

A survey of graph-based algorithms for discovering business processes



Riyanarto Sarno ^{a,1,*}, Kelly Rossa Sungkono ^{a,2}

^a Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia

¹ riyanarto@if.its.ac.id; ² kelly@its.ac.id

* corresponding author

ARTICLE INFO

Article history

Received November 2, 2018

Revised June 15, 2019

Accepted July 4, 2019

Available online July 30, 2019

Keywords

Survey

Graph database

Process discovery

Quality

ABSTRACT

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 analyses 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 expansion. Other than that, this research also gives outline explanations about graph-based algorithms and their focus issues. Based on the evaluations, the graph-based algorithm has high performance and less time complexity than Alpha Miner algorithm.



This is an open access article under the [CC-BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



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 that, algorithms for automatically detecting a business process model are needed. A set of those algorithms is called process discovery. Several topics implement process discovery, such as business [1]–[8], fraud [9]–[11], and advertising [12]. Several findings process combination of tuples to depict relationships and conditions, such as invisible tasks, parallel relationships, and non-free choice, and combine them into a process model [13]–[16]. Other researches calculate the frequency of activities occurrences to determine the conditions and relationships [5], [17]–[19]. Probabilities of the Hidden Markov Model are also used to decide the relationships in a process model [1], [20].

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-database algorithm has improved, so there are a group of graph-database algorithms containing a graph-based algorithm of the parallel process [21], a graph-based algorithm of processes containing non-free choice which is proposed by the author, a graph-based algorithm of processes containing invisible task [22].

Graph-based algorithms which are analyzed by this paper are limited because the graph utilization in process discovery is rare. Graph algorithm is usually implemented in knowledge mining [23]–[26] and

image classification [27]–[31]. Yan [19] proposed a graph algorithm for process mining; however, this algorithm is not discussed in this paper because the graph is used for decomposing the log and that paper uses rules of the heuristic miner for depicting a process model.

This research analyzes all of the graph-based algorithms. Several questions guide this research to analyze; Q(1) What are issues that are handled by graph-based algorithms? Q(2) How graph-based algorithms handle those issues? Q(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 in [4], [32] to determine the performance of the obtained process model. Fitness is a measurement of completeness of a model based on the process 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 [33] and its expansions, i.e. Alpha++ [15] that concerns with non-free choice constructs, Alpha# [34] 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.

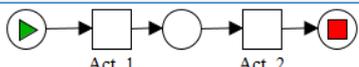
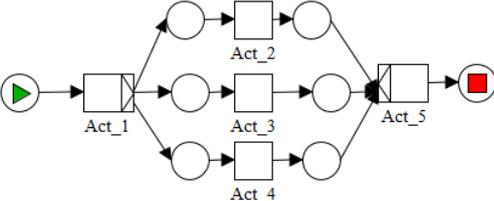
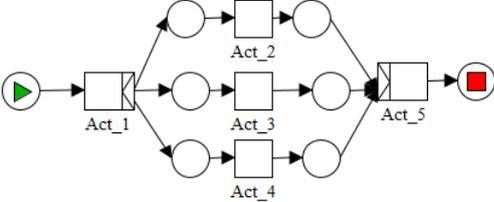
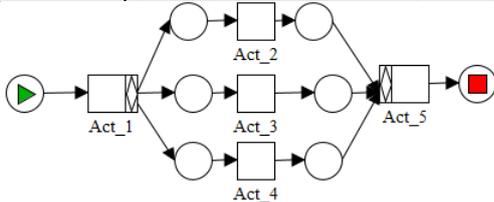
2. Method

2.1. Issues in Discovering Process Models

2.1.1. Parallel Relationships

In a process model, an activity has a relationship with other activities. There is a condition when two activities are related to each other for all processes, or 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. Table 1 explains both of sequential and parallel relationships.

Table 1. A process model containing parallel relationships.

1. Sequential Relationships	
<i>Processes in the Log</i>	<i>Process model</i>
{Act_1, Act_2}, {Act_1, Act_2}, {Act_1, Act_2}	
2. Parallel relationships	
2.1 XOR relationships	
{Act_1, Act_2, Act_5}, {Act_1, Act_3, Act_5}, {Act_1, Act_3, Act_5}	
2.2 AND relationships	
{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}	
2.3 OR relationships	
{Act_1, Act_2, Act_3, Act_5}, {Act_1, Act_3, Act_4, Act_5}, {Act_1, Act_4, Act_2, Act_5}	

This research uses YAWL notation [35], [36] 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 which 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 the XOR relationship if only one of them is selected in a process. As seen in Table 1, there is only one of the 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 a different order of executions. The example log is shown in Table 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 Table 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 [37], [38].

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 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], [15]. The simplified example is shown in Table 2. Based on the event log, Act_5 always executed when Act_2 is chosen, vice versa for Act_6. The real example is part of choosing transportation online. In the application, there is two option of transportation online, such as a motorcycle and a car. Even if there are two choices, when a customer bought large stuff, he chooses a car rather than a motorcycle. Conversely, if a customer bought small or no stuff, most likely he chooses a motorcycle. The relationship between the selection of transportation online and the selection of stuff is depicted by non-free choice.

Table 2. A process model containing non-free choice relationships.

Non-free choice Relationships	
Processes in the Log	Process model
{Act_1, Act_2, Act_4, Act_5, Act_7}, {Act_1, Act_3, Act_4, Act_6, Act_7}	

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 Table 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 the activity of previous selection is chosen, process discovery algorithm detects those relationships as NONFREE CHOICE.

2.1.3. Invisible tasks

Besides non-free choice, several conditions cannot be handled by both parallel and sequential relationships. Those conditions are skip condition, redo condition, and switch condition.

The first condition is a skip 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. Table 3 explains a skip condition in the event log and the process model. As shown in Table 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. Table 3 explains a redo condition in the event log and the process model. As shown in Table 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.

Table 3. A process model containing invisible tasks relationships.

Invisible Tasks	
Invisible Tasks for handling skip conditions	
Processes in the Log	Process model
{Act_1, Act_2, Act_3}, {Act_1, Act_3}, {Act_1, Act_3}	
Invisible tasks for handling redo conditions	
{Act_1, Act_2, Act_3, Act_2, Act_3, Act_4}, {Act_1, Act_2, Act_3, Act_2, Act_3, Act_4}	
Invisible tasks for handling switch conditions	
{Act_1, Act_2, Act_4, Act_6}, {Act_1, Act_2, Act_5, Act_6}, {Act_1, Act_3, Act_5, Act_6}	

2.2. Description of Method

2.2.1. Alpha Miner

Alpha Miner algorithm is a deterministic process discovery algorithm that develops causality of activities based on the event log [39]. Alpha Miner discovers a process model that has sequence relationships or parallel relationships, such as XOR relationship and AND relationship. The alpha algorithm utilizes workflow-nets in the form of Petri Nets [40], [41].

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

1. All of the activities in *ActGroup1* and *ActGroup2* are stored in the event log.
2. All of the activities in *ActGroup1* have casual dependencies with all activities in *ActGroup2*. A casual 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 casual dependency with *R* because activity *K* is followed by activity *R* and activity *R* is not followed by activity *K*.
3. All of the 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.
5. For example, if there are two tuples, (K,R) and (K,S) , it can be combined into a tuple, $(K, \{R,S\})$.

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. 1. This process model has an XOR relationship between activity R and activity S.

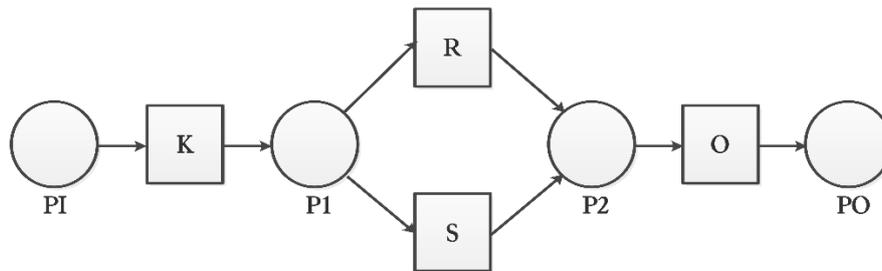


Fig. 1. A process model by the alpha miner.

2.2.2. Alpha++

Alpha++ improves Alpha Miner to depicting non-free choice in a process model forming a 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 the Alpha++ algorithm is showed in Fig. 2. The first step and the second step are the parts of the Alpha algorithm. Alpha++ adds the third and the four steps for creating non-free choice constructs.

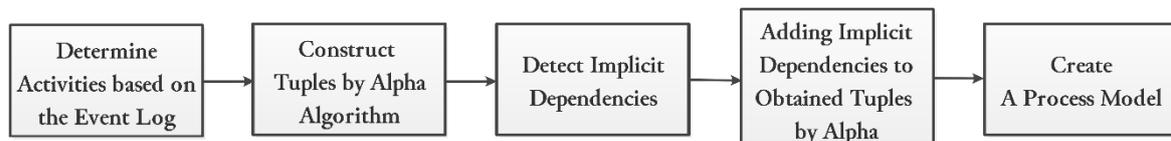


Fig. 2. Steps of Alpha++ Algorithm

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 aims to detect invisible prime tasks from event logs. This algorithm is derived from the 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 to the Alpha++ algorithm. The different is Alpha# adds invisible tasks, meanwhile Alpha++ adds implicit dependencies. The steps of Alpha# can be seen in Fig. 3. There are several steps to detect invisible tasks by Alpha# algorithm.

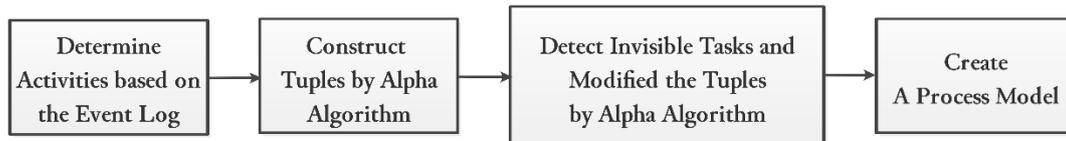


Fig. 3. Steps of Alpha# Algorithm

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\$ 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. 4. Alpha\$ improves the rules of mendacious dependencies in a# 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#.

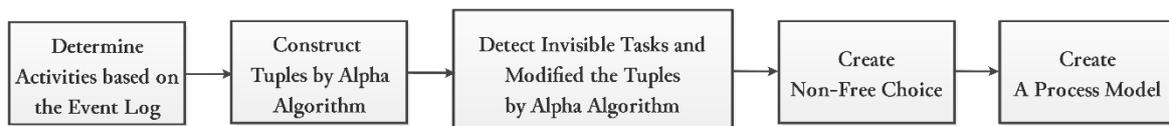


Fig. 4. Steps of Alpha\$ Algorithm

2.2.5. 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 the graph-based algorithm for the parallel processes, such as XOR, OR, and AND. Table 4 describes the step-by-step of a graph-based algorithm for the parallel process.

Based on Table 4, 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. After that, the research discovers AND relationship. The activities are included in AND relationship if the number of outgoing arcs of the activity is the same as the number of outgoing arcs of its previous activity. Lastly, the 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.

Table 4. Graph-based algorithm for the parallel process.

No	Steps of algorithm
The input is an event log that contains names of activities, number of case or process, and execution time of activities	
1	Storing event log in the format of graph database following rules in Table 7.
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 the previous nodes, as well as the 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 the 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 the 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 the 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
The output is a graph process model containing parallel relationships	

2.2.6. Graph-Based Algorithm for Processes Containing Non-Free Choice

Graph-based algorithm for processes containing non-free choice is an expansion algorithm of a 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 5 shows the pseudocode of the graph-based algorithm for processes containing non-free choice. This algorithm creates an implicit dependency between two activities if that activity is in the same process and the beginning activity of the implicit dependency is executed before the end activity. This statement can be seen in the fifth step. The final result of the algorithm is a graph process model.

Table 5. Graph-based algorithm for processes containing a non-free choice

No	Steps of algorithm
The input is an event log that contains names of activities, number of case or process, and execution time of activities	
1	Converting event log following rules in Table 7.
2	Creating a graph process model following rules in Table 4.
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 the first list of Table 7:
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 execution time 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
The output is a graph process model containing a non-free choice	

2.2.7 Graph-Based Algorithm for Processes Containing Invisible Task

Graph-based algorithm for processes containing invisible task is an expansion algorithm of a graph-based algorithm for parallel processes. This algorithm adds the rule to obtain an invisible task in the graph-based algorithm for parallel processes. Table 6 shows the pseudocode of the graph-based algorithm for processes containing invisible tasks. This algorithm has specific steps. Those steps are executed after the steps of the 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 is formed a graph process model.

Table 6. Graph-based Algorithm for Processes Containing Invisible Tasks

No	Steps of algorithm
The input is an event log that contains names of activities, number of case or process, and execution time of activities	
1	Converting event log following rules in Table 7.
2	Creating a graph process model following rules in Table 4.
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 <i>Invisible_Task</i>
7	Creating a graph sequence that fulfills {node act_i, relation_a, <i>Invisible_Task</i> }
8	Creating a graph sequence that fulfills { <i>Invisible_Task</i> , relation_b, node act_2}
7	Deleting relation_b that connects node act_i and node act_2
The output is a graph process model containing invisible tasks	

2.2.8. Graph-Based Approach of Processes Containing Non-Free Choice and Invisible Task

The graph-based approach of processes containing non-free choice and the invisible task is a combination of a graph-based algorithm of non-free choice and graph-based algorithm of an invisible task. This algorithm applies the steps of an invisible task in the graph-based algorithm and then applies steps of non-free choice. The obtained process model is formed in the graph process model.

All of the graph-based algorithms contain steps 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. Table 7 shows pseudocode for converting event logs. Even though the steps for converting event logs needs several times and costs, these steps decrease the time complexity of graph-based algorithms.

Table 7. A pseudocode of constructing a graph-database based on the event log.

No	Steps of algorithm
The input is an event log that contains names of activities, number of case or process, and execution time of activities	
1	Creating two lists 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 ^{a)} =1 to maximal_id ^{b)} :
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 connects node act1 and act2
The output is a graph database having SEQUENCE relation.	

^{a)} id = the index storing of activities in the first list.
^{b)} maximal_id = the maximal index storing in the first list.

3. Results and Discussion

This research evaluates the graph-based algorithm and Alpha algorithm using four-event logs as the data set. The detailed data set is shown in Table 8. The complex event log of all event logs is the event log of port container handling. It is because this event log has a non-free choice and invisible tasks. All of the processes in those event logs are complete and right processes, so there are no noises in there. 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 9.

Table 8. Event log for evaluation.

The name of the process	The number of cases	Issues					Noise (Y/N)
		<i>XOR</i> (Number Max Branch)	<i>OR</i> (Number Max Branch)	<i>AND</i> (Number Max Branch)	<i>Non-Free Choice</i>	<i>Invisible Task</i>	
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
The subprocess of Retail (Selling process and Recording Item Sales Journal)	50			v (4 branches)			N

Based on Table 9, all of the graph-based algorithms have less time complexity than Alpha Miner and its expansions. Meanwhile, there are differences in terms of performance. The results of Alpha Miner and Alpha++ have higher fitness and higher precision than those of graph-based algorithm of parallel process and non-free choice. It is because Alpha Miner and Alpha++ can handle skip condition without an invisible task, while graph-based cannot. Then, both of graph-based non-free choice invisible task and Alpha\$ has higher fitness and precision of the port event log because those algorithms can handle non-free choice and invisible task. Alpha Miner and its expansions cannot handle OR relation, so in the certificate event log, all of the Alpha algorithms have low fitness and precision. Meanwhile, graph-database algorithms have high fitness and precision because those can handle OR relation. Based on Table 9, the number of branches of OR relation affects the performance of graph-based algorithms. Graph-based algorithms cannot handle OR relation with two branches because the number of arcs is similar to AND relation in the graph-database. There is no problem to depict a process model of subprocess of retail because the issue is only AND relation.

Table 9. Performance metrics and time complexity.

Methods	Event Logs	Performance Metrics		Time Complexity
		<i>Fitness</i> ^{a)} (0.0 – 1.0)	<i>Precision</i> ^{b)} (0.0 – 1.0)	
Alpha Miner	Port container handling process	0.4	0.42	$O(n^4)$
	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	$O(n^4)$
	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	$O(n^4)$
	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	$O(n^4)$
	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	$O(n^2)$
	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	$O(n^3)$
	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	$O(n^2)$
	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	$O(n^3)$
	Certificate Formation process	1.0	0.5	
	Cotton Production	0.6	1.0	
	Subprocess of Retail	1.0	1.0	

^{a)} fitness = the metric of calculating the capability of delineating a log process into a model.

^{b)} precision = the metric of calculating the suitability of extracted processes of a model with processes of a log.

4. 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 the 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\$. Graph-based algorithms have less time complexity than Alpha Miner algorithm and its expansions. Then, out of all graph-based algorithms, the graph-based algorithm of non-free choice and the invisible task has higher fitness and precision. Meanwhile, all of the graph-based algorithms cannot handle OR relation with two branches. For future work, the graph-based algorithm of non-free choice and invisible tasks can be improved to handle OR relation with two branches. Then, the event log can be stored as a graph database directly, e.g., using Neo4j.

References

- [1] R. Sarno and K. R. Sungkono, "Coupled Hidden Markov Model for Process Discovery of Non-Free Choice and Invisible Prime Tasks," *Procedia Comput. Sci.*, vol. 124, pp. 134–141, 2018, doi: [10.1016/j.procs.2017.12.139](https://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," *Int. Rev. Comput. Softw.*, vol. 11, no. 6, pp. 539–547, 2016, doi: [10.15866/irecos.v11i6.9555](https://doi.org/10.15866/irecos.v11i6.9555).
- [3] R. Sarno and K. R. Sungkono, "Hidden Markov Model for Process Mining of Parallel Business Processes," *Int. Rev. Comput. Softw.*, vol. 11, no. 4, pp. 290–300, 2016, doi: [10.15866/irecos.v11i4.8700](https://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," *Int. J. Innov. Comput. Inf. Control*, vol. 14, no. 4, 2018, available at: [10.24507/ijicic.14.04.1285](https://doi.org/10.24507/ijicic.14.04.1285).
- [5] S. K. L. M. vanden Broucke and J. De Weerd, "Fodina: A robust and flexible heuristic process discovery technique," *Decis. Support Syst.*, vol. 100, pp. 109–118, 2017, doi: [10.1016/j.dss.2017.04.005](https://doi.org/10.1016/j.dss.2017.04.005).
- [6] R. Conforti, M. Dumas, L. García-Bañuelos, and M. La Rosa, "BPMN Miner: Automated discovery of BPMN process models with hierarchical structure," *Inf. Syst.*, vol. 56, pp. 284–303, 2016, doi: [10.1016/j.is.2015.07.004](https://doi.org/10.1016/j.is.2015.07.004).
- [7] F. M. Maggi, C. Di Ciccio, C. Di Francescomarino, and T. Kala, "Parallel algorithms for the automated discovery of declarative process models," *Inf. Syst.*, vol. 74, pp. 136–152, 2018, doi: [10.1016/j.is.2017.12.002](https://doi.org/10.1016/j.is.2017.12.002).
- [8] D. Chapela-Campa, M. Mucientes, and M. Lama, "Mining Frequent Patterns in Process Models," *Inf. Sci.*, vol. 472, pp. 235–257, 2019, doi: [10.1016/j.ins.2018.09.011](https://doi.org/10.1016/j.ins.2018.09.011).
- [9] 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, doi: [10.1109/ICSITech.2017.8257117](https://doi.org/10.1109/ICSITech.2017.8257117).
- [10] R. Sarno, R. D. Dewandono, T. Ahmad, M. F. Naufal, and F. Sinaga, "Hybrid Association Rule Learning and Process mining for Fraud Detection," *IAENG Int. J. Comput. Sci.*, vol. 42, no. 2, pp. 59–72, 2015, available at: [Google Scholar](https://scholar.google.com/citations?view_op=view_citation&hl=en&user=IAENG).
- [11] 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, doi: [10.1109/ICoICT.2014.6914045](https://doi.org/10.1109/ICoICT.2014.6914045).
- [12] 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, doi: [10.1109/sccc.2016.7836000](https://doi.org/10.1109/sccc.2016.7836000).
- [13] Z. He, Y. Du, L. U. Wang, L. Qi, and H. Sun, "An Alpha-FL Algorithm for Discovering Free Loop Structures From Incomplete Event Logs," *IEEE Access*, vol. 6, pp. 27895–27901, 2018, doi: [10.1109/ACCESS.2018.2840818](https://doi.org/10.1109/ACCESS.2018.2840818).

- [14] S. Akshay, L. Helouet, and R. Phawade, "Combining Free Choice and Time in Petri Nets," in *Proceedings of the International Workshop on Temporal Representation and Reasoning*, 2016, vol. 2016–Decem, pp. 120–129, doi: [10.1109/TIME.2016.20](https://doi.org/10.1109/TIME.2016.20).
- [15] L. Wen, W. M. P. van der Aalst, J. Wang, and J. Sun, "Mining process models with non-free-choice constructs," *Data Min. Knowl. Discov.*, vol. 15, no. 2, pp. 145–180, 2007, doi: [10.1007/s10618-007-0065-y](https://doi.org/10.1007/s10618-007-0065-y).
- [16] Q. Guo, L. Wen, J. Wang, Z. Yan, and P. S. Yu, "Mining Invisible Tasks in Non-free-choice Constructs," 2015, pp. 109–125, doi: [10.1007/978-3-319-23063-4_7](https://doi.org/10.1007/978-3-319-23063-4_7).
- [17] A. P. Kurniati, G. Kusuma, and G. Wisudiawan, "Implementing Heuristic Miner for Different Types of Event Logs," vol. 11, no. 8, pp. 5523–5529, 2016, available at: [Google Scholar](https://scholar.google.com/).
- [18] 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, Comput. Electron. Control.*, vol. 14, no. 1, pp. 349–360, 2016, doi: [10.12928/telkomnika.v14i1.3257](https://doi.org/10.12928/telkomnika.v14i1.3257).
- [19] Z. Yan *et al.*, "Decomposed and parallel process discovery: A framework and application," *Futur. Gener. Comput. Syst.*, vol. 98, pp. 392–405, 2019, doi: [10.1016/j.future.2019.03.048](https://doi.org/10.1016/j.future.2019.03.048).
- [20] K. R. Sungkono and R. Sarno, "CHMM for discovering intentional process model from event logs by considering sequence of activities," in *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2017, pp. 1–6, doi: <https://doi.org/10.1109/EECSI.2017.8239194>.
- [21] R. Sarno, K. R. Sungkono, and R. Septiarakhman, "Graph-Based Approach for Modeling and Matching Parallel Business Processes," *Int. Inf. Inst. (Tokyo). Inf.*, vol. 21, no. 5, pp. 1603–1614, 2018, available at: [Google Scholar](https://scholar.google.com/).
- [22] R. Sarno, K. R. Sungkono, R. Johanes, and D. Sunaryono, "Graph-Based Algorithms for Discovering A Process Model Containing Invisible Tasks," *Intell. Networks Syst. Soc.*, vol. 12, no. 2, pp. 85–94, 2019, available at: [Google Scholar](https://scholar.google.com/).
- [23] S. Velampalli and M. V Jonnalagedda, "Graph based knowledge discovery using MapReduce and SUBDUE algorithm," *Data Knowl. Eng.*, vol. 111, pp. 103–113, 2017, doi: [10.1016/j.datak.2017.08.001](https://doi.org/10.1016/j.datak.2017.08.001).
- [24] Q. Zhang, X. Song, Y. Yang, H. Ma, and R. Shibasaki, "Visual graph mining for graph matching," *Comput. Vis. Image Underst.*, vol. 178, pp. 16–29, 2019, doi: [10.1016/j.cviu.2018.11.002](https://doi.org/10.1016/j.cviu.2018.11.002).
- [25] M. İğde, Y. Kavurucu, and A. Mutlu, "Graph Representation of Relational Database for Concept Discovery," *Procedia - Soc. Behav. Sci.*, vol. 195, pp. 1981–1989, 2015, doi: [10.1016/j.sbspro.2015.06.212](https://doi.org/10.1016/j.sbspro.2015.06.212).
- [26] P. Braun, A. Cuzzocrea, C. K. Leung, A. G. M. Pazdor, and K. Tran, "Knowledge Discovery from Social Graph Data," *Procedia Comput. Sci.*, vol. 96, pp. 682–691, 2016, doi: [10.1016/j.procs.2016.08.250](https://doi.org/10.1016/j.procs.2016.08.250).
- [27] Z. Mohammadpoory, M. Nasrolahzadeh, N. Mahmoodian, and J. Haddadnia, "Automatic identification of diabetic retinopathy stages by using fundus images and visibility graph method," *Measurement*, vol. 140, pp. 133–141, 2019, doi: [10.1016/j.measurement.2019.02.089](https://doi.org/10.1016/j.measurement.2019.02.089).
- [28] Z. Chen, B. Jiang, J. Tang, and B. Luo, "Image Set Representation and Classification with Attributed Covariate-Relation Graph Model and Graph Sparse Representation Classification," *Neurocomputing*, vol. 226, pp. 262–268, 2017, doi: [10.1016/j.neucom.2016.12.004](https://doi.org/10.1016/j.neucom.2016.12.004).
- [29] R. Zhu, F. Dornaika, and Y. Ruichek, "Joint graph based embedding and feature weighting for image classification," *Pattern Recognit.*, vol. 93, pp. 458–469, 2019, doi: [10.1016/j.patcog.2019.05.004](https://doi.org/10.1016/j.patcog.2019.05.004).
- [30] H. Yuan, J. Li, L. L. Lai, and Y. Y. Tang, "Graph-based multiple rank regression for image classification," *Neurocomputing*, vol. 315, pp. 394–404, 2018, doi: [10.1016/j.neucom.2018.07.032](https://doi.org/10.1016/j.neucom.2018.07.032).
- [31] Y. Shao, N. Sang, C. Gao, and L. Ma, "Spatial and class structure regularized sparse representation graph for semi-supervised hyperspectral image classification," *Pattern Recognit.*, vol. 81, pp. 81–94, 2018, doi: [10.1016/j.patcog.2018.03.027](https://doi.org/10.1016/j.patcog.2018.03.027).
- [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," in *OTM Conferences (1)*, 2012, vol. 7565, no. 1, pp. 305–322, doi: [10.1007/978-3-642-33606-5_19](https://doi.org/10.1007/978-3-642-33606-5_19).

- [33] 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 α -algorithm to mine short loops," *Eindhoven Univ. Technol. Eindhoven*, pp. 1–25, 2004, doi: [10.1016/j.is.2011.01.003](https://doi.org/10.1016/j.is.2011.01.003).
- [34] L. Wen, J. Wang, W. M. P. van der Aalst, B. Huang, and J. Sun, "Mining process models with prime invisible tasks," *Data Knowl. Eng.*, vol. 69, no. 10, pp. 999–1021, Oct. 2010, doi: [10.1016/j.datak.2010.06.001](https://doi.org/10.1016/j.datak.2010.06.001).
- [35] W. M. P. Van Der Aalst and A. H. M. Hofstede, "YAWL : yet another workflow language," vol. 30, pp. 245–275, 2005, doi: [10.1016/j.is.2004.02.002](https://doi.org/10.1016/j.is.2004.02.002).
- [36] E. Börger, "Approaches to modeling business processes : a critical analysis of BPMN , workflow patterns and YAWL," pp. 305–318, 2012, doi: [10.1007/s10270-011-0214-z](https://doi.org/10.1007/s10270-011-0214-z).
- [37] R. Sarno, Y. A. Effendi, and F. Haryadita, "Modified Time-Based Heuristics Miner for Parallel Business Processes," *Int. Rev. Comput. Softw.*, vol. 11, no. 3, pp. 249–260, 2016, doi: [10.15866/irecos.v11i3.8717](https://doi.org/10.15866/irecos.v11i3.8717).
- [38] R. Sarno, Kartini, W. A. Wibowo, and A. Solichah, "Time based Discovery of parallel business processes," *Proceeding - 2015 Int. Conf. Comput. Control. Informatics Its Appl. Emerg. Trends Era Internet Things, IC3INA 2015*, pp. 28–33, 2016, doi: [10.1109/IC3INA.2015.7377741](https://doi.org/10.1109/IC3INA.2015.7377741).
- [39] W. M. P. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004, doi: [10.1109/TKDE.2004.47](https://doi.org/10.1109/TKDE.2004.47).
- [40] A. Rozinat, R. S. Mans, M. Song, and W. M. P. van der Aalst, "Discovering colored Petri nets from event logs," *Int. J. Softw. Tools Technol. Transf.*, vol. 10, no. 1, 2008, doi: [10.1007/s10009-007-0051-0](https://doi.org/10.1007/s10009-007-0051-0).
- [41] R. Sarno, B. A. Sanjoyo, I. Mukhlash, and H. M. Astuti, "Petri Net model of ERP business process variation for small and medium enterprises," *J. Theor. Appl. Inf. Technol.*, vol. 54, no. 1, pp. 31–38, 2013, available at: [Google Scholar](https://scholar.google.com/).