# Graph-based Process Mining for Measuring Quality of Business Process Model

Kelly Rossa Sungkono
Department of Informatics
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
kelly@its.ac.id

Riyanarto Sarno
Department of Informatics
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
riyanarto@if.its.ac.id

Fara Dinda Mutia Kinanggit
Department of Informatics
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
faradinda96@gmail.com

Irsyadhani Dwi Shubhi
Department of Informatics
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
irsyadhanishubhi@gmail.com

Khofifah Nurlaela
Department of Informatics
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
khofifahnurlaela11@gmail.com

2022 International Conference on Informatics Electrical and Electronics (ICIEE) | 978-1-6654-8622-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICIEE55596.2022.10010272

***Abstract*— Process mining can be used to model business processes. In process mining, graph-based can be used as an option in making a process model that is efficient based on quality. This research proposes a method of measuring fitness, precision, simplicity, and generalization. Measurement evaluation is carried out using two processes, port container handling and waste handling. Based on the research, the results of the fitness and precision port container handling using the Proposed Graph-based Quality Measurement and ETM Quality Measurement methods are 1.00. Meanwhile, the generalization obtained from both methods is 0.96. Simplicity generated by the Proposed Graph-based Quality Measurement method is 0.87 meanwhile the result of ETM Quality Measurement method is 1.00. In the case study of waste handling, based on both methods, the fitness is 1.00 and the generalization is 0.83. Precision obtained from the Proposed Graph-based Quality Measurement method is 0.95 and 0.75 from ETM Quality Measurement method. The simplicity obtained from the Graph-based Quality Measurement method is 0.95 meanwhile from ETM Quality Measurement method is 1.00. The simplicity calculation results are different because ETM cannot detect invisible tasks, while Graph-based Quality Measurement can discover them.**

***Keywords*— *business process, graph, process, process discovery, process mining, process model, quality measurement***

## I. INTRODUCTION

Over the years, the Business Process management (BPM) field has fragmented significantly, resulting in self-sustaining sub areas of BPM study such as business process [1]. The old BPM strategy has been supplanted in recent years by Case Management with process discovery methods [2]. Process discovery is a rapidly emerging scientific field that focuses on the development of theory and strategies for acquiring and conveying knowledge about real-world process execution and other organizational activity norms [3].

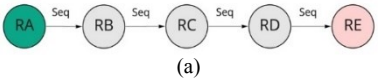
Process discovery approaches use event logs to automatically infer process models [4]–[6]. It is widely considered as the most popular method of process mining [7]. Process discovery is a crucial discipline in process mining that entails the creation of process models using event data that has been captured [3]. Event data that has been captured is created during the execution of (business) processes [8]. Event data, i.e. also known as event log, is the starting point for process mining. Each entry in the log relates to an activity and is associated with a certain case. These event logs can be used to conduct three types of process mining approaches for various purposes: discovery, conformance, and enhancement [9]. An algorithm that based on graph is one which uses a graph-database to represent a process model. Graph-based algorithms opted to process graph-database because it can contain not only actions but also its relationships. Its ability to store relationships results in minimal time complexity [10].

Existing quality measurement research on business process model [11], [12] shows several weaknesses, including (1) the process model must be changed to another process model to be able to calculate simplicity and generalization, and (2) it has not accommodated Invisible Tasks and Non - Free Choice. In calculating the ETM quality measurement, the first thing to do is to change the discovery event log process into a tree process model, then calculate the results. The results obtained are used as a reference for calculations using the graph method later. To calculate the quality measurement automatedly, Neo4j application is used. The Neo4j application is used because it has the advantage of being able to create event logs and process models in one application. Neo4js query methods are written in Cypher language which is query language designed by Neo Technology[13]. There are two datasets that used in this implementation, the first has 165 data records and the second has 68 data records. In the initial method, using the process tree method, it was found that the process model has not been able to handle the Non-Free Choice and Invisible Task relations. An invisible task is a task added to the process model to aid in the visualization of real processes[14]. The relationship between Non-Free Choice and Invisible Task can affect the calculation of quality measurement. So to deal with this problem, we tried to use another method, namely graph-based process discovery with Neo4j tools. Graph-based quality measurement calculations are able to handle Non-Free Choice and Invisible Tasks by modifying the cypher rule.

## II. PRELIMINARIES

### A. Relation Types of Business Process Model

The Business Process Model has two types of basic relation activities : (1) Sequence Relation (2) Parallel Relation. Sequence Relation link one activity and another activity. Meanwhile, Parallel Relation link one activity and other activities. There are three types of Parallel Relations, XOR, OR, and AND. Exactly one activity is chosen by XOR and carried out. OR chooses one or more activities to be carried out. AND permits some activities to be executed [15]. Fig. 1 examples of relation types in business process model.




(a)

978-1-6654-8622-4/22/$31.00 ©2022 IEEE

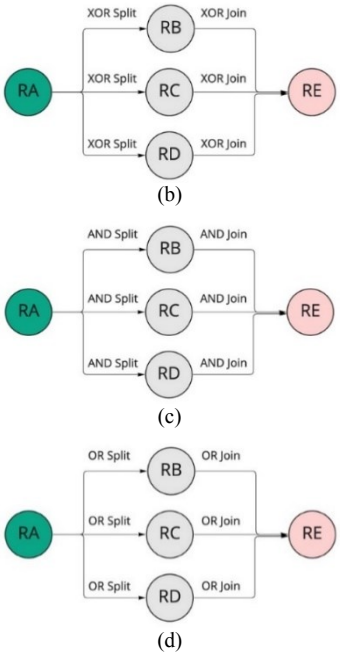



Fig. 1. Relation types of business process model example (a) Sequence relation, (b) XOR relation, (c) AND relation, and (d) OR relations

### B. *Quality Measurement of Business Process Model*

There are numerous attributes that can be measured to see how well a process model describes observable behavior. Fitness, precision, simplicity, and generalization are the four attributes [16], [17].

**Fitness:** The extent to which the identified model can properly repeat the cases recorded in the log is measured by fitness [18]. Eq. 1 is the formula for measuring fitness quality.

$$Fitness = \frac{n\,(captured\_traces)}{n\,(traces\_in\_event\_log)} \quad (1)$$

**Precision:** Precision seeks to detect models that are overly generic and penalizes undesirable behavior [19]. Eq. 2 is the formula for measuring precision quality.

$$Precision = \frac{n\,(captured\_traces)}{n\,(traces\_in\_model)} \quad (2)$$

**Generalization:** The models generalization reveals how well it can support unexpected traces [20]. Eq. 3 is the formula for measuring generalization quality.

$$Generalization = 1 - \frac{\sum_1^{\#no}(\sqrt{\#execution})^{-1}}{\#no} \quad (3)$$

if $\#execution$<100 = $\frac{\#execution}{traceInEventLog} x100$

where:

$\#no$ = number of operators

$\#execution$ = number of operators executed in traces of the event log

$traceInEventLog$ = number of traces in the event log

**Simplicity:** In terms of readability, simplicity represents the complexity of a process model [18]. Eq. 4 is the formula for measuring generalization quality.

$$Simplicity = 1 - \frac{\#da+\#mo}{\#no} \quad (4)$$

where:

$\#da$ = number of redundant activities

$\#mo$ = number of operators not reflected in traces of the event log

$\#no$ = number of operators

## III. RESEARCH METHOD

The workflow in this research can be seen in Fig. 2 It consists of two steps. The first step is to discover process models and the second step is to measure its quality.

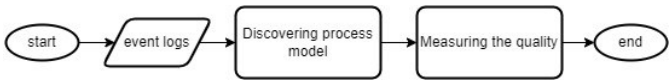



Fig. 2. Research Method Workflow

### A. *Discovering Process Model*

The first stage is to discover a process model. A graph model built from the event log loaded into Neo4j. The function used to load the event log is the create function. This create function is used because it creates a node based on all the data in the event log. Event logs will be contained in 2 labels, namely LogBase and LogModel. LogBase consists of all the columns contained in the event logs, while LogModel only consists of the CaseID column and the name of the activity. Fig. 3 displays the query structure for loading event logs in Neo4j.

```
LOAD CSV with headers FROM "file:///*.csv" AS line
create (:LogBase{CaseId, NameActivity, Time})

LOAD CSV with headers FROM "file:///*.csv" AS line
create (:LogModel{CaseId, NameActivity})
```

Fig. 3. Load event logs into Neo4j

The second step is to build a sequence relation in the process model. The sequence relation connects the node with the next node. The merge function is used to build a new relation based on a match with the node conditions. Fig. 4 describes the query structure to build a sequence relation between two consecutive nodes.

```
//CASE SEQUENCE DISCOVERY
MATCH (node1:LogBase)
WITH COLLECT(node1) AS Caselist
UNWIND RANGE(0,Size(Caselist) - 2) as idx
WITH Caselist[idx] AS list1, Caselist[idx+1] AS list2
MATCH (node2:LogModel), (node3: LogModel)
WHERE list1.CaseId = list2.CaseId AND list1.Name =
node2.Name AND list2.Name = node3.Name AND
list1.CaseId = node2.CaseId AND list2.CaseId = node3.CaseId
MERGE (node2)-[NEXT]->( node3)
```

Fig. 4. Discover sequence relations in Neo4j

The next step is discovery of the control-flow pattern. The control-flow patterns that will be formed are XOR, AND, Invisible Task, and Non-Free Choice relations.

There are two kinds of XOR relations, they are XORSPLIT and XORJOIN. XORSPLIT is formed when the number of relations leaving the origin node is more than one and the relations entering the destination node are equal to one. Meanwhile, XORJOIN is formed when the number of relations entering the destination node is more than one and the relations leaving the origin node are equal to one. If there is a node with such conditions, it will form a new relation and the old relation will be deleted. Fig. 5 shows the cypher structure to form XORSPLIT and XORJOIN relations in the graph process model.

Invisible task is formed when there are two relations that are connected but between the two relations there is no node that separates them. So, a new named the invisible task is created. Fig. 6 shows the cypher structure to make a new invisible task node in a graph-based process model.

```
//XOR SPLIT
MATCH (node1)-[r]->( node2)
WHERE size((node1)-->())>1 AND size((node2)<--())=1
AND ( size((node2)-->())=1 OR size((node2)-->())>1 )
CREATE (node1)-[:XORSPLIT]->( node2)
DELETE r

//XOR JOIN
MATCH (node1)-[r]->( node2)
WHERE ( size((node1)-->())=1 OR size((node1)-->())>1 )
AND size((node2)<--())>1
CREATE (node1)-[:XORJOIN]->( node2)
```

Fig. 5. Discover XOR relations in Neo4j

```
//Invisible Task
MATCH (node3)<-[r]-(node1)-[s]->(node2)
WHERE size((node2)<--())=1 AND size((node3)<--())>1
MERGE (i:Invisible_Task {Name: "Inv_Task1"})
MERGE (node1)-[:NEXT]->(i)
MERGE (i)-[:NEXT]->(node3)
DELETE r
```

Fig. 6. Discover invisible task in Neo4j

There are two kinds of AND relations, they are ANDSPLIT and ANDJOIN. ANDSPLIT is formed when the number of relations leaving the origin node is more than one and the number is the same as the relations that enter the destination node. While ANDJOIN is formed when the number of relations that enter the destination node is more than one and is equal to the number of relations that leave the origin node. If there is a node with such conditions, it will form a new relation and the old relation will be deleted. Fig. 7 shows the cypher structure to form ANDSPLIT and ANDJOIN relations in the graph process model.

```
//AND SPLIT
MATCH (node2)<-[r]-(node1)-[s]->(node3)
WHERE size((node1)-->())>1 AND size((node3)--
>())=size((node1)-->()) AND size((node2)-->()) =
size((node1)-->()) AND not (node2)-[:SEQUENCE]->(a)
AND not (node3)-[:SEQUENCE]->( node1)
MERGE (node1)-[:ANDSPLIT]->( node2)
MERGE (node1)-[:ANDSPLIT]->( node3)
DELETE r,s

//AND JOIN
MATCH (node2)-[r]->( node1)<-[s]-( node3)
MATCH (node2)-[t]-( node3)
WHERE size((node1)<--())>1 AND size((node3)--
>())=size((node1)<--()) AND size((node2)--
>())=size((node1)<--()) AND not ()-[:ANDSPLIT]->( node1)
MERGE (node1)<-[:ANDJOIN]-( node3)
MERGE (node1)<-[:ANDJOIN]-( node2)
DELETE r,s,t
```

Fig. 7. Discover AND relations in Neo4j

A Non-Free Choice relationship is indicated by the presence of a node whose previous relation is XORJOIN and the following relation is XORSPLIT. If there is a node with such conditions, the previous node and the next node are connected by a new relationship, named non-free-choice. Fig. 8 describes the cypher structure to find non-free-choice relations in the graph process model.

```
//Non Free Choice
MATCH (node1)-[b:XORJOIN]->()
MATCH ()-[c:XORSPLIT]->(node2)
MATCH (k:LogBase),(l: LogBase)
WHERE node1.Name<> node2.Name AND k.Name=
node1.Name AND l.Name= node2.Name AND
k.CaseId=l.CaseId AND k.Time<l.Time
MERGE (node1)-[:NONFREECHOICE]->( node2)
```

Fig. 8. Discover non-free-choice relations in Neo4j

### *B. Measuring Quality of Fitness and Precision*

Calculation of fitness and precision is done simultaneously. The calculation steps are as follows:

1. Calculates fitness / precision of a model process is done by calculating fitness / precision on each relation contained in the process model.
2. The formula for each relation is shown at Eq. 1 for fitness and Eq. 2 for precision.
3. Traces of each relation can be viewed based on the origin and destination nodes.
4. Overall fitness / precision formula:
   Sum of results of all relations divided by the number of relations.

Fig. 9 displays the cypher structure used to calculate the number of relationships that exist in the graph model process.

```
MATCH p=(a)-[r]->(b)
WITH COLLECT (DISTINCT type(r)) AS TypeCaselist, n
CREATE (n:Temp {Name: "Relation"})
WITH n, TypeCaselist, size(TypeCaselist) AS Count
SET n.countR = Count
RETURN TypeCaselist AS RelationType, Count
```

Fig. 9. Counting relation in process model

Fig. 10 displays the cypher structure that is used to see nodes that have sequence relations that exist in the graph model process.

```
MATCH (a:LogModel)-[r]->(b:LogModel)
WHERE type(r) = 'SEQUENCE'
MATCH (c:LogBase)-->(d: LogBase)
WHERE a.Name = c.Name
RETURN source_node, dest_node
```

Fig. 10. Discover sequence relation in process model and event log

Fig. 11 displays the cypher structure that is used to see nodes that have XORSPLIT relations which exist in the graph model process.

```
//XOR SPLIT in Process Model
MATCH (a:LogModel)-[r]->(b)
WHERE type(r) = 'XORSPLIT'
RETURN source_node, dest_node

//XOR SPLIT in Event Logs
MATCH (a:LogModel)-[r]->(b)
WHERE type(r) = 'XORSPLIT'
MATCH (c:LogBase)-->(d)
WHERE a.Name = c.Name
RETURN source_node, dest_node
```

Fig. 11. Discover XOR SPLIT relation in process model and event log

```
//XOR JOIN in Process Model
MATCH (node1)-[r]->(node2:LogModel)
WHERE type(r) = 'XORJOIN'
RETURN source_node, dest_node

//XOR JOIN in Event Logs
MATCH (node1)-[r]->(node2:LogModel)
WHERE type(r) = 'XORJOIN'
MATCH (node3)-->(node4:LogBase)
WHERE a.Name = c.Name
RETURN source_node, dest_node
```

Fig. 12. Discover XOR JOIN relation in process model and event log

Fig. 12 displays the cypher structure that is used to see nodes that have XORJOIN relations which exist in the graph model process. Fig. 13. displays the cypher structure that is used to see nodes that have ANDSPLIT relations which exist in the graph model process.

```
//AND SPLIT in Process Model
MATCH (a)-[r]->(b:LogModel)
WHERE type(r) = 'ANDSPLIT'
WITH COLLECT (b.Name) AS dest_pm, a
RETURN DISTINCT a.Name AS src_pm, dest_pm

//AND SPLIT in Event Logs
MATCH (a)-[r]->(b:LogModel)
WHERE type(r) = 'ANDSPLIT'
MATCH (c:LogBase)-->(d)-->(e)
WHERE b.Name = d.Name AND NOT c.Name = b.Name
RETURN DISTINCT c.Name AS scr_el, d.Name AS dest_el
```

Fig. 13. Discover AND SPLIT relation in process model and event log

Fig. 14 displays the cypher structure that is used to see nodes that have ANDJOIN relations which exist in the graph of model process.

```
//AND JOIN in Process Model
MATCH (a:LogModel)-[r]->(b)
WHERE type(r) = 'ANDJOIN'
WITH COLLECT (a.Name) AS src_pm, b
RETURN source_node, dest_node

//AND JOIN in Event Logs
MATCH (a)-[r]->(b:LogModel)
WHERE type(r) = 'ANDSPLIT'
MATCH (c:LogBase)-->(d)-->(e)
WHERE b.Name = d.Name AND NOT c.Name = b.Name
RETURN source_node, dest_node
```

Fig. 14. Discover AND JOIN relation in process model and event log

Fig. 15 displays the cypher structure that is used to see nodes that have non-free-choice relations in the graph of model process.

```
//NON FREE CHOICE in Process Model
MATCH (a:LogModel)-[r]->(b)
WHERE type(r) = 'NONFREECHOICE'
RETURN source_node, dest_node

//NON FREE CHOICE in Event Logs
MATCH (a:LogModel)-[r]->(b)
WHERE type(r) = 'NONFREECHOICE'
MATCH (c:LogBase)-->(d), (n: LogBase)-->(m: LogBase)
WHERE c.Name = a.Name AND d.Name = n.Name AND
c.CaseId = n.CaseId
RETURN source_node, dest_node
```

Fig. 15. Discover non-free-choice relation in process model and event log

### *C. Calculate Generalization*

Generalization calculation requires several steps. These stages include calculating the number of operators executed on the process model, calculating the total execution that is carried out, and calculating the generalization value.

The first stage is to count the number of operators executed by the process model. The operators are nodes AND relations. Node can be counted by counting the number of nodes that are executed for each activity name. Fig. 16 displays a cypher structure to count the number of nodes executed for each activity name. The relation is calculated based on any relation which exists in the tree process model. If the number of executions is less than 100, it is necessary to normalize all execution results.

```
MATCH (a:LogBase), (b:LogModel)
WHERE a.Name = b.Name
WITH count(a.Name) AS countA, b
SET b.count = countA
```

Fig. 16. Counting the number of nodes in the process model

Fig. 17 displays the cypher structure that is used to calculate the number of relations that came out of the initial node that was successfully executed.

```
MATCH (a:LogBase)-[r]->(), (b:LogModel)
WHERE NOT ()-->(a) AND a.Name=b.Name
WITH count(r) AS countA, b
```

Fig. 17. Counting many relationships that leave from the initial node

Fig. 18 displays the cypher structure that is used to calculate the number of relations that enter the end node that has been successfully executed.

```
MATCH ()-[r]->(a:LogBase), (b:LogModel)
WHERE NOT (a)-->() AND a.Name=b.Name
WITH count(a) AS countA, b
```

Fig. 18. Counting the number of incoming relations to the end node

Fig. 19 displays the cypher structure that is used to calculate the number of sequence relations which enter the successfully executed node.

```
MATCH (a:LogModel)-[r:SEQUENCE]->(b:LogModel),
(c:LogModel)-[s:SEQUENCE]->(d:LogModel)
WHERE a.Name = d.Name
WITH COLLECT (DISTINCT a) AS CaseCaselist
UNWIND RANGE(0,Size(CaseCaselist)-1) as idx
WITH CaseCaselist[idx] AS list1
MATCH (m:LogBase)-[s]->(n:LogBase)
WHERE n.Name = list1.Name
WITH count(s) AS countA, list1
SET list1.in = countA
```

Fig. 19. Counting the number of incoming relations to the sequence relation

Fig. 20 displays the cypher structure that is used to calculate the number of XOR relations that enter the successfully executed node.

```
MATCH (a)-[r:XORSPLIT]->(b)
WITH COLLECT (b) AS CaseCaselist, a
UNWIND RANGE(0,Size(CaseCaselist)-1) as idx
WITH CaseCaselist[idx] AS list1, a
MATCH (m:LogBase)-[s]->(n:LogBase)
WHERE n.Name = list1.Name
WITH count(s) AS countA, list1
SET list1.in = countA
```

Fig. 20. Counting the number of incoming relations to the XOR relation

Fig. 21 displays the cypher structure that is used to count the number of relations and incoming nodes that have been successfully executed.

```
MATCH (a)-[r:ANDSPLIT]->(b:CaseActivity)
WITH COLLECT (b) AS CaseCaselist
UNWIND RANGE(0,Size(CaseCaselist)-1) as idx
WITH CaseCaselist[idx] AS list1
MATCH (m:LogBase)-[s]->(n:LogBase)
WHERE n.Name = list1.Name
WITH count(s) AS countA, list1
SET list1.in = countA
```

Fig. 21. Counting the number of incoming relations to the AND relation

Fig. 22 displays the cypher structure used to calculate the total execution of all operators according to the generalization formula in Eq. 3.

```
MATCH (a:LogModel)
WITH a, sqrt(a.execute) AS Execution, -1 AS e
SET a.powerExecute = Execution^e
```

Fig. 22. Calculate Power Execute

Fig. 23 displays the cypher structure used to calculate the generalization value according to the generalization formula on Eq. 3.

```
MATCH (a:LogModel)
WITH sum(a.powerExecute) AS a, sum(a.powerIn) AS b,
sum(a.powerOut) AS c
MATCH (n:Temp)
WITH sum(n.powerExecute) AS d, a, b, c
WITH a+b+c+d AS sumall
RETURN 1 - (sumall/15) AS Generalization
```

Fig. 23. Calculate generalization

### D. Simplicity

Calculation of simplicity requires several stages. These stages include calculating redundant activity and calculating the number of operators that is not reflected in traces of the event log.

Calculating redundant activity can be seen based on the similarity of activity names in process models with different process id. If there is no node that has an activity name with a similarity equal to 100, then the value of the redundant activity is equal to 0. Fig. 24 displays the cypher structure to find redundant activity in the graph process model.

```
MATCH (a:CaseActivity)
MATCH (b:CaseActivity)
WHERE id(a) <> id(b)
WITH a, b, apoc.text.clean(a.Name) as norm1,
apoc.text.clean(b.Name) as norm2
with toInteger(apoc.text.jaroWinklerDistance(norm1,
norm2) * 100) as similarity, a, b
with a, b, similarity where similarity = 100
RETURN a.Name as aname, b.Name as bname,
```

Fig. 24. Discover redundant activity in the graph process model

The next step is the number of operators not reflected in traces of the event log. The number of operators not reflected in traces of the event log can be calculated based on the origin and destination nodes. If the origin and destination nodes in the event log are not the same as the origin and destination nodes in the process model, then the number of operators not reflected will increase. The number of operators is calculated based on any relation that exists in the tree process model. Fig. 25 displays the cypher structure used to calculate the simplicity value.

```
MATCH (a: LogModel)
RETURN 1 - ((a.DA + a.MO)/a.NO) AS SIMPLICITY
```

Fig. 25. Calculate the simplicity of graph process model

## IV. EXPERIMENT

### A. Materials

This research uses two case studies. They are waste handling and port container handling. The event log used must contain the CaseID, the activity name, and the time when the activity was done. In the event logs of waste handling there are 7 cases and 6 traces, while in the event log of port handling there are 30 cases and 3 traces.

### B. Result

The results of the discovery of the graph-based process model for the port container handling case study is shown in Fig. 27. The graph model contains sequence relations, XOR relations, invisible task, and non-free-choice relations.

The results of the discovery of a graph-based process model for the case study of waste handling is shown in Fig. 28. In the graph model, there are sequence, XOR relations, AND relations, invisible task, and non-free-choice relations.

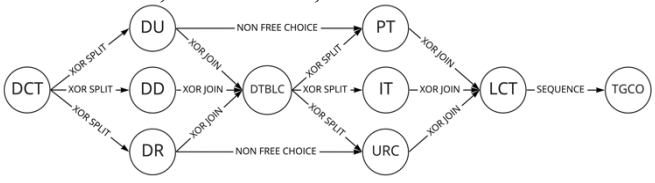



Fig. 27. Process graph model for port container handling case studies

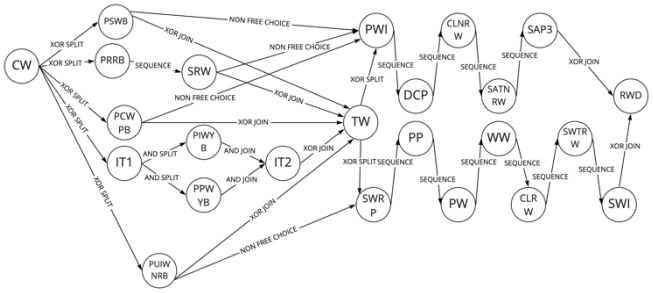



Fig. 28. Process graph model for case study of waste handling

After the process of the graph model is successfully discovered, the next step is to measure the quality of the graph process models. The quality measures considered are fitness, precision, generalization, and simplicity.

In the first case study, the port container handling, the results of the fitness and precision calculations are 1. The generalization value is equal to 0.96 and the simplicity value is 0.87. Meanwhile, for the waste handling case study, the results of the fitness calculation are 1. The precision value is 0.95. The result of the generalization value is equal to 0.83 and the simplicity value is equal to 0.95.

The results of the calculations above are the results of calculations performed by using graph-based quality measurement. To ensure the calculation results are correct, we also perform ETM quality measurement using the formula in Eq. 1 for fitness, Eq. 2 for precision, Eq. 3 for generalization, and Eq. 4 for simplicity. The calculations of generalization and simplicity are done using ETM quality measurement based on the tree model of the process that is formed. Table I and Table II displays the comparison of measurement quality on graph-based quality measurement and using ETM quality measurement.

TABLE I. RESULT OF MEASURMENT QUALITY FROM PORT CONTAINER HANDLING

| Method | Measurement Quality | | | |
|---|---|---|---|---|
| | ***Fitness*** | ***Precision*** | ***Generalization*** | ***Simplicity*** |
| Graph-based Quality Measurement | 1.00 | 1.00 | 0.96 | 0.87 |
| ETM quality measurement | 1.00 | 1.00 | 0.96 | 1.00 |

TABLE II. RESULT OF MEASURMENT QUALITY FROM WASTE HANDLING

| Method | Measurement Quality | | | |
|---|---|---|---|---|
| | ***Fitness*** | ***Precision*** | ***Generalization*** | ***Simplicity*** |
| Graph-based Quality Measurement | 1.00 | 0.95 | 0.83 | 0.95 |
| ETM quality measurement | 1.00 | 0.75 | 0.83 | 1.00 |

Based on the calculation results shown in Table I and Table II, it is shown that the results of fitness and generalization for both methods are the same. The results of the generalization calculation between the two methods can be the same because both are calculated based on the tree process model that was made previously. As for the simplicity calculation results, although both are also calculated based on the same process tree model, in the ETM quality measurement cannot detect Invisible Tasks whereas graph-based quality measurement can detect them. The results of the calculation of precision for the two methods are different because traces of ETM quality measurement is seen based on the entire

process model, while trace of graph-based quality measurement is based on each relationship that exists in the process model.

## V. CONCLUSION

This research proposes about Graph-based Quality Measurement Method. Measurement quality that is measured includes fitness, precision, generalization, and simplicity. This measurement is carried out by two methods, they are proposed Graph-based Quality Measurement and ETM quality measurement.

There are two case studies used in this research, namely case studies of port container handling and waste handling. Based on the research, the results of the fitness and precision port container handling using the proposed Graph-based Quality Measurement and ETM Quality Measurement methods are 1.00. Meanwhile, the generalization obtained from both methods is 0.96. Simplicity generated by the Proposed Graph-based Quality Measurement method is 0.87 hence the result of ETM quality measurement method is 1.00. In the case study of waste handling, based on both methods, the fitness is 1.00 and the generalization is 0.83. Precision obtained from the Proposed Graph-based Quality Measurement method is 0.95 and 0.75 from ETM quality measurement method. The simplicity obtained from the Graph-based Quality Measurement method is 0.95 henceforth from ETM quality measurement method is 1.00. Simplicity calculation results are different between the two methods. It is caused by the invisible task in the graph. The different result in precision caused by the different calculation approach between the two methods.

The results of this study are expected to be a guide for quality measurement of the business process model in the future where the implementation of quality measurement is done using a graph-based method.

## ACKNOWLEDGMENT

This research was funded by Lembaga Pengelola Dana Pendidikan (LPDP) under Riset Inovatif-Produktif (RISPRO) Invitation Program, the Indonesian Ministry of Education and Culture under Penelitian Terapan Unggulan Perguruan Tinggi (PTUPT) Program, and Institut Teknologi Sepuluh Nopember (ITS) under Penelitian Dana Departemen and project scheme of the Publication Writing and IPR Incentive Program (PPHKI).

## REFERENCES

[1] M. Klun and P. Trkman, "Business process management – at the crossroads," *Business Process Management Journal*, vol. 24, no. 3. Emerald Group Publishing Ltd., pp. 786–813, 2018. doi: 10.1108/BPMJ-11-2016-0226.

[2] P. Dymora, M. Koryl, and M. Mazurek, "Process discovery in business process management optimization," *Information (Switzerland)*, vol. 10, no. 9, Sep. 2019, doi: 10.3390/info10090270.

[3] D. Schuster, S. J. van Zelst, and W. M. P. van der Aalst, "Cortado—An Interactive Tool for Data-Driven Process Discovery and Modeling," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2021, vol. 12734 LNCS. doi: 10.1007/978-3-030-76983-3_23.

[4] R. Sarno, K. R. Sungkono, M. Taufiqulsa'di, H. Darmawan, A. Fahmi, and K. Triyana, "Improving efficiency for discovering business processes containing invisible tasks in non-free choice," *Journal of Big Data*, vol. 8, no. 1, 2021, doi: 10.1186/s40537-021-00487-x.

[5] K. R. Sungkono, R. Sarno, F. D. Amadea, and A. F. Irbah, "Graph-Based Process Model Discovery for Mining Invisible Non-Prime Tasks in Hybrid Choice-Parallel Relationships," *International Journal of Intelligent Engineering and Systems*, vol. 14, no. 3, 2021, doi: 10.22266/ijies2021.0630.35.

[6] 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, 2020, doi: 10.22266/ijies2020.0430.13.

[7] F. Mannhardt, M. de Leoni, H. A. Reijers, and W. M. P. van der Aalst, "Data-driven process discovery - Revealing conditional infrequent behavior from event logs," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2017, vol. 10253 LNCS. doi: 10.1007/978-3-319-59536-8_34.

[8] A. Berti, S. J. van Zelst, W. M. P. van der Aalst, and F. Gesellschaf, "Process mining for python (PM4py): Bridging the gap between process- And data science," in *CEUR Workshop Proceedings*, 2019, vol. 2374.

[9] H. R'Bigui and C. Cho, "The state-of-the-art of business process mining challenges," *International Journal of Business Process Integration and Management*, vol. 8, no. 4, 2017, doi: 10.1504/IJBPIM.2017.10009731.

[10] 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, 2019, doi: 10.26555/ijain.v5i2.296.

[11] A. Rozinat, M. Veloso, and W. M. P. van der Aalst, "Evaluating the Quality of Discovered Process Models," *Information Systems Journal*, vol. 16, no. Section 2, 2008.

[12] Y. A. Effendi, R. Sarno, and D. V. Marsha, "Improved fuzzy miner algorithm for business process discovery," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 19, no. 6, 2021, doi: 10.12928/TELKOMNIKA.v19i6.19015.

[13] J. Guia, V. G. Soares, and J. Bernardino, "Graph databases: Neo4j Analysis," in *ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems*, 2017, vol. 1. doi: 10.5220/0006356003510356.

[14] 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, 2016, doi: 10.15866/irecos.v11i6.9555.

[15] van Dongen, Boudewijn F., et al. "Conformance checking of mixed-paradigm process models." Information Systems 102 (2021): 101685.

[16] W. L. J. Lee, H. M. W. Verbeek, J. Munoz-Gama, W. M. P. van der Aalst, and M. Sepúlveda, "Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining," *Information Sciences*, vol. 466, 2018, doi: 10.1016/j.ins.2018.07.026.

[17] B. F. van Dongen, J. de Smedt, C. di Ciccio, and J. Mendling, "Conformance checking of mixed-paradigm process models," *Information Systems*, vol. 102, 2021, doi: 10.1016/j.is.2020.101685.

[18] A. Bogarín, R. Cerezo, and C. Romero, "Discovering learning processes using inductive miner: A case study with learning management systems (LMSs)," *Psicothema*, vol. 30, no. 3, 2018, doi: 10.7334/psicothema2018.116.

[19] C. Rinner, E. Helm, R. Dunkl, H. Kittler, and S. Rinderle-Ma, "Process mining and conformance checking of long running processes in the context of melanoma surveillance," *International Journal of Environmental Research and Public Health*, vol. 15, no. 12, 2018, doi: 10.3390/ijerph15122809.

[20] A. Polyvyanyy *et al.*, "Entropia: A family of entropy-based conformance checking measures for process mining," in *CEUR Workshop Proceedings*, 2020, vol. 2703.