

# A Survey of Deterministic Graph-based Algorithms for Discovering Business Processes

Riyanarto Sarno<sup>1,\*</sup>, Kelly R. Sungkono<sup>2</sup>

Department of Informatics, Institut Teknologi Sepuluh Nopember, Jl. Raya ITS Surabaya 60111, Indonesia

<sup>1</sup> riyanarto@if.its.ac.id; <sup>2</sup> kelsungkono@gmail.com

\* corresponding author

## ARTICLE INFO

## ABSTRACT (10PT)

### Article history

Received

Revised

Accepted

### Keywords

Survey

Graph database

Process discovery

Quality

Algorithms of process discovery help analysts to understand business processes and problems in a system by creating a process model based on a log of the system. There are existing algorithms of process discovery, namely Graph-based. Of all algorithms, there are algorithms that process graph-database to depict a process model. Those algorithms claimed that those have less time complexity because of the graph-database ability to store relationships. This research analyzes Graph-based algorithms by measuring the time complexity and performance metrics and comparing them with a widely used algorithm, i.e. Alpha Miner and its expansions. Other than that, this research also gives outline explanations about Graph-based algorithms and their focus issues. Based on the evaluations, the graph-based algorithms have high performance and those have lower time complexity than Alpha Miner and its expansions.

## 1. Introduction

Analysts use their business process model to find out the latest business process and become a reference for future development. Unfortunately, modification of a business process model becomes forgotten in a system that has fast-growing requirements. Thus, changes in a process model are not in line with changes in the system. Because of the change incompatibility, algorithms for detecting a business process model automatically are needed. A set of those algorithms is called process discovery. Process discovery is implemented in many sectors, such as business [1]–[6], fraud [7]–[9], advertising [10], and medical [11].

Out of all algorithms, there is a Graph-based algorithm that depicts a process model by processing a graph-database. This algorithm chose a graph-database to be processed because a graph-database can store not only activities but also their relationships. The ability for storing relationships is claimed to produce low time complexity. The Graph-based algorithm has improved, so there are a group of graph-based algorithms containing a Graph-based algorithm of parallel process, a Graph-based algorithm of processes containing non-free choice, a Graph-based algorithm of processes containing invisible task, and a Graph-based algorithm of processes containing non-free choice and invisible task.

This research analyzes all of Graph-based algorithms. There are several questions that guide this research to analyze.

**Question (1)** : What are issues that are handled by Graph-based algorithms?

**Question (2)** : How Graph-based algorithms handle those issues?

**Question (3)** : How is the quality of Graph-based algorithms in the context of time complexity and performance of its results?

To answer the last question, this research uses fitness and precision measurements mentioned in Sungkono *et al.* [4], Buijs *et al.* [12]. Those measurements determine the performance of the

obtained process model. Fitness is a measurement of completeness of a model based on processes in a log. Precision is a measurement of conformity of model behavior with a log. This research compares graph-database algorithms with widely used algorithm, Alpha miner [13] and its expansions, i.e. Alpha++ [14] that concerns with non-free choice constructs, Alpha# [15] that detects invisible tasks for describing some special conditions, and Alpha\$ [16] that combines Alpha++ and Alpha# for detecting non-free choice in invisible tasks.

## 2. Related Works

Responding **Question (1)** in the introduction, this section will describe about three issues of process discovery that are handled by Graph-based algorithms. Those issues are adopted from problems that are handled by existing algorithms, Alpha miner [13] and its expansions. Those issues are also described in the research of workflow pattern [17]. Section 2.1 explains those issues and simple overviews of each issue. Other than that, in this section especially Section 2.2, the existing algorithms, Alpha miner [13] and its expansions, are described. The explanation of those algorithms gives the understanding of the general steps to form a process model. Those algorithms will be used as the comparison of Graph-based algorithms in time complexity and performance measurements. The results of the comparison are the answer of **Question (3)**.

### 2.1 Issues in Discovering Process Models

There are three issues that are handled by Graph-based algorithms. Those issues are parallel relationships, non-free choice, and invisible tasks. Parallel relationships contain AND, XOR, and OR relationships. Those relationships accommodate the behavior of activities carried out by one of them or run in unison. Afterward, non-free choice accommodates the behavior of selected activity whose execution depends on other selected activities. Lastly, invisible tasks accommodate a depiction of specific cases that cannot be described only by activities in the log.

#### 2.1.1 Parallel Relationships

In a process model, an activity has a relationship with other activities [18]. There is a condition when two activities are related to each other for all processes or an activity have relationships with more than one activity. A sequential relationship is a condition when an activity always followed by the same activity for all processes. On the contrary, a parallel relationship is a condition when an activity has different related activities.

Fig. 1 explains both of sequential and parallel relationships. This research uses YAWL notation [19], [20] to depict those relationships. In the first event log, activity Act\_1 always followed by Act\_2 for those three processes. This condition is called a sequential relationship, that is depicted by a place (a circle) and connectors between the place and activities. Parallel relationships [21] are divided into three categories. First, activities are included in XOR relationship if only one of them is selected in a process. As seen on Fig. 1, there is only one of activities {Act\_2, Act\_3, Act\_4} that is executed in every process. The triangle signs in both of Act\_1 and Act\_5 describe XOR relationships by using YAWL. Secondly, AND relationships occur if all activities are executed in every process with different order of executions. The example log is shown in Fig. 1, wherein {Act\_2, Act\_3, Act\_4} are executed with different sequences. Lastly, OR relationships depict conditions that cannot be handled by AND relationships and XOR relationships. The example is shown in Fig. 1. All processes only execute two out of three activities, i.e. {Act\_2, Act\_3, Act\_4}. This condition does not meet the rule of XOR relationship and AND relationship. Because of that, this condition is depicted by OR relationship. The OR relationship is denoted by diamond signs in YAWL notation.

Process discovery determines AND or OR relationships in two ways. The first way is considering the sequence of activities, and the second way is considering the time execution of activities. Mostly process discovery algorithm, such as Graph-based algorithm, chooses the first way. Meanwhile, there are researches that determine those relationships by using the second way [22], [23].


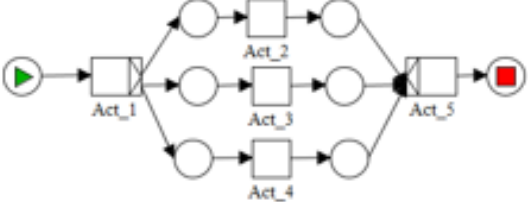
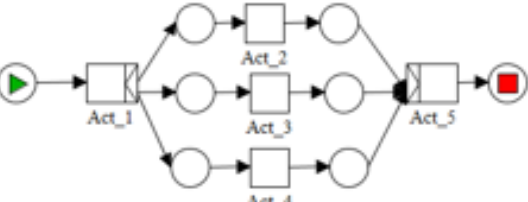
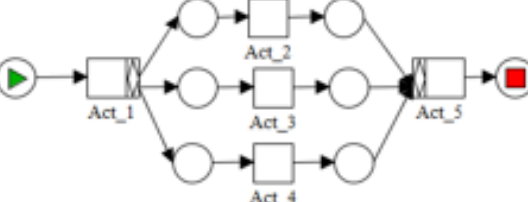
<b>1. Sequential Relationships</b>	
<i>Event Log</i>	{Act_1, Act_2}, {Act_1, Act2}, {Act_1,Act_2}
<i>Process Model</i>	
<b>2. Parallel relationships</b>	
<b>2.1. XOR Relationships</b>	
<i>Event Log</i>	{Act_1, Act_2, Act_5}, {Act_1, Act_3, Act_5}, {Act_1, Act_3, Act_5}
<i>Process Model</i>	
<b>2.2. AND Relationships</b>	
<i>Event Log</i>	{Act_1, Act_2, Act_3, Act_4, Act_5}, {Act_1, Act_3, Act_4, Act_2, Act_5}, {Act_1, Act_4, Act_2, Act_3, Act_5},
<i>Process Model</i>	
<b>2.3. OR Relationships</b>	
<i>Event Log</i>	{Act_1, Act_2, Act_3, Act_5}, {Act_1, Act_3, Act_4, Act_5}, {Act_1, Act_4, Act_2, Act_5},
<i>Process Model</i>	

Fig. 1. A Process Model by Alpha Miner

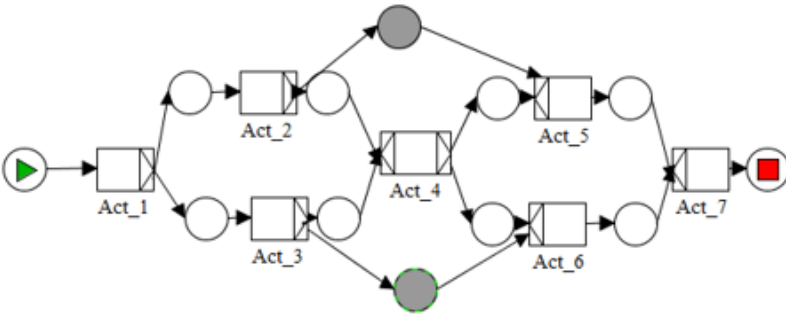
<b>Non-Free Choice Relationships</b>	
<i>Event Log</i>	{Act_1, Act_2, Act_4, Act_5, Act_7}, {Act_1, Act_3, Act_4, Act_6, Act_7},
<i>Process Model</i>	

Fig. 2. A Process Model including non-free choice

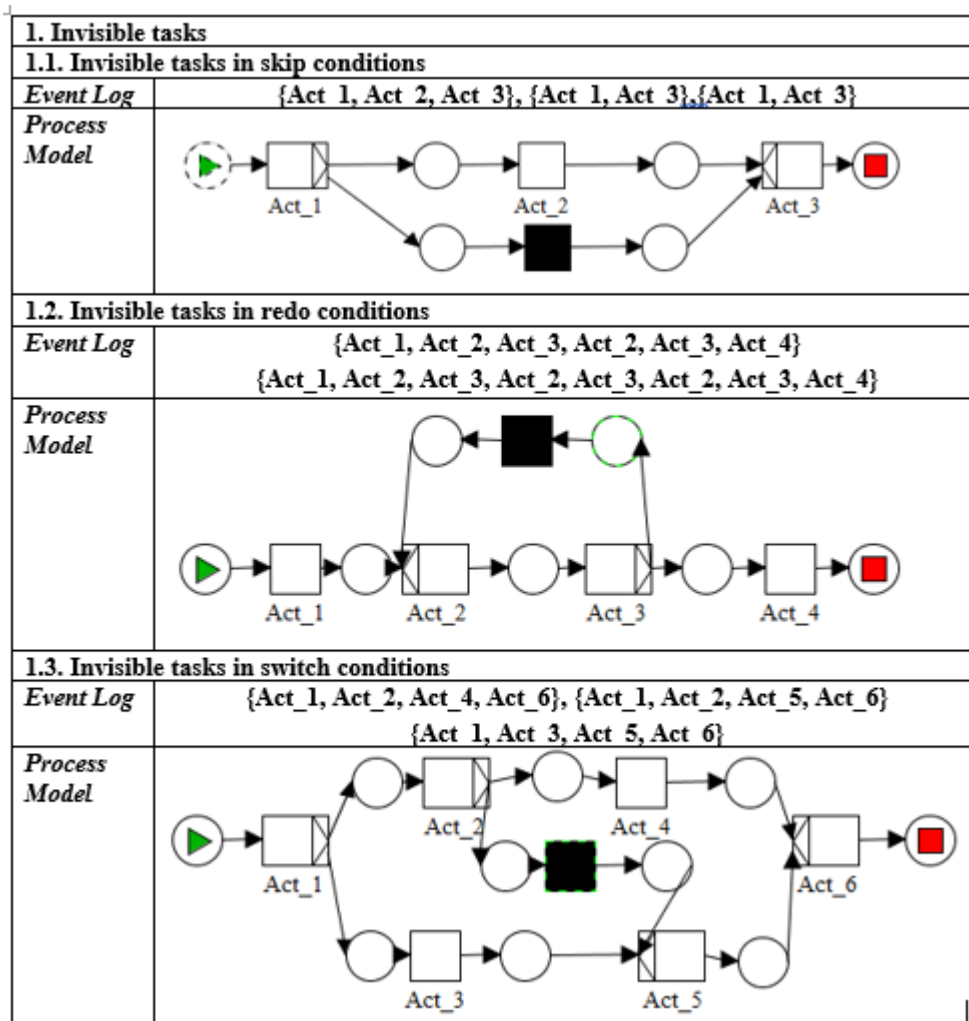


Fig. 3. A Process Model including invisible tasks

### 2.1.2 Non-Free Choice

With the development of processes, both parallel and sequential relationships cannot handle all conditions. There are several conditions requiring special depiction. One of the conditions is the selection of an activity in a parallel relationship is influenced by the selection of activities in the previous parallel relationship. This condition triggers a non-free choice.

A non-free choice is an additional implicit dependency in a process model for describing the election dependence between an activity and its previous activity [1], [14]. The simplified example is shown in Fig. 2. Based on the event log, Act<sub>5</sub> always executed when Act<sub>2</sub> is chosen, vice versa for Act<sub>6</sub>. The real example is the part of choosing a transportation online. In the application, there are two options of transportation online, such as a motorcycle and a car. Even if there are two choices, when a customer bought large kinds of stuff, he chooses a car rather than a motorcycle. Conversely, if a customer bought small or no kind of stuff, most likely he chooses a motorcycle. The relationship between the selection of transportation online and the selection of the kinds of stuff is depicted by non-free choice.

There are several ways to depict a non-free choice in a process model. Both of YAWL and Petri Net model uses a place and arcs to connect the place and the activities. The additional place is depicted by a grey circle in Fig. 2. In the graph model, the non-free choice is depicted by an arc with the name is "NONFREE CHOICE".

Process discovery algorithms determine non-free choice by observing the behavior of activities in the process model. If an activity of selection is executed when an activity of previous selection is chosen, process discovery algorithm detects those relationships as NONFREE CHOICE.

### 2.1.3 Invisible tasks

Besides non-free choice, there are several conditions that cannot be handled by both parallel and sequential relationships. Those conditions are skip condition, redo condition, and switch condition. Those conditions need invisible tasks [15] in the depiction of the model.

The first condition is a skipped condition. Skip condition happens if several processes skipped one or more activities. The skip condition is detected by comparing the processes with other processes. If a process executes two activities, such as *Act\_1* and *Act\_3*, and another process executes other activities between *Act\_1* and *Act\_3*, this is called skip condition. Fig. 3 explains a skip condition in the event log and the process model. As shown in Fig. 3, there is a skip condition when activity *Act\_1* can directly be followed by *Act\_3*. An invisible task is added to depict this condition.

The second condition is a redo condition. Redo condition happens if several activities in a process are executed more than one time. The redo condition is detected by calculating the execution frequency of activities in a process. If a process executes two activities, such as *Act\_2* and *Act\_3*, and those activities are stored more than one time in a process, this condition is called redo condition. Fig. 3 explains a redo condition in the event log and the process model. As shown in Fig. 3, there is a redo condition when activity *Act\_2* and *Act\_3* has more than one execution time in a process. An invisible task is added to depict this condition.

The second condition is a redo condition. Redo condition happens if several activities in a process are executed more than one time. The redo condition is detected by calculating the execution frequency of activities in a process. If a process executes two activities, such as *Act\_2* and *Act\_3*, and those activities are stored more than one time in a process, this condition is called redo condition. Fig. 3 explains a redo condition in the event log and the process model. As shown in Fig. 3, there is a redo condition when activity *Act\_2* and *Act\_3* has more than one execution time in a process. An invisible task is added to depict this condition.

## 2.2 Algorithms

### 2.2.1 Alpha Miner

Alpha Miner algorithm is a deterministic process discovery algorithm that develops causality of activities based on the event log [24]. Alpha Miner discovers a process model that has sequence relationships or parallel relationships, such as XOR relationship and AND relationship. Alpha algorithm utilizes workflow-nets in the form of Petri Nets [25], [26].

Alpha Miner creates tuples for constructing a process model. There are some rules of determining a tuple (*ActGroup1*, *ActGroup2*). There can be one or more activities in *ActGroup1* or *ActGroup2*. Those rules are:

1. All of activities in *ActGroup1* and *ActGroup2* are stored in the event log.
2. All of activities in *ActGroup1* have casual dependencies with all activities in *ActGroup2*. A causal dependency denoted by  $\rightarrow$  occurs if an activity is followed by another activity, but another activity is not followed by the activity. For example, based on a process *KR*, *K* has a causal dependency with *R* because activity *K* is followed by activity *R* and activity *R* is not followed by activity *K*.
3. All of activities in *ActGroup1* do not have casual dependencies each other, likewise all activities in *ActGroup2*.
4. If there are two tuples that have the same activity in *ActGroup1* or the same activity in *ActGroup2*, those tuples can be combined into a tuple. For example, if there are two tuples, (*K,R*) and (*K,S*), it can be combined into a tuple, (*K*, {*R,S*}).

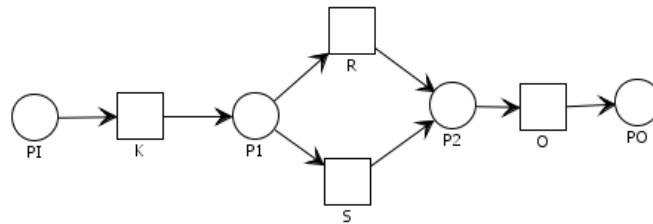


Fig. 4. A Process Model by Alpha Miner

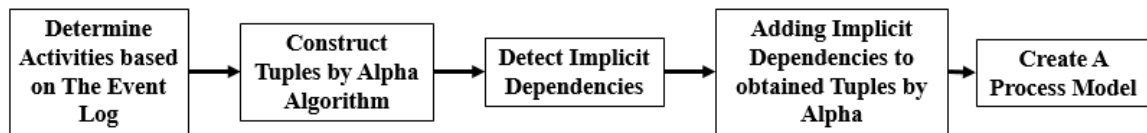


Fig. 5. Steps of Alpha++ Algorithm

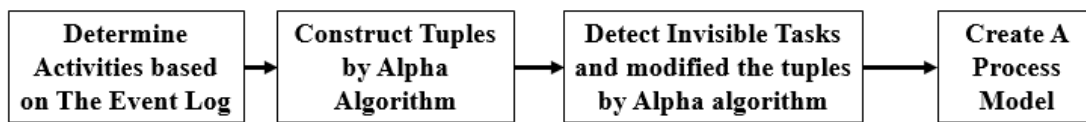


Fig. 6. Steps of Alpha# Algorithm

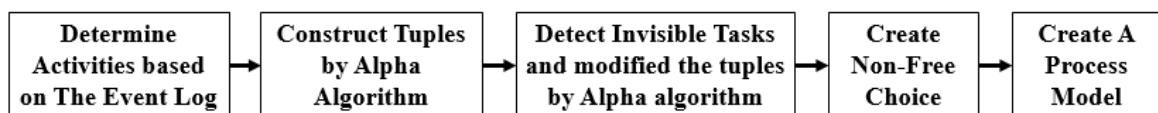


Fig. 7. Steps of Alpha\$ Algorithm

Those obtained tuples are arranged into a process model. To arrange into a Petri-Net process model, Alpha Miner defines those tuples to create places, activities, and arcs. There is an initial place, an ending place, and a place between  $i$  and  $ActGroup2$  for each tuple and arcs connect activities and places. For example, if there are two tuples,  $(K, \{R, S\})$ , and  $(\{R, S\}, O)$ , there are four places (an initial place, an ending place, and two places for each tuple) and four arcs that are used to build the process model. The process model based on those two tuples is shown in Fig. 4. This process model has XOR relationship between activity R and activity S.

### 2.2.2 Alpha++

Alpha++ [14] improves Alpha Miner to depicting non-free choice in a process model forming Petri Net model. The non-free choice is depicted by adding implicit dependencies. Alpha++ forms implicit dependencies by adding extra-arcs and extra-places to connect the activities. The steps of Alpha++ algorithm is showed in Fig. 5. The first step and the second step are the parts of Alpha algorithm. Alpha++ adds the third and the four steps for creating non-free choice constructs.

There are three rules to depict the implicit dependency as the form of a non-free choice. All implicit dependencies are added into obtained tuples of Alpha Miner. The rule of depicting the dependencies are:

- A first implicit dependency of  $task a$  and  $task b$  occurs if  $task a$  is a task of a parallel AND relationship that has an explicit dependency with another activity and both of another activity and  $task b$  are tasks of an XOR relation.
- A second implicit dependency of  $task a$  and  $task b$  occurs if  $task a$  is a former or latter activity of AND relation and it has an indirect relationship with  $task b$ .
- A third implicit dependency of  $task a$  and  $task b$  occurs if  $task a$  has an indirect relationship with  $task b$ , both of  $task a$  and  $task b$  are activities of XOR relation and  $task a$  has different XOR relation with  $task b$ .

### 2.2.3 Alpha#

Alpha# algorithm [15], [4] aims to detect invisible prime tasks from event logs. This algorithm is derived from Alpha algorithm. Alpha# divides the prime tasks into three types, SKIP, REDO and SWITCH. There are several steps of Alpha# algorithm for obtaining invisible prime tasks.

The steps of Alpha# algorithm are similar with Alpha++ algorithm. The different is Alpha# adds invisible tasks, meanwhile Alpha++ adds implicit dependencies. The steps of Alpha# can be seen in Fig. 6. There are several steps to detect invisible tasks by Alpha# algorithm.

First, Alpha# detects all mendacious dependencies between tasks and identifies redundant mendacious dependencies. Based on the discovered mendacious dependencies, Alpha# algorithm constructs invisible prime tasks. Besides, Alpha# algorithm also ensures that newly discovered dependencies are not composed by the others. Then, Alpha# algorithm combines new casual and parallel relations between invisible tasks with ones between invisible tasks and visible tasks. Finally, the set of visible tasks and invisible tasks establishes a process model.

### 2.2.4 Alpha\$

Alpha\$ [16] algorithm is a combination of alpha++ and alpha#. Alpha\$ algorithm aims to construct a process model including invisible tasks and non-free choice. Alpha\$ algorithm uses Petri Net for depicting the process model.

There are several steps to construct a process model using Alpha\$ algorithm. The steps are shown in Fig. 7. Alpha\$ improves the rules of mendacious dependencies in Alpha# algorithm by adding a rule to generate invisible tasks involved in a parallel construct. The improvement rules can solve a condition that cannot be handled by Alpha#.

**Table 1** Graph-based Algorithm for Parallel Process

No	Steps of algorithm
	<b>The input</b> is an event log that contains names of activities, number of case or process, and time execution of activities
1	Storing event log in the format of graph database following rules in Table 4.
2	For a graph sequence that fulfills a format {node act1 – relation - node act2}: if the number of the next node of node act1 is more than 1 and the number of previous node, as well as next node, of node act2 is 1: Creating XORSPLIT relation that connects act1 and act2
3	For a graph sequence that fulfills a format {node act1 – relation - node act2}: if the number of the next node of node act1 is 1 and the number of previous node of node act2 is more than 1: Creating XORSPLIT relation that connects act1 and act2
3	for a graph sequence that fulfills a format { node act1 – relation - node act2 – relation - node act3}: if the number of the next node of node act1 is more than 1, the number of the next node of node act3 is same with that of node act1, and act1 is not the next node of both of node act2 and node act3: Creating ANDSPLIT relation that connects act1 and act2 and ANDSPLIT relation that connects act1 and act3
4	for a graph sequence that fulfills a format { node act2 – relation - node act3 – relation - node act1}: if the number of the previous node of node act1 is more than 1, the number of the next node of node act3 is same with the number of previous node of node act1, and node act1 has not ANDSPLIT relation: Creating ANDJOIN relation that connects act2 and act1 and ANDJOIN relation that connects act3 and act1
5	for a graph sequence that fulfills a format { node act1 – relation - node act2 – relation - node act3}: if the number of the next node of node act1 is more than 1, the number of the next node of node act3 is more than 1 and less than that of node act1, and act1 is not the next node of both of node act2 and node act3: Creating ORSPLIT relation that connects act1 and act2 and ORSPLIT relation that connects act1 and act3
6	for a graph sequence that fulfills a format { node act2 – relation - node act3 – relation - node act1}: if the number of the previous node of node act1 is more than 1, the number of the next node of node act3 is more than 1 and less than the number of previous node of node act1, and node act1 has not ORSPLIT relation: Creating ORJOIN relation that connects act2 and act1 and ORJOIN relation that connects act3 and act1
	<b>The output</b> is a graph process model containing parallel relationships

### 3. Deterministic Graph-based Algorithms

In process discovery, deterministic algorithms depict all relationships of activities in an event log into a process model [27]. Prior deterministic algorithm of process discovery is Alpha [24]. Alpha algorithm has improved, such as Alpha++ [14], Alpha# [15], and Alpha\$ [16]. Graph-based algorithms are categorized as deterministic algorithms. It is because the formulation of those algorithms refers to Alpha Miner and its expansions. There are four deterministic Graph-based algorithms, i.e. Graph-based algorithm for parallel process, the algorithm for processes containing non-free choice, the algorithm for processes containing invisible tasks, and for processes containing invisible task and non-free choice. The description of Graph-based algorithms is used to respond **Question (2)**.

#### 3.1 Graph-based Algorithm for Parallel Process

Graph-based Algorithm for Parallel Process [21] constructs a graph process model that contains parallel relationships by implementing several rules in a graph-database. There are three parallel relationships that are handled by Graph-based algorithm for parallel process, such as XOR, OR, and AND. Table 1 describes step-by-step of Graph-based algorithm for parallel process.

Based on Table 1, there are several steps. The first step is storing an event log in the format of graph-database. There is a storing process because the research cannot keep a log as a graph-database automatically. Then, the research discovers XOR relationship. To depict a parallel relationship, a process model needs *Split* sign and *Join* sign. The *split* sign is used to denote the beginning of a parallel relationship, and the *join* sign is used to denote the end of a parallel relationship. XOR relationship occurs if several activities have only one outgoing activity. Thereafter, the research discovers AND relationship. The activities are included in AND relationship if the number of outgoing arcs of the activity is same with the number of outgoing arcs of its previous activity. Lastly, OR relationship is discovered following the condition, i.e. activities having the number of outgoing arcs less than the total number of outgoing arcs of previous activity and more than 1 arc.

#### 3.1 Graph-based Algorithm for Processes Containing Non-Free Choice

Graph-based algorithm for processes containing non-free choice [28] is an expansion algorithm of Graph-based algorithm for parallel processes. This algorithm adds the rule to obtain non-free choice in the Graph-based algorithm for parallel processes.

Table 2 shows the pseudocode of Graph-based algorithm for processes containing non-free choice. This algorithm creates an implicit dependency between two activities if those activity are in same process and the beginning activity of the implicit dependency is executed before the end activity. This statement can be seen on sixth step. The final result of the algorithm is a graph process model.

#### 3.2 Graph-based Algorithm for Processes Containing Invisible Task

Graph-based algorithm for processes containing invisible task [29] is an expansion algorithm of Graph-based algorithm for parallel processes. This algorithm adds the rule to obtain invisible task in the Graph-based algorithm for parallel processes. Table 3 shows the pseudocode of Graph-based algorithm for processes containing invisible tasks. This algorithm has specific steps. Those steps are executed after the steps of Graph-based algorithm for parallel processes are executed. This algorithm will add invisible task between two activities if the beginning activity has more than one outgoing relationship and the name of the relationships are different. The obtained process model has formed a graph process model.

**Table 2** Graph-based Algorithm for Processes Containing Non-Free Choice

No	Steps of algorithm
	The <b>input</b> is an event log that contains names of activities, number of case or process, and time execution of activities
1	<i>Converting event log following rules in Table 4.</i>
2	<i>Creating a graph process model following rules in Table 1</i>
3	For a sequence that fulfills {node act1, relation XORJoin, node actafter}:



4	For a sequence that fulfills { node actbefore, relation XOR Split, node act2}:
5	For 2 nodes, initialized by actfirst and actsecond, that are obtained from a first list of Table 4:
6	If the name of node act1 is same with the name of node actfirst, the name of node act2 is same with the name of node actsecond, the number of case of node actfirst is same with node actsecond and the time execution of node actfirst is before the time execution of node actsecond:
7	Creating non-free choice relation that connects node act_1 and node act_2
<b>The output</b> is a graph process model containing non-free choice	

**Table 3** Graph-based Algorithm for Processes Containing Invisible Tasks

No	Steps of algorithm
The <b>input</b> is an event log that contains names of activities, number of case or process, and time execution of activities	
1	Converting event log following rules in Table 4.
2	Creating a graph process model following rules in Table 1
3	For a graph sequence that fulfills {node act_i, relation_a, node act_1}:
4	For a graph sequence that fulfills {node act_i, relation_b, node act_2}:
5	if relation_a has "SPLIT" fragment and relation_b has "JOIN" fragment:
6	Creating additional node naming Invisible_Task
7	Creating a graph sequence that fulfills {node act_i, relation_a, Invisible_Task}
8	Creating a graph sequence that fulfills {Invisible_Task, relation_b, node act_2}
7	Deleting relation_b that connects node act_i and node act_2
<b>The output</b> is a graph process model containing invisible tasks	

**Table 4** A pseudocode to Constructing A Graph-Database based on The Event Log

No	Steps of algorithm
The <b>input</b> is an event log that contains names of activities, number of case or process, and time execution of activities	
1	Creating two list of nodes, i.e. 1) a list containing all activities and their information in the event log, and 2) a list containing irredundant activities
2	For id=1 to maximal_id:
3	act1 is a node that has id as its index storing and act2 is a node that has id+1 as its index storing
4	For actbefore and actafter as nodes in the second list:
5	if the number of case of act1 is same with that of act2, the name of act1 is same with that of actbefore, and the name of activity of act2 is same with that of node actafter:
6	Creating SEQUENCE relation that connect node act1 and act2
<b>The output</b> is a graph database having SEQUENCE relation.	

where : id = the index storing of activities in the first list  
 maximal\_id = the maximal index storing in the first list

**Table 5.** Event Log for Evaluation

The name of process	The number of cases	Issues					Noise (Y/N)
		XOR (Number Max Branch)	OR (Number Max Branch)	AND (Number Max Branch)	Non-Free Choice	Invisible Task	
Port container handling process	200	v (3 branches)			v	v (skip condition)	N
Certificate Formation process	50		v (3 branches)				N
Cotton Production	60	v (2 branches)	v (2 branches)				N
Subprocess of Retail (Selling process and Recording Item Sales Journal)	50			v (4 branches)			N

### 3.3 Graph-based Algorithm of Processes Containing Non-Free Choice and Invisible Task

Graph-based algorithm of processes containing non-free choice and invisible task [30] are a combination of Graph-based algorithm of non-free choice and Graph-based algorithm of invisible task. This algorithm applies steps of invisible task in Graph-based algorithm of invisible task and then applies steps of non-free choice. The obtained process model is formed in graph process model.

### 3.4 Converting Event Log into A Graph Database

All of Graph-based algorithms [21], [30] contain steps for converting event logs. Table 4 shows a pseudocode for converting event logs. The input of Graph-based algorithms is the event log that includes case identifications, activities, and execution times. The format of the event log is CSV format. The output is a graph database.

## 4. Results and Analysis

This research evaluates Graph-based algorithm and Alpha algorithm using four event logs as the data set. The information of data set is shown in Table 5. The complex event log of all event logs is the event log of port container handling. It is because this event log has non-free choice and invisible tasks. All of processes in those event logs are complete and right processes, so there are no noises in there.

This research evaluates those algorithms by comparing each Graph-based algorithm that has same ability with Alpha algorithm or its expansions. Graph-based algorithm for parallel processes, for processes containing non-free choice, for processes containing invisible tasks, for processes containing invisible tasks and non-free choice, are compared with Alpha Miner, Alpha++, Alpha#, and Alpha\$. The performance metrics are calculated based on those event logs. Performance metrics that are used in this paper are fitness and precision. The performance metrics and the time complexity are shown in Table 6.

**Table 6.** Performance Metrics and Time Complexity

Methods	Event Logs	Performance Metrics		Time Complexity
		Fitness (0.0 – 1.0)	Precision (0.0 – 1.0)	
Alpha Miner	Port container handling process	0.4	0.42	O (n <sup>4</sup> )
	Certificate formation process	0.0	0.0	
	Cotton Production	0.6	1.0	
	Subprocess of Retail	1.0	1.0	
Alpha++	Port container handling process	0.4	0.63	O (n <sup>4</sup> )
	Certificate formation process	0.0	0.0	
	Cotton Production	0.6	1.0	
	Subprocess of Retail	1.0	1.0	
Alpha#	Port container handling process	1.0	0.28	O (n <sup>4</sup> )
	Certificate formation process	0.0	0.0	
	Cotton Production	0.6	1.0	
	Subprocess of Retail	1.0	1.0	
Alpha\$	Port container handling process	1.0	0.83	O (n <sup>4</sup> )
	Certificate formation process	0.0	0.0	
	Cotton Production	0.6	1.0	
	Subprocess of Retail	1.0	1.0	
Graph-based Parallel Process	Port container handling process	0.2	0.33	O (n <sup>2</sup> )

Methods	Event Logs	Performance Metrics		Time Complexity
		Fitness (0.0 – 1.0)	Precision (0.0 – 1.0)	
	Certificate formation process	1.0	0.5	
	Cotton Production	0,6	1.0	
	Subprocess of Retail	1.0	1.0	
Graph-based Non-Free Choice	Port container handling process	0.2	0.5	O (n <sup>3</sup> )
	Certificate formation process	1.0	0.5	
	Cotton Production	0,6	1.0	
	Subprocess of Retail	1.0	1.0	
Graph-based Invisible Task	Port container handling process	1.0	0.28	O (n <sup>2</sup> )
	Certificate formation process	1.0	0.5	
	Cotton Production	0.6	1.0	
	Subprocess of Retail	1.0	1.0	
Graph-based Non-Free Choice and Invisible Task	Port container handling process	1.0	0.83	O (n <sup>3</sup> )
	Certificate formation process	1.0	0.5	
	Cotton Production	0.6	1.0	
	Subprocess of Retail	1.0	1.0	

Where: fitness : the metric of calculating the capability of delineating a log processes into a model  
precision : the metric of calculating the suitability of extracted processes of a model with processes of a log

Based on Table 6, there are differences in terms of performance. Comparing Alpha Miner and Graph-based for parallel processes, Alpha Miner has higher fitness and higher precision in the container handling process and Graph-based for parallel processes has higher fitness and higher precision in the certificate formation processes; therefore, Graph-based for parallel processes cannot depict the right parallel relationships if the process has skip condition, and Alpha Miner cannot depict OR relation. Comparing Alpha++ and Graph-based algorithm for processes containing non-free choice, Alpha++ has higher fitness and higher precision in the container handling process and Graph-based for processes containing non-free choice has higher fitness and higher precision in the certificate formation processes. Same as the previous comparison, Graph-based for processes containing non-free choice cannot depict the right parallel relationships if there is a skip condition, and Alpha++ cannot depict OR relation. For other algorithms, they both have high fitness and high precision in the processes of port container handling, cotton production, and subprocess of Retail. On the other hand, in the process of certificate formation, Graph-based algorithms have higher fitness and higher precision than Alpha# and Alpha\$ because two Alpha algorithms cannot depict OR relation. However, broadly speaking, all of Graph-based algorithms are as effective as Alpha Miner and its expansions because they have high fitness and high precision.

Based on Table 6, it can be concluded that all Graph-based algorithms are more efficient than Alpha Miner and its expansions in the term of the time complexity. It is because the highest time complexity of all Graph-based algorithms, O (n<sup>3</sup>), is lower than the highest time complexity of Alpha Miner and its expansions, O (n<sup>4</sup>). Alpha Miner and its expansions have high time complexity because when taking the data of the event log, they did not record relationships of activities directly. Therefore, they always check the possible relationships of the combination of activities in the event log. On the contrary, Graph-based algorithms record the relationships when taking the data of the event log, so in the process discovery, those algorithms only analyze the relationships directly, instead of finding the possible relationships of the activities combination.

## 5. Conclusion

Graph-based algorithms are algorithms for discovering a process model by storing both of activities and their relationships in a graph database and processing the graph database into a process

model. Graph-based algorithms handle many aspects, such as parallel relationship, non-free choice constructs, invisible tasks, and non-free choice in invisible tasks.

This research evaluates Graph-based algorithms in context of time complexity, fitness, and precision. The evaluation is comparing Graph-based algorithms and Alpha Miner and its expansions, such as Alpha++, Alpha#, and Alpha\$. Based on the evaluation, Graph-based algorithms are effective as Alpha Miner and its expansions. It can be seen that all of those algorithms have high fitness and precision. However, all of Graph-based algorithms are more efficient because those algorithms have less time complexities than Alpha Miner and its expansions. The time complexities of Graph-based algorithms are  $O(n^2)$  and  $O(n^3)$ , whereas Alpha Miner and its expansions have  $O(n^4)$  as their time complexities.

For future work, this research will form heuristic [22], [31] Graph-based algorithms that consider the frequency of appearance of the processes in the formation of process models. Thereafter, this research will store the event log as a graph database directly, e.g. using Neo4j, to reduce the steps of the process discovery.

### Acknowledgment

Authors give a deep thank to Institut Teknologi Sepuluh Nopember, the Ministry of Research, Technology and Higher Education of Indonesia, *Direktorat Riset dan Pengabdian Masyarakat*, and *Direktorat Jenderal Penguatan Riset dan Pengembangan Kementerian Riset, Teknologi dan Pendidikan Tinggi Republik Indonesia* for supporting the research.

### References

- [1] R. Sarno and K. R. Sungkono, "Coupled Hidden Markov Model for Process Discovery of Non-Free Choice and Invisible Prime Tasks," *Procedia Computer Science*, vol. 124, pp. 134–141, 2018. <http://doi.org/10.1016/j.procs.2017.12.139>.
- [2] R. Sarno and K. R. Sungkono, "Coupled Hidden Markov Model for Process Mining of Invisible Prime Tasks," *International Review on Computers and Software (IRECOS)*, vol. 11, no. 6, pp. 539–547, 2016. <http://doi.org/10.15866/irecos.v11i6.9555>.
- [3] R. Sarno and K. R. Sungkono, "Hidden Markov Model for Process Mining of Parallel Business Processes," *International Review on Computers and Software (IRECOS)*, vol. 11, no. 4, pp. 290–300, 2016. <http://doi.org/10.15866/irecos.v11i4.8700>.
- [4] K. R. Sungkono and R. Sarno, "Constructing Control-Flow Patterns Containing Invisible Task and Non-Free Choice Based on Declarative Model," *International Journal of Innovative Computing, Information and Control (IJICIC)*, vol. 14, no. 4, 2018.
- [5] K. R. Sungkono, R. Sarno, and N. F. Ariyani, "Refining business process ontology model with invisible prime tasks using SWRL rules," in *2017 11th International Conference on Information Communication Technology and System (ICTS)*, 2017, pp. 215–220. <http://doi.org/10.1109/ICTS.2017.8265673>.
- [6] R. Sarno, W. A. Wibowo, Kartini, Y. A. Effendi, and K. R. Sungkono, "Determining Model Using Non-Linear Heuristics Miner and Control-Flow Pattern," *TELKOMNIKA (Telecommunication, Computing, Electronics and Control)*, vol. 14, no. 1, pp. 349–360, 2016. <http://doi.org/10.12928/telkomnika.v14i1.3257>.
- [7] K. R. Sungkono and R. Sarno, "Patterns of fraud detection using coupled Hidden Markov Model," in *2017 3rd International Conference on Science in Information Technology (ICSITech)*, 2017, pp. 235–240. <http://doi.org/10.1109/ICSITech.2017.8257117>.
- [8] R. Sarno, R. D. Dewandono, T. Ahmad, M. F. Naufal, and F. Sinaga, "Hybrid Association Rule Learning and Process mining for Fraud Detection," *IAENG International Journal of Computer Science*, vol. 42, no. 2, pp. 59–72, 2015.
- [9] S. Huda, R. Sarno, T. Ahmad, and H. A. Santoso, "Identification of Process-based Fraud Patterns in Credit Application," in *2014 2nd International Conference on Information and Communication Technology (ICoICT)*, 2014, pp. 84–89. <http://doi.org/10.1109/ICoICT.2014.6914045>.
- [10] A. S. Osses, L. Q. Da Silva, B. F. Cobo, and M. Arias, "Business process analysis in advertising: An extension to a methodology based on process mining projects," in *Computer Science Society (SCCC), 2016 35th International Conference of the Chilean*, 2016, pp. 1–12. <http://doi.org/10.1109/sccc.2016.7836000>.
- [11] W. Chomyat and W. Premchaiswadi, "Process mining on medical treatment history using conformance checking," in *ICT and Knowledge Engineering (ICT&KE), 2016 14th International*

- Conference, 2016, pp. 77–83. <http://doi.org/10.1109/ictke.2016.7804102>.
- [12] 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,” in *OTM Conferences (1)*, 2012, vol. 7565, no. 1, pp. 305–322. [http://doi.org/10.1007/978-3-642-33606-5\\_19](http://doi.org/10.1007/978-3-642-33606-5_19).
- [13] A. K. A. De Medeiros, B. F. Van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters, “Process mining: Extending the  $\alpha$ -algorithm to mine short loops,” *Eindhoven University of Technology Eindhoven*, pp. 1–25, 2004. <http://doi.org/10.1016/j.is.2011.01.003>.
- [14] 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, 2007. <http://doi.org/10.1007/s10618-007-0065-y>.
- [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, Oct. 2010. <http://doi.org/10.1016/j.datak.2010.06.001>.
- [16] Q. Guo, L. Wen, J. Wang, Z. Yan, and P. S. Yu, “Mining Invisible Tasks in Non-free-choice Constructs,” in *Lecture Notes in Computer Science*, Springer International Publishing, 2015, pp. 109–125. [http://doi.org/10.1007/978-3-319-23063-4\\_7](http://doi.org/10.1007/978-3-319-23063-4_7).
- [17] N. Russell, A. H. M. Ter Hofstede, W. M. P. van der Aalst, and N. Mulyar, “WORKFLOW CONTROL-FLOW PATTERNS A Revised View,” *BPM Center Report*, vol. 2, pp. 06–22, 2006. <http://doi.org/10.1.1.93.6974>.
- [18] W. M. P. Van Der Aalst, *Process Mining Discovery, Conformance and Enhancement of Business Processes*. Germany: Springer, 2011.
- [19] W. M. P. Van Der Aalst and A. H. M. Hofstede, “YAWL : yet another workflow language,” vol. 30, pp. 245–275, 2005. <http://doi.org/10.1016/j.is.2004.02.002>.
- [20] E. Börger, “Approaches to modeling business processes : a critical analysis of BPMN , workflow patterns and YAWL,” pp. 305–318, 2012. <http://doi.org/10.1007/s10270-011-0214-z>.
- [21] R. Sarno, K. R. Sungkono, and R. Septiarakhman, “Graph-Based Approach for Modeling and Matching Parallel Business Processes,” *International Information Institute (Tokyo). Information*, vol. 21, no. 5, pp. 1603–1614, 2018.
- [22] R. Sarno, Y. A. Effendi, and F. Haryadita, “Modified Time-Based Heuristics Miner for Parallel Business Processes,” *International Review on Computers and Software (IRECOS)*, vol. 11, no. 3, pp. 249–260, 2016. <http://doi.org/10.15866/irecos.v11i3.8717>.
- [23] R. Sarno, Kartini, W. A. Wibowo, and A. Solichah, “Time based Discovery of parallel business processes,” *Proceeding - 2015 International Conference on Computer, Control, Informatics and Its Applications: Emerging Trends in the Era of Internet of Things, IC3INA 2015*, pp. 28–33, 2016. <http://doi.org/10.1109/IC3INA.2015.7377741>.
- [24] W. M. P. van der Aalst, T. Weijters, and L. Maruster, “Workflow mining: discovering process models from event logs,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004. <http://doi.org/10.1109/TKDE.2004.47>.
- [25] A. Rozinat, R. S. Mans, M. Song, and W. M. P. van der Aalst, “Discovering colored Petri nets from event logs,” *International Journal on Software Tools for Technology Transfer*, vol. 10, no. 1, 2008. <http://doi.org/10.1007/s10009-007-0051-0>.
- [26] R. Sarno, B. A. Sanjoyo, I. Mukhlash, and H. M. Astuti, “Petri Net model of ERP business process variation for small and medium enterprises,” *Journal of Theoretical & Applied Information Technology*, vol. 54, no. 1, pp. 31–38, 2013.
- [27] M. Werner and N. Gehrke, “Process Mining,” *WISU - die Zeitschrift für den Wirtschaftsstudenten 7/13*, pp. 1–16, 2013.
- [28] R. Sarno, K. R. Sungkono, A. Y. Hadiwijaya, and C. Faticah, “An Algorithm for Discovering Process Models Containing Non-Free Choice Using Graph Database,” *Intelligent Networks and Systems Society (INASS)*.
- [29] R. Sarno, K. R. Sungkono, R. Johaness, and D. Sunaryono, “Graph-Based Algorithms for Discovering A Process Model Containing Invisible Tasks,” *Intelligent Networks and Systems Society (INASS)*.
- [30] H. Dermawan, R. Sarno, and K. R. Sungkono, “Graph-based Algorithms for Discovering Non-free-choice in Invisible Tasks of Business Process,” *International Journal of Electrical and Computer Engineering (IJECE)*.
- [31] A. J. M. M. Weijters, W. M. P. van der Aalst, and A. K. A. De Medeiros, “Process Mining with the Heuristics Miner Algorithm,” *Technische Universiteit Eindhoven, Tech. Rep. WP*, vol. 166, no. July 2017, pp. 1–34, 2006.