Received: October 22, 2018

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Graph-Based Algorithms for Discovering a Process Model Containing Invisible Tasks

Riyanarto Sarno[1]* Kelly Rossa Sungkono[1] Reynaldo Johanes[1] Dwi Sunaryono[1]

[1]*Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia*
* Corresponding author's Email: riyanarto@if.its.ac.id

Abstract: An event log records the business processes of a company. Modeling event logs aim to help users in analyzing business processes. One of the problems in modeling event logs automatically is the addition of invisible tasks. Invisible tasks are dummy activities, other than activities of an event log, that are added to a process model to describe a correct process model. This research proposes a graph-based algorithm to mine the data from an event log. From the data, the graph-based algorithm establishes an additional-invisible-task process model by converting all of the processes in the event log into a link list and adding invisible tasks and operators for parallel relations, such as XOR Split or XOR Join. The experimental analysis explains that the fitness of the discovered process models by the graph-based algorithm was as high as that of compared algorithms, such as Alpha# models, Alpha$ models, CHMM-NCIT models, and CHMM-IT models. Furthermore, the graph-based algorithm is more efficient than existing algorithms. This was proven by the time complexity of the graph-based, which is $O(n^2)$ while both of Alpha# and Alpha$ algorithm have a time complexity of $O(n^4)$ and both of CHMM-IT and CHMM-NCIT algorithm have $O(n^3)$.

Keywords: Graph-database, Invisible tasks, Process discovery.

1. Introduction

A repository of facts or processes of an organization is called an event log. The fact consists of the name of a task, the executor of the task, and the time of the task. These facts are processed for process analysis and problem discovery [1]. To sum up, an event log can be utilized to analyze and find problems in business processes.

Because some facts in an event log keep growing, process discovery facilitates the process of analyzing the facts by arranging them in a process model. Process discovery, as an element of process mining, can handle many different issues in many sectors: business [2 - 5], and fraud [6, 7], and medical [8]. One of the problems in process discovery that are addressed in this research is handling invisible tasks.

Invisible tasks are dummy activities, other than activities of an event log, that are added to a process model. One reason for adding invisible tasks is accommodating Split and Join relations in parallel processes. There are several algorithms that handle invisible tasks, such as Alpha# [9], Alpha$ [10], Coupled Hidden Markov Model-Nonfree Choice Invisible Task (CHMM-NCIT) [2] and Coupled Hidden Markov Model-Invisible Task (CHMM-IT) [5].

Both of Alpha# [9] and Alpha$ [10] determine invisible tasks by forming tuples and analyzing the tuples with a number of rules. Those algorithms only take activities from the log, so the relations between activities are determined by checking all possible relationships based on all activities. Those checking processes prolongs the processing time of Alpha#. Furthermore, both of Coupled Hidden Markov Model-Nonfree Choice Invisible Task (CHMM-NCIT) [2] and Coupled Hidden Markov Model-Invisible Task (CHMM-IT) [5] use Coupled Hidden Markov Model (CHMM) and utilize Baum-Welch algorithm to determine the weights of variables of CHMM. However, those algorithm check sequences of the event log and both of hidden variables and observed variables, so the time

complexities of CHMM-NCIT and CHMM-IT are high.

In this research, a graph-based algorithm is proposed to automatically model an event log containing invisible tasks by converting the event log into a graph-database and processing the graph-database into a process model. Graph-database is chosen because it can store only activities but also their relations, so it negates the steps of checking all possibilities relationships which are expected to reduce the time complexity. This graph-based algorithm is applied in Neo4j [11, 12] by using Cypher syntax [13]. As a summary, the major contributions of this research are:

[1] The graph-based algorithm is an algorithm that can depict a process model containing invisible tasks. The fitness value of the obtained process model by using the proposed graph-database verifies the contribution.

[2] The time complexity of the graph-based algorithm is low because the graph-based algorithm uses a graph-database that stores not only activities but also their relationships. The additional of invisible task is not can be checked directly from the relationship without looking for possible relationships between activities continuously. The time complexity of the graph-based algorithm will be compared with other existing algorithms of invisible task to prove this statement.

A focused issue in this research is detecting invisible tasks in the process models. Because of that, the other issues of process discovery are not discussed in this research. Therefore, the event log as the experiment data only contain the issue of invisible tasks.

The graph-based algorithm is evaluated by comparing the correctness and the time complexity of results yielded by Alpha#, Alpha$, CHMM-NCIT, and CHMM-IT algorithms. The correctness was calculated using a fitness measurement. Fitness measurement calculates completeness of facts that are depicted in a process model.

2. Problem statement and preliminaries

The following works are related to the graph-based algorithm, including invisible tasks and graph database.

2.1 Business process model

A set of connected activities which is created for a particular benefit is a business process. The business process records what activities are carried out when activities occur, and what conditions, initial or final conditions, during the execution. The business process can be transformed into an image, which is called a business process model.

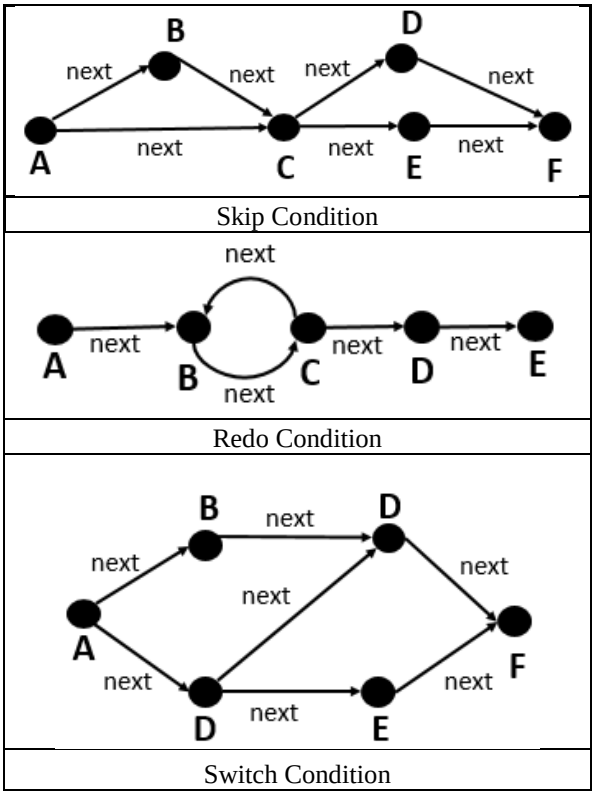

Figure. 1 Three types of invisible tasks: (a) skip condition, (b) redo condition, and (c) switch condition

A benefit of a business process model is illustrating the process clearly. There are considerable templates that represent a process model, for example, Unified Modeling Language (UML) [14], Causal Net [15], Business Process Model and Notation (BPMN) [16], Petri Net [17], and so forth. Every type of model needs diverse characteristic, for example, Petri net utilizes tokens to join activities in its model. On the other hand, Causal Net depicts its activities by making them associated straightforwardly.

2.2 Invisible tasks

Invisible tasks are unrecorded activities in an event log that show up in a business process model. There are several situations that trigger a process model to display invisible tasks. These situations distinguish different types of invisible tasks. Based on the Alpha# algorithm [9], there are three types of invisible tasks.

2.1.1. Skip and redo invisible tasks

The first invisible task represents the skip condition. A skip condition occurs when activities in sequential relations are not executed in separated processes. An example of a skip condition is shown in Figure. 1. There are four traces as the input data of a model in Figure. 1, i.e. [ABCDF, ACDF, ABCEF, ACEF]. As shown in Figure. 1, there is a skip condition when activity A can directly go to activity C. A skip invisible task will be added between those activities to depicting a skip condition.

Then, there is another invisible task that represents the redo condition. A redo condition occurs when two or more activities are executed repeatedly. An example of a redo condition is shown in Figure. 1. There are three traces in the example, i.e. [ABCBCD, ABCBCBCD, ABCBCD]. Based on these traces, both activity B and activity C are executed repeatedly. A redo invisible task will be added between those activities to depicting a redo condition.

2.1.2. Switch invisible tasks

The last invisible task represents the switch condition. A switch condition occurs when there is an activity of which it is uncertain where it is heading or where it came from. An example of a switch condition is shown in Figure. 1. There are three traces in the example, i.e. [ABCF, ADEF, ABEF]. Based on those traces, both activity B and activity C are in a switch condition. To providing the switch condition, an invisible task is added between activity B and C.

2.3 Existing process discovery algorithms of invisible tasks

Alpha# [12] is an algorithm used in process. There are several algorithms of discovering a process model containing invisible task. Those algorithms are Alpha# [9], Alpha$ [10], Coupled Hidden Markov Model-Nonfree Choice Invisible Task (CHMM-NCIT) [2] and Coupled Hidden Markov Model-Invisible Task (CHMM-IT) [5]. The contribution of Alpha# and CHMM-IT is detecting invisible tasks in the process model, while Alpha$ and CHMM-NCIT can depict invisible task and non-free choice in the process model. Both of Alpha# and Alpha$ uses tuples that denote the relationships of activity. Then, both of CHMM-IT and CHMM-HRIT utilize Coupled Hidden Markov Model to determine the relationships of activity.

There are several steps of Alpha# or Alpha$ algorithm. First, the activities, including an initial activity and the final activity, of the event log are determined. For example, based on four traces that are used to depict the model of skip condition in Figure. 1, T_w= { A, B, C, D, E, F }, while the initial activity is A and the final activity is F. The second step is a set of activity pairs (D_M) which fulfils the first rule is created. The first rule is *a* as the first activity of relationships with *b* and other activities (for example activity *k*) and *b* as the end activity of relationships with *a* and other activities (for example activity *l*) and there is no relationship between activity *l* as the first activity and activity *k* as the end activity. Based on the example, D_M = { (A, C) } because A has relationship with B, C also has a relationship with B, and there is no relationship between B and itself.

The third step is R_M as D_M that fulfils the second rule is determined. The rule is occurring a relationship between two activities (for example activity *k* and *l*) and *a* ~> *k* and *l* ~> *b*. This example doesn't have relationships that fulfil the second rule, so R_M is empty. The fourth step is arcs (YI) of the model and places (XI) that are divided into P_{in} which contains the first pairs for building relationships of invisible tasks and P_{out} that contains the end pairs are determined. Based on the example, P_{in} of XI = { (A,B) } and P_{out} of XI = { (B,C) }, so Y_I = { { (A, B), (B, C) } }. The fifth step is activities that are in both of P_{in} and P_{out} are replaced with invisible tasks and the change is stored in D_S. D_S is { (A, Invisible_Task), (Invisible_Task, C) }. The last step is the process model is constructed with the result in D_S.

CHMM-IT and CHMM-NCIT are also have several steps to depict invisible tasks. First, the Coupled Hidden Markov Model is constructed. Then, probabilities of the Coupled Hidden Markov Model is trained with the Baum-Welch algorithm [5]. Then, several rules are executed to depict the invisible tasks. The steps to train the Coupled Hidden Markov Model involves sequences of the event log and two types of nodes, such as hidden and observed variables, so the time complexity of those algorithms is increased.

2.4 Graph database

Graph databases provide connection of data that can be a guided chart for analysts [18]. Graph databases are not similar to relational databases. A relational database provides connection of tables as a group of data with similar characteristics, while a graph database forms connector of each data that is

stored as a node. For discovering a process model, the data is activities or events.

Table 1. An example of event log in database

Case_ Number	Activity_ Label	Starting_ Time	End_ Time
CN01	A	08:20	08:25
CN01	B	08:27	08:47
CN01	C	08:48	09:00
CN01	D	09:00	09:30
CN01	E	09:40	10:13
CN01	G	10:28	11:01
CN02	A	11:05	11:15
CN02	C	11:18	11:30
CN02	D	13:00	13:32
CN02	F	13:35	14:08
CN02	G	14:20	14:55

where :

Case_Number : identity number of processes of an event log

Activity_Label : names of events or activities that are executed in the processes

Starting_Time : the start time of execution of events or activities

End_Time : the end time of execution of events or activities

This research compares a relational database with a graph database. A relational database has a table with columns and rows, where the attributes must be determined first. Hence, it is difficult for users to change the attributes. An example of a relational database is shown in Table 1, where Case_Number, Activity_Label, Starting_Time, and End_Time are the attributes.

In a relational database, relationships between rows are not determined, so the relational database must be modified by adding a new column, for example, Previous_Act, which denotes the previous activity. The modification can be seen in Table 2. The additional column is determined manually by users by looking at the database or forming an additional query. Because the determination of Previous_Act is determined in each case, not the whole data, the Previous_Act column of the first activity is zero.

Modifying a relational database consumes a large amount of memory and time. On the other hand, a graph database is easy to modify because it is flexible and without modifying the table, a graph database automatically defines the relations between activities and their previous or next activities.

The nodes in a graph database are represented by a row in the relational database. Attributes from the relational database are saved in the nodes in a graph database. For example, a graph database that represents Table 1 is shown in a model of skip condition, which is visualized on Figure. 1.

Table 2. The modified event log

CN	AL	Starting_ Time	End_ Time	Previous_ Act
CN01	A	08:20	08:25	-
CN01	B	08:27	08:47	A
CN01	C	08:48	09:00	B
CN01	D	09:00	09:30	C
CN01	E	09:40	10:13	D
CN01	G	10:28	11:01	E
CN02	A	11:05	11:15	-
CN02	C	11:18	11:30	A
CN02	D	13:00	13:32	C
CN02	F	13:35	14:08	D
CN02	G	14:20	14:55	F

where :

CN : Case_Number

AL : Activity_Label

Previous_act : activities that are executed before an activity that is described in the line of the event log

\- : no previous activity of the activity that is in the same row

2.5 Control-flow pattern

This section explains how control-flow patterns [19] can capture elementary aspects of processes These patterns closely match the elementary control flow concepts used in this research. These patterns are a sequence and parallel patterns. The parallel patterns consist of exclusive choice, simple merge, parallel split, synchronization, multi-choice, and structured synchronizing merge.

The sequence is a point about a relationship in which one activity is processed after another activity has been executed in the same process. The exclusive choice is a pattern that has XOR Split as the point of the workflow and simple merge is a pattern that has XOR Join as the point of the workflow. An AND Split is a point in parallel split pattern and an AND Join is a point in synchronization pattern. Finally, multi-choice has OR Split as its point and structured synchronizing merge has OR Join as its point.

2.6 Fitness

Fitness is a measurement of quality of a process model based on the suitability of the model and an event log. If a number of processes that are depicted in the model is higher, the fitness value is higher.

To calculate the fitness, Eq. (1) shows the calculation. Based on this equation, n_{right_cases} is the total number of processes that were depicted in

the process model, while n_{total_cases} is the number of processes of the event log.
For the example, based on the model of switch condition in Figure. 1 (c), all traces, i.e. trace ABCF, trace ADEF, trace ABEF can be depicted. Because of that, n_{right_cases} is three and n_{total_cases} is three. In conclusion, the fitness value of the model of switch condition is 1.0.

$$Fitness\ (F) = \frac{n_{right\_cases}}{n_{total\_cases}} \qquad (1)$$

3. Proposed method

In this section, graph-based algorithm is explained to modeling the event log into a process model in Neo4j. Graph-based algorithm and Alpha# algorithm are compared based on their time complexity and fitness measurement.

The first step of graph-based algorithm is converting the event log from CSV file into a graph-database. After that, graph-based algorithm adds parallel relationships and invisible tasks based on its pseudocode in Table 3. The application of the pseudocode should be in an order of rows of Table 3.

The sequence pattern is the starting point of the pseudocode and the invisible task is the end point. First, all activities are depicted as a model with sequence relationships based on the pseudocode of sequence pattern. Activity is a list of variants of activities and all_activity is a list of all activities in the event log. For example, based on traces [A,B,C], [A,B,C], [A,C], activity has { A, B, C } and all_activity has { A, B, C, A, B, C, A, C }. The model is established by adding sequence relationships if activities in activity are sequence in each process of all_activity. The obtained sequence relationships based on the example are A-[:SEQUENCE]->B, B-[:SEQUENCE]->C, and A-[:SEQUENCE]->C.

Secondly, the pseudocode of exclusive choice until structured synchronizing merge patterns are executed. Based on the example in previous paragraph, the pseudocode of exclusive choice and simple merge patterns is fulfilled. The result is A-[:XORSPLIT]->B, B-[:XORJOIN]->C, A-[:XORJOIN]->C.
Finally, the pseudocode of invisible task is executed. An invisible task is created when an activity has SPLIT relationships and JOIN relationships. The invisible task is added between JOIN relationships between the activity and other activities. Based on the previous paragraph, activity A has two types of relationships, i.e. XORSPLIT and XORJOIN. Then, the invisible task is added between activity A and activity C because their relationship is XORJOIN. The final result is A-[:XORSPLIT]->B, B-[:XORJOIN]->C, A-[:XORSPLIT]->Inv_Task, Inv_Task-[:XORJOIN]->C. The illustration of this result is shown in a submodel of Figure. 2 that is starting from activity A and ending at activity C.

Table 3. Pseudocode of graph-based algorithm

Pattern	**Pseudocode**
Sequence	*for idx=0 until idx=count_activity_ all_processes:* *if all_activity[idx] and all_activity [idx+1] are in one process:* *for i=0 until i=count_variant_ activity:* *if activity[i]=all_activity [idx]:* *first_act = activity[i]* *if activity[i]=all_activity [idx+1]* *second_act = activity[i]* *Create first_act-[:SEQUENCE]-> second_act*
Exclusive Choice	*Match activity[i]-[:relation1]- >activity[i+1]* *if outgoing(activity[i])>1 and ingoing(activity[i+1])=1 and (outgoing(activity[i+1]=1 or outgoing(activity[i+1]>1):* *Create activity[i]-[:XORSPLIT]->activity [i+1]* *Delete [:relation1] from activity[i]-[:relation1]- >activity[i+1]*
Simple Merge	*Match activity[i]-[:relation1]- >activity[i+1]* *if (outgoing(activity[i])=1 or outgoing(activity[i])=>1) and ingoing(activity[i+1])>1:* *Create activity[i]-[:XORJOIN]->activity [i+1]* *Delete [:relation1] from activity[i]-[:relation1]- >activity[i+1]*

Parallel Split	*Match activity[i]-[:relation1]->activity[i+1]* *Match activity[i]-[:relation1]->activity[i+2]* *if outgoing(activity[i])>1 and outgoing(activity[i+2])=outgoing(activity[i]) and not (activity[i+1]-[:relation1]->activity[i] or activity[i+2]-[:relation1]->activity[i]):* *Create activity[i]-[:ANDSPLIT]->activity[i+1]* *Create activity[i]-[:ANDSPLIT]->activity[i+2]* *Delete [:relation1] from activity[i]-[:relation1]- >activity[i+1]* *Delete [:relation1] from activity[i]-[:relation1]- >activity[i+2]*
Synchron-ization	*Match activity[i+1]-[:relation1]->activity[i]* *Match activity[i+2]-[:relation1]->activity[i]* *if ingoing(activity[i])>1 and outgoing(activity[i+2])= ingoing(activity[i]):* *Create activity[i+1]-[:ANDJOIN]->activity[i]* *Create activity[i+2]-[:ANDJOIN]->activity[i]* *Delete [:relation1] from activity[i+1]-[:relation1]->activity[i]* *Delete [:relation1] from activity[i+2]-[:relation1]->activity[i]*
Multi Choice	*Match activity[i]-[:relation1]->activity[i+1]* *Match activity[i]-[:relation1]->activity[i+2]* *if outgoing(activity[i])>1 and (outgoing(activity[i+2])<outgoing(activity[i]) and outgoing(activity[i+2])>1) and not (activity[i+1]- [:relation1]->activity[i] or activity[i+2]- [:relation1]->activity[i]):* *Create activity[i]-[:ORSPLIT]->activity[i+1]* *Create activity[i]-[:ORSPLIT]->activity[i+2]* *Delete [:relation1] from activity[i]-[:relation1]- >activity[i+1]*
Structural Synchron-ization Merge	*Match activity[i+1]-[:relation1]->activity[i]* *Match activity[i+2]-[:relation1]->activity[i]* *if ingoing(activity[i])>1 and (outgoing(activity[i+2])< ingoing(activity[i]) and outgoing(activity[i+2]) >1):* *Create activity[i+1]-[:ORJOIN]->activity[i]* *Create activity[i+2]-[:ORJOIN]->activity[i]* *Delete [:relation1] from activity[i+1]-[:relation1]->activity[i]* *Delete [:relation1] from activity[i+2]-[:relation1]->activity[i]*
Invisible Task (Skip, Switch, or Redo)	*Match activity[i]-[:relation1]->activity[i+1]* *Match activity[i]-[:relation2]->activity[i+2]* *if (:relation1=:XORSPLIT and :relation2=:XORJOIN) or (:relation1=:ORSPLIT and :relation2=:ORJOIN):* *Create invisible_task* *Create activity[i]-[:relation1]->invisible_task* *Create invisible_task-[:relation2]->activity[i+2]* *Delete [:relation2] from activity[i]-[:relation2]- >activity[i+2]*

where :

activity[i]-[] : activity [i] as a beginning activity of a relationship

[]->activity[i] : activity [i] as an end activity of a relationship

[]->activity[i+1] : activity [i+1] as an end activity of a relationship

[]->activity[i+2] : activity [i+2] as an end activity of a relationship

count_activity : sum of count for all activities

create : cypher syntax for making only activities or activities and their relationships

delete : cypher syntax for deleteing relationships or activities

i: invisible task : name of the invisible task

ingoing(activity) : sum of relations that come to the activity

match : cypher syntax for searching the statement that is followed this syntax

outgoing(activity): sum of relations that go from the activity

:relation1, :relation2 : initial variable for determining relationships in the graph-database

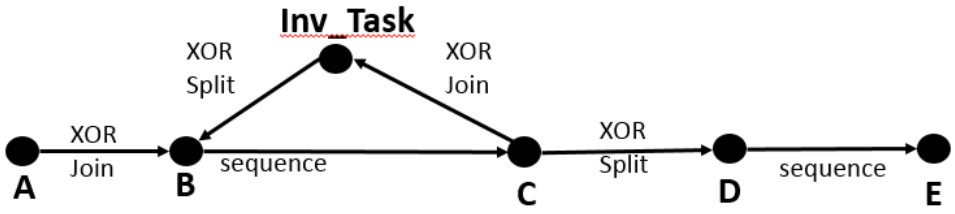

Figure. 2 Result of modeling a process model by graph-based algorithm in redo condition

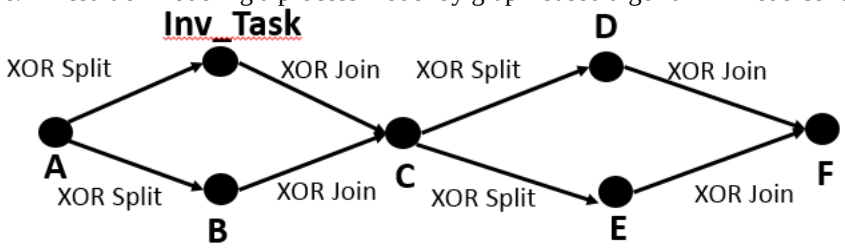

Figure. 3 Result of modeling a process model by graph-based algorithm in skip condition

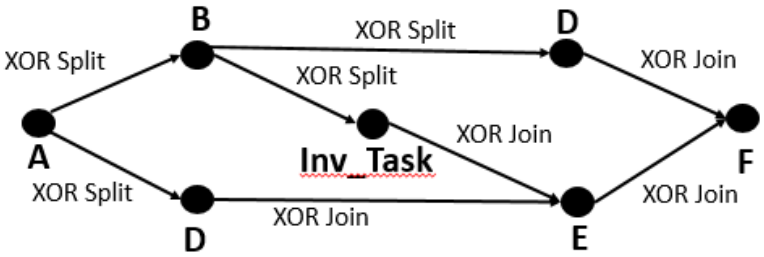

Figure. 4 Result of modeling a process model by graph-based algorithm in switch condition

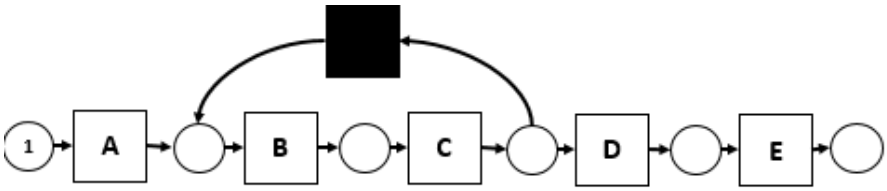

Figure. 5 Result of redo condition by Alpha# and Alpha$

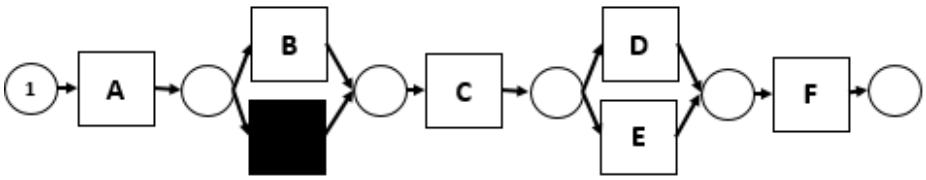

Figure. 6 Result of skip condition by Alpha# and Alpha$

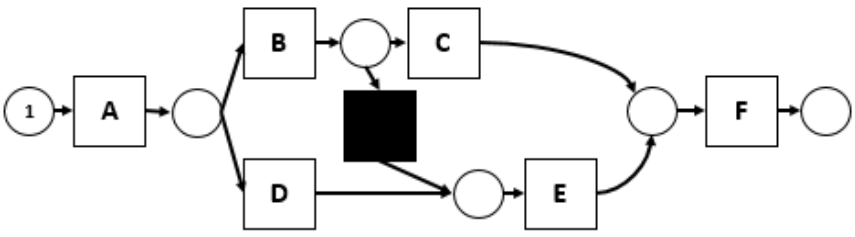

Figure. 7 Result of switch condition by Alpha# and Alpha$

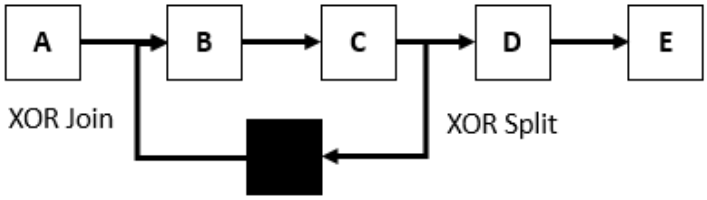

Figure. 8 Result of redo condition by CHMM-IT and CHMM-NCIT

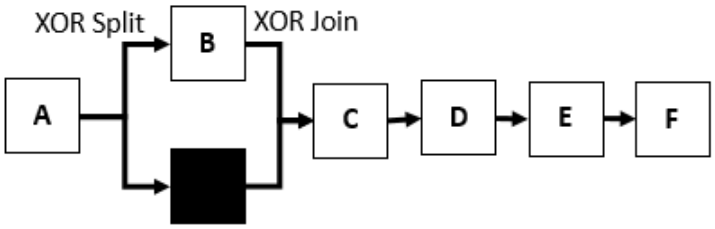

Figure. 9 Result of skip condition by CHMM-IT and CHMM-NCIT

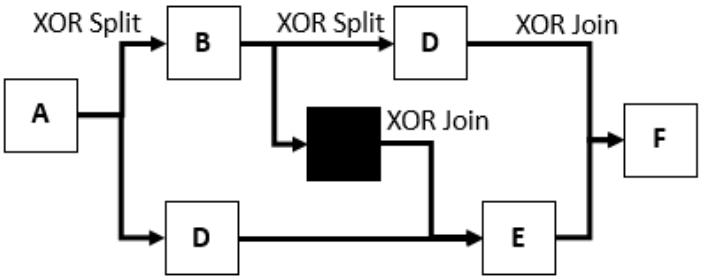

Figure. 10 Result of switch condition by CHMM-IT and CHMM-NCIT

Table 4. Fitness values of results constructed by different algorithms and time complexity of algorithms

Algorithms	Condition			Time Complexity
	Redo	Skip	Switch	
Graph-based	1.0	1.0	1.0	$O(n^2)$
Alpha#	1.0	1.0	1.0	$O(n^4)$
Alpha$	1.0	1.0	1.0	$O(n^4)$
CHMM-IT	1.0	1.0	1.0	$O(n^3)$
CHMM-NCIT	1.0	1.0	1.0	$O(n^3)$

4. Result and analysis

4.1 Business process model

This evaluation uses simulation event logs that describe skip condition, switch condition and redo condition. All of the event logs have 60 cases, wherein a redo-condition event log has three traces, a skip-condition event log has four traces, and a switch-condition event log has three traces. Traces of redo-condition event log are ABCDE, ABCBCDE, ABCBCBCBCDE. Traces of skip-condition event log are ACDF, ABCDF, ACEF, ABCEF. Traces of switch-condition event log are ABCF, ABEF, ADEF. All of the event logs are processed by two methods, i.e. graph-based algorithm and Alpha#.

First, the graph-based algorithm converts the event log in CSV format to a graph-database. Then, graph-based algorithm determines parallel relationships and invisible tasks in the graph-database for obtaining a process model containing invisible tasks. In this evaluation, there are three process models that are obtained by graph-based algorithm. Process models of Redo Condition, Skip Condition, and Switch Condition by graph-based algorithm are shown in Figure. 2, 3, and 4.

A process model of Figure. 2 has an invisible task between activity B and activity C. Invisible task is depicted between activity A and activity C in a process model of Figure. 3. The invisible task accommodates a skip condition of activity B. Based on the traces, a skip condition of activity B is discovered because several traces do not execute directly activity C after activity C, but execute activity B after activity A and before activity C. Then, invisible task in Figure. 4 is depicted between activity B and E to describe a switch condition.

After the process models of the graph-based algorithm were obtained, the models of Alpha#, Alpha$, CHMM-NCIT, and CHMM-IT were obtained. Figure. 5 and 8 show the result of Redo

condition, Figure. 6 and 9 show the result of Skip condition, and Figure. 7 and 10 show the result of Switch condition. Same as the results of the graph-based algorithm, an invisible task in both of Figure. 5 and 8 occurs between activity B and activity C, an invisible task in both of Figure. 6 and 9 occurs between activity A and activity C to depict skip condition of activity B and an invisible task in both of Figure. 7 and 10 occurs between activity B and E to depict switch condition.

4.2 Analysis

This evaluation compares the results of four different algorithms based on fitness and time complexity. The compared methods are the graph-based algorithm with Alpha# algorithm, Alpha$ algorithm, CHMM-IT algorithm, and CHMM-NCIT algorithms. The fitness values produced by these four algorithms were counted (see Table 4).

Based on the traces of the event log to build a model in redo condition, Figure. 2, 5 and 7 depict all those traces. Because of this condition, the fitness values of the graph-based algorithm and Alpha# are 1.0. Models that are established by all algorithms in switch condition and skip condition also depict all traces. It caused the fitness values of those conditions are 1.0. The values can be seen in Table 4. These high fitness values prove that graph-based algorithms can model all of the processes from the event log in a process model.

The second evaluation concerned the time complexity of each method. The time complexity of each algorithm is shown in Table 4. The graph-based algorithm gets $O(n^2)$. The detailed time complexity of graph-based algorithm is $O(n^2)$ for converting event log into a graph-database, $O(n^2)$ for executing exclusive choice until structured synchronizing merge patterns, and $O(n^2)$ for determining invisible tasks.

As the comparison algorithm, both of Alpha# and Alpha$ consist of ten steps, wherein the first step needs O(n) time complexity, the second step needs $O(n^4)$, the third step needs $O(n^2)$ time complexity, the fourth step needs $O(n^2)$ time complexity, the fifth needs $O(n^2)$ time complexity, and the last step need O(n) time complexity. Because of that, Alpha# and Alpha$ spends $O(n^4)$. Then, both of CHMM-NCIT and CHMM-IT have several steps. The first step needs $O(n^2)$ time complexity, the second step needs $O(n^3)$ time complexity, and the last step needs $O(n^2)$ time complexity. Overall, CHMM-NCIT and CHMM-IT spend $O(n^3)$.

Based on Table 4, the graph-based algorithm has the lowest time complexity. This is because the graph-based algorithm already stores the relations between activities before executing the rules for determining invisible tasks.

5. Conclusion

Graph-based algorithm models business processes containing invisible tasks based on an event log. Graph-based algorithm uses graph-database for storing activities and their relationships based on the log.

Firstly, graph-based algorithm converts event log in CSV format into a graph-database based on the event log. Next, the graph database is improved by adding invisible tasks and operators of parallel relations to construct a process model containing the invisible tasks. The process model that contains invisible tasks is the final result of the graph-based algorithm.

The results from the graph-based algorithm and from other existing algorithms were compared. The experiment conducted in this research showed that they all had high fitness. However, the graph-based algorithm is the most efficient method as proven by the time complexity of the graph-based algorithm ($O(n^2)$), while both of Alpha# and Alpha# have a time complexity of $O(n^4)$ and both of CHMM-NCIT and CHMM-IT have a time complexity of $O(n^3)$.

The graph-based algorithm does not consider anomalies in the event log. For future research, this method can be developed to consider anomalies and this method is tested in a large-scale event log.

Acknowledgments

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] K. R. Sungkono and R. Sarno, "CHMM for discovering intentional process model from event logs by considering sequence of activities", In: *Proc. of 2017 4th International Conference on Electrical Engineering, Computer Science and Informatics*, pp. 1–6, 2017.

[2] 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.

[3] 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*, Vol. 14, No. 4, 2018.

[4] K. R. Sungkono, R. Sarno, and N. F. Ariyani, "Refining business process ontology model with invisible prime tasks using SWRL rules", In: *Proc. of 2017 11th International Conference on Information Communication Technology and System*, pp. 215–220, 2017.

[5] R. Sarno and K. R. Sungkono, "Coupled Hidden Markov Model for Process Mining of Invisible Prime Tasks", *International Review on Computers and Software*, Vol. 11, No. 6, pp. 539–547, 2016.

[6] D. Rahmawati, M. A. Yaqin, and R. Sarno, "Fraud detection on event logs of goods and services procurement business process using Heuristics Miner algorithm", In: *Proc. of 2016 International Conference on Information Communication Technology and Systems*, pp. 249–254, 2016.

[7] K. R. Sungkono and R. Sarno, "Patterns of fraud detection using coupled Hidden Markov Model", In: Proc. of *2017 3rd International Conference on Science in Information Technology*, pp. 235–240, 2017.

[8] T. Erdogan and A. Tarhan, "Process Mining for Healthcare Process Analytics", In: *Proc. of Software Measurement and the International Conference on Software Process and Product Measurement*, pp. 125–130, 2016.

[9] 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.

[10] 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, pp. 109–125, 2016.

[11] Z. Besri and A. Boulmakoul, "Framework for organizational structure re-design by assessing logistics business processes in harbor container terminals", *Transportation Research Procedia*, Vol. 22, pp. 164–173, 2017.

[12] J. Webber and I. Robinson, *A programmatic introduction to neo4j*. Addison-Wesley Professional, 2018.

[13] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An Evolving Query Language for Property Graphs", In: *Proc. of ACM SIGMOD International Conference on Management of Data*, 2018.

[14] Meiliana, S. Karim, S. Liawatimena, A. Trisetyarso, B. S. Abbas, and W. Suparta, "Automating functional and structural software size measurement based on XML structure of UML sequence diagram", In: *Proc. of 2017 IEEE International Conference on Cybernetics and Computational Intelligence*, pp. 24–28, 2017.

[15] W. M. P. van der Aalst, A. Adriansyah, and B. F. Van Dongen, "Causal nets: a modeling language tailored towards process discovery", In: *Proc. of International Conference on Concurrency Theory*, pp. 28–42, 2011.

[16] A. A. Kalenkova, W. M. P. van der Aalst, I. A. Lomazova, and V. A. Rubin, "Process mining using BPMN: relating event logs and process models", *Software & Systems Modeling*, Vol. 16, No. 4, pp. 1019–1048, 2017.

[17] J. Dai, G. Su, Y. Sun, S. Ye, P. Liao, and Y. Sun, "Application of advanced Petri net in personalized learning", In: *Proc. of the 9th International Conference on E-Education, E-Business, E-Management and E-Learning*, pp. 1–6, 2018.

[18] J. Joishi and A. Sureka, "Graph or Relational Databases: A Speed Comparison for Process Mining Algorithm", *arXiv preprint arXiv:1701.00072*, pp. 1–22, 2016.

[19] R. Sarno, W. A. Wibowo, F. Haryadita, Y. A. Effendi, and K. R. Sungkono, "Determining Process Model Using Time-Based Process Mining and Control-Flow Pattern", *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, Vol. 14, No. 1, pp. 349–360, 2016.