)$, D produces redo to (B,E) which causes create link to existing XOR. For these reasons, D has an NFC link to XOR gateway because D does not have C child. NFC has the same configuration such as XOR and AND gateways (see Figure 16).*

6) *One-loop* (Δ)

Δ indicates the occurrence of a single task one-loop free SWF-net Alpha++ loop [14]. This pattern occurs inside a redundant task (L_0L) during log extraction when a repeating task is found. Δ is processed at the end of discovery after filtering, which is executed through firing a parent-input and child-output set (see Figure 13) as expressed in the following Lemma 19:

$$Iff\ (T \in L_0L < T >) \rightarrow \left(T_\rho \leftarrow {}^{*>}; {}^{*>}{}_\partial \leftarrow T; T_\partial \leftarrow {}^{<*}; {}^{<*}{}_\partial \leftarrow ({}^{*>}, T_\partial)|T_\partial : G\right) \tag{19}$$

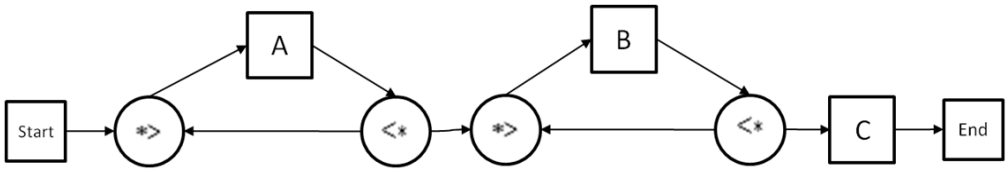


FIGURE 13. SWF-net for *one-loop*

Example 4.5. *AABC, ABBC. Tasks A and B are repeating tasks and will create an XOR loop.*

7) *Two-loop* ($\diamondsuit$)

$\diamondsuit$ indicates a locked short-redo loop between two interconnected tasks, called a two-loop free sound SWF-net loop in Alpha++ [14], as expressed in Lemma 20. $\diamondsuit$ is produced with condition α having a direction only to the parent because it has only one child, N (see Figure 14).

$$\begin{aligned} &Iff\ \left(N_\partial \ni (\alpha,\beta)|(\rho) \leq (\alpha); N \in \alpha_\partial; |N_\partial| \geq 2; |\alpha_\partial| = 1; \alpha \notin \beta_\partial; \beta \notin \alpha_\partial\right) \rightarrow \\ &\left(N_\partial \leftarrow G_{S_{(*)}}; G_{S_{(*)\partial}} \leftarrow \alpha; \alpha_\partial \leftarrow G_{S_{(*)}}; G_{J_{(*)\partial}} \leftarrow (\alpha,\beta); \alpha_\partial \leftarrow G_{S_{(*)}}\right) \end{aligned} \tag{20}$$

Example 4.6. *ABAC, ABAD. $\diamondsuit$ occurs when the tuple parent $A(B,C,D)$ has many children, and one of its children has only one child as its parent such as B. So, the close vice versa causality creates a* two-loop *pattern such as B to A.*

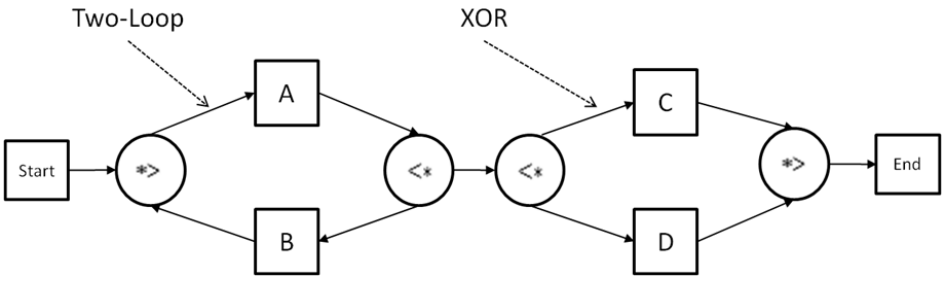


FIGURE 14. SWF-net for two-loop

8) *Skip* $(< \# >)$

Skip is a causal relation that creates a jump XOR alternative. Skip occurs when a tuple task α is the precedent of β but not vice versa, as expressed in the following Lemma 21:

$$\begin{aligned} &\textit{Iff } (N_\partial \ni (\alpha,\beta)|(N) \leq (\alpha); |N_\partial| \geq 2; \beta \in \alpha_\partial; \alpha \notin \beta_\partial \rightarrow \\ &(N_\partial \leftarrow {}^{<*}; {}^{<*}{}_\partial \leftarrow (\alpha, {}^{*>}); {}^{*>}{}_\partial \leftarrow \beta) \end{aligned} \tag{21}$$

Example 4.7. *ABC, AC. A parent tuple has children B and C, whereas B is a causality of C so A needs a skip to C. Skip is shown in Figure 15.*

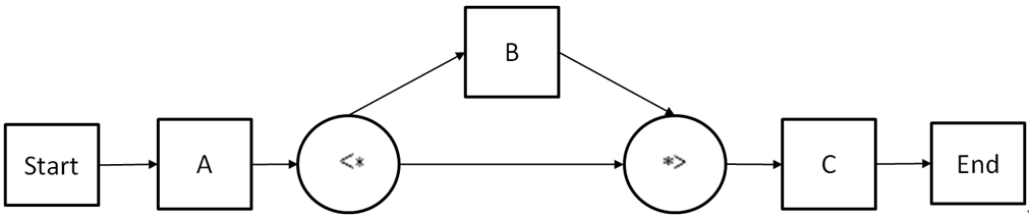


FIGURE 15. SWF-net for skip

9) *AND* $(< \& >)$

AND occurs when two or more tasks are absolutely parallel, where a tuple task α precedes β and vice versa, as expressed in the following Lemma 22:

$$\begin{aligned} &\textit{Iff } (N_\partial \ni (\alpha,\beta)|W(N) \leq W(\alpha); |N_\partial| \geq 2; \alpha_\partial = \beta_\partial) \rightarrow \\ &(N_\partial \leftarrow <\&; <\&_\partial \leftarrow (\alpha,\beta); (\alpha_\partial,\beta_\partial) \leftarrow \&>); \&>_\partial \leftarrow (\alpha_\partial,\beta_\partial|(\alpha_\partial,\beta_\partial) \neq (\alpha,\beta)) \end{aligned} \tag{22}$$

Example 4.8. *ABC, BAC. Task A has a child B and B has a child A, as shown in Figure 16.*

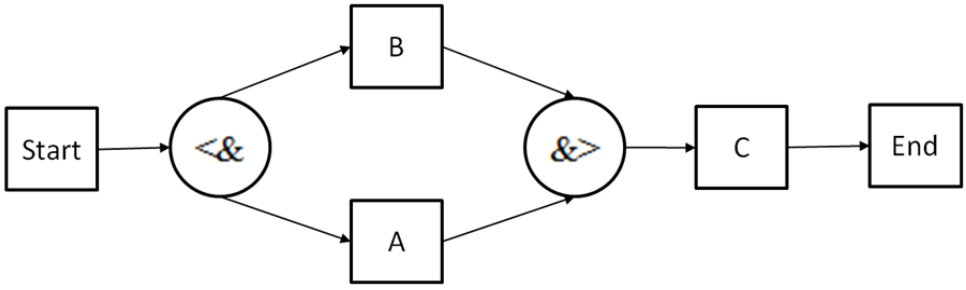


FIGURE 16. SWF-net for *AND*

10) *XOR* ($< * >$)

XOR occurs when two or more tasks are an exclusive alternative so that only one of them can be executed. This occurs when node α has no child β and node β also has no child α and their children are equality, as expressed in the following Lemma 23:

$$\begin{aligned} &\textit{Iff } (N_\partial \ni (\alpha,\beta)|\beta \notin \alpha_\partial; \alpha \notin \beta_\partial; \alpha_\partial = \beta_\partial) \rightarrow \\ &(N_\partial \leftarrow G_{<*}; G_{<*_\partial} \leftarrow (\alpha,\beta); (\alpha_\partial,\beta_\partial) \leftarrow G_{*>}); G_{*>_\partial} \leftarrow (\alpha_\partial,\beta_\partial|\alpha_\partial,\beta_\partial \neq (\alpha,\beta)) \end{aligned} \tag{23}$$

Example 4.9. *AC, BC. See Figure 16.*

11) *OR* ($< \% >$)

OR occurs when two or more tasks have two or more multi-choice [29] constructed from AND and SKIP logic. It is caused by the set of children of parent tuple that is not equal, so they have multi-causality variants. Since the exclusive AND cannot skip the SKIP causality that occurs on the members, the possible logic is only OR.

The method of OR analysis used here is simpler than using many decision analyses from AND and XOR patterns [29], as described as Lemma 24 followed:

$$\begin{aligned} &\textit{Iff } (N_\partial \ni (\alpha,\beta)|W(N) \leq W(\alpha); |N_\partial| \geq 2; \beta \in \alpha_\partial; \alpha \in \beta_\partial; \alpha_\partial \neq \beta_\partial) \\ &\quad \rightarrow (N_\partial \leftarrow \textit{Replace}(< \& > \leftarrow < \% >)) \end{aligned} \tag{24}$$

Example 4.10. *ACD, BCD, ABCD, ACBD. See Figure 16 as configuration and the result is shown as Table 6.*

4.6. **Concurrent loops in Alpha-T algorithm.** The Alpha-T discovery analysis works over concurrent looping inside ALT, where all enabled and non-inner-gate tasks and join gateway nodes inside ALT are parent sets for tuple analysis as defined in Expression 25.

By Expression 25, if the node is a gateway, it will induct the causality analysis on the join gateway having the benefit to analyze the set of child nodes of children that is contained in the parallel or exclusive gateway, so the analysis will be able to align the relationship of parallel sequences with successors and reduce the number of tasks to analyze. So, it will avoid overfitting pattern that only analyzed independently such worked on Alpha.

Concurrent discovery works well through an iterator loop and concurrent data type support, which are provided by the programming language engine. By using the Java SDK that is used in the Alpha-T implementation various concurrent data types are provided, including: ConcurrentLinkedQueue, ConcurrentLinkedDeque, ConcurrentSkipList.

```
function discoveryLoop(∀N)
    iter ← iterator(∀N)
    While (iter : hasNext()):
        Node n = iter.next()
        iff (n is task OR n is gateway_join)
        iff (n is not gateway_loop)
        iff (n is enable AND n is non_inner_gate)
            execute_pattern(n)
```
(25)

To simplify the matcher for logic pattern, Expression 26 is a configuration summary for all Alpha-T patterns to be executed,

function execute_pattern(n)
n: *ALT nodes,* α: *predecessor,* β: *successor*
Iff $|n_\partial| = 1$
    (1) *Serial Direct*
Else Iff $|n_\partial| > 1$
    Iff $\omega(n) > \omega(\alpha)$
        Iff n *is non_inner_gate AND* α *is non_inner_gate*
            (2) *Reply*
        Else
            (3) *free_choice*
    Else Iff $\beta \notin \alpha_\vartheta$ *AND* $\alpha \notin \beta_\vartheta$
        Iff $|\alpha_\vartheta| = 1$ *AND* $n \in \alpha_\vartheta$
            (7) *Two_Loop*
        Else Iff $\alpha_\vartheta \neq \beta_\vartheta$
            (3) *Free_choice*
        Else
            (10) *XOR/*(6) *Non_Free_Choice*
    Else Iff $\beta \in \alpha_\vartheta$ *AND* $\alpha \in \beta_\vartheta$
        (9) *AND*
        Iff $\alpha_\partial \neq \beta_\partial$
            (11) *Replace* $(AND \leftarrow OR)$
    Else Iff $(\beta \in \alpha_\partial$ *AND* $\alpha \in \beta_\vartheta)$
        (8) *Skip*
**Iff* $N \in L_o L$
    (6) *One_Loop* (26)

In accordance with $\log L_1$, it is illustrated how a concurrent loop from Alpha-T works to implement all logic patterns from the sorted $\forall N$, as shown in Figure 17. This illustrated the start node (in yellow color) will be analyzed first, because the start node has only one child, task A, so by pattern analysis it is a singleton and produces an SD logic pattern and an SD gateway will be added to ALT.

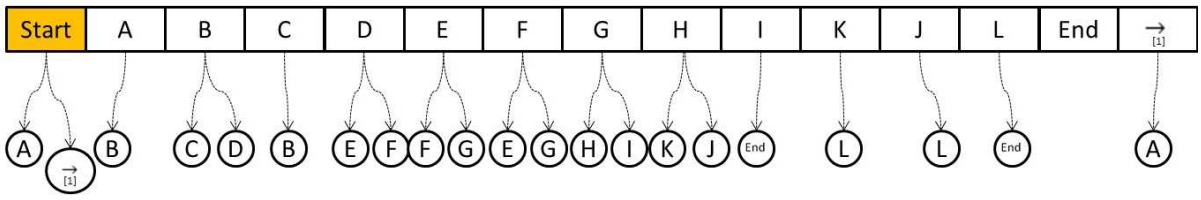


FIGURE 17. First illustration for concurrent pattern analysis ALT for L_1

The next looping trace is node A. Because A is a non-inner-gate task, A is executed independently. Meanwhile, A has a singleton pattern with child B and also produces an SD logic pattern as Figure 18.

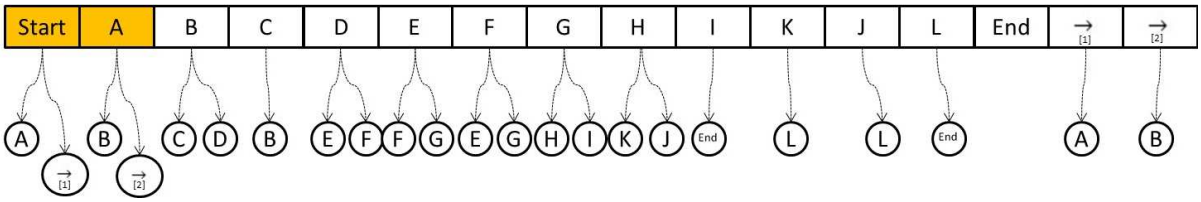


FIGURE 18. Second illustration for concurrent pattern analysis ALT for L_1

In the next loop, B is a non-inner-gate task, so it is executed. B has children C and D, while C has a causality relation with parent B so that it produces a two-loop pattern, as shown in Figure 19. When a two-loop pattern is produced, $*>$ as input and $<*$ as output are created. $\diamondsuit$ pattern configures the task member considered in Lemma 20.

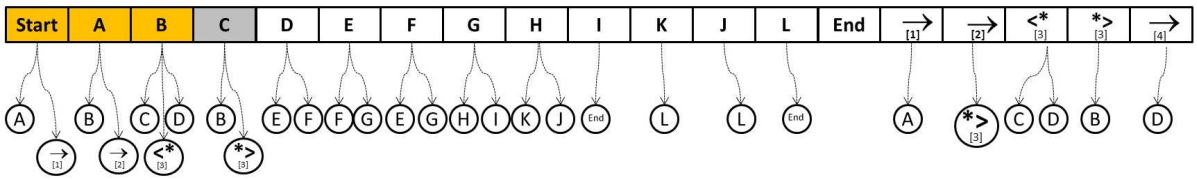


FIGURE 19. Third illustration for concurrent pattern analysis ALT for L_1

Because task C is a loop inside a two-loop gateway pair, task C is set as inner-gate with task B and is not executed in the next loop, as shown in Figure 19 (grey color).

Next, D is executed. Because D has two exclusive parallel children (F, E), it will produce an AND gateway. $<\&$ and $\&>$ gateways will be added to ALT and (E, F) will not be executed. AND-join gateway inherits all children from node task (E, F) except itself as shown in Figure 20.

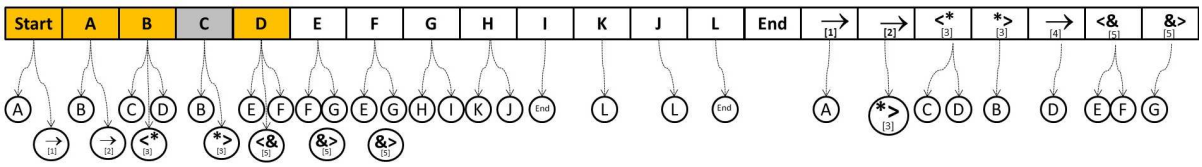


FIGURE 20. Fourth illustration for concurrent pattern analysis ALT for L_1

The execution of the ALT logic pattern is done in a concurrent loop until all parent nodes with limited non-inner-gate tasks, enabled join gateway, and loop have been inspected as shown in Figure 21.

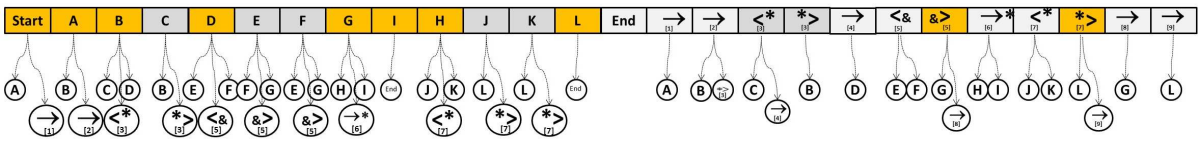


FIGURE 21. Fifth illustration for concurrent pattern analysis ALT for L_1

4.7. **Filtering on firing.** Upon fulfillment of the SWF-net firing, evaluation is done for the gateway firing. Moreover, by using Expression 6 a filter can be implemented to reduce the number of ALT members and also firing number. Firing is done by keeping each transition task with the places that have been produced and deleting all child tasks.

Similarly, on the gateway-join that has been inducted to execute pattern, to avoid redundancy links that cause over-fitting, filtering is done as referring to the filtering in Alpha. Single causality is removed if the task transition is covered by a multi-place gateway, as on serial $A \rightarrow [2]$ that has child $(B, *>[3])$ so that task B is removed, as shown in Figure 22 (B is colored red), and all others adjacency task child with place are removed too, else for the loop nodes.

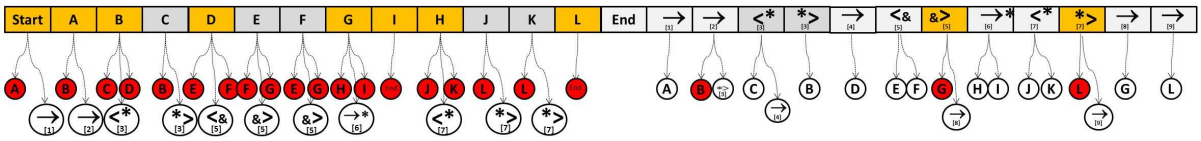


FIGURE 22. Firing filtering inside ALT for L_1

In addition, there exist patterns that are not executed simultaneously on the concurrent loop, which are: *OR*, *One-Loop* and *Free-Join*. Such a free-join is produced by the *End* node (yellow-white color) that has more than 1 non-inner-gates parent such as (I, L) so it is produced ${}^{*}\!\rightarrow_{\partial}[10] \leftarrow$ *End* (yellow color). As shown in Figure 23, final firing post-processing is represented.

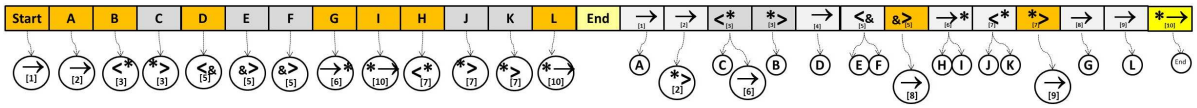


FIGURE 23. Final set of nodes inside ALT for L_1

The result of discovery process is presented to the graph visualization that shows a workflow model as shown in Figure 24, and simplified to the main result as in Figure 5.

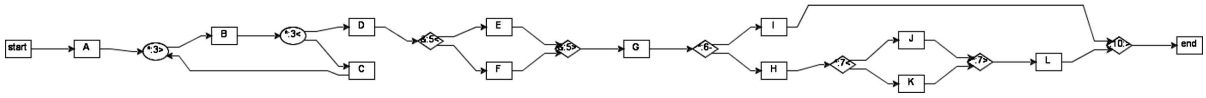


FIGURE 24. Graph of workflow model for final discovery of event log L_1

5. **Result and Analysis.** As proven by ProM testing that was used to show the discovery result from Alpha and the graph model of Alpha-T, both had the same correctness when visual checked with the reference model from YAWL. The computation performances of Alpha and Alpha-T based on time complexity are shown in Table 4. The numbers of time variables taken into account were: l number of logs, t number of tasks, and p number of logic patterns. As a consideration from another time complexity analysis, cyclomatic complexity measurement could be adopted for the concurrent analysis in the discovery process [30].

For the first comparison, an analysis was done based on time complexity within the preprocessing stage to construct the base footprint of the tuple task relation pattern from the event log. Alpha extracts the tuple relation locally, log per log, which takes $\Omega(t)$.

TABLE 4. Time complexity comparison between Alpha and Alpha-T

Process Step	Alpha		Alpha-T	
	Sub-process	Time complexity	Sub-process	Time complexity
Log extraction (steps 1 to 3)	Footprint matrix, trace each log by follow tuple relation and update footprint matrix	$\Omega(t_1 t_2^2) = \Omega(t^3)$ t_1: trace task for each log t_2: set follow footprint matrix	Trace event log and set follow tuple relation inside ALT with parent node and children relation	$\Omega(lt_1t_2) = \Omega(lt^2)$ l: trace event log t_1: trace task for each log t_2: set nodes and children in ALT
Task weighting	–	–	Updating weight of task nodes	$\Omega(t)$ t: get parent children weight + 1
Task sorting	–	–	Sorting parent nodes by weight using hash-map	$\Omega(t_1 \log(t) + t_2^2)$ t_1: sort parent nodes t_2: sort child nodes
Set places (steps 4-6)	Update causality footprint matrix	$\Omega(2t_1t_2) = \Omega(2t^2)$ t_1: check row to column follow t_2: check column to row follow	Trace task parent and children causality relation	$\Omega(pt_1t_2t_3t_4) = \Omega(pt^4)$ t_1: get ALT parent t_2: get ALT child t_3: get ALT grandchild t_4: set tuple grandchild p: number of logic
	Place set (X_w)	$\Omega(t_1^2 + t_2^3)$ t_1: trace row-column single causality t_2: trace column multi-causality tuple	Set places of split and join gateways of children	$\Omega(2t)$ t: get number of grandchildren from tuple child nodes
	Place set (P_w)	$\Omega((t_1 + t_2)t_3) = \Omega(2t^2)$ t_1: trace start task t_2: trace finish task t_3: get places (t_1, t_2)	Filtering	$\Omega(t^2)$ t: check parent to children relation to fulfill SWF-net soundness
	Analyze logic pattern of place	$\Omega(p(t_1^2t_2^2)) = \Omega(pt^4)$ p: number of logic patterns t_1: get pairs in multi-causality t_2: check causality logic	Configure output place for free-choice & termination	$\Omega(t^2)$ t: check children to parent relation to fulfill SWF-net soundness
Firing set (F_w)	Pair set $< P, T >$	$\Omega(t^3)$ t: get token set inside place map, t^3, due on multi-causality places	–	–
Discovery graph	$\alpha_w = \{P_w, T_w, F_w\}$	$\Omega(l(pt^4 + 4t^3 + 3t^2))$	Trace overall ALT nodes and their children on $(N \rightarrow N_\vartheta)$	$\Omega(t_1t_2) = \Omega(t^2)$ t_1: trace parent nodes t_2: get child nodes

Its tuple relation is updated to the follow footprint matrix, which takes $\Omega(t^2)$ so that the footprint construction process takes $\Omega(t^3)$. Then the footprint is converted to the causality matrix with checking the follow matrix, which takes $\Omega(t^2)$, so the preprocessing stage finally requires $\Omega(t^3 + t^2)$.

Meanwhile, within Alpha-T, the overall follow tuple relations are constructed inside ALT based on the global relations from the GTPR, which takes $\Omega(lt^2)$. Meanwhile, addition of the weighting of the ALT step takes $\Omega(t)$, so the tuple construction takes

$\Omega(lt^3)$. Afterwards, the hashing/ordering of the ALT takes $\Omega(t \log t)$, so that the time complexity of the Alpha-T preprocessing stage is $\Omega(lt^3 + t \log t)$.

For the next comparison, an analysis was done of the processing stage. Alpha sets the X_w place sets by checking for single causality, which takes $\Omega(t^2)$, while checking for multi-causality takes $\Omega(t^3)$. Furthermore, to fulfill the behavior of the workflow, the place set is examined for causality logic inside the multi-causality map, so the time complexity for X_w is $\Omega(pt^4)$. Thereafter, for the optimization of place set Y_w, filtering is done to check the single causality that is covered by multi-causality place on input and output pairs, which takes $\Omega(2t^3)$.

Finally, to construct the final place set P_w, the set of places that have causality with a start and finish task transition, Y_w, is added, which takes $\Omega(2t^2)$. The total time complexity of Alpha for the establishment of the place sets is $\Omega\big(pt^4 + 2t^3 + 2t^2\big)$.

Within Alpha-T, determining place sets is executed directly by the pattern of causality logic between the parent node and the children tuples. The concurrent looping on the parent node inside ALT has time complexity $\Omega(t)$ and tracing the children has $\Omega(t)$, while the children tuple examination to develop the set of place gateways by getting the grandchildren requires $\Omega(t)$. The determination of the behavior logic is a constant number of logic patterns; therefore, the time complexity is $\Omega(pt^3)$. Furthermore, to set the child pair of the gateway children takes $\Omega(t)$, so Alpha-T processing has time complexity $\Omega(pt^4)$.

Within the post-processing stage in Alpha, the discovery process requires a firing step to create a link between place set and task transition, requiring $\Omega(t^2)$ for single causality and $\Omega(t^3)$ for multi-causality places. All stages of the Alpha discovery process are repeated until the entire event log has been processed. Because Alpha discovery $\alpha_w = \{P_w, T_w, F_w\}$ the total time complexity is $\Omega\big(l\big(pt^4 + 4t^3 + 3t^2\big)\big)$.

Meanwhile, Alpha-T post-processing requires a filtering step to fulfill the SWF-net gateway behavior pattern and reduce the redundant links, which takes $\Omega(t^2)$. The final result of this stage is a simple dataset flat-tree that shows the direct links in a graph model between parent node links as a vertex and child node links as an edge. Therefore, completion of Alpha-T discovery takes $\Omega\left(lt^3 + t \log t + pt^4 + t^2\right)$.

However, in the general discovery process, the number of events that is produced by the number of data transactions has the highest value, while the numbers of task combinations and logic patterns are limited and generally constant. With O time complexity analysis on limit-to-limit approximation [31], O can be determined by the following equation:

$$Alpha(O) = \lim_{x\to\infty} \lim_{y\to 1\ldots} \lim_{z\to 4\ldots 6} \sum_{n=1}^{x} \sum_{t=1}^{y} \sum_{p=1}^{z} n \left(pt^4 + 4t^3 + 3t^2\right)_{ntp} \approx pt^4 + 4t^3 + 3t^2 \tag{27}$$

$$Alpha\text{-}T(O) = \lim_{x\to\infty} \lim_{y\to 1\ldots} \lim_{z\to 8} \sum_{n=1}^{x} \sum_{t=1}^{y} \sum_{p=1}^{z} \left(nt^3 + t \log t + pt^4 + 4t^2\right)_{ntp} \approx t^3 \tag{28}$$

For the highest number of events in Alpha, as defined by Lemma 27, it has been shown that the increasing in the number of events greatly affects the preprocessing, processing and post-processing stages, that cause significantly increasing the computing load. Similarly, complex concurrency and the number of tasks also have a major impact on the time complexity because there are many steps that are directly affected by the number of events.

Lemma 28 of Alpha-T shows that Alpha-T is able to minimize the influence of the number of events by localizing only in the preprocessing stage because it is analyzed on global causality. Meanwhile, the complexity of the task causality and the number of tasks

will affect without adding significant computational load on the processing stage because there are not many steps that are influenced directly by the number of events.

Other than that, by using Alpha-T, the post-processing to firing and visualize will do on the simple graph relation without complex mapping process between task and place of gateway. Furthermore, to show the quality of discovery from Alpha-T, Table 5 shows the comparison of many discovery results on invisible and concurrent patterns that are produced from *.mxml event logs. The testing uses many algorithms that are available in ProM 6.7., including: Alpha, Alpha++, Heuristic, Genetic, *evolutionary tree miner* (ETM) [32], and also *inductive miner* (IM) [33]. Two basic parameters that are examined are *Completeness* (Cp) [9] and *Correctness* (Cr) [10].

TABLE 5. Comparison of discovery result between Alpha-T and others

Concurrent	Alpha		Alpha++		Heuristic		Genetic		ETM		IM		Alpha-T	
Pattern	Cp	Cr	Cp	Cr	Cp	Cr	Cp	Cr	Cp	Cr	Cp	Cr	Cp	Cr
Multi-Switch						x		x						
Multi-Skip		x				x		x		x				
Multi-Reply				x		x		x		x		x		
Free-choice										x		x		
Multi-Loop	x	x	x	x		x		x		x		x		
XOR/NFC		x		x		x		x		x				
Multi-Choice						x		x						

Meanwhile, by the comparison testing it is shown that Alpha-T has the best discovery result on structure and behavior model as proven in Table 6. Many algorithms that are not based on the direct mapping in pattern matching have the worst result. Whereas on testing with a large task and event log variants, such as referring to *Coselog* project data testing [34], Alpha-T has over-fitting if compared with IM, which gives the better simplification and generalization result because based on a heuristic. However, on the other side, Alpha-T has better precision because it has more logic pattern which can be examined than IM.

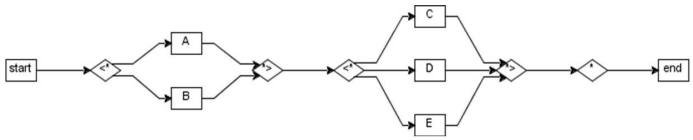
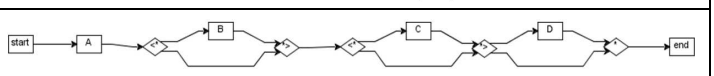
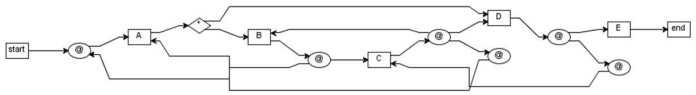
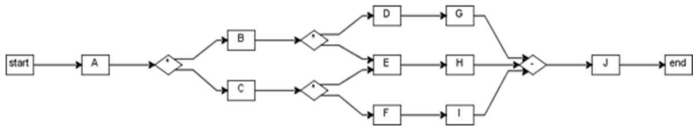
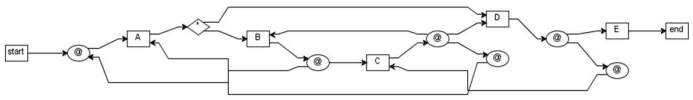
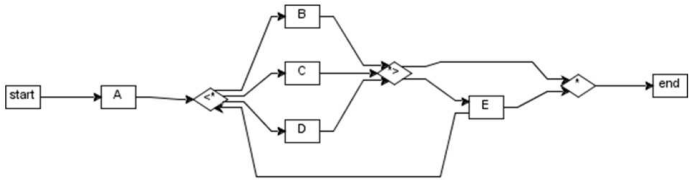
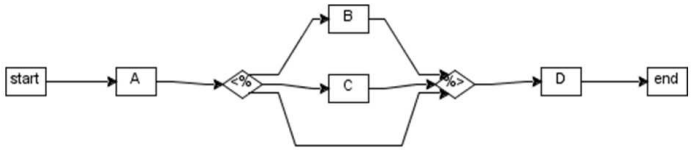
Furthermore, Alpha-T has the better result than the other, because it executes the pattern on global event log, and also do induction to pattern analysis on the gateway places. This mechanism has not been applying by other algorithms. The induction has been reducing step to do the concurrent analysis of all transition. Because it does not analyze independently many causality relations, it will keep off over-fitting of *place* on the model from discovery result.

6. **Conclusion.** From the test results it can be concluded that the novel Alpha-T algorithm using GTPR based on the ALT structure is an approach that is able to accurately perform discovery from a structural and behavioral perspective while conforming to the SWF-net and BPMN model. Other than that, the step preprocessing in addition to the depth weighting process has benefit to generalize logic analysis that has not been applying to Alpha and other advanced algorithms.

Meanwhile, by direct pattern executing on global causality, this causes Alpha-T has more benefit in pattern matching precision and also to be advanced by adding new pattern analysis on the next study. Then by result comparison, Alpha-T is better than many algorithms on correctness and completeness.

Furthermore, Alpha-T has successfully conducted discovery on a business process under certainty with complex concurrent transitions and places. It reduced the number of steps

TABLE 6. Discovery result from *Alpha-T* simulated by *JGraph*

Concurrent Pattern	Event Log	Alpha-T Discovery
Multi Switch	AC, AD, BC, BD, AE, BE	
Multi-Skip	AC, ABC, ABD, ABCD	
Multi-Reply	ABABCDE ABCDCDE ABCBCDE ABCBCADE	
Multi Free-choice	ABDGJ, ABEHJ, ACEHJ, ACFIJ.	
Multi-Loop	ABACCDE, ABBACDCE.	
Multi-XOR/NFC	ABE, ACE, ADE, ABEC, ABED	
Multi-Choice (AND-OR)	ABCD, ACBD, ABD, ACD, AD	

by localizing computation complexity in the pre-processing stage, and also minimizing complexity firing mapping of task and place gateway in post-processing. In fact, Alpha-T has time complexity $O(t^3)$, so that it performs better than Alpha and other deterministic, which has time complexity $O(t^4)$.

Therefore, it is proven, Alpha-T has not only high-quality discovery for producing workflow model suitable for SWF-net structure and BPMN behavior, but also high performance.

For further development of the Alpha-T algorithm, we will develop an advanced logic to solve complex concurrent logic on business processes under uncertainty [34,35], which until now cannot be solved by most discovery algorithms [36]. To improve the discovery quality, we will analyze the heuristic influence based on frequency and time causality [37], and also the task weight causality used to adjust discovery view level, considered with the four discovery level perspectives which are fitness, simplification, precision, and generalization [32]. Also, to support implementation, we plan to develop several feature and tools that cover various standard business process models, such as YAWL and BPMN, for conformance evaluation and business process auditing [38] as ProM plugins library [15].

Acknowledgment. This work is partially supported by *Indonesia Endowment Fund for Education* (LPDP-KeMenKeu). The authors also gratefully acknowledge the helpful comments and suggestions of the supervisor, reviewers and editors, which have improved the presentation.

REFERENCES

[1] M. de Leoni, W. M. P. van der Aalst and M. Dees, A general process mining framework for correlating, predicting and clustering dynamic behavior based on event log, *Information Systems*, vol.56, pp.235-257, 2016.

[2] A. Rozinat and W. M. P. van der Aalst, Conformance checking of processes based on monitoring real behavior, *Information Systems*, vol.33, no.1, pp.64-95, 2008.

[3] N. Mundbrod and M. Reichert, Process-aware task management support for knowledge-intensive business processes: Findings, challenges, requirements, *IEEE the 18th International Enterprise Distributed Object Computing*, Ulm, Germany, pp.116-125, 2014.

[4] C. K. H. Lee, K. L. Choy, G. T. S. Ho and C. H. Y. Lam, A slippery genetic algorithm-based process mining system for achieving better quality assurance in the garment industry, *Expert Syst. Appl.*, vol.46, pp.236-248, 2016.

[5] M. Werner and N. Gehrke, Multilevel process mining for financial audits, *IEEE Trans. Serv. Comput.*, vol.8, no.6, pp.820-832, 2015.

[6] Á. Rebuge and D. R. Ferreira, Business process analysis in healthcare environments: A methodology based on process mining, *Information Systems*, vol.37, pp.99-116, 2012.

[7] M. Jans, M. Alles and M. Vasarhelyi, The case for process mining in auditing: Sources of value added and areas of application, *International Journal of Accounting Information Systems*, vol.14, no.1, pp.1-20, 2013.

[8] G. Sedrakyan, J. D. Weerdt and M. Snoeck, Process-mining enabled feedback: 'Tell me what I did wrong' vs. 'tell me how to do it right', *Comput. Human Behav.*, vol.57, pp.352-376, 2016.

[9] A. Rozinat, M. Veloso and W. M. P. van der Aalst, Evaluating the quality of discovered process models, *The 2nd International Workshop on the Induction of Process Models*, Antwerp, Belgium, pp.1-8, 2008.

[10] S. Suriadi, R. Andrews, A. H. M. ter Hofstede and M. T. Wynn, Event log imperfection patterns for process mining towards a systematic approach to cleaning event log, *Information Systems*, pp.1-20, 2016.

[11] C. Favre, D. Fahland and H. Völzer, The relationship between workflow graphs and free-choice workflow nets, *Information Systems*, vol.47, pp.197-219, 2015.

[12] E. Börger, Approaches to modeling business processes: A critical analysis of BPMN, workflow patterns and YAWL, *Softw. Syst. Model.*, pp.305-318, 2012.

[13] J. Li, D. Liu and B. Yang, Process mining: Extending α-algorithm to mine duplicate tasks in process logs, *Advances in Web and Network Technologies, and Information Management*, vol.4537, 2007.

[14] A. K. A. de Medeiros, B. F. van Dongen and W. M. P. van der Aalst, *Process Mining: Extending the α-Algorithm to Mine Short Loops*, Technische Universiteit Eindhoven, 2004.

[15] L. Wen, J. Wang, W. M. P. van der Aalst, B. Huang and J. Sun, Mining process models with prime invisible tasks, *Data & Knowledge Engineering*, vol.69, no.10, pp.999-1021, 2010.

[16] W. G. Christian and W. M. P. van der Aalst, Fuzzy mining – Adaptive process simplification based on multi-perspective metrics, *Proc. of the 5th International Conference on Business Process Management*, Brisbane, Australia, pp.328-343, 2007.

[17] M. S. Saravanan and R. J. R. Sree, Evaluation of process models using heuristic miner and disjunctive workflow schema algorithm for dyeing process, *International Journal of Information Technology Convergence and Services*, vol.1, no.3, 2011.

[18] W. M. P. van der Aalst, A. K. A. de Medeiros and A. J. M. M. Weijters, Genetic process mining, *Applications and Theory of Petri Nets, Lecture Notes in Computer Science*, vol.3536, 2005.

[19] S. Goedertier, D. Martens, J. Vanthienen and B. Baesens, Robust process discovery with artificial negative events, *The Journal of Machine Learning Research*, vol.10, pp.1305-1340, 2009.

[20] Z. He, F. Gu, C. Zhao, X. Liu, J. Wu and J. Wang, Conditional discriminative pattern mining: Concepts and algorithms, *Information Sciences*, vol.375, pp.1-15, 2017.

[21] F. Feng, J. Cho, W. Pedrycz, H. Fujita and T. Herawan, Soft set based association rule mining, *Knowledge-Based Syst.*, vol.111, pp.268-282, 2016.

[22] R. Sarno, R. D. Dewandono, T. Ahmad and M. F. Naufal, Hybrid association rule learning and process mining for fraud detection, *IAENG International Journal of Computer Science*, 2015.
[23] J. Wang, R. K. Wong, J. Ding, Q. Guo and L. Wen, Efficient selection of process mining algorithms, *IEEE Trans. Services Computing*, pp.484-496, 2012.
[24] W. M. P van der Aalst, Business process configuration in the cloud: How to support and analyze multi-tenant processes?, *The 9th IEEE European Conference on Web Services*, pp.3-10, 2011.
[25] W. M. P. van der Aalst, *Process Mining Discovery, Conformance and Enhancement of Business Processes*, Springer-Verlag Berlin Heidelberg, 2011.
[26] H. Hermawan and R. Sarno, Developing distributed system with service resource oriented architecture, *TELKOMNIKA (Telecommunication, Computing, Electronics and Control)*, vol.10, no.2, pp.389-399, 2012.
[27] R. Sarno, W. A. Wibowo, D. Sunaryono and A. Munif, Developing workflow patterns based on functional subnets and control-flow patterns, *International Conference on Science in Information Technology (ICSITech)*, 2015.
[28] Q. Guo, L. Wen, J. Wang, Z. Yan and P. S. Yu, Mining invisible tasks in non-free-choice constructs, *Business Process Management, Lecture Notes in Computer Science*, vol.9253, 2015.
[29] R. Sarno, P. Sari, H. Ginardi, D. Sunaryono and I. Mukhlash, Decision mining for multi choice workflow patterns, *Proc. of International Conference on Computer, Control, Informatics and Its Applications: "Recent Challenges in Computer, Control and Informatics"*, Jakarta, 2013.
[30] B. Mikolajczak and J. L. Chen, Workflow mining alpha algorithm – A complexity study, intelligent information processing and web mining, *Advances in Soft Computing*, vol.31, 2005.
[31] D. H. Greenee and D. E. Knuth, *Mathematics for the Analysis of Algorithms*, Birkhauser, Boston, 2008.
[32] J. C. A. M. Buijs, B. F. van Dongen and W. M. P. van der Aalst, On the role of fitness, precision, generalization and simplicity in process discovery, *Lecture Notes in Computer Science*, vol.7565, 2012.
[33] S. J. J. Leemans, D. Fahland and W. M. P. van der Aalst, Discovering block-structured process models from event logs containing infrequent behaviour, *Business Process Management Workshops*, vol.171, 2013.
[34] W. M. P. van der Aalst, Configurable services in the cloud: Supporting variability while enabling cross-organizational process mining, *CoopIS 2010, Lecture Notes in Computer Science*, vol.6426, pp.8-25, 2010.
[35] W. M. P. van der Aalst, Process mining in the large: A tutorial, *eBiSS 2013: Business Intelligence*, pp.33-76, 2014.
[36] E. Rojas, J. Munoz-Gamaa, M. Sepúlveda and D. Capurro, Methodological review process mining in healthcare: A literature review, *Journal of Biomedical Informatics*, vol.61, pp.224-236, 2016.
[37] R. Sarno, W. A. Wibowo, K. Kartini and F. Haryadita, Determining model using non-linear heuristics miner and control-flow pattern, *TELKOMNIKA (Telecommunication, Computing, Electronics and Control)*, vol.14, no.1, pp.349-359, 2016.
[38] M. Schultz, Audit-focused mining – New views on integrating process mining and internal control, *ISACA Journal*, vol.3, no.1, pp.1-6, 2014.