Received: December 10, 2020. Revised: March 25, 2021.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Graph-Based Process Model Discovery for Mining Invisible Non-Prime Tasks in Hybrid Choice-Parallel Relationships

Kelly R. Sungkono[1]* **Riyanarto Sarno[1]** **Fransisca D. Amadea[2]** **Ariqoh F. Irbah[2]**

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

Abstract: Process discovery is a study to form a process model based on an event log. The output of process discovery is a process model that contains linked activities based on performed business processes. There are several kinds of relationships and additional invisible tasks in forming a process model. The current invisible tasks, called invisible prime tasks, are divided into three conditions, i.e. skip, redo and switch. However, the need for invisible task is not only in those conditions but also in a situation, which is hybrid choice-parallel relationships. Hybrid choice-parallel relationships occur when there are choice relationships, namely XOR, pile up with parallel relationships for all branching activities (AND) or parallel relationships for several branching activities (OR). This research proposes graph-based Invisible-Task Prime and Non-Prime (ITPNP) algorithm that mines invisible prime tasks and invisible non-prime task in the cased of hybrid parallel relationships. Graph-based ITPNP algorithm creates rules for detecting invisible task needs based on relationships occurring in the graph model. The evaluation is implemented with seven processes and four comparison methods, i.e. Alpha#, Alpha Mining, Fodina, and Graph-based Invisible Task. Among all algorithms, graph-based ITPNP obtains one of fitness for all event logs and one of precision in four event logs. On the other hand, Fodina algorithm has 0.6 precision which is lower than graph-based ITPNP in the event logs requiring invisible non-prime tasks of hybrid XOR-OR. Besides, Graph-based Invisible Task has low fitness, i.e. 0.67 in Hazardous Waste Handling processes that depict choice relationships with additional invisible tasks for hybrid XOR-AND relationships.

Keywords: Graph database, Invisible task, Process discovery.

1. Introduction

Process discovery is a study to form a process model based on an event log. The output of process discovery is a process model that contains linked activities based on performed business processes. The input of process discovery is an event log which is a collection of lists of activities. An event log [1] consists of at least three fields, i.e., an identification of case, a name of activity and a time (start time or end time). An aim of process discovery is creating rules for depicting processes in an event log as a process model.

There are several kinds of relationships and additional of invisible tasks in forming a process model. Those relationships are a sequence relationship [2], looping relationship [3, 4], choice relationship (XOR), parallel relationship for all branching activities (AND), or parallel relationship for several activities (OR) [5, 6], and a non-free choice relationship [7–9]. The widely used invisible tasks are called invisible prime tasks [10], which handle skip cases, redo cases, and switch cases.

Many algorithms are developed to automatic discover those relationships or invisible tasks. α algorithm, the first process discovery algorithm, acquaints XOR and AND relationships and forms those relationships automatically by using tuple rule-based [11]. However, α algorithm form limited relationships, so that there are various research developments of this algorithm. Medeiros et.al. proposes α+[3] for processing event logs

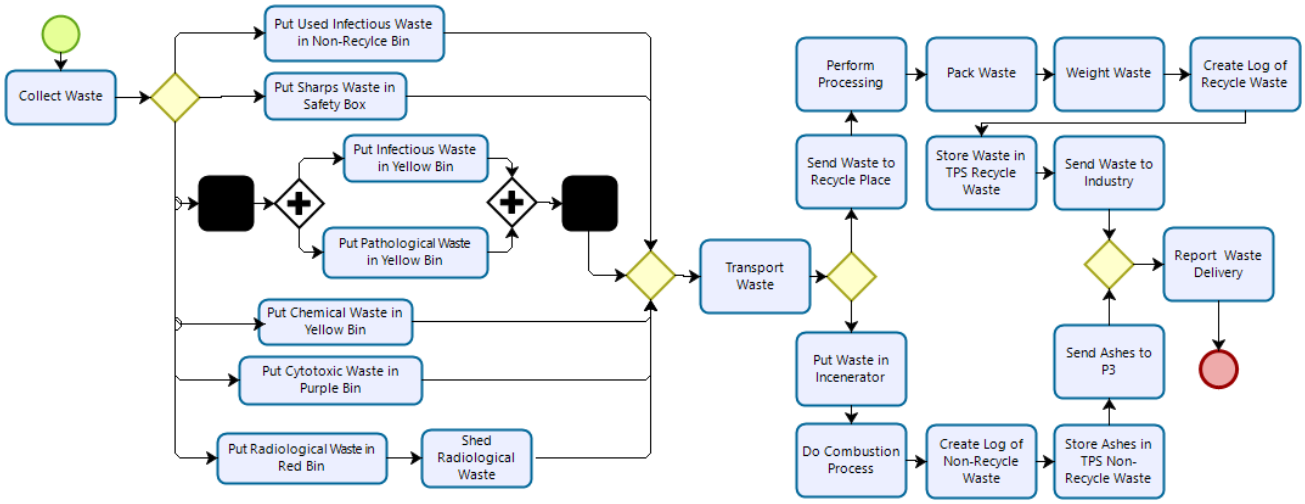

Figure. 1 Hazardous waste processes

containing short loops; thereafter, van der Aalst et.al. develops $\alpha++$ [7] for depicting non-free choice relationship. In addition, Guo presents $\alpha\#$ [10] to form skip cases, redo cases and switch cases of business processes by adding invisible prime tasks in the process model. Latest development is from Sun et.al. who introduces AlphaMining [12] to depict extended short-loops that cannot be handled by $\alpha+$. α algorithm and its development algorithms depict all relationships of processes storing in an event log. On the other hand, other existing algorithms, such as Heuristic Miner [13] and Fodina [14], assign a threshold to filter several activities relationships for obtaining a general process model, can be called as a "lasagna"-like process model [15]. Both those algorithms use frequency of activities to determine relationships displayed in a process model and specify kinds of those relationships. There are also algorithms which utilize a graph-database to create discovery rules and form a process model. Sarno et.al. creates graph-based rules [16] in Neo4j [17], a graph-database platform, to mine parallel relationships; and then those rules are enhanced by Waspada et.al. [5]. Sarno et.al. expand their rules to depict non-free choice [18] and invisible prime tasks [19].

Apart from existing cases, there is another condition that requires an invisible task. It is a hybrid of parallel relationships. This paper states a hybrid of parallel relationships is a condition where different parallel relationships are needed simultaneously. This condition happened in hospital hazardous waste handling processes. Fig.1 gives an overview of this hazardous waste handling. Before processing hazardous wastes, those should be sorting and storing in different types of rubbish bin. This condition indicates XOR relationship. When storing those wastes, infectious waste and pathological waste thrown together in a yellow rubbish bin. This condition declares AND relationship. Two different relationships cannot be formed directly without a task in the middle of them; therefore, an invisible task should be added. The black boxes denoted invisible tasks. $\alpha\#$ algorithm [10] introduces invisible tasks for skip condition, redo condition and switch condition. Those invisible tasks are called invisible prime tasks. Graph-based invisible task algorithm [19] refers to $\alpha\#$ algorithm rules to create graph-based rules of discovering a process model. Both $\alpha\#$ algorithm and graph-based invisible task algorithm handle those three conditions; however, the hybrid of parallel relationships is different from those conditions. The inability of $\alpha\#$ algorithm and graph-based invisible task algorithm to depict invisible task in a hybrid parallel relationship is the main motivation of this research to propose a new process discovery algorithm.

This research proposes graph-based Invisible-Task Prime and Non-Prime (ITPNP) algorithm that mines not only invisible prime tasks but also an invisible non-prime task in the cased of hybrid parallel relationships. The main contributions are as follows.

1. Introducing a hybrid of choice-parallel relationships as a new condition that needs invisible tasks.
2. Proposing graph-based ITPNP algorithm that depicts invisible non-prime tasks in the case of a hybrid of parallel relationships. The proposed algorithm expands rules of existing algorithm, i.e. graph-based invisible task algorithm [19], to depict invisible tasks in a hybrid of choice-parallel relationships.
3. Verifying the proposed rules by using several processes, including the hazardous waste

handling processes, and several measurements, i.e. fitness and precision.

The rest of this paper is organized as follows: Section 2 explains the rules of graph-based invisible task algorithm and basic concepts of existing invisible prime tasks and a graph model. This section also introduces types of hybrid parallel relationships as the challenge problem in this research. A detail description of graph-based ITPNP algorithm is written in Section 3. Next, Section 4 shows the experiment of graph-based ITPNP algorithm which is compared with existing process discovery algorithms based on several study cases and measurements. The last section, Section 5, concludes the contribution and the result of the proposed method.

2. Related work

A. Graph Process Model

A process model is a model made from classifying together similar processes, in other words, a description of a process at the type level [7]. Process models are a balance of workflow diagrams and descriptions of workflows [8]. The workflow diagrams themselves holds an important role as those are the first thing that people will see when analyzing the model and the diagrams will visualize the descriptions of the workflows, thus making it easier to understand the description of the model. However, it still needs a detailed explanation of the process modeling.

A graph is a data structure consisting of edges and vertices [9]. Graph databases such as Neo4j are optimized for data where the links between the things are both numerous and as important as the things themselves [10]. In Neo4j, vertices are called nodes and edges are the representation of relationships between vertices. Graph database technology is an effective tool for modeling data focused on the relationship between event logs [11]. Graph databases form the connection between each data that is stored as a node. An event log containing activity and time is needed to build a graph-database process model.

B. Invisible Prime Task

The common problem in creating process models is the existence of invisible tasks. The invisible task is created when there are dummy activities in a process model. The common type of it is prime invisible task. Prime invisible task consists of invisible task skip, redo and switch [10] as in Figs. 2-4. The grey squares are invisible tasks.

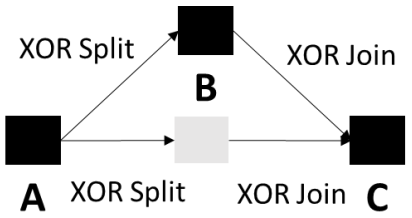

Figure. 2 Invisible Prime Task for Skip Cases

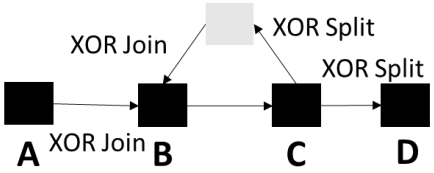

Figure. 3 Invisible Prime Task for Redo Cases

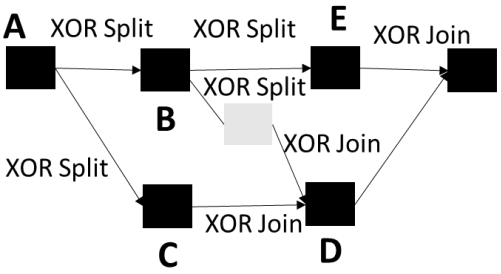

Figure. 4 Invisible Prime Task for Switch Cases

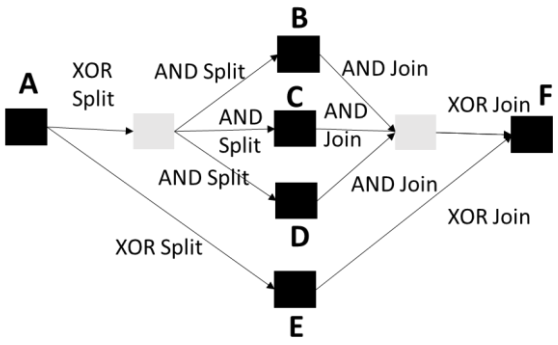

Figure. 5 Hybrid Parallel Invisible Task XOR AND

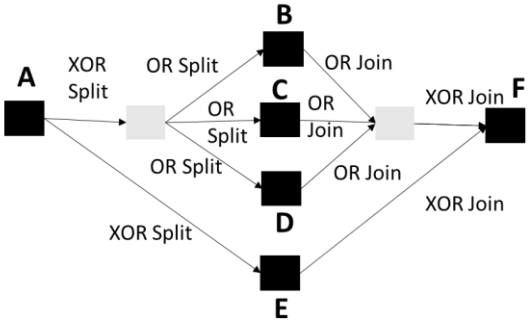

Figure. 6 Hybrid Parallel Invisible Task XOR OR

C. Invisible Non-Prime Task for Hybrid Choice-Parallel Relationships

Invisible tasks for hybrid choice-parallel are invisible tasks that exist when there is either XOR OR relationship or XOR AND relationships from and to an activity of a process model. The examples of those are depicted on Figs. 5 and 6.

D. Existing algorithm: Graph-based Invisible Task algorithm

Graph-based Invisible Task Algorithm [19] is an algorithm that utilizes graph-database to mine invisible prime tasks in skip, redo and switch cases. The rules of graph-based invisible task algorithm.

Alpha# [10] only check the activities from the event log, so the relations between activities are determined by checking all possible relationships based on all activities. Those checking processes increase the processing time of Alpha# [19].

The first step of the graph-based algorithm is converting the event log from a comma separated values (CSV) file into a graph database. After that, the graph-based algorithm adds parallel relationships and invisible tasks based on its rule, that is applied in order, in Table 1. The invisible task is created when there are SPLIT relationships and JOIN relationships coming to the same node.

E. Measurements

This research utilizes two measurements, i.e. Fitness and Precision. Fitness calculates how many traces in an event log which are depicted in a discovered process model. On the other hand, Precision measures how fit the obtained process model with traces of the event log. The equations of them are shown in Eq. 1 and Eq. 2.

$$FT = \frac{matchTraces}{n(traces\_eventLog)} \quad (1)$$

$$PC = \frac{matchTraces}{n(traces\_processM)} \quad (2)$$

where:

- FT = fitness
- PC = precision
- $matchTraces$ = traces of event logs which are depicted in a process model
- $n(traces\_eventLog)$ = amount of traces of an event log
- $n(traces\_processM)$ = amount of traces of a process model

Table 1. Graph-Based Invisible Task Algorithm

Mine Relationship / Taask	Cypher
Sequence	MATCH (c:Activity) WITH COLLECT(c) AS Caselist UNWIND RANGE(0,Size(Caselist) - 2) as idx WITH Caselist[idx] AS s1, Caselist[idx+1] AS s2 MATCH (b:CaseActivity),(a:CaseActivity) WHERE s1.CaseId = s2.CaseId AND s1.Name = a.Name AND s2.Name = b.Name MERGE (a)-[r:SEQUENCE]->(b)
XOR (XOR Split)	MATCH (bef)-[r]->(aft) WHERE size((bef)-->())>1 AND size((aft)<--())=1 AND (size((aft)-->())=1 OR size((aft)-->())>1) CREATE (bef)-[:XORSPLIT]->(aft) DELETE r
XOR (XOR Join)	MATCH (bef)-[r]->(aft) WHERE (size((bef)-->())=1 OR size((bef)-->())>1) AND size((aft)<--())>1 CREATE (bef)-[:XORJOIN]->(aft) DELETE r
AND (AND Split)	MATCH (aft1)<-[r]-(bef)-[s]->(aft2) WHERE size((bef)-->())>1 AND size((aft2)<--())=size((bef)-->()) AND size((aft1)<--())=size((bef)-->()) AND not (aft1)-[:SEQUENCE]->(bef) AND not (aft2)-[:SEQUENCE]->(bef) MERGE (aft1)<-[:ANDSPLIT]-(bef)-[:ANDSPLIT]->(aft2) DELETE r,s
AND (AND Join)	MATCH (aft1)-[r]->(bef)<-[s]-(aft2) WHERE size((bef)<--())>1 AND size((aft2)-->())=size((bef)<--()) AND size((aft1)-->())=size((bef)<--()) AND not ()-[:ANDSPLIT]->(bef) MERGE (aft1)-[:ANDJOIN]->(bef)<-[:ANDJOIN]-(aft2) DELETE r,s
OR (OR Split)	MATCH (aft1)<-[r]-(bef)-[s]->(aft2) WHERE size((bef)-->())>1 AND size((aft2)-->())<size((bef)-->()) AND size((aft2)-->())>1 AND not ((aft1)-[:SEQUENCE]->(bef) OR (aft2)-[:SEQUENCE]->(bef)) MERGE (aft1)<-[:ORSPLIT]-(bef)-[:ORSPLIT]->(aft2) DELETE r,s
OR (OR Join)	MATCH (aft1)-[r]->(bef)<-[s]-(aft2) WHERE size((bef)<--())>1 AND size((aft2)-->())<size((bef)<--()) AND size((aft2)-->())>1 MERGE (aft1)-[:ORJOIN]->(bef)<-[:ORJOIN]-(aft2) DELETE r,s

Invisible Task	MATCH (aft2)<-[r]-(bef)-[s]->(aft1) WHERE (TYPE(s)='XORSPLIT' AND TYPE(r)='XORJOIN') MERGE (i:Invisible_Task {Name: "Inv_Task"}) MERGE (bef)-[:XORSPLIT]->(i) MERGE (aft2)<-[:XORJOIN]-(i) DELETE r

3. Methods

Rules of graph-based ITPNP algorithm is described in Table 2.

First, graph-based ITPNP algorithm collects activities based on the event log and creates unique activities based on those collected events as a set of P_Model nodes. Referring to the collected events, rules of sequence relationships are executed for P_Model nodes. A graph model containing sequence relationships are obtained in this step.

Secondly, the obtained graph-model is developed with rules of mining invisible tasks, including invisible tasks of hybrid choice-parallel relationships. Referring to the number of previous edges and next edges of an activity, a choice relationship (XOR) occurs when an activity has many next activities or many previous activities. Those activities only have one previous edge and one following edge. On the other hand, parallel relationships (AND or OR relationships) occur when an activity has many next activities or many previous activities. Those activities have more than one previous edge and more than one following edge. Hybrid choice-parallel relationships occur when choice relationships (XOR) pile up with parallel relationships (AND or OR). The pile-up of choice-parallel relationships means that an activity has many next activities or previous activities which have XOR relationship and many next activities or pervious activities which have AND or OR relationships. Graph-based ITPNP algorithm determines hybrid choice parallel relationships if an activity, denoted as X, has many next activities, denoted as $k, l \in Y$, and $\forall k$ has one previous edge and one following edge while $\forall l$ have more than one previous edge and more than one following edge. The invisible tasks are added between X and l.

Then, rules of choice relationships and parallel relationships are implemented. The final graph model is obtained after all these rules are executed.

4. Results and discussion

This research evaluates graph-based ITPNP algorithm by comparing with other algorithms in six event logs. Those event logs are four simulation event logs and two real event logs. The simulation log means the event log is formed based on a process model depicted by authors; however, the real event logs depict real processes, which are Port Container Handling processes and Hazardous Waste Handling processes.

D01 consists of three traces which depict redo cases. In addition, D02 contains two traces about skip cases. Switch cases are recorded as D03. Hybrid XOR-AND are stored as D04, whereas Hybrid XOR-OR are stored as D05. Port Container Handling processes, which are 230 cases, are represented as D06. On the other hand, D07 that contains 400 cases, is an event log of Hazardous Waste Handling processes.

Fitness and Precision are chosen measurements to compare graph-based ITPNP algorithm with others. Based on Table 3, graph-based ITPNP algorithm have high fitness and precision for all event logs, followed by Fodina [22] which get best scores in six event logs. Fodina has low fitness and precision in D05 because this algorithm has not been able to depict OR relationships. Graph-based Invisible Task struggles in processes which require invisible tasks for XOR – AND relationships, thus it has low precision in D04 and D07.

Table 2. Graph-based ITPNP algorithm

Pseudocode	**Cypher**
1. Load Data	
Load data from comma-separated values to stores all processes, containing id of case (CaseId), name of activity (Name) and a time (Time)	LOAD CSV with headers FROM *name_file* AS line Merge (:Activity {CaseId:line.Case_ID, Name:line.Activity, Time:line.Time})
Load data from comma-separated values and Store list of activities in PModel node	LOAD CSV with headers FROM *name_file* AS line Merge (:PModel {Name:line.Activity })

2. Mine Sequence Relationships	
For ind_actA ← 0 : amount_activity_Activity **if** activity_Activity[ind_actA] $_{Case\_ID}$ = activity_Activity[ind_actA + 1] $_{Case\_ID}$ **For** ind_actP ← 0 : amount_activity_PModel **if** activity_Pmodel[ind_actP] $_{Name}$ = activity_Activity[ind_act] $_{Name}$ front_act ← activity_Pmodel **else** end_act ← activity_Pmodel create sequence relationship from front_act to end_act **return** a modified graph model which has sequence relationships	MATCH (c:Activity) WITH COLLECT(c) AS Caselist UNWIND RANGE(0,Size(Caselist) - 2) as idx WITH Caselist[idx] AS s1, Caselist[idx+1] AS s2 MATCH (b:CaseActivity), (a:CaseActivity) WHERE s1.CaseId = s2.CaseId AND s1.Name = a.Name AND s2.Name = b.Name MERGE (a)-[r:SEQUENCE]->(b)
3. Mine Invisible Task in Hybrid Choice-Parallel Relationship	
For a graph which have relationships from activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+1] and activity_Pmodel[ind_actP+2] **if** *Before*(activity_Pmodel[ind_actP+1]) = 1 and *Before*(activity_Pmodel[ind_actP+2]) > 1 create *Invisible_Task* create sequence relationship from activity_Pmodel[ind_actP] to *Invisible_Task* create sequence relationship from *Invisible_Task* to activity_Pmodel[ind_actP + 2] **return** a modified graph model which involves invisible tasks	MATCH (aft2)<-[r]-(bef)-[s]->(aft1) WHERE size((aft1)<--())=1 AND size((aft2)<--())>1 MERGE (i:Invisible_Task {Name: "Inv_Task1"}) MERGE (bef)-[:SEQUENCE]->(i) MERGE (i)-[:SEQUENCE]->(aft2) DELETE r
For a graph which have relationships from activity_Pmodel[ind_actP+1] and activity_Pmodel[ind_actP+2] to from activity_Pmodel[ind_actP] **if** *Next*(activity_Pmodel[ind_actP+1]) = 1 and *Next*(activity_Pmodel[ind_actP+2]) > 1 create *Invisible_Task* create sequence relationship from activity_Pmodel[ind_actP + 2] to *Invisible_Task* create sequence relationship from *Invisible_Task* to activity_Pmodel[ind_actP] **return** a modified graph model which involves invisible tasks	MATCH (bef2)-[r]->(aft)<-[s]-(bef1) WHERE size((bef1)-->())=1 AND size((bef2)-->())>1 MERGE (n:Invisible_Task {Name: "Inv_Task2"}) MERGE (bef2)-[:SEQUENCE]->(n) MERGE (n)-[:SEQUENCE]->(aft) DELETE r
4. Mine XOR Relationship	
For a graph which have relationships from activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+1] **if** *Next*(activity_Pmodel[ind_actP]) > 1 and *Before*(activity_Pmodel[ind_actP+2])= 1 and *Next*(activity_Pmodel[ind_actP+1]) =1 or *Next*(activity_Pmodel[ind_actP]+1) > 1 create *XOR_SPLIT* relationship from activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+1] **return** a modified graph model which consists of XOR_SPLIT relationships	MATCH (bef)-[r]->(aft) WHERE size((bef)-->())>1 AND size((aft)<--())=1 AND (size((aft)-->())=1 OR size((aft)-->())>1) CREATE (bef)-[:XORSPLIT]->(aft) DELETE r
For a graph which have relationships from activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+1] **if** *Next*(activity_Pmodel[ind_actP]) =1 or *Next*(activity_Pmodel[ind_actP]) > 1 and *Before*(activity_Pmodel[ind_actP+1]) > 1 create *XOR_JOIN* relationship from activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+1] **return** a modified graph model which consists of XOR_JOIN relationships	MATCH (bef)-[r]->(aft) WHERE (size((bef)-->())=1 OR size((bef)-->())>1) AND size((aft)<--())>1 CREATE (bef)-[:XORJOIN]->(aft) DELETE r

5. Mine AND Relationship	
For a graph which have relationships from activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+1] and activity_Pmodel[ind_actP+2] **if** *Next*(activity_Pmodel[ind_actP]) > 1 and *Next*(activity_Pmodel[ind_actP+1]) =$Next$(activity_Pmodel[ind_actP]) and *Next*(activity_Pmodel[ind_actP+2]) =$Next$(activity_Pmodel[ind_actP]) and no relationship from activity_Pmodel[ind_actP+1] to activity_Pmodel[ind_actP] and no relationship from activity_Pmodel[ind_actP+2] to activity_Pmodel[ind_actP] create *AND_SPLIT* relationship from activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+1] create *AND_SPLIT* relationship from en activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+2] **return** a modified graph model which involves AND_SPLIT relationships	MATCH (aft1)<-[r]-(bef)-[s]->(aft2) WHERE size((bef)-->())>1 AND size((aft2)-->())=size((bef)-->()) AND size((aft1)-->())=size((bef)-->()) AND not (aft1)-[:SEQUENCE]->(bef) AND not (aft2)-[:SEQUENCE]->(bef) MERGE (bef)-[:ANDSPLIT]->(aft1) MERGE (bef)-[:ANDSPLIT]->(aft2) DELETE r,s
For a graph which have relationships from activity_Pmodel[ind_actP+1] and activity_Pmodel[ind_actP+2] to activity_Pmodel[ind_actP] **if** *Before*(activity_Pmodel[ind_actP]) > 1 and *Before*(activity_Pmodel[ind_actP+1]) =$Before$(activity_Pmodel[ind_actP]) and *Before*(activity_Pmodel[ind_actP+2]) =$Before$(activity_Pmodel[ind_actP]) and no *AND_SPLIT* relationship to activity_Pmodel[ind_actP] create *AND_JOIN* relationship from activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+1] create *AND_JOIN* relationship from n activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+2] **return** a modified graph model which involves AND_JOIN relationships	MATCH (aft1)-[r]->(bef)<-[s]-(aft2) MATCH (aft1)-[t]-(aft2) WHERE size((bef)<--())>1 AND size((aft2)-->())=size((bef)<--()) AND size((aft1)-->())=size((bef)<--()) AND not ()-[:ANDSPLIT]->(bef) MERGE (bef)<-[:ANDJOIN]-(aft2) MERGE (bef)<-[:ANDJOIN]-(aft1) DELETE r,s,t
6. Mine OR Relationship	
For a graph which have relationships from activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+1] and activity_Pmodel[ind_actP+2] **if** *Next*(activity_Pmodel[ind_actP]) > 1 and *Next*(activity_Pmodel[ind_actP+1])< $Next$(activity_Pmodel[ind_actP]) and *Next*(activity_Pmodel[ind_actP+1]) > 1 and *Next*(activity_Pmodel[ind_actP+2])< $Next$(activity_Pmodel[ind_actP]) and *Next*(activity_Pmodel[ind_actP+2]) > 1 and no relationship from activity_Pmodel[ind_actP+1] to activity_Pmodel[ind_actP] and no relationship from activity_Pmodel[ind_actP+2] to activity_Pmodel[ind_actP] create *OR_SPLIT* relationship from activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+1] create *OR_SPLIT* relationship from en activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+2] **return** a modified graph model which involves OR_SPLIT relationships	MATCH (aft1)<-[r]-(bef)-[s]->(aft2) WHERE size((bef)-->())>1 AND size((aft2)-->())<size((bef)-->()) AND size((aft2)-->())>1 AND size((aft1)-->())<size((bef)-->()) AND size((aft1)-->())>1 AND not (aft1)-[:SEQUENCE]->(bef) AND not (aft2)-[:SEQUENCE]->(bef) MERGE (bef)-[:ORSPLIT]->(aft2) MERGE (bef)-[:ORSPLIT]->(aft1) DELETE r,s
For a graph which have relationships from activity_Pmodel[ind_actP+1] and activity_Pmodel[ind_actP+2] to activity_Pmodel[ind_actP] **if** *Before*(activity_Pmodel[ind_actP]) > 1 and *Before*(activity_Pmodel[ind_actP+1])< $Before$(activity_Pmodel[ind_actP]) and *Before*(activity_Pmodel[ind_actP+1]) > 1 and *Before*(activity_Pmodel[ind_actP+2])< $Before$(activity_Pmodel[ind_actP]) and and	MATCH (aft1)-[r]->(bef)<-[s]-(aft2) MATCH (aft1)-[t]-(aft2) WHERE size((bef)<--())>1 AND size((aft2)-->())<size((bef)<--()) AND size((aft2)-->())>1 AND size((aft1)-->())<size((bef)<--()) AND size((aft1)-->())>1 AND not ()-[:ORSPLIT]->(bef)

Before(activity_Pmodel[ind_actP+2]) > 1 and no *OR_SPLIT* relationship to activity_Pmodel[ind_actP] create *AND_JOIN* relationship from activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+1] create *AND_JOIN* relationship from n activity_Pmodel[ind_actP] to activity_Pmodel[ind_actP+2] **return** a modified graph model which involves OR_JOIN relationships	MERGE (aft1)-[:ORJOIN]->(bef) MERGE (aft2)-[:ORJOIN]->(bef) DELETE r,s,t

where: amount_activity_Activity = the total of all activities which are stored as Activity nodes in a graph-database

activity_Activity[ind_actA] $_{Case\_ID}$ = identification of case of an activity

activity_Activity[ind_actA + 1] $_{Case\_ID}$ = identification of case of the next activity

amount_activity_PModel = the total of variations of activities which are stored as PModel nodes in a graph-database

Before(activity_Pmodel[ind_actP+1]) = the number of previous edges of an activity denoted as activity_Pmodel[ind_actP+1]

Next(activity_Pmodel[ind_actP+1]) = the number of following edges of an activity denoted as activity_Pmodel[ind_actP+1]

Table 3. Performance comparison result

	Event Logs						
Algorithms	D01	D02	D03	D04	D05	D06	D07
Alpha# [10]	F= 1 P= 1	F= 1 P= 1	F= 1 P= 1	F= 1 P= 0.6	F= 0.25 P= 1	F= 1 P= 0.33	F= - P= -
Alpha Mining [12]	F= 1 P= 0.75	F= 1 P= 1	F= 0 P= 0	F= - P= -	F= 0 P= 0	F= 1 P= 0.33	F= - P= -
Fodina [21]	F= 1 P= 1	F= 1 P= 1	F= 1 P= 1	F= 1 P= 1	F= 0.5 P= 0.4	F= 1 P= 0.33	F= 1 P= 0.5
Graph-based Invisible Task [19]	F= 1 P= 1	F= 1 P= 1	F= 1 P= 1	F= 1 P= 0.6	F= 1 P= 1	F= 1 P= 0.33	F= 0.67 P= 0.33
Graph-based ITPNP (Proposed Method)	F= 1 P= 1	F= 1 P= 1	F= 1 P= 1	F= 1 P= 1	F= 1 P= 1	F= 1 P= 0.33	F= 1 P= 0.5

where:
D01: Simulation Process Containing Invisible Prime Tasks for a Redo Case
D02: Simulation Process Containing Invisible Prime Tasks for a Skip Case
D03: Simulation Process Containing Invisible Prime Tasks for a Switch Case
D04: Simulation Process Containing Invisible Non-Prime Tasks for Hybrid XOR-AND
D05: Simulation Process Containing Invisible Non-Prime Tasks for Hybrid XOR-OR
D06: Port Container Handling Processes
D07: Hazardous Waste Handling Processes
F: Fitness
P: Precision

All algorithms cannot get high precision for D06 and D07 because those event logs contains non-free choice relationships [8, 20]. This weakness will be improved by graph-based ITPNP algorithm on the next research.

5. Conclusion

This research proposes graph-based ITPNP algorithm to handle invisible tasks of hybrid choice-parallel relationships. Hybrid choice-parallel relationships occur when there are choice

relationships (XOR) pile up with parallel relationships (AND or OR). The rules for detecting invisible task needs are based on relationships occurring in the graph model.

This research compares the process models from graph-based ITPNP algorithm and other existing algorithms by measuring their fitness and precision. The evaluation shows that graph-based ITPNP can depict invisible tasks for hybrid XOR-AND and for hybrid XOR-OR by having one fitness value and one precision value. In contrast, Fodina can only define invisible tasks for hybrid XOR-AND, and graph-based invisible task algorithm can only depict invisible tasks for hybrid XOR-OR. Graph-based ITPNP not only forms invisible tasks for hybrid choice-parallel relationships, but also invisible tasks for redo, skip and switch cases. Process models of graph-based ITPNP algorithm obtain one of fitness and one of precision for event logs containing redo, skip and switch cases. It can be concluded that graph-based ITPNP can depict invisible prime tasks for redo condition, skip condition, switch condition and hybrid choice-parallel relationships.

For Port Container Handling and Hazardous Waste Handling processes, graph-based ITPNP failed to reach a higher result in the precision measurement. The low result because the algorithm was unable to detect the non-free choice relationship. This weakness will be the future work of this research.

Conflicts of Interest

The authors declare no conflict of interest.

Author Contributions

Conceptualization, K.R. Sungkono; supervision, R. Sarno; methodology, K.R. Sungkono, F.D. Amadea, and A.F. Irbah; cypher creation, F.D. Amadea, and A.F. Irbah; validation, K.R. Sungkono; writing—original draft preparation, F.D. Amadea, and A.F. Irbah; writing— review and editing, K.R. Sungkono.

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 Penelitian Dana Departemen managed by Institut Teknologi Sepuluh Nopember (ITS) research grant (Contract No. 1664/PKS/ITS/2020).

References

[1] A.-W. Scheer and A.-W. Scheer, "Process Mining", *Unternehmung 4.0*, pp. 85–102, 2020, doi: 10.1007/978-3-658-27694-2_5.

[2] R. Sarno and K. R. Sungkono, "Hidden Markov Model for Process Mining of Parallel Business Processes", *International Review on Computers and Software (IRECOS)*, Vol. 11, No. 4, pp. 290–300, 2016, doi: 10.15866/irecos.v11i4.8700.

[3] 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 University of Technology Eindhoven*, pp. 1–25, 2004, doi: 10.1016/j.is.2011.01.003.

[4] D. Chapela-Campa, M. Mucientes, and M. Lama, "Mining Frequent Patterns in Process Models", 2017, Accessed: Apr. 19, 2019. [Online]. Available: http://arxiv.org/abs/1710.05693.

[5] I. Waspada, R. Sarno, and K. R. Sungkono, "An Improved Method of Parallel Model Detection for Graph-Based Process Model Discovery", *International Journal of Intelligent Engineering and Systems*, Vol. 13, No. 2, pp. 127–138, 2020, doi: 10.22266/ijies2020.0430.13.

[6] D. Zhang, A. Kuhnle, J. Richardson, and M. Sensoy, "Process Discovery for Structured Program Synthesis", *arXiv preprint arXiv:2008.05804*, 2020, [Online]. Available: http://arxiv.org/abs/2008.05804.

[7] L. Wen, W. M. P. van der Aalst, J. Wang, and J. Sun, "Mining process models with non-free-choice constructs", *Data Mining and Knowledge Discovery*, Vol. 15, No. 2, pp. 145–180, 2007, doi: 10.1007/s10618-007-0065-y.

[8] Q. Guo, L. Wen, J. Wang, Z. Yan, and P. S. Yu, "Mining Invisible Tasks in Non-free-choice Constructs", *Lecture Notes in Computer Science*, Springer International Publishing, 2015, pp. 109–125.

[9] L. Hakim, R. Sarno, and K. R. Sungkono, "Modified Alpha++ Algorithm for Discovering the Hybrid of Non-Free Choice and Invisible Task of Business Processes", *International Journal of Intelligent Engineering and Systems*, Vol. 12, No. 3, pp. 31–40, 2019.

[10] 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, doi: 10.1016/j.datak.2010.06.001.

[11] R. Sarno, R. D. Dewandono, T. Ahmad, M. F.

Naufal, and F. Sinaga, "Hybrid Association Rule Learning and Process Mining for Fraud Detection", *IAENG International Journal of Computer Science*, Vol. 42, No. 2, 2015.

[12] H. W. Sun, W. Liu, L. Qi, Y. Y. Du, X. Ren, and X. Y. Liu, "A process mining algorithm to mixed multiple-concurrency short-loop structures", *Information Sciences*, Vol. 542, pp. 453–475, 2021, doi: 10.1016/j.ins.2020.07.003.

[13] A. P. Kurniati, G. Kusuma, and G. Wisudiawan, "Implementing Heuristic Miner for Different Types of Event Logs", *International Journal of Applied Engineering Research*, Vol. 11, No. 8, pp. 5523–5529, 2016.

[14] S. K. L. M. vanden Broucke, and J. De Weerdt, "Fodina: A robust and flexible heuristic process discovery technique", *Decision Support Systems*, Vol. 100, pp. 109–118, 2017, doi: 10.1016/j.dss.2017.04.005.

[15] W. Van Der Aalst, "Process mining: Overview and opportunities", *ACM Transactions on Management Information Systems (TMIS)*, Vol. 3, No. 2, pp. 1–17, 2012.

[16] R. Sarno, K. R. Sungkono, and R. Septiarakhman, "Graph-Based Approach for Modeling and Matching Parallel Business Processes", *International Information Institute (Tokyo). Information*, Vol. 21, No. 5, pp. 1603--1614, 2018.

[17] J. Celko, "Graph Databases," *Joe Celko's Complete Guide to NoSQL*, pp. 27–46, 2014, doi: 10.1016/b978-0-12-407192-6.00003-0.

[18] R. Sarno, and K. R. Sungkono, "A survey of graph-based algorithms for discovering business processes", *International Journal of Advances in Intelligent Informatics*, Vol. 5, No. 2, pp. 137–149, 2019.

[19] R. Sarno, K. R. Sungkono, R. Johanes, and D. Sunaryono, "Graph-Based Algorithms for Discovering a Process Model Containing Invisible Tasks", *International Journal of Intelligent Engineering and Systems*, Vol. 12, No. 2, pp. 85–94, 2019.

[20] L. Hakim, R. Sarno, and K. R. Sungkono, "Modified Alpha++ Algorithm for Discovering the Hybrid of Non-Free Choice and Invisible Task of Business Processes", *International Journal of Intelligent Engineering and Systems*, Vol. 12, No. 3, pp. 31–40, 2019, doi: 10.22266/ijies2019.0630.04.

[21] S. K. L. M. vanden Broucke, and J. De Weerdt, "Fodina: A robust and flexible heuristic process discovery technique", *Decision Support Systems*, Vol. 100, pp. 109–118, 2017, doi: 10.1016/j.dss.2017.04.005.