Improving Efficiency for Discovering Business Processes Containing Invisible tasks in Non-free Choice

Riyanarto Sarno (riyanarto@if.its.ac.id)
Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia
Kelly Rossa Sungkono
Institut Teknologi Sepuluh Nopember https://orcid.org/0000-0003-3030-3566
Muhammad Taufiqulsa'di
Institut Teknologi Sepuluh Nopember
Hendra Darmawan
Institut Teknologi Sepuluh Nopember
Achmad Fahmi
Universitas Airlangga
Kuwat Triyana
Universitas Gadjah Mada

Research

Keywords: business process, graph-database, invisible task, non-free-choice, process discovery

Posted Date: September 16th, 2020

DOI: https://doi.org/10.21203/rs.3.rs-71558/v1

Version of Record: A version of this preprint was published at Journal of Big Data on August 24th, 2021.
See the published version at https://doi.org/10.1186/s40537-021-00487-x.

Improving Efficiency for Discovering Business Processes Containing Invisible tasks in Non-free Choice

Riyanarto Sarno[1*], Kelly Rossa Sungkono[1], Muhammad Taufiqulsa'di[1], Hendra Darmawan[1], Achmad Fahmi[2,3], Kuwat Triyana[4]
*Corresponding authors' email: riyanarto@if.its.ac.id
[1]Department of Informatics Engineering, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia
[2]Department of Neurosurgery, Faculty of Medicine, Universitas Airlangga, Surabaya, Indonesia
[3]Dr Soetomo General Academic Hospital, Medical Center. Indonesia
[4]Department of Physics, Universitas Gadjah Mada

Abstract

Process discovery helps companies to automatically discover their existing business processes based on the huge, stored event log. The algorithms of process discovery have been developed rapidly to discover several types of relations, i.e., choice relations, non-free choice relations with invisible tasks. Invisible tasks in non-free choice, introduced by $\alpha^{\$}$ method, is a type of relation that combines the non-free choice and the invisible task. $\alpha^{\$}$ proposed rules of ordering relations of two activities for determining invisible tasks in non-free choice. The event log records sequences of activities, so the rules of $\alpha^{\$}$ check the combination of invisible task within non-free choice. The checking processes is time consuming, and results in high computing times of $\alpha^{\$}$. This research proposes Graph-based Invisible Task (GIT) method to discover efficiently invisible tasks in non-free choice. GIT method develops sequences of business activities as graphs and determines rules to discover invisible tasks in non-free choice based on relations of the graphs. The analysis of the graph relations by rules of GIT is more efficient than the iterative process of checking combined activities by $\alpha^{\$}$. This research measures the time efficiency of storing the event log and discovering a process model to evaluate GIT algorithm. Storing a streaming event log in a graph-database has the lowest computing time than storing in other databases, i.e., SQL and MongoDB. Discovering a process model by GIT algorithm has less time complexity than that by $\alpha^{\$}$, wherein GIT obtains $O(n^3)$ and $\alpha^{\$}$ obtains $O(n^4)$. In terms of computing time, GIT algorithm is 0.89 faster on batch event log and 0.85 seconds faster on streaming event log than $\alpha^{\$}$. Those results of the evaluation show a significant improvement of GIT method in term of time efficiency.

Keywords: business process, graph-database, invisible task, non-free-choice, process discovery

1. Introduction

The rapid development of information systems leads to a growing amount of information [1] that obtains massive stored data [2,3]. In the presence of various types of data, events that are happened in the systems are stored in so-called event log [4]. Monitoring business processes from a massive event log brings a challenge related to Big Data. Process mining is a discipline of gathering the event log and processing the log into a process model for monitoring, including capturing anomalies [5,6] or bottlenecks [7]. The technique of constructing a process model by process mining is called process discovery.

The algorithms of process discovery have been developed rapidly to discover several types of relations [8–13]. Table 1 describes the existing algorithms which can discover types of relations depicted by the columns. The tick signs explain that the algorithms can discover the related relations depicted by the columns. $\alpha^{\$}$ [14] is the development of α algorithm that discovers a new issue, i.e., invisible tasks in non-free choice.

Invisible tasks in non-free choice occur when choice activities that involve invisible tasks are not a choice-free relation (their execution depends on previous activities. In importing Port Container Handling processes, cranes move containers to the truck based on the types of containers. For the dry containers, cranes directly move without doing other previous activities. If there is no invisible task, the process of dry container transfer indicates a skip sequence, one of the anomalies of the business process. If there is no non-free choice, cranes can do wrong activities against the dry container. Another example is the process of selling medicines. If the customer needs a tax invoice of the purchase, the store prints the

invoice before ends the transaction. An invisible task is added before the activity of the end transaction, so non-free choice relations can be added to link the event of payment without tax and an invisible task and connect the event of payment with tax and an event of tax printing.

Table 1. Algorithms of Process Discovery

Algorithms	**Types of Relations**						
	Sequence	**Looping**	**XOR, AND**	**OR**	**Invisible Prime Task (Skip, Redo, Switch)**	***Non-Free Choice***	***Invisible Prime Task in Non-Free Choice***
Heuristic Miner [8]	√	√	√				
Inductive Miner [9]	√	√	√	√			
RPST [10]	√	√	√				
Fodina [11]	√	√	√				
α [15]	√		√				
α^{+} [16]	√	√	√				
α^{++} [14]	√	√	√			√	
$\alpha^{\#}$ [17]	√	√	√		√		
$\alpha^{\$}$ [4]	√	√	√		√	√	√
HMM-Parallel Tasks [18]	√	√	√	√			
CHMM-Invisible Tasks [19]	√	√	√	√	√		

$\alpha^{\$}$ stores the dependencies of activities in the form of pairs of activities and collects traces of activities based on the event log and combines α^{++} rules and $\alpha^{\#}$ rules to construct invisible task in non-free choice. $\alpha^{\$}$ has several rules of ordering relations. The rules check all pairs of activities because there is no information of relations of those activities. The checking process produces high computing time.

This research proposes GIT, the expansion of Graph-based Invisible Task, to effectively construct invisible tasks in non-free choice. This research utilizes graph capabilities of storing relations in forming rules of invisible tasks in non-free choice. Storing of direct relations can simplify rules of process discovery because several types of relations are detected based on other discovered relations. For example, if $\alpha^{\$}$ checks all pairs of activities to determine non-free-choice, the GIT algorithm only checks choice relations because non-free-choice is constructed based on choice relations. Next, if $\alpha^{\$}$ check all pairs of activities to determine XOR relations as choice relations, the GIT algorithm establishes an XOR relation if the next activity of activity has only one next relation. This research hypothesizes that storing relations can accelerate the computing time for discovery.

Existing graph-based process discovery methods [13,20–22] do not directly interact with the system, so those methods import the event log of the system and convert the log to graph-database. The conversion process causes ineffective computing time. Besides creating

new rules, the GIT algorithm also integrates graph-database and the system platform, i.e., Laravel. The aim of integration is storing all logs of the system directly in the graph database. The integration can reduce the computing time.

This research is evaluated in medical store processes that already implemented an enterprise resource planning (ERP) system. The medical store processes, especially payment processes, contain the condition of invisible tasks in non-free choice. The experiment is measured by several aspects, which are:

(1) the quality model of GIT and $\alpha^{\$}$ based on fitness and precision [23],
(2) the time complexity and computing time of GIT and $\alpha^{\$}$, and
(3) the computing time of storing the event log by using graph-database, SQL, or MongoDB in streaming event log and batch event log.

2. Literature Review

2.1 Event Log

An event log consists of several *cases,* which identify the executed processes. Each *case* has several *attributes,* such as identification of case (CaseId), name of activities (Activity), the execution time of the activities (Timestamp), and actors who carried out activities (Resource) [24]. For example, an event log $\mathcal{E}$ = { {C1, A, 2018-08-05-15:54, Admin}, {C1, B, 2018-08-05-16:54, Customer}, {C1, D, 2018-08-05-16:54, Customer}, {C2, A, 2018-08-06-15:54, Admin}, {C2, B, 2018-08-06-16:54, Customer}, {C2, D, 2018-08-06-16:54, Customer}, {C3, A , 2018-08-07-15:54, Admin}, {C3, C , 2018-08-07-16:54, Customer}, {C3, D, 2018-08-07-16:54, Customer}. The event log $\mathcal{E}$ has three cases with id C1, C2, and C3. Both C1 and C2 have the same sequence of activities, which is $A \rightarrow B \rightarrow D$ and C3 has $A \rightarrow C \rightarrow D$.

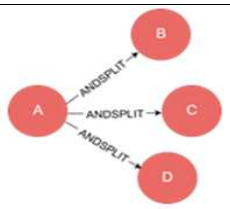
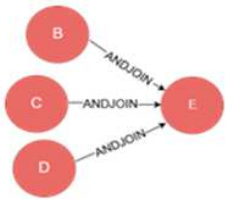
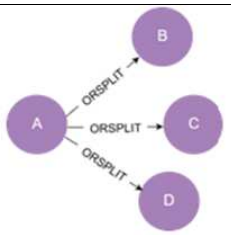
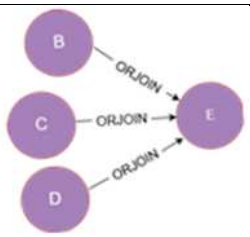
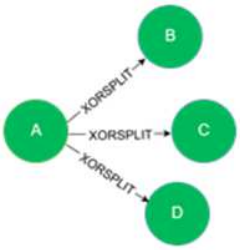
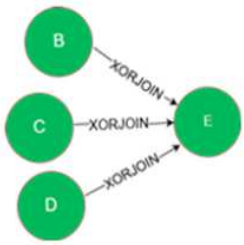
An event log contains several *traces,* which are unique sequences of activities. The event log $\mathcal{E}$ has $(A \rightarrow B \rightarrow D)_2$ and $(A \rightarrow C \rightarrow D)_1$, where 2 and 1 identify the occurrence numbers. There are two traces of $\mathcal{E}$, i.e., $(A \rightarrow B \rightarrow D)$ and $(A \rightarrow C \rightarrow D)$.

2.2 Control-flow Patterns

The process model consists of two types of relations, namely sequence relations and parallel relations. Sequence relations are used to describe processes that run straight in sequence or sequentially, whereas the parallel relations are used to describe a branched process. There are three types of control-flow patterns [25] to represent parallel relations, namely AND, OR, and XOR [26],[18]. Table **2** shows examples of relations and differences between control-flow patterns in graph models.

Table 2.Control-flow Patterns in Process Model

Relations Types	The example of cases	The Representation of Graph Model
Sequence	[A, B, C, D] [A, B, C, D] [A, B, C, D]	

AND Split | AND Join	[A, B, C, D, E] [A, B, D, C, E] [A, C, B, D, E] [A, C, D, B, E] [A, D, B, C, E] [A, D, C, B, E]		
OR Split | OR Join	[A, B, C, E] [A, B, D, E] [A, C, B, E] [A, C, D, E] [A, D, B, E] [A, D, C, E]		
XOR Split | XOR Join	[A, B, E] [A, C, E] [A, D, E] [A, B, E] [A, C, E] [A, D, E] [A, B, E]		

2.3 Invisible Task

An invisible task is a task that does not exist in the event log but should exist in process discovery and usually serve as a routing purpose [17]. As seen in Figure 1, invisible task discovery is divided into three types, which skip invisible task discovery, redo invisible task discovery, and switch invisible task discovery. Skip invisible task happens when an activity or several activities are skipped in the process. Redo invisible task happens when several activities are executed multiple times in the process. Lastly, switch invisible task happens when there is a stack of parallel relations, such as XOR Split relation and XOR Join relation, AND Split relation and AND Join relation, or OR Split relation and OR Join relation.

2.4 Non-free Choice

The last step is non-free-choice discovery. Non-free-choice relation connects an activity in one choice relation with another activity in the next choice relation. This relation shows that the next activity of choice relation cannot be freely chosen and is influenced by the previous choice relation activity. Figure 2 is an example of a non-free choice relation. As depicted in Figure 2, node F and node E cannot be freely chosen, even though the relations between D to F and D to E are XORSPLIT. Choice of node F can only be taken if node C is executed and vice versa for node E. Non-free-choice relation is represented by NON FREE CHOICE relation.

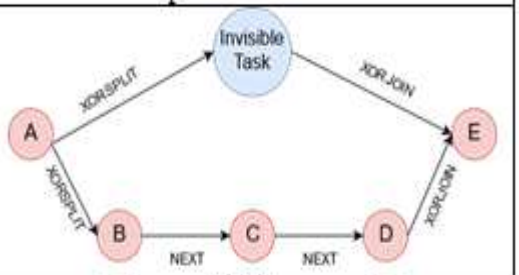
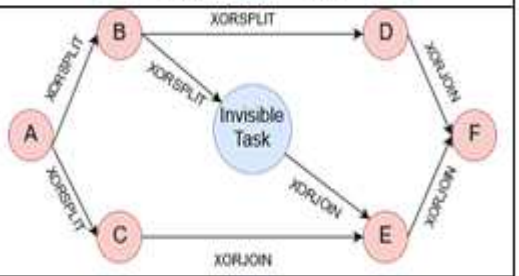
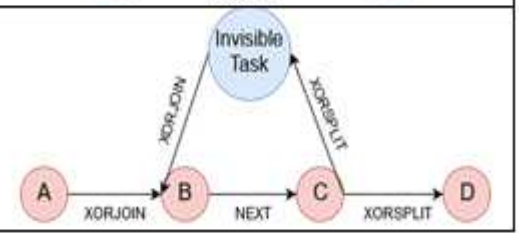
Conditions	Traces	Graph Process Models
Skip	[A, B, C, D, E] [A, E]	
Switch	[A, B, D, E] [A, B, F, E] [A, C, D, E]	
Redo	[A, B, C, D] [A, B, C, B, C, D]	

Figure 1. The Types of Invisible Tasks

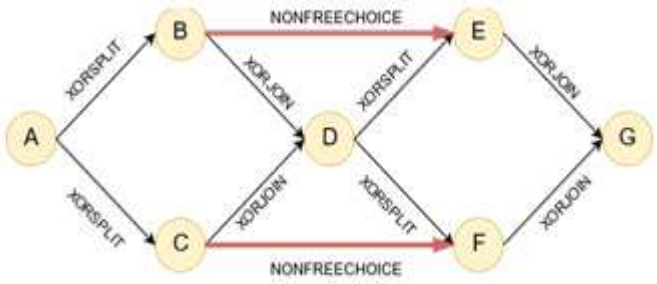
Condition	Traces	Graph Process Model
Non-free-choice	[A, B, D, E, G] [A, C, D, F, G]	

Figure 2. Non-free-choice

2.5 Invisible Task in Non-free Choice

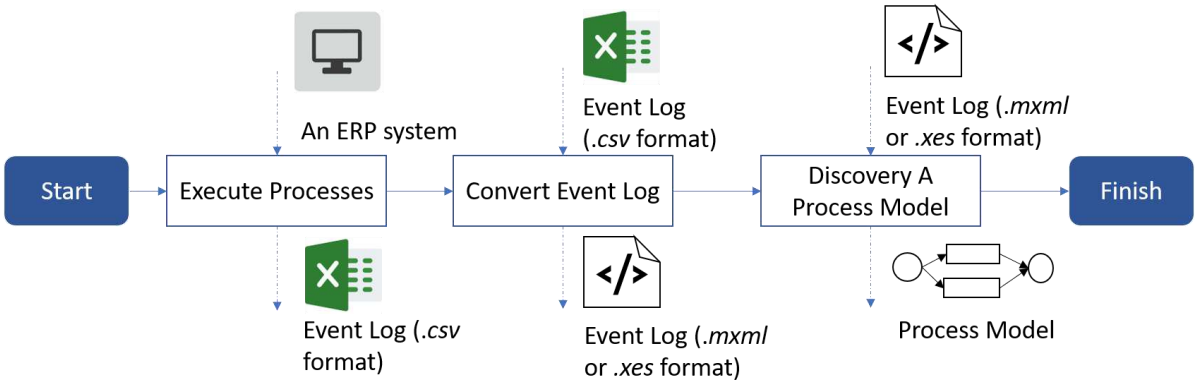
As mentioned in Section 2.3, an invisible task is a task that does not become observable in the event log. In Section 2.4, this research has discussed non-free choice as a relation of the choice construct (XOR). This relation shows that the next activity of choice relation cannot be freely chosen. The invisible task in non-free choice means that a process model contains both invisible task and non-free choice.

In this research, the invisible task in non-free choice happens in the payment process. The customer from a corporation usually needs proof of tax invoice when buying high prices of medical equipment. To accommodate the request, the system will give the proof of tax invoice to the customer who bought medical equipment at high prices. Because the proof of tax invoice is not always issued in every process, so the process model needs an invisible task: skip condition. Then, the relation between invoice issuance and purchasing is called non-free

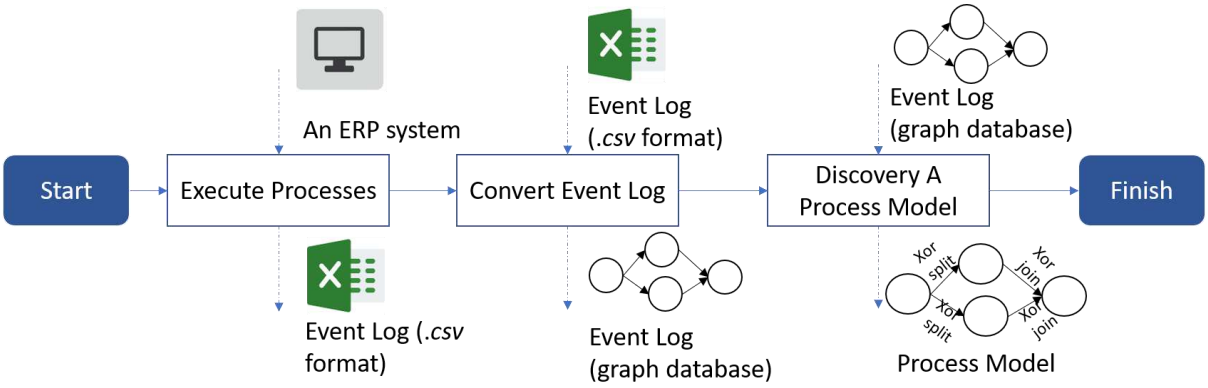
choice. Formation of an invisible task and non-free choice at once is called an invisible task in non-free choice.

3. Methodology

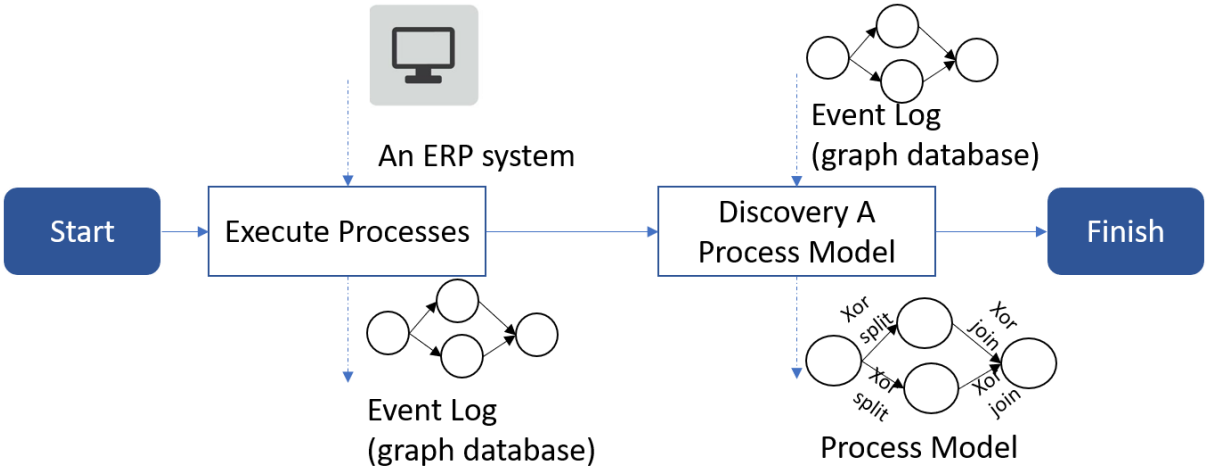
This research proposes an event log storage algorithm into a graph-database for direct process discovery without exporting event logs from the ERP database and transforming .csv file into *.mxml* or *.xes*. The storage algorithm will make the discovery process more efficient than previous methods. Figure 3 shows a comparison of the previous method with the proposed method. The first one shows the steps of process discovery proposed by Aalst [15,27]. The event log needs to be extracted from the database in the form of .csv. Then, the .csv file needs to be converted into .mxml or .xes by using Disco Tool. .Mxml file is supported by ProM 5, and .xes file is supported by ProM 6. Then, the .mxml or .xes file is imported to ProM Tool to conduct process discovery. The second one shows process discovery steps using graph-database Neo4J proposed by Darmawan, et al [28]. The event log needs to be extracted from the database in the form of .csv. Then, the .csv file is imported to Neo4J to conduct process discovery. The last one shows the steps for discovering the process model of the proposed method. By integrating Neo4J and ERP using Laravel, the event log stored in the database ERP is directly processed to execute process discovery.



(a)



(b)



(c)

Figure 3.The Comparison Process Discovery Methods (a) The Pioneer of Process Discovery Method by Aalst, et al [15,27], (b) The Process Discovery using Graph-database [28], and (c) GIT Process Discovery Model

3.1 The Integration Stage

The integration stage is connecting the ERP system with Neo4J, a graph database platform, in storing an event log of the system directly in the form of a graph database. The used ERP system in this research is built by Laravel. This research uses *ClientBuilderfrom the GraphAwareLibrary* to start the integration and *ClientBuilder* to make a connection between the system and the graph database.

3.2 Process Discovery

a. The Algorithm of Logging Stage

The logging stage in a graph-database starts when the user logs into the ERP system until the user logs out. The recording is done using the algorithm in Table **3**. First, the algorithm checks the log that has been saved in Neo4J. If the last log is found in the graph-database, the login activity is entered in the last case + 1. However, if the last log is not found, then the login activity is recorded as the first case. The Neo4J connection is made using the GraphAwarelibrary by connecting ClientBuilder from the GraphAware. At that moment, the current case will be included in the session to be used in later stages.

b. Constructing Sequence Relation and Invisible Task

Table **4** is used to discover sequence relations of the event log. GIT stores all variations of activities as nodes in the graph database. Then, GIT creates the sequence relations of those nodes based on activities of the event log (called an *event*). A sequence relation is discovered between an *event* and its next *event* with the same case id.

Table 3. The Algorithm of Logging Stage

1	last_log_case = DB::table('event_logs')->latest('case')->first();
2	if last_log_case is empty then
3	set case_id to 1
4	else
5	set case_id to last_log_case+1
6	case_id++
7	endif
8	set case_id, activity, user_id, company_id, value, division, and timestamp
9	create event log
10	Session(['case'=>case_id])
11	Store the event log into graph_database
12	caseData <- MERGE activity and value
13	modelBusiness <- MERGE activity and value

Table 4. The Algorithm of GIT for Discovering Sequence Relations

Algorithm	**for** i = 0 **until** *count_event-1* **do**
	if event[i]$_{CaseID}$ = event[i + 1]$_{CaseID}$ **then**
	for j = 0 **until** *count_variants_activities* **do**
	if event[i]$_{Name}$ = nodeAsact[j]$_{Name}$ **then**
	a = nodeAsact[j]
	if event[i + 1]$_{Name}$ = nodeAsact[j]$_{Name}$ **then**
	b = nodeAsact[j]
	endif
	endfor
	[a]-[NEXT]->[b]
	endif
	endfor

Output	A graph model with sequential relations

where:

event	:	activity in the event log
nodeAsact	:	node of graph database as activity
count_event-1	:	the number of events minus 1 in the event log
count_variants_activities	:	the number of variaties of activities that are arranged into a process model in the graph database
event[i]$_{CaseID}$	:	CaseID of an *i*-order event
nodeAsact[j]$_{Name}$	:	Name of activity in the graph database

Cypher queries in Table **5** are a part of the GIT method. Based on Table **5**, the steps of detecting skip, switch and redo invisible tasks are same. GIT algorithm chooses relations of activities based on its rules and inserts invisible tasks in the middle of those relations. The algorithm is implemented before control-flow pattern discovery.

The next step is the discovery of the control-flow patterns, i.e., AND, OR, and XOR, as can be seen in Table **6**. Control-flow patterns, that is described in Table **6**, are used to determine types of parallel relations. Parallel relations can be divided into three types, which are XOR, AND, and OR. XOR relations only allows one activity to be executed in a process. AND relations allow all activities to be executed. Lastly, OR relations are relations between XOR and AND, and it only allows parallel activities that are not categorized into XOR and AND relations. Each relation is divided into two forms, which are split and join. A split

happens when there are multiple activities to be chosen. A join happens when multiple activities that can be chosen in split relations are closing into one or more activities.

Table 5. The Algorithm of GIT for Invisible Prime Task Discovery

Conditions	Algorithm

SKIP
REDO
SWITCH
	//discovering invisible task in sequence relations
	for i = 0 **until** *count_variants_activities* **do**
	n = nodeAsact[i]
	for a **in** ((n)-->(a)) **do**
	if size(n-->()) > size(()-->(n)) **and** size(()-->(a)) > size((a)-->())
	and not((a)-->(n)) and size((n)-->()) > 1 **then**
	(n)-[:INVISIBLE]->(a)
	endif
	endfor
	endfor
	
	//discovering invisible task in relations that contain split and join
	for i = 0 **until** *count_variants_activities* **do**
	if not((nodeAsact[i])-[:SPLIT]->()) **then**
	continue to loop
	endif
	n = nodeAsact[i]
	for a **in** ((n)-[:INVISIBLE]->(a)) **do**
	x = INVISIBLE TASK
	(n)-[:SPLIT]->(x)
	(x)-[:JOIN]->(a)
	endfor
	endfor

Output	A graph model with invisible task nodes

where:

x	:	node of graph database as invisible task
[:SPLIT]	:	a split relation of activities (XORSplit, ORSplit or ANDSplit)
[:JOIN]	:	a join relation of activities (XORJoin, ORJoin or ANDJoin)
[:INVISIBLE]	:	a temporary relation to determine the need for invisible tasks
size()	:	the number of occurrences of the specified relation in the graph database

Table 6. The Algorithm of GIT for Discovering Control-Flow Patterns

Types	Algorithm

XORSPLIT	**for** i = 0 **until** *count_variants_activities* **do**
	n = nodeAsact[i]
	if size((n)-->()) <= 1 **then**
	continue to loop
	endif
	for a **in** ((n)-->(a)) **do**
	if size((a)<--()) = 1 **then**
	(n)-[:XORSPLIT]->(a)
	endif
	endfor
	endfor

XORJOIN	**for** i = 0 **until** *count_variants_activities* **do**
	n = nodeAsact[i]
	if size(()-->(n)) <= 1 **then**
	continue to loop
	for a **in** ((a)-->(n)) **do**
	if size((a)-->()) = 1 **then**
	(a)-[:XORJOIN]->(n)
	endif
	endfor
	endfor

ANDSPLIT || AND JOIN || ORSPLIT || ORJOIN	**for** i = 0 **until** *count_variants_activities* **do**
	n = nodeAsact[i]
	if size(()-->(n)) > 1 **then**
	nodesBefore = 0
	for a **in** ((a)-->(n)) **do**
	for b **in** ((b)-->(n) and a != b) **do**
	nodesBefore ++
	endfor
	if nodesBefore-1 == size(()-->(n)) **then**
	(a)-[:ANDJOIN]->(n)
	else then
	(a)-[:ORJOIN]->(n)
	endif
	endfor
	continue to loop
	endif
	if size(n)-->(a)) > 1 **then**
	nodesAfter = 0
	for a **in** ((n)-->(a)) **do**
	for b **in** ((n)-->(b) and a != b) **do**
	nodesAfter++
	endfor
	if nodesAfter-1 == size((n)-->()) **then**
	(n)-[:ANDSPLIT]->(a)
	else then
	(n)-[:ORSPLIT]->(a)
	endif
	endfor
	continue to loop
	endif
	endfor

where:

n	: node of the graph database as activity
a, b	: other node in the model
nodesAfter	: all nodes after node n
nodesBefore	: all nodes before node n

The last step is the discovery of invisible tasks in non-free choice. Non-free-choice relation connects an activity in one choice relation with another activity in the next choice relation. This relation shows that the next activity of choice relation cannot be freely chosen

and is influenced by the activity of the previous choice relation. Figure **2** is an example of non-free-choice relation and discovery by using GIT. As depicted in Figure **2**, node F and node E cannot be freely chosen even though the relation between D to F and D to E are XORSPLIT; instead, the choice of node F can only be taken if the previous activities taken is node C and choice of node E can only be taken if the previous activities taken is node B. Non-free-choice relation is represented by NONFREECHOICE relation. Using the algorithm in Table **7**, GIT algorithm finds some nodes with outgoing relation of XORJOIN, some nodes with ingoing relation of XORSPLIT, detect invisible task, and lastly match them against some nodes inside activity label which has the same name.

Table 7. The Algorithm of GIT for Discovering Invisible Task in Non-free Choice

Types	Algorithm

Invisible Task in Non-free Choice	**for** i = 0 **until** *count_variants_activities* **do**
	n = nodeAsact[i]
	if not((n)-[:XORSPLIT]->() and ()-[:XORJOIN]->(n)) **then**
	continue to loop
	endif
	path = 0
	for a,b **in** (a)-[xorjoin]->(n)-[xorsplit]->(b) **do**
	if b == INVISIBLE TASK **then**
	b = (b)-->(x)
	return x
	endif
	for j = 0 **until** *count_event-1* **do**
	if (a:case[j])--->(n:case[j])-->(x: case[j]) and x != b **then**
	path++
	if path > 0 **then**
	break
	endif
	endif
	if path == 1 **then**
	(a)-[:NON_FREE_CHOICE]->(b)
	endif
	endfor
	endfor

where:

n : node of the graph database as activity
a, b : other node in the model
path : all path from b to a and connected by n on case[j]

4. Results and Discussion

This research evaluates a medical store that already implemented ERP. The medical processes consist of 45 names of activities, 86 traces, and 906 cases. The result of GIT algorithm is shown in Figure **4**. The full name of activities presented in Figure **4** is shown in Table **8**. In Figure **4**, the non-free choice relations occur between activity *Direct Payment* and an invisible task and between activity *Payment with E-facture* and *Print Tax Invoice.* Thus, this research is verified to be able to detect invisible task in non-free choice.

GIT algorithm was tested by comparing it with other algorithms: $\alpha^{\$}$ and α^{++}, in the value of fitness [29], precision [30], and complexity algorithm. The fitness value is obtained by counting the number of cases described in the model divided by the total cases in the event log. The precision value is obtained by counting the number of traces discovered in the model divided by the number of traces present in the event log.

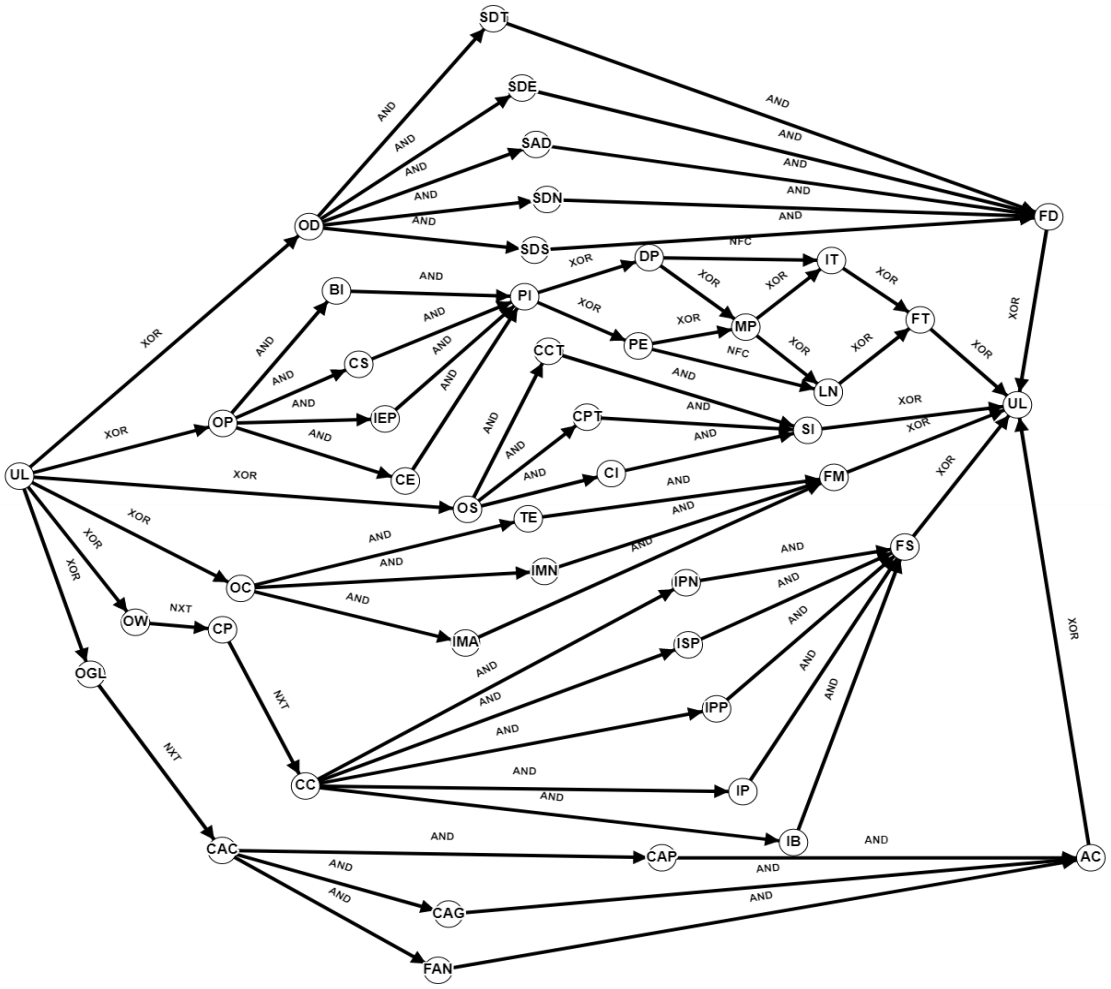


Figure 4. The Process Model by GIT algorithm

Table 8. List of Activity Names Initialized in Figure 4

No	Alias	Real Task Name	No	Alias	Real Task Name
1	UL	User Login	24	BI	Buy Items
2	OD	Open Discount	25	IEP	Input Expedition Price
3	SDE	Set Discount End Time	26	PI	Purchasing Items
4	SDN	Set Discount Name	27	PE	Payment with E-facture
5	SDT	Set Discount Type	28	MP	Make Payment
6	SDS	Set Discount Start Time	29	LN	Print Tax Invoice
7	SAD	Set Amount of Discount	30	FT	Finish Transaction
8	FD	Finish Discount	31	DP	Direct Payment
9	UL	User Logout	32	OC	Open Capital
10	OS	Open Selling	33	IMN	Input Modal Name
11	CI	Choose Item	34	IMA	Input Modal Amount
12	CPT	ChoosePayment Type	35	TE	Type Explanation

13	CCT	Choose Customer Type	36	FM	Finish Modal
14	SI	Sell Item	37	OW	Open Warehouse
15	OGL	Open General Ledger	38	CP	Choose Product
16	CAC	Choose Account Code	39	CC	Choose Category
17	CAG	Choose Account Group	40	IPN	Input Product Name
18	CAP	Choose Account Parent	41	IP	Insert Picture
19	FAN	Fill Account Name	42	IB	Input Barcode
20	AA	Add Account	43	IPP	Input Purchase Price
21	OP	Open Purchasing	44	ISP	Input Selling Price
22	CS	Choose supplier	45	FP	Finish Product
23	CE	Choose Expedition			

Figure **5** shows the comparison of results from the previous methods with the method proposed in this study. The α^{++} algorithm obtains 0.996 fitness and 0.964 precision because this algorithm cannot discover the invisible task in the model. Meanwhile, $\alpha^{\$}$ algorithm and GIT algorithm obtained fitness and precision results of 1. In addition to comparing the results of fitness and precision, the time complexity of the algorithms is measured. The time complexity of $\alpha^{\$}$ is $O(n^4)$ [31], while the time complexity of GIT is $O(n^3)$, with the explanation steps as listed in

Table 9.

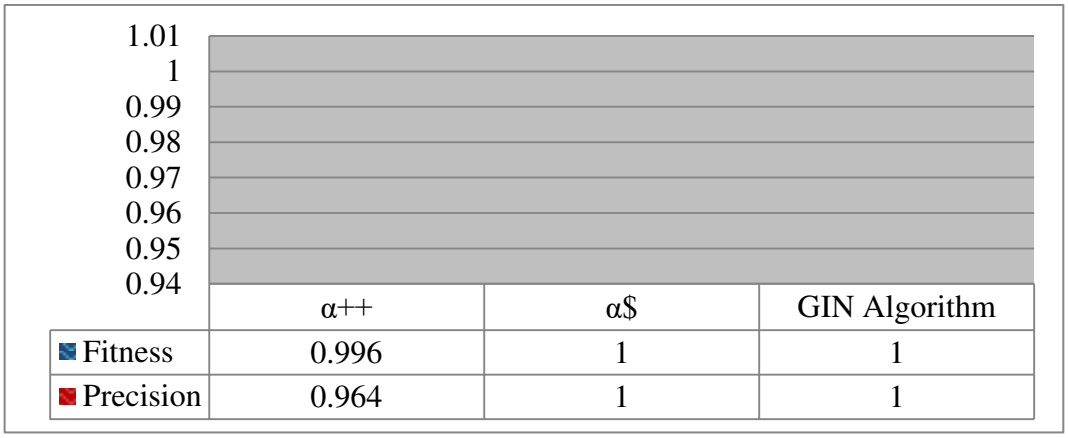


Figure 5. Result Comparison between the Previous Methods and the Proposed Method

Table 9. Comparison of Algorithm Complexity

GIT Algorithm	
Steps	**Complexity**
Storing and Creating Sequence Relations of Activities (Table **4**)	$O(n^2)$
Creating Invisible Prime Tasks (Table **5**)	$O(n^2)$
Creating parallel relations, i.e., XOR, OR, and AND (Table **6**)	$O(n^3)$
Creating parallel relations, i.e., XOR, OR, and AND (Table **7**)	$O(n^3)$
Total Complexity	$O(n^3)$

Table 10. Result of Computing Time

Type of Event Log	Used Cases	Total Case	Total Traces	Total Events	Computing Time of Storing (s)			Computing Time of Discovery (s)	
					Graph-Database	MySQL-CSV	MongoDB-CSV	GIT	$\alpha^{\$}$
Batch	UC1	120	6	840	42	32.9	0.06	2	2.8
	UC2	120	12	840	45	30.5	0.034	2	2.8
	UC3	120	63	1010	52	40.2	0.098	2	3.1
	UC4	120	86	1078	57	41.5	0.15	3	3.7
		240	86	2153	134	71.4	0.085	3	3.9
		480	86	4305	551	94.5	0.405	3	4.1
		960	86	8601	2049	281.8	0.767	3	3.8
	Mean				**418.57**	**84.69**	**0.23**	**2.57**	**3.46**
Streaming	UC1	6	6	42	(Mean) 0.085	(Mean) 0.126 s	(Mean) 0.007	2	2.8
	UC2	12	12	84	(Mean) 0.005	(Mean) 0.147	(Mean) 0.006	2	2.8
	UC3	66	66	301	(Mean) 0.001	(Mean) 0.152	(Mean) 0.006	2	3.1
	UC4	86	86	974	(Mean) 0.001	(Mean) 0.143	(Mean) 0.004	3	3.7
	Mean				**0.023**	**0.142**	**0.006**	**2.25**	**3.1**

where:

UC1 : Cases that contain processes starting from User Login activity to Open Selling activity

UC2 : Cases that contain processes starting from User Login activity to Open Selling and Open Capital activity

UC3 : Cases that contain processes starting from User Login activity to Open Selling, Open Capital, Open Discount and Open Warehouse activities

UC4 : All cases on the event log

The experiment is executed in Ryzen 2700U with 8GB RAM. The result is described in Table **10**. Based on Table **10**, while using a batch event log, the graph database has the highest storing time among other databases. This condition happens because in the insert bulk operation of all data in a graph database is entered almost the same time in one command, making the database rules must recheck the path for every row of the preceding merge clause.

In the first trial streaming event log, graph database stores slower than MongoDB. For the next trial, the graph database has the fastest computing time than the others. Based on the whole case, the graph database consumes 0.001 seconds on average, MySQL 0.143 seconds

on average, and MongoDB 0.004 seconds on average. Graph database caches plan that can store the rules beforehand, making the next operations performed faster. Because of the process of storing rules, the graph database needs high computing time in the first trial.

The experiment shows graph database is suitable to store streaming data, rather than batch data. Then, the average computing time of GIT algorithm is faster than $\alpha^{\$}$, where GIT is 0.89 seconds faster than $\alpha\$$ in the batch event log and 0.85 seconds faster than $\alpha^{\$}$ in streaming event log. It can be concluded that GIT algorithm is more efficient than $\alpha^{\$}$ in terms of process discovery.

5. Conclusion

This research proposes an algorithm called GIT algorithm, which utilizes a graph database to model invisible tasks in non-free choice efficiently. The capability of storing relations in a graph database can simplify the rules of process discovery because several relations of a process model can be constructed based on other discovered relations. GIT algorithm improves Graph-based Invisible Task by combining non-free choice rules with invisible task rules from Graph-based Invisible Task.

This research compares GIT algorithm to $\alpha^{\$}$ and α^{++} based on the qualities of their models. The chosen quality measurements are fitness and precision. The experiments show that the fitness and precision of α^{++} are 0.996 and 0.964. However, both of GIT and $\alpha^{\$}$ have high fitness and precision, i.e., 1. This research calculates computing time and time complexity of GIT and $\alpha^{\$}$ and computing time of graph database, MySQL, and MongoDB. Graph database has the highest computing time of storing a streaming event log than other databases. The complexity of $\alpha^{\$}$ is $O(n^4)$; meanwhile, the complexity of GIT is $O(n^3)$. The average computing time of GIT algorithm is faster than $\alpha^{\$}$, where GIT has is 0.89 seconds faster on batch event log and 0.85 seconds faster on streaming event log than $\alpha^{\$}$. It can be concluded that the process discovery of GIT algorithm is more efficient than that of $\alpha^{\$}$ algorithm, and the graph database is suitable on the streaming event log.

DECLARATIONS

Abbreviations

ERP : Enterprise Resource Planning
GIT : Graph-based Invisible Task

Availability of data and materials

The used raw dataset in this research is not publicly available. Readers can contact the corresponding author if they want to access the data

Competing interests

The authors declare that they have no competing interests.

Funding

This research is partially funded by Indonesian Ministry of Research and Technology/National Agency for Research and Innovation under PPKI program managed by Institut Teknologi Sepuluh Nopember.

Authors' contribution

RS formulated rules of GIT algorithm and checked the manuscript. KRS analyzed the efficiency of GIT algorithm and was a major contributor in writing the manuscript. MT and HD implemented the rules of GIT algorithm and performed the evaluation of GIT algorithm, AF describes the flow of medical store processes. KT checks the analysis of evaluation.

Acknowledgments

This research was funded by the Indonesian Ministry of Education and Culture, and the Indonesian Ministry of Research and Technology/National Agency for Research and Innovation under PPKI program managed by Institut Teknologi Sepuluh Nopember, and under RISPRO Invitation program managed by LPDP, and under World Class University Program managed by Institut Teknologi Bandung.

Authors' information
[1]Jalan Teknik Kimia – Gedung Departemen Teknik Informatika, Kampus Institut Teknologi Sepuluh Nopember, Jalan Raya ITS, Sukolilo, Surabaya 60111, Indonesia. [2]Kampus A Universitas Airlangga, Jalan Mayjen. Prof. Dr. Moestopo 47 Surabaya 60131, Indonesia. [3]Jalan Mayjen Prof. Dr. Moestopo 6-8, Airlangga, Surabaya 60286, Indonesia. [4]Jalan Grafika 2, Yogyakarta 55281, Indonesia.

REFERENCES

1. Mantzaris A V., Walker TG, Taylor CE, Ehling D. Adaptive network diagram constructions for representing big data event streams on monitoring dashboards. Journal of Big Data [Internet]. Springer International Publishing; 2019;6. Available from: https://doi.org/10.1186/s40537-019-0187-2
2. Ismail A, Truong HL, Kastner W. Manufacturing process data analysis pipelines: a requirements analysis and survey. Journal of Big Data [Internet]. Springer International Publishing; 2019;6:1–26. Available from: https://doi.org/10.1186/s40537-018-0162-3
3. Hasan MM, Popp J, Oláh J. Current landscape and influence of big data on finance. Journal of Big Data [Internet]. Springer International Publishing; 2020;7. Available from: https://doi.org/10.1186/s40537-020-00291-z
4. Guo Q, B LW, Wang J, Yan Z, Yu PS. Mining Invisible Tasks in Non-free-choice Constructs. International Conference on Business Process Management Springer. 2015. p. 109–10.
5. Sarno R, Sinaga F, Sungkono KR. Anomaly detection in business processes using process mining and fuzzy association rule learning. Journal of Big Data. Springer International Publishing; 2020;7:1–19.
6. Al Jallad K, Aljnidi M, Desouki MS. Anomaly detection optimization using big data and deep learning to reduce false-positive. Journal of Big Data [Internet]. 2020;7:68. Available from: https://doi.org/10.1186/s40537-020-00346-1
7. Eboreime EA, Idika O, Omitiran K, Eboreime O, Ibisomi L. Primary healthcare planning, bottleneck analysis and performance improvement: An evaluation of processes and outcomes in a Nigerian context. Evaluation and Program Planning [Internet]. 2019;77:101712. Available from: http://www.sciencedirect.com/science/article/pii/S014971891930182X
8. Weijters AJMM, Aalst WMP Van Der. Process Mining with the Heuristics Miner-algorithm. Technische Universiteit Eindhoven, Tech Rep WP. 2006;166:1–34.
9. Leemans SJJ, Fahland D, van der Aalst WMP. Discovering Block-Structured Process Models from Incomplete Event Logs. International Conference on Applications and Theory of Petri Nets and Concurrency [Internet]. 2014;9698:91–110. Available from: http://link.springer.com/10.1007/978-3-319-07734-5_6
10. Yan Z, Sun B, Chen Y, Wen L, Hu L, Wang J, et al. Decomposed and parallel process discovery: A framework and application. Future Generation Computer Systems [Internet]. 2019;98:392–405. Available from: http://www.sciencedirect.com/science/article/pii/S0167739X18327110
11. vanden Broucke SKLM, De Weerdt J. Fodina: A robust and flexible heuristic process discovery technique. Decision Support Systems. Elsevier B.V.; 2017;100:109–18.
12. Hermawan, Sarno R. A More Efficient Deterministic Algorithm In Process. International Journal of Innovative Computing, Information and Control. 2018;14:971–95.
13. Sarno R, Sungkono KR, Johanes R, Sunaryono D. Graph-Based Algorithms for Discovering a Process Model Containing Invisible Tasks. International Journal of Intelligent Engineering and Systems (IJIES). 2019;12:85–94.
14. Guo Q, Wen L, Wang J, Yan Z, Yu PS. Mining Invisible Tasks in Non-free-choice Constructs. Lecture Notes in Computer Science. Springer International Publishing; 2015. p. 109–25.
15. van der Aalst WMP, Weijters T, Maruster L. Workflow mining: discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering. 2004;16:1128–42.
16. De Medeiros AKA, Van Dongen BF, van der Aalst WMP, Weijters AJMM. Process mining: Extending the α-algorithm to mine short loops. Eindhoven University of Technology Eindhoven. 2004;1–25.
17. Wen L, Wang J, van der Aalst WMP, Huang B, Sun J. Mining process models with prime invisible tasks. Data & Knowledge Engineering. Elsevier; 2010;69:999–1021.
18. Sarno R, Sungkono KR. Hidden Markov Model for Process Mining of Parallel Business Processes. International Review on Computers and Software (IRECOS). 2016;11:290–300.
19. Sarno R, Sungkono KR. Coupled Hidden Markov Model for Process Mining of Invisible Prime Tasks. International Review on Computers and Software (IRECOS). 2016;11:539–47.
20. Sarno R, Sungkono KR, Septiarakhman R. Graph-Based Approach for Modeling and Matching Parallel Business Processes. International Information Institute (Tokyo) Information. 2018;21:1603--1614.
21. Iğde M, Kavurucu Y, Mutlu A. Graph Representation of Relational Database for Concept Discovery. Procedia - Social and Behavioral Sciences. 2015;195:1981–9.
22. Wiratmo A, Sungkono KR, Sarno R. Graph-based Algorithm for Checking Wrong Indirect Relationships of Process Model Containing Non-Free Choice. Telkomnika. 2020;18:106–13.

23. Sungkono KR, Sarno R. Constructing Control-Flow Patterns Containing Invisible Task and Non-Free Choice Based on Declarative Model. International Journal of Innovative Computing, Information and Control (IJICIC). 2018;14.
24. Sungkono KR, Sarno R. Patterns of fraud detection using coupled Hidden Markov Model. 2017 3rd International Conference on Science in Information Technology (ICSITech). Bandung: IEEE; 2017. p. 235–40.
25. Russell N, Hofstede AHM, Aalst WMP Van Der, Mulyar N. Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22. Netherlands; 2006;6–22.
26. Sarno R, Sungkono KR. Coupled Hidden Markov Model for Process Discovery of Non-Free Choice and Invisible Prime Tasks. 4th Information Systems International Conference. Elsevier B.V.; 2017. p. 134–41.
27. Van Der Aalst WMP. Process Mining Discovery, Conformance and Enhancement of Business Processes. Germany: Springer; 2011.
28. Darmawan H, Sarno R, Ahmadiyah AS, Sungkono KR, Wahyuni CS. Anomaly Detection Based on Control-flow Pattern of Parallel Business Processes. TELKOMNIKA. 2018;16:2808–15.
29. Buijs JCAM, Van Dongen BF, van Der Aalst WMP. On the role of fitness, precision, generalization and simplicity in process discovery. OTM Conferences (1). 2012. p. 305–22.
30. Buijs JCAM, Dongen BF Van, Aalst WMP Van Der. Quality Dimensions in Process Discovery : The Importance of Fitness , Precision , Generalization and Simplicity. 2014;23:1–39.
31. Sarno R, Sungkono KR. A survey of graph-based algorithms for discovering business processes. International Journal of Advances in Intelligent Informatics. 2019;5:137–49.

Figures

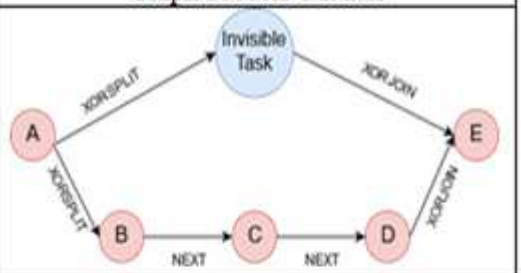
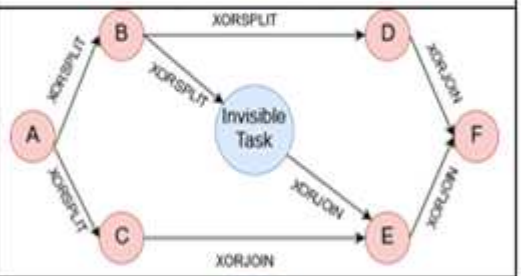
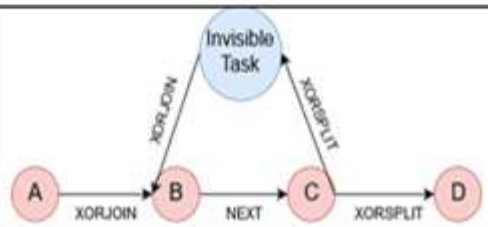
Conditions	Traces	Graph Process Models
Skip	[A, B, C, D, E] [A, E]	
Switch	[A, B, D, E] [A, B, F, E] [A, C, D, E]	
Redo	[A, B, C, D] [A, B, C, B, C, D]	

Figure 1

The Types of Invisible Tasks

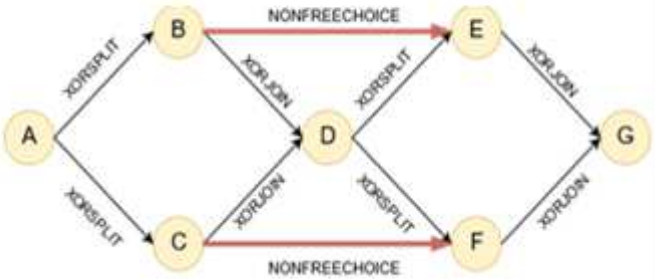
Condition	Traces	Graph Process Model
Non-free-choice	[A, B, D, E, G] [A, C, D, F, G]	

Figure 2

Non-free-choice

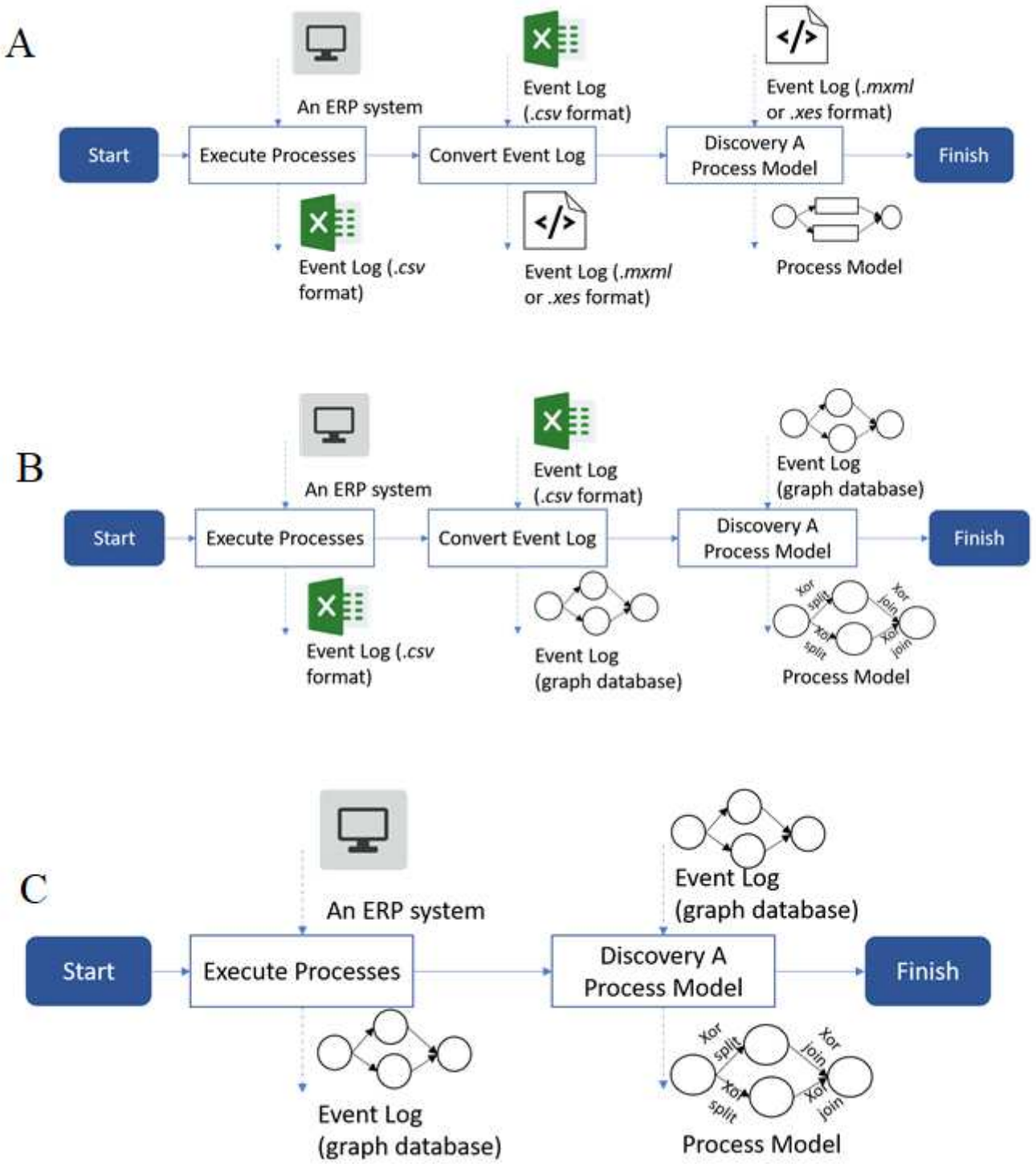


Figure 3

The Comparison Process Discovery Methods (a) The Pioneer of Process Discovery Method by Aalst, et al [15,27], (b) The Process Discovery using Graph-database [28], and (c) GIT Process Discovery Model

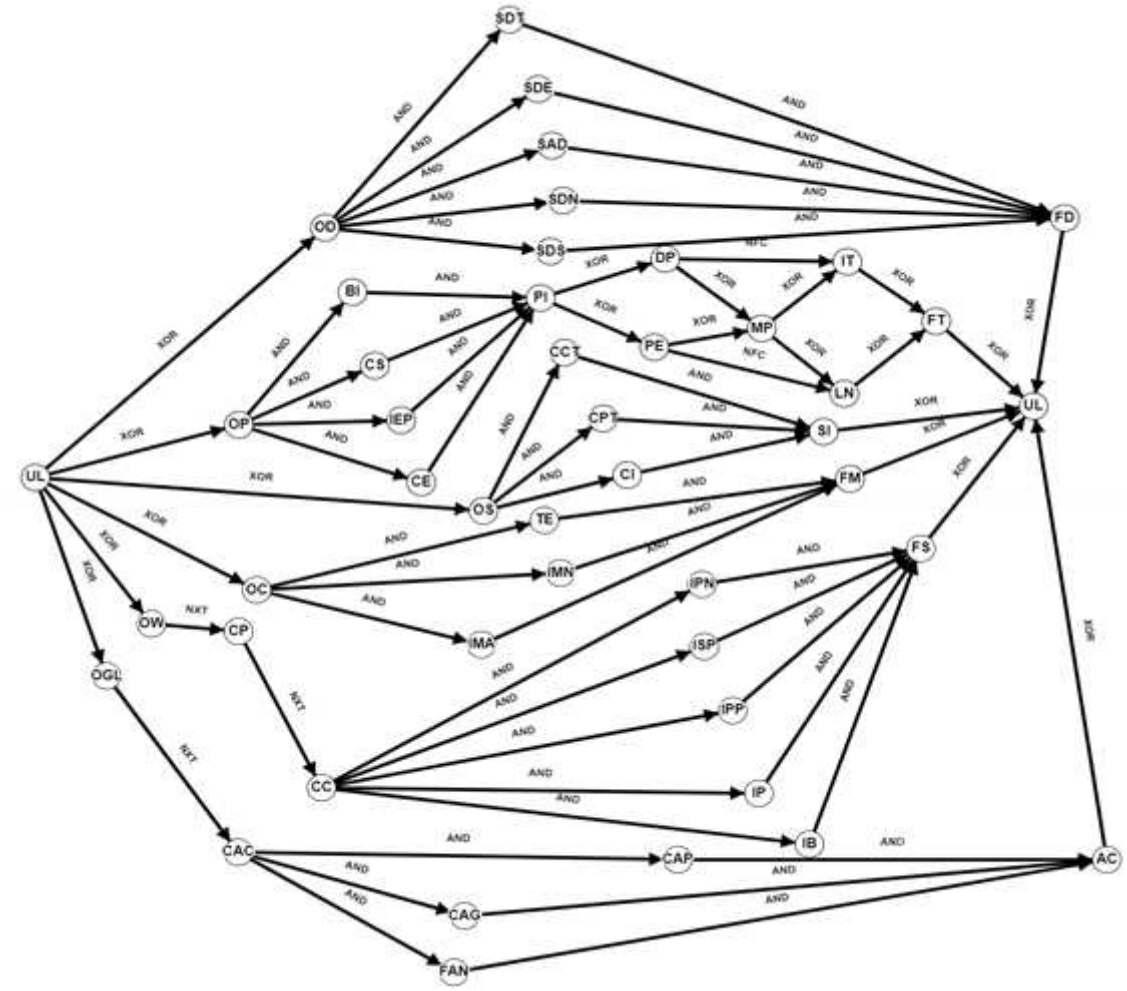


Figure 4

The Process Model by GIT algorithm

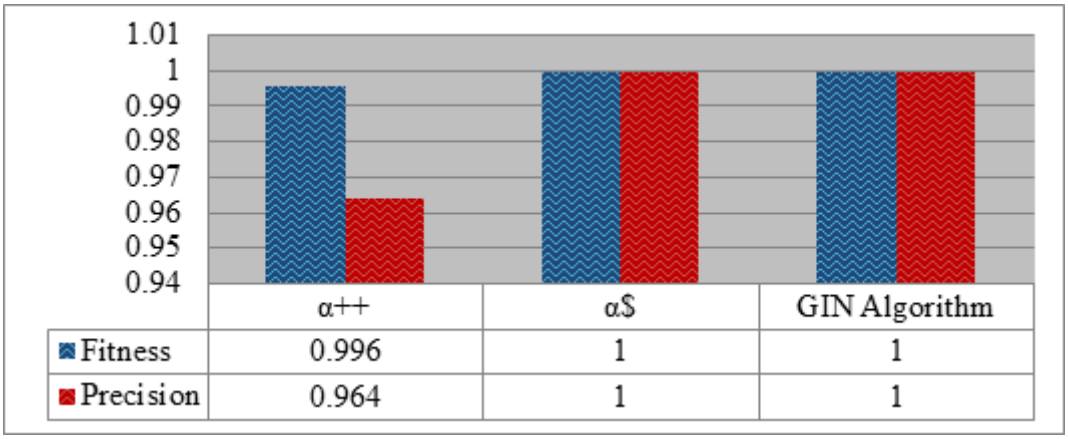


Figure 5

Result Comparison between the Previous Methods and the Proposed Method