Received: December 3, 2019. Revised: January 2, 2020.

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

An Improved Method of Parallel Model Detection for Graph-Based Process Model Discovery

Indra Waspada[1,2*] **Riyanarto Sarno**[1] **Kelly Rossa Sungkono**[1]

[1]*Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia*
[2]*Department of Informatics, Universitas Diponegoro, Semarang, Indonesia*
* Corresponding author's Email: indrawaspada@lecturer.undip.ac.id

Abstract: The existing method of graph-based process model discovery has weaknesses in detecting parallel relationship (XOR, AND, and OR). The algorithm only works on a particular graph structure, so it must be reconfigured when applied to other different structures. To answer this problem, this paper proposes an improved method of parallel model detection, which is designed in two phases. The first one consists of three steps; firstly is to count and record the value of relationship frequency into every node in a graph model. Then, the second step implements the algorithm to discover the concurrent relationship. The third step detects all possible split and join relationships. Based on the first phase, then a consistent and robust parallel discovery algorithm can be developed. The first parallel algorithm is to identify the XOR relationship. This algorithm is designed with the rule that the XOR pattern cannot have a concurrent relationship between its branch nodes. Next, the algorithm for detecting AND and OR must detect the existence of any concurrent relationship in its branches. Then, AND and OR pattern is differentiated by their unique characteristic of relationship frequency at branch nodes. To verify the ability of the proposed methods in which the existing method fails, we have designed four scenarios. Scenario 1 and 2 consecutively were arranged with two and three branches parallel model. Scenario 3 located the AND and OR inside the XOR pattern. In scenario 4 the sequence relationships were inserted between split and join of parallel patterns. The experimental results show that the proposed method successfully recognizes and differentiates XOR, AND and OR patterns correctly in all scenarios. It also sounds in all discovered model and get 100% fitness.

Keywords: Parallel model detection, Graph-based process model discovery, Relationship frequency, Concurrent relationship.

1. Introduction

The automatic process model discovery of the event log is an important aspect of the organization [1]. The produced process model is a real reflection of the field conditions obtained based on the event log [2]. The discovered model can be beneficial, starting from inspection and finding valuable insights to observing the conformance with the reference model.

Several methods are known for discovering business process models from event logs, including alpha miner [2], heuristic miner [3], inductive miner [4], fuzzy miner [5], split miner [6, 7], and graph-based miner [8, 9]. Graph-based miner algorithms outperform others in lower time complexity [10]. This algorithm is applied to Neo4j graph database, which stores activities and relationships. Several studies have used graph-based discovery models to detect anomalies [11, 12], the model also able to be combined with data perspectives for decision mining [13].

In previous graph-based miner studies, several algorithms have been introduced to find process models, including detecting parallel processes [8, 11], recognize and insert the invisible task [9], and discovering Non-free Choice [14].

The existing method use indegree and outdegree of nodes depicted in the Process Model as the basis of parallel detection. This approach encounters some difficulties when the structures of two or more patterns in the model are figured out similarly, i.e., a concurrent relationship is the same as a split or join pattern, OR pattern looks similarly with AND pattern, etc. Another problem also arises when the concurrent relationship potentially disguises the parallel pattern that makes it unable to be detected. So it is clear that the existing method has a weakness in the ability to discover parallel process models.

To overcome these weaknesses, we propose a more reliable method by improving the previous one as the contribution of this paper. There are six main contributions of this research; they are:

1. Introducing relation frequency values for every relation in the graph sequence model. These values then are summed up and labeled to every node as an input relationship frequency and output relationship frequency. Both kinds of node's frequency are utilized to differentiate between AND and OR.
2. Proposing an algorithm for detecting graph-based concurrent relationships. The proposed algorithm accommodates the difference between a concurrent relationship with a short-loop pattern by investigating the pattern found in the model with the real condition in the trace.
3. Detecting all possible split and join relationships as the foundation of next step parallel pattern detection.
4. Proposing an algorithm to detect an XOR relationship based on the absence of concurrent relations
5. Proposing an algorithm to detect AND relations based on input and output relationship frequency value between its branches nodes.
6. Proposing an algorithm to detect OR relations based on input and output relationship frequency value between its branches nodes.

The top three contributions are the first phase that aims as a means of supporting the algorithm of parallel process discovery. The next three steps are the second phase, which is the parallel process discovery algorithms. All six are designed and implemented to produce reliable methods used in any parallel cases.

This study examines the ability of our new proposed method in several scenarios of event log cases. The results obtained from all experiments indicate that the proposed method successfully detects and distinguishes the XOR, AND and OR parallel relations in all cases.

The next four sections organized as follows. Definitions and existing method are discussed in Section 2. The proposed method is discussed in Section 3. We then discuss and summarize the results of some scenarios for parallel process discovery in Section 4. Finally, the conclusion of this work is presented in Section 5.

2. Research method

2.1 Automated process discovery

Automatic process model discovery techniques utilize event logs as input and generate business process models that closely match behavior observed in the event log or implied by traces in the event log. The event log is obtained from business activities.

Definition 1 (Event, attribute): Given a set of all possible event $\mathcal{E}$, and let AN be a set of attribute names. For any event $e \in \mathcal{E}$ and name $n \in AN$, $\#_n(e)$ is the value of attribute n for event e. So $\#_{activity}(e)$ is the activity associated with event e, and $\#_{time}(e)$ is the timestamp of event e.

The event log consists of cases and cases consist of events. Events for a case are represented in the form of a trace, e.g., a sequence of unique events. In addition, cases, such as events, can have attributes.

Definition 2 (Case, trace, event log): Given a set of case $\mathcal{C}$, for any case $c \in \mathcal{C}$ and name $n \in AN$: $\#\_n(c)$ is the value of attribute n for case c. Each case has a special mandatory attribute trace, $\#_{trace}(c) \in \mathcal{E}^*$. $\hat{c} = \#_{trace}(c)$ refers to the trace of a case. A trace is a finite sequence of events $\sigma \in \mathcal{E}^*$ such that each event appears only once. An event log is a set of cases $L \subseteq \mathcal{C}$ such that each event appears at most once in the entire log.

2.2 Graph database

A graph database is a database management system with Create, Read, Update, and Delete methods that expose a graph data model [15]. The structure of the data looks like a directed graph in mathematics. The graph database consists of nodes and vertices. A node is a point that contains all information from an object, while vertices represent the relationship between objects.

Graph databases are more flexible than relational databases because they can be developed without the need for any adjustments or changes to the initial design. Relationships of graph databases are stored at the level of individual records, whereas in a relational database the structure is defined at a higher level (table definition).

Received: December 3, 2019. Revised: January 2, 2020.

2.3 Graph-based process model

Process modeling helps us to understand the process better and identify and prevent problems from occurring. Process models are a requirement for analyzing, redesigning, or process automation [16].

A graph-based process model is a process model that is obtained through a graph-based algorithm applied to a graph database. The advantage of implementing a graph-based algorithm in business process modeling is because the graph database stores not only activities but also relationships so that the algorithm designed can produce a low time complexity [10].

2.4 Parallel process model

Activities in the process model have relations with other activities. If an activity in the event log is obtained always and only followed by one type of the same activity then the type of relationship is sequential. Whereas the parallel type is when there is more than one different type of activity connected to the same activity.

To illustrate the relationship between the event log and the corresponding parallel model. Table 1 was showed an example of some event log trace for sequential and parallel relations. Then each relation is depicted using YAWL notation and graph (Neo4j). Table 2 describes the sequential relationship. Table 3 presents the XOR parallel relationship. Table 4 is the AND relationship, and the OR relationship in Table 5.

Table 1. Sample of traces

Patterns	Sample of traces in event log
Sequence	{Act_1,Act_2},{Act_1,Act_2}, {Act_1,Act_2}
XOR	{Act_1,**Act_2**,Act_5}, {Act_1,**Act_3**,Act_5}, {Act_1,**Act_4**,Act_5}
AND	{Act_1,**Act_2,Act_3,Act_4**,Act_5}, {Act_1,**Act_4,Act_3,Act_2**,Act_5}, {Act_1,**Act_4,Act_2,Act_3**,Act_5}, {Act_1,**Act_3,Act_2,Act_4**,Act_5}
OR	{Act_1,**Act_2,Act_3**,Act_5}, {Act_1,**Act_3,Act_4**,Act_5}, {Act_1,**Act_4,Act_2**,Act_5}, {Act_1,**Act_4,Act_3,Act_2**,Act_5}, {Act_1,**Act_2,Act_4,Act_3**,Act_5}

Table 2. Sequence process model in YAWL and Graph

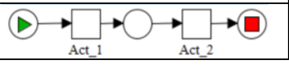
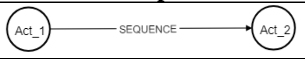
YAWL	Graph
	

Table 3. XOR relationship of Parallel process model

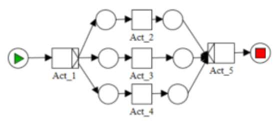
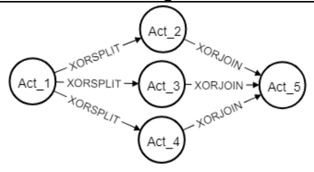
YAWL	Graph
	 Act_1 XORSPLIT XORSPLIT XORSPLIT Act_2 Act_3 Act_4 XORJOIN XORJOIN XORJOIN Act_5

Table 4. AND relationship of parallel process model

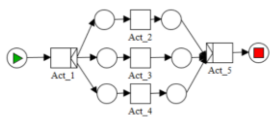
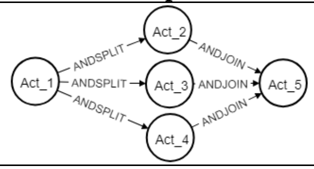
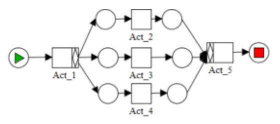
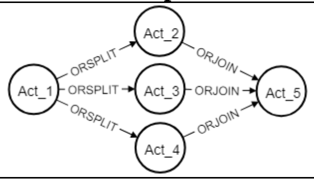
YAWL	Graph
	 Act_1 ANDSPLIT ANDSPLIT ANDSPLIT Act_2 Act_3 Act_4 ANDJOIN ANDJOIN ANDJOIN Act_5

Table 5. OR relationship of parallel process model

YAWL	Graph
	 Act_1 ORSPLIT ORSPLIT ORSPLIT Act_2 Act_3 Act_4 ORJOIN ORJOIN ORJOIN Act_5

2.5 Existing graph-based method for parallel process discovery

The following will be discussed the algorithms used in [8–11, 13, 14]. These algorithms are executed in 3 steps:

1. *Load the event log as graph nodes*. At this stage, the event log data is needed to be loaded into the graph database as two types of nodes. The first type of node is Trace. It represents the **process instant** of all events (with its accompanying attributes) from the log. The second type of node is **Model**. This node will be used to mount each unique activity name (with its supporting properties) as a **process mode**l.
2. *Create a sequence relationship model*. By referring to the definition of a directly-follows graph (DFG) [17] as depicted in Definition 3 , we create DFG in graph database by matching a sequential pair of nodes in each unique case id. A relationship is used to connect both nodes, and then they are named as Sequence Relationship. The detailed algorithm explained in Table 6.

Definition 3 (Directly-Follows Graph): With an event log L where $L \in \mathbb{B}(\mathcal{A}^*)$, the directly-follows graph of L is written as $G(L) = (A_L, \mapsto_L, A_L^{start}, A_L^{end})$ with:

- $A_L = \{a \in \sigma \mid \sigma \in L\}$ is the set of activities in L

- $\mapsto_L = \{(a,b) \in A\ x\ A\ |a >_L b\}$ is the directly follows relation,
- $A_L^{start} = \{a \in A\ |\exists_{\sigma \in L^a} = first(\sigma)\}$ is the set of start activities, and
- $A_L^{end} = \{a \in A\ |\exists_{\sigma \in L^a} = last(\sigma)\}$ is the set of end activities

A directly-follows graph $G(L).\ a \mapsto_L b$ exist if a was directly followed by b somewhere in any sub log L.

3. *Parallel detection and creation*. The final step is the implementation of the algorithm for parallel discovery. The main idea of the existing algorithm is utilizing the indegree and outdegree of nodes which is depicted in the process model. Based on it, then the XOR-Split can be discovered with a rule that every branch should have $indegree = 1$ and $outdegree \geq 1$. The algorithm for detecting and creating this XOR relationship detailed in Table 7.

   The algorithm of AND-Split detector is characterized with a rule that all outdegree of nodes in AND pattern should be equal, as we can see in Table 8. For OR-Split pattern, the algorithm set a rule to detect a condition of $outdegree_{branch} < outdegree_{gateway}$ as the characteristics. The OR algorithm can be seen in Table 9.

Table 6. Existing method for Sequence discovery

Relation ship	Pseudocode	Cipher Syntax
Sequ-ence	For idx as index of records of all activities in the traces:	`MATCH (c:Activity) WITH COLLECT(c) AS Caselist UNWIND RANGE(0,Size(Caselist) - 2) as idx`
	s1 = nodes[idx], s2 = nodes[idx+1]	`WITH Caselist[idx] AS s1, Caselist[idx+1] AS s2`
	For "a" as each node in the model, and "b" as each node in the model:	`MATCH (a:CaseActivity),(b:CaseActivity)`
	If s1.Case_ID = s2.CaseID, and s1.Activity = a.Activity, and s2.Activity = b.activity:	`WHERE s1.CaseId = s2.CaseId AND s1.Name = a.Name AND s2.Name = b.Name`
	Create sequence relationship between node a and node b	`MERGE (a)-[r:SEQUENCE]->(b)`

Table 7. Existing method for XOR relationship discovery

Relation ship	Pseudocode	Cipher Syntax
XOR Split	For nodes a and b and their relationships:	`MATCH (a)-[r]->(b)`
	If : the outdegree of a> 1 and the indegree of b = 1	`WHERE size((a)-->())>1 AND size((b)<--())=1`
	and the outdegree of b> = 1:	`AND (size((b)-->())=1 OR size((b)-->())>1 )`
	Create an XORSplit relation between nodes a and b	`CREATE (a)-[:XORSPLIT]->(b) DELETE r`
XOR Join	For nodes a and b and their relationships:	`MATCH (b)-[r]->(a)`
	If the output relation a> = 1 and the input relation b> 1:	`WHERE size((a)<--())>1 AND ( size((b)-->())=1)`
	Create an XORJoin relation between nodes a and b	`CREATE (b)-[:XORJOIN]->(a) DELETE r`

Table 8. Existing method for AND relationship discovery

Relation ship	Pseudocode	Cipher Syntax
AND Split	For nodes a, b, c and their relationships:	`MATCH (a)<-[r]-(b)-[s]->(c)`
	If: the outdegree of b> 1,	`WHERE size((b)-->())>1`
	and the outdegree of c = outdegree of b = outdegree of a,	`AND size((c)-->())=size((b)-->()) AND size((a)-->())=size((b)-->())`
	and b is not the next node of a or b:	`AND not (a)-[:SEQUENCE]->(b) AND not (c)-[:SEQUENCE]->(b)`
	Create an ANDSplit relation between b and a and between b and c	`MERGE (a)<-[:ANDSPLIT]-(b) DELETE r,s`

AND Join	For nodes a, b, c and their relationships:	`MATCH (a)-[r]->(b)<-[s]-(c)`
	If : the indegree of b> 1, and the indegree of c = indegree of b = indegree of a,	`WHERE size((b)<--())>1 AND size((c)-->())=size((b)<--()) AND size((a)-->())=size((b)<--())`
	Create AND Join relation between a with b and between c and b	`MERGE (a)-[:ANDJOIN]->(b) DELETE r,s`

Table 9. Existing method for OR relationship discovery

Relation ship	Pseudocode	Cipher Syntax
OR Split	For nodes a, n, b and their relationships:	`MATCH (a)<-[r]-(n)-[k]->(b)`
	If : the outdegree of b < the outdegree of n, and the outdegree of b > 1,	`WHERE (size((b)-->())< size((n)-->()) and (size((b)-->())>1))`
	and n is not the next node of nodes a and b:	`and not (b)-[:SEQUENCE]->(n) and not (a)-[:SEQUENCE]->(n)`
	Create an ORSplit relation between nodes a and n nodes b	`MERGE (n)-[:ORSPLIT]->(a) MERGE (n)-[:ORSPLIT]->(b) DELETE r,k`
OR Join	For nodes a, n, b and their relationships:	`MATCH (a)-[r]->(n)<-[k]-(b)-[l]->(a)`
	If : the indegree of n> 1 and the outdegree of b <the indegree of n, and the outdegree of b> 1:	`WHERE size((n)<--())>1 and ( size((b)-->()) < size((n)<--()) and size((b)-->()) > 1 )`
	Create an ORJoin relation between node a and n and between node b and n	`MERGE (a)-[:ORJOIN]->(n) MERGE (b)-[:ORJOIN]->(n) DELETE r,k,l`

3. The proposed method

Our proposed method following 9 steps which start with getting the input of an event log data and end with the resulting output of the graph-based process model.

1. *Load event logs as graph nodes*. This step is similar to the existing method as described in section 2.5.
2. *Create a sequential relationship and its frequency*. Here we add two things to the existing method. First, the directly-follows graph is not only created in the process model but also in the traces. This trace with sequence pattern will make benefit when we use it for matching the purpose, i.e, in concurrent relationship detection. Second, each time a relationship established in the graph model, a frequency counter of the relation also increased. This kind of value is termed in [6] as directly-follows frequency (DFF) as defined in definition 4.

Definition 4 (Directly-Follows Frequency): Given an event log $\mathcal{E}$, and two events label l_1,l_2 $\in L$, the directly-follows frequency between l_1 and l_2 ($|l_1 \rightarrow l_2|$) is $\left|\left\{\left(e_i, e_j\right) \in \mathcal{E} \; x \; \mathcal{E} \mid e_i^l = l_1 \wedge e_j^l = l_2 \wedge \exists_t \in \mathcal{E} \left[\exists e_x \in t\left[e_x = e_i \wedge e_{x+1} = e_j\right]\right]\right\}\right|$

Having two additional conditions, then we design the algorithm in Table 10.

3. *Counting the frequency relationship of each node*. Based on the DFF value, it is summed up in every node as their value of input frequency relations (ifr) and output frequency relations (ofr). The algorithm is presented in Tables 11 and 12.
4. *Identify a concurrent relationship*. Concurrent relations are detected when there is a direct relationship from node A to node B and vice versa. This condition can arise in models because in reality both of them actually work in parallel. It is necessary to pay attention that in the graph process model, the pattern of concurrent relations looks similar to the short-loops pattern. We use the definition of short-loop and concurrent relationships as defined in [6]. So that the algorithm we designed must be able to distinguish between the two. To distinguish them we search the candidate pattern in the graph-based process model and do pattern matching with its real trace in graph-based process instances. The detailed algorithm is presented in Table 13.

Definition 5 (Short-loop): Given two activities a and b, a short-loop ($a \circlearrowright b$) exists iff meet the requirement in (1) and (2) [6].

$$|a \rightarrow a| = 0 \wedge |b \rightarrow b| = 0 \qquad (1)$$

$$|a \leftrightarrow b| + |b \leftrightarrow a| \neq 0 \qquad (2)$$

The rule in Condition 1 gives a constraint that a and b are not allowed in a self-loop. Condition 2 ensure the pattern of short-loop $a \circlearrowright b$.

Definition 6 (Concurrent relationship): Given activities a and b, they are concurrent $a||b$ iff meet condition in (3),(4), and (5) [6].

$$|a \rightarrow b| > 0 \ \wedge \ |b \rightarrow a| > 0 \qquad (3)$$

$$|a \leftrightarrow b| + |b \leftrightarrow a| = 0 \qquad (4)$$

$$\frac{\left||a\rightarrow b| - |b\rightarrow a|\right|}{|a\rightarrow b| + |b\rightarrow a|} < \varepsilon \quad (\varepsilon \in [0,1]) \qquad (5)$$

Condition 3 is the main prerequisite for $a||b$. Condition 4 is the requirement to avoid a short-loop. Condition 5 is required when we require the frequency of both directions of concurrent relationships are in specific similar (threshold, ε) value.

5. *Identify potential Split and Join relationship.* All sequential relations which are split or joint relationship need to be recognized. With the concurrent relationship was detected in the previous step, we can make a rule for Split and Join that must in sequential relations. These Split and Join are the main foundation of the parallel detection algorithm being executed in the next phase. The algorithm for split relations is presented in Table 14 and the join relationship can be seen in Table 15.
6. *XOR identification and creation*. The XOR relation can be identified by utilizing its unique characteristics compared to other parallel relations. This algorithm is designed with the requirement that between branch nodes in the XOR relation cannot have a concurrent relationship. The proposed algorithm for XOR identification and creation is depicted in Table 16.
7. *AND identification and creation*. Both algorithms of detecting AND and OR require concurrent relationships in their branches, but the AND pattern is characterized by the condition when the relationship frequency (ifr or ofr) values at the branches is equal. The proposed algorithm is described in Table 17.
8. *OR identification and creation*. The algorithm to find OR relationship is design by detecting concurrent relationships in branches and having a branching pattern with gateway nodes that have a higher relation frequency value than each branch node. A more detail description of the proposed algorithm can be seen in Table 18.
9. *Remove all Concurrent relationships*. We remove all concurrent relationship process model.

Table 10. Proposed algorithm for Sequence discovery

Relationship	Pseudocode	Cipher Syntax
Sequence	For idx as index of records of all activities in the trace of process instant:	`MATCH (c:Trace) WITH COLLECT(c) AS Caselist UNWIND RANGE(0,Size(Caselist) - 2) as idx`
	Prepare the DFG from events in s1 and s2 where ($s_1 \mapsto s_2$)	`WITH Caselist[idx] AS s1, Caselist[idx+1] AS s2`
	For all nodes with label Model are assigned to a and b:	`MATCH (a:Model) MATCH (b:Model)`
	If: Both events in same Case Id,	`WHERE s1.CaseId = s2.CaseId`
	Find the corresponding activity in the Model with the same name as S1 name	`AND s1.Name = a.Name`
	Find the corresponding activity in the Model with same name S2 name	`AND s2.Name = b.Name`
	Create sequence relationship in **process instant**	`MERGE (s1)-[q:SEQUENCE {dff:0}]->(s2)`
	Create sequence relationship in **process model**	`MERGE (a)-[r:SEQUENCE {dff:0}]->(b)`
	Count DFF in Model	`with a,r,b,count((a)-->(b)) as dff set r.dff = dff`

Table 11. Counting input relationship frequencies

Pseudocode	Cipher Syntax
For all nodes a in process model with pattern ($a \mapsfrom x$) and input relation i:	`match (a:Model)<-[i]-(x)`
Sum up the frequency from all input relation	`with a, collect(i) as ilist, sum(i.dff) as ifr`
Update node a with the total input frequency value	`set a.ifr = ifr;`

Table 12. Counting output relationship frequencies

Pseudocode	Cipher Syntax
For all nodes a in process model with pattern ($a \mapsto x$) and output relation o:	`match (a:Model)-[o]->(x)`
Sum up the frequency from all output relation	`with a, collect(o) as olist, sum(o.dff) as ofr`
Update node a with the total output frequency value	`set a.ofr = ofr;`

Table 13. Concurrent relationship detection

Relationship	Pseudocode	Cipher Syntax
Concurrent relationship	For nodes with pattern ($x \mapsto y \mapsto z$) in Trace	`MATCH (x:Trace)-[p:SEQUENCE]->(y:Trace)-[q:SEQUENCE]->(z:Trace)`
	For nodes with pattern ($a \mapsto b \mapsto c$) in Model	`MATCH (a:Model)-[r:SEQUENCE]->(b:Model)-[s:SEQUENCE]->(c:Model)`
	If in Model: c = a, If (x=a) in Trace: x <> z	`WHERE c.Name = a.Name AND x.Name = a.Name AND x.Name <> z.Name`
	Create concurrent relationship in Model	`MERGE (a)-[:CONCURRENT {dff:r.dff}]->(b) DELETE r`

Table 14. Split relationship detection

Pseudocode	Cipher Syntax
For all nodes n in process model with pattern ($a \mapsfrom n \mapsto b$) with SEQUENCE relationship:	`MATCH (a:Model)<-[r:SEQUENCE]-(n)-[s:SEQUENCE]->(b)`
Update the relationship into SPLIT	`WITH a,count(a) as asum,n,r,s MERGE (a)<-[:SPLIT {dff:r.dff}]-(n) DELETE r`

Table 15. Join relationship detection

Pseudocode	Cipher Syntax
For all nodes n in process model with pattern ($a \mapsto n \mapsfrom b$) with SEQUENCE relationship:	`MATCH (a:Model)-[r:SEQUENCE]->(n)<-[s:SEQUENCE]-(b)`
Update the relationship into JOIN	`WITH a,count(a) as asum,n,r,s MERGE (a)-[:JOIN {dff:r.dff}]->(n) DELETE r`

Table 16. Proposed algorithm for XOR relationship detection

Relationship	Pseudocode	Cipher Syntax
XOR Split	For nodes with pattern ($a \mapsfrom n \mapsto b$) in Model and have Split relationship:	`MATCH (a:Model)<-[r:SPLIT]-(n)-[:SPLIT]->(b)`
	If: a and b **do not have** a CONCURRENT relationship:	`WHERE NOT (a)<-[:CONCURRENT]->(b)`
	Create XORSPLIT relationship between a and n	`WITH a,count(a) as asum,n,r MERGE (n)-[:XORSPLIT {dff:r.dff}]->(a) DELETE r`
XOR Join	For nodes with pattern ($a \mapsto b \mapsfrom c$) in Model and have a Join relationship:	`MATCH (a:Model)-[r:JOIN]->(n)<-[:JOIN]-(b)`
	If: a and b **do not have** a CONCURRENT relationship:	`WHERE AND NOT (a)<-[:CONCURRENT]->(b)`
	Create XORJOIN relationship between a and n	`WITH a,count(a) as asum,n,r MERGE (n)<-[:XORJOIN {dff:r.dff}]-(a) DELETE r`

Table 17. Proposed algorithm for AND relationship detection

Relationship	Pseudocode	Cipher Syntax
AND Split	For nodes with pattern ($a \mapsfrom b \mapsto c$) in Model and Split relationship:	`MATCH (a: Model)<-[r:SPLIT]-(n)-[:SPLIT]->(b)`

Received: December 3, 2019. Revised: January 2, 2020.

	If
	a has a concurrent relationship with b,
		`WHERE`
		
		`(a)<-[:CONCURRENT]->(b)`

	and all branches nodes' frequency is the same,	`AND(a.ifr)=(b.ifr)=(n.ofr)`

	and no incoming relation to node from its branch:	`AND`
		`(not ((a)-->(n))`
		`AND not ((b)-->(n)) )`

	Create ANDSPLIT relationship between a and n	`WITH a, count(a) as asum, n, r`
		`MERGE (a)<-[:ANDSPLIT {dff:r.dff}]-(n)`
		`DELETE r`

AND Join	For nodes with pattern ($a \mapsto b \mapsfrom c$) in Model and Join relationship:	`MATCH (a: Model)-[r:JOIN]->(n)<-[:JOIN]-(b)`

	If
	a has concurrent relationship with b,
		`WHERE`
		`(a)<-[:CONCURRENT]->(b)`

	and all branches nodes' frequency is the same,	`AND(a.ofr)=(b.ofr)=(n.ifr)`

	and no outgoing relation from node to its branch:	`AND`
		`(not ((a)<--(n))`
		`AND not ((b)<--(n)) )`

	Create ANDJOIN relationship between a and n	`WITH a, count(a) as asum, n, r`
		`MERGE (a)-[:ANDJOIN {dff:r.dff}]->(n)`
		`DELETE r`

Table 18. Proposed algorithm for OR relationship detection

Relationship	Pseudocode	Cipher Syntax
OR Split	For nodes with pattern a<=n=>b:	`match (a: Model)<-[r:SPLIT]-(n)-->(b)`

	If:
	a has concurrent relationship with *b*,
		`where`
		`(a)<-[:CONCURRENT]->(b)`

	and output relationship frequencies of gateway's node is larger than each of its branch node's relationship frequencies input,	`AND (a.ifr)<n.ofr`
		`AND (b.ifr)<n.ofr`

	and no incoming relation to node from its branch:	`AND`
		`(not ((a)-->(n))`
		`AND not ((b)-->(n)))`

	Create ORSPLIT relationship between a and n	`WITH a, count(a) as asum, r, n`
		`MERGE (n)-[:ORSPLIT {dff:r.dff}]->(a)`
		`DELETE r`

OR Join	For nodes with pattern ($a \mapsto b \mapsfrom c$) in Model and Join relationship:	`MATCH (a: Model)-[r:JOIN]->(x)<--(b)`

	If:
	a has a concurrent relationship with b, and Not ANDJOIN:
		`where`
		`(a)<-[:CONCURRENT]->(b)`

	and input relationship frequencies of gateway's node is larger than each of its branch node's relationship frequencies output,	`AND (a.ofr)<x.ifr`
		`and (b.ofr)<x.ifr`

	and no outgoing relation from node to its branch:	`AND (not ((a)<--(n)) AND not ((b)<--(n)) )`

	Create ORSPLIT relationship between a and n	`with a, count(a) as asum, x, r`
		`MERGE (a)-[:ORJOIN {dff:r.dff}]->(x)`
		`DELETE r`

4. Results and discussion

Experiments to compare and prove the ability of the proposed method in detecting parallel processes are carried out through several scenarios that provide event log data to identify XOR, AND, and OR types. Scenario 1 is a case with a 2-branch structure, scenario 2 uses 3-branches, scenario 3 with a nested structure, and scenario 4 inserts a sequential relation in a parallel pattern. For each scenario, three graph model visualizations are displayed, which are the directly-follows frequency result, parallel discovery using the existing method, and parallel discovery using the proposed method. We use some symbols to mark the results of the experiments. The symbols state in Table 19.

In order to get the soundness and fitness of each discovered model, we utilize prom tool. Prom provides some plugin, i.e., the Woflan [18] plugin to analyze the soundness, and conformance checking of DPN [19] plugin to get the fitness.

Table 19. Symbols to mark the experiment result

Symbol	Meaning
√	The pattern succeed to recognize
x	The pattern fail to recognize
*[pattern]	The expected pattern is recognize as another pattern

4.1 Scenario 1

Scenario 1 is an event log that contains XOR, AND, and OR parallel processes with two branches. The results of sequential pair detection can be seen in Fig. 1. Here, we represent the sequence of a graph process model in directly-follows frequency (DFF). Supposedly, the E-F-G-H will be detected as AND relationship while the H-I-J-K is OR relationship. We can see that both of them have identical structures and degrees, though they have a different in concurrency frequency.

With this scenario, the experimental results show that the existing method able to detect XOR and AND relationship. But the AND algorithm also detects H-I-J-K as AND because it only analyzes the degree, as presented in Fig. 2 (a). Meanwhile, the OR algorithm unable to recognize the OR pattern using rule describe in Table 9, this is due to the value of outdegree in brach of H-I-J-K, which is I and J have the same value to H. We can see the result of the existing method in Fig. 2 (b).

The proposed method tested in scenario 1 succeeds to detects and distinguishes all parallel process models correctly. This method differentiates the AND and OR patterns using an algorithm that basically utilizes the concurrent relationship frequency. The results of the experiment using the proposed method in scenario 1 can be seen in Fig. 3.

4.2 Scenario 2

Scenario 2 uses event log data similar to scenario 1 but with three branches. The detection results of graph-based sequential pairs are presented in Fig. 4. We can see that the concurrent relationship has no different from the regular sequence relationship. It is the main problem when the existing method relies on the degree of nodes. As a result, the existing method fails to detect OR as AND, and cannot detect OR as shown in Figs. 5 (a) and (b).

The proposed method takes benefit of the concurrency existence and the relationship frequency of nodes to detect parallel processes. The XOR, AND, and OR are successfully identified as we can see in Fig. 6.

Table 20. Experiment result for scenario 1

Expected Relationship	Existing Method	Proposed Method
XOR-Split	√	√
AND-Split	√	√
OR-Split	*AND	√
XOR-Join	√	√
AND-Join	√	√
OR-Join	*AND	√
Soundness	Yes	Yes
Fitness	0.98	1.00

Table 21. Experiment result for scenario 2

Expected Relationship	Existing Method	Proposed Method
XOR-Split	√	√
AND-Split	√	√
OR-Split	x	√
XOR-Join	√	√
AND-Join	√	√
OR-Join	x	√
Soundness	Yes	Yes
Fitness	0.96	1.00

Table 22. Experiment result for scenario 3

Expected Relationship	Existing Method	Proposed Method
XOR-Split	*AND	√
AND-Split	√	√
OR-Split	x	√
XOR-Join	√	√
AND-Join	√	√
OR-Join	x	√
Soundness	No	Yes
Fitness	-	1.00

Table 23. Experiment result for scenario 4

Expected Relationship	Existing Method	Proposed Method
XOR-Split	√	√
AND-Split	x	√
XOR-Join	√	√
AND-Join	x	√
Soundness	No	Yes
Fitness	-	1.00

4.3 Scenario 3

Scenario 3 is an event log containing the AND and OR processes in XOR. The results of sequential pair detection can be seen in Fig. 7. There is a challenge for the parallel detection algorithm to not misguided by the nested structure. It is supposed that B-D-C and J-I-K are XOR-Split and XOR-Join, C-E-F-I is an AND, and the D-G-H-J is detected as OR.

Scenario 1

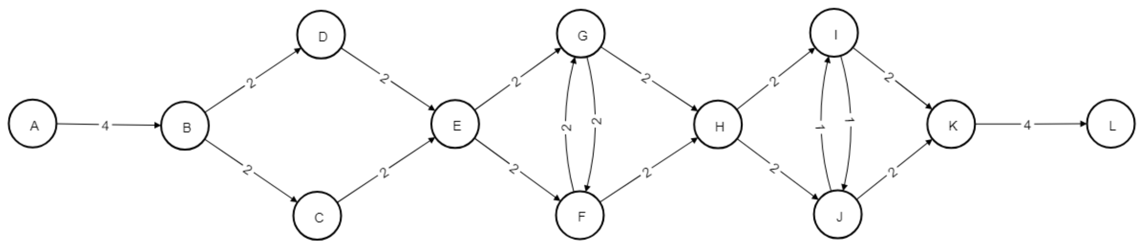


Figure. 1 Result of scenario 1 in directly-follows frequency (DFF)

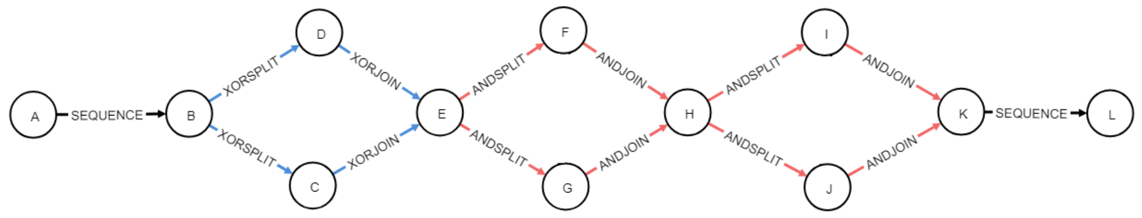


Figure. 2 Scenario 1 results with Existing method: detects OR as AND relationship and fail to detect OR relationship

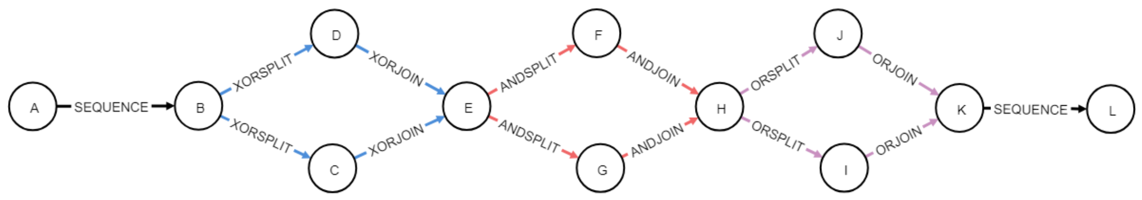


Figure. 3 Scenario 1 results using the proposed method succeed detecting all parallel

Scenario 2

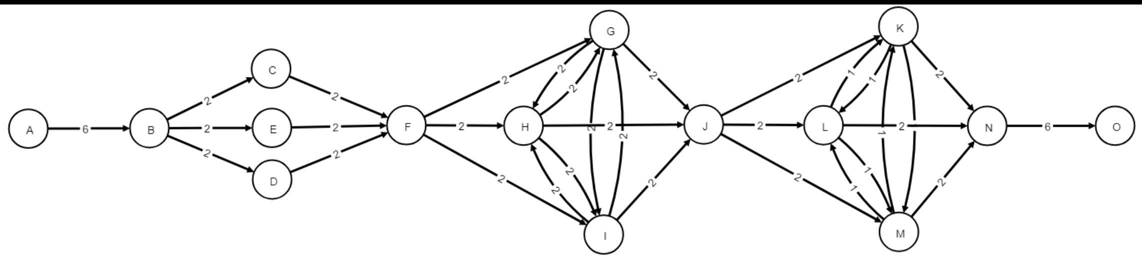


Figure. 4 Result of scenario 2 in directly-follows frequency (DFF)

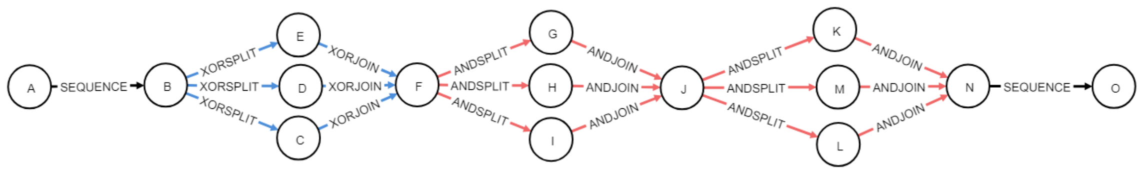


Figure. 5 Scenario 2 results with Existing method: detects OR as AND and fail to detect OR relationship

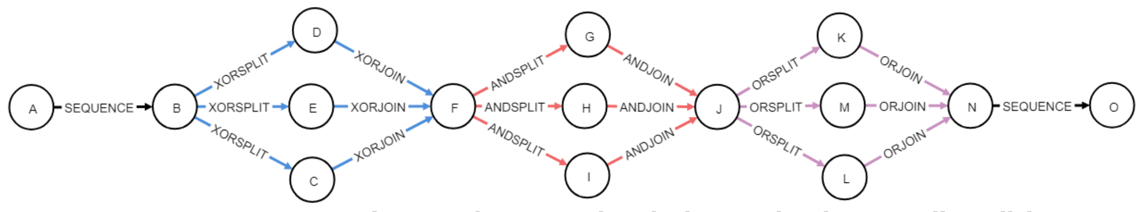


Figure. 6 Scenario 2 results using the proposed method succeed in detecting all parallel

Scenario 3

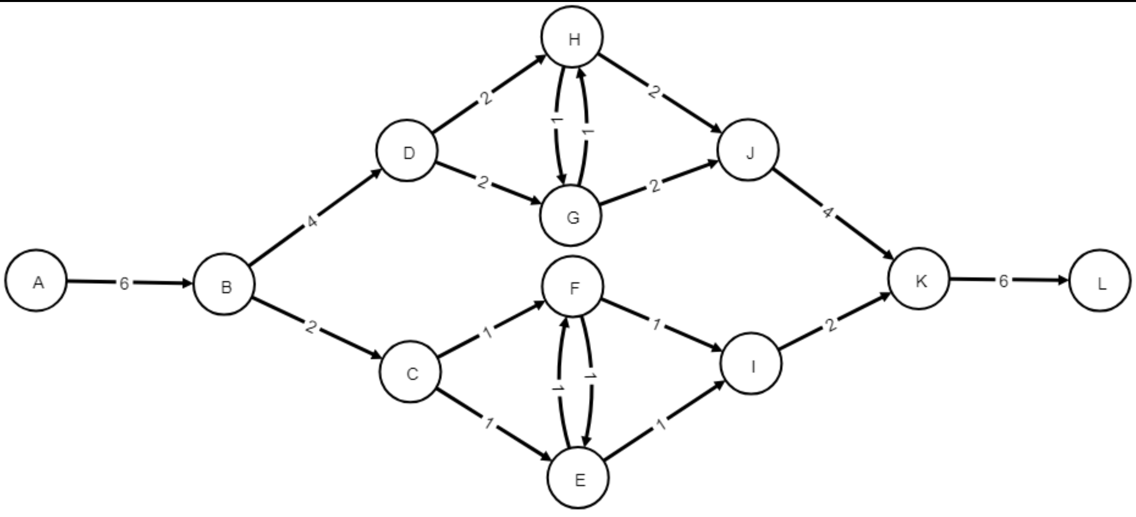


Figure. 7 Result of scenario 3 in directly-follows frequency (DFF)

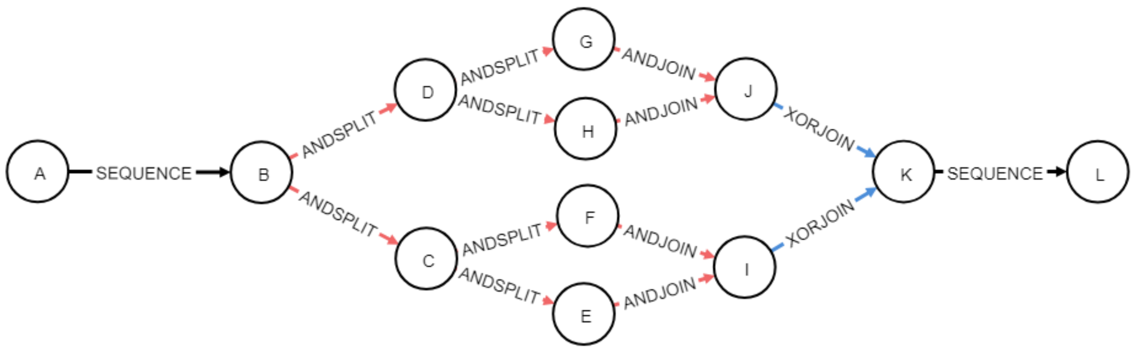


Figure. 8 Scenario 3 results using Existing method: detect OR as AND, detects XORSplit as ANDSplit, and fail to detect OR relationship

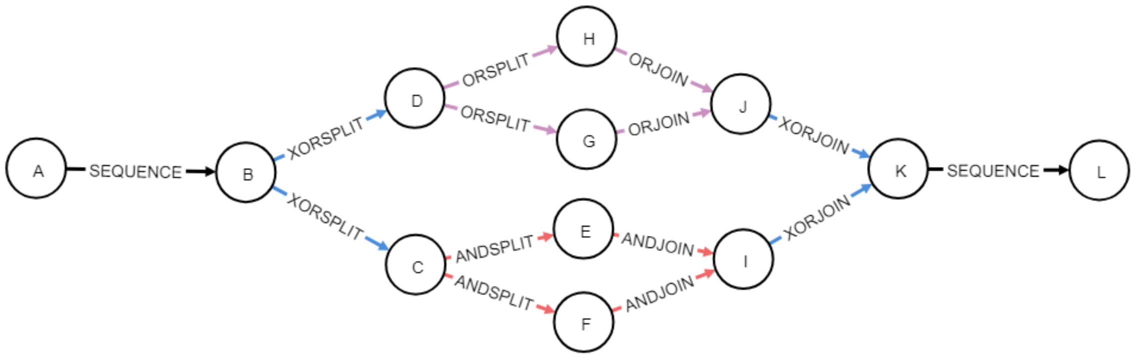


Figure. 9 Scenario 3 results using the proposed method succeed in detecting all parallel

The results of the existing method are presented in Figs. 8 (a) and (b). The AND algorithm is incorrectly detecting D-G-H-J as AND, while the OR algorithm still unable to recognize the OR. Moreover, the AND algorithm detects the XOR-Split pattern in B-D-C as AND-Split due to the outdegree of D and C are detected two, which is caused by the following nested parallel.

The proposed method shows that it is reliable in a nested structure. This method takes advantage of concurrent relationship existence to differentiate the XOR and the other parallel patterns strictly. The experiment successfully detects all parallel processes correctly as shown in Fig. 9.

4.4 Scenario 4

In scenario 4 the event log contains the AND process in XOR with the Sequence inserted in it. The resulting sequential pair model can be seen in Fig. 10. This scenario has a main pitfall from the

Received: December 3, 2019. Revised: January 2, 2020.

Scenario 4

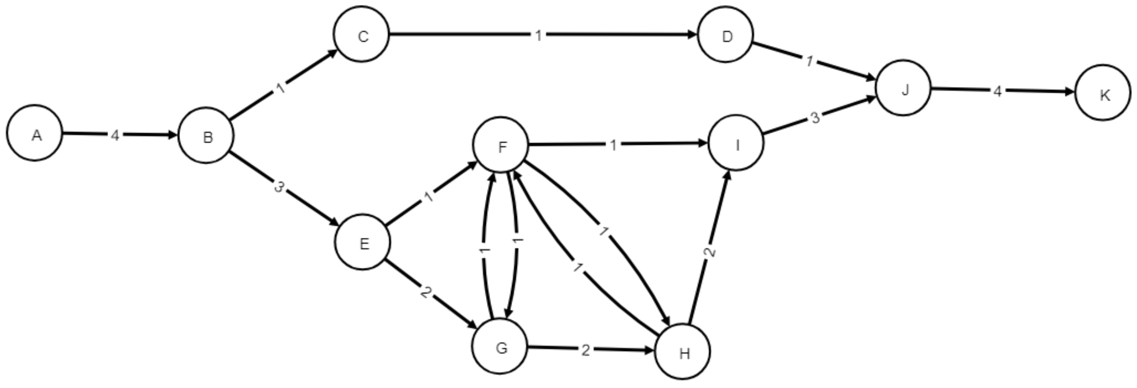


Figure. 10 Result of scenario 4 in directly-follows frequency (DFF)

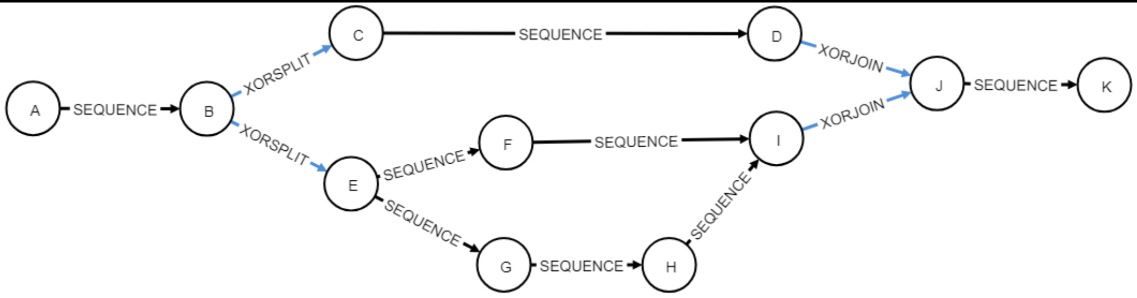


Figure. 11 Scenario 4 results using Existing method fail to detect AND relationship

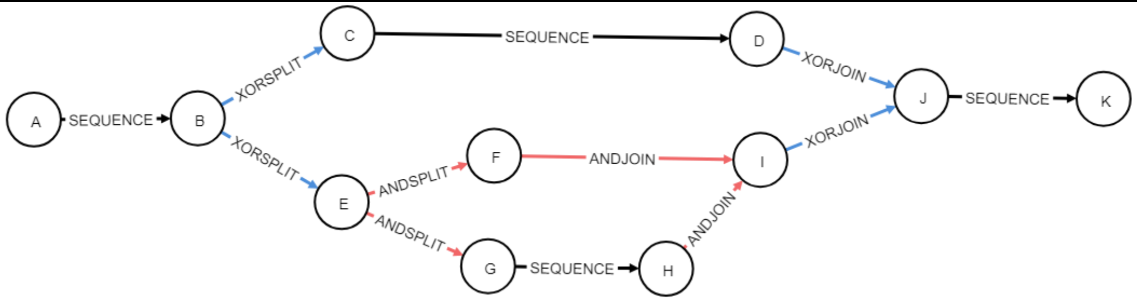


Figure. 12 Scenario 4 results using the proposed method succeed in detecting all parallel

concurrent relationship. We can see in E-F-G-H that F seems to have 3 outdegrees, in which 2 of them are a concurrent relationship with G and H. So, using the existing method will produce a model in Fig. 11, which cannot detect both AND and OR.

Experiments using the proposed method show the relevant results, i.e., all parallel patterns and inserted sequences correctly recognized, as we can see in Fig. 12.

5. Conclusion

We propose an improved method to discover parallel process patterns. Our strategy is to split the discovery process into two phases. The first is to enrich the process model semantically. This phase executes in three steps, namely counting the relationship frequency of each node, identifying concurrent relationships, and detecting split and join relationships.

The second phase is the process of executing parallel detection algorithms (XOR, AND, and OR). Among the three of them, XOR is the one supposed to have no concurrency. The difference between AND and OR is characterized by the value of relationship frequencies in branch nodes. The AND pattern should have the equal frequency for all of its nodes, which the OR is not.

To verify the ability of both methods in various cases in which the existing method fails, we have design four scenarios. The experiment result shows some problems with the existing method. In scenario 1 it detects OR pattern as AND, this discovered model results in a sound model with 0.98 score in fitness. Scenario 2 is also sound, and 0.96 for the fitness score. The existing method makes wrong detection in scenario 3 and 4 that results in not sound model, so we cannot measure the fitness score.

The experiment with our proposed method able to recognize all parallel patterns and successfully differentiates AND and OR patterns in all scenarios correctly. All model discovered also sound and get fitness value 1.00. This result proof that the proposed method successfully improves the existing method and outperform perfectly in all given scenarios.

For future work, further research needs to carry out to advance this method's ability. Some of them are the ability to deal with noise, detect invisible tasks, and handle the non-free choice condition.

References

[1] W. van der Aalst, "Process mining: discovering and improving Spaghetti and Lasagna processes", In: *Proc. of 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, No. c, pp. 1–7, 2011.

[2] W. M. P. van der Aalst, A. J. M. M. Weijters, and L. Maruster, "Workflow Mining: Discovering process models from event logs", *IEEE Trans. Knowl. Data Eng.*, Vol. 16, No. 9, pp. 1128–1142, 2004.

[3] A. J. M. M. Weijters and J. T. S. Ribeiro, "Flexible heuristics miner (FHM)", In: *Proc. of IEEE SSCI 2011: Symposium Series on Computational Intelligence - CIDM 2011: 2011 IEEE Symposium on Computational Intelligence and Data Mining*, pp. 310–317, 2011.

[4] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering Block-Structured Process Models From Event Logs - A Constructive Approach", In: *Proc. of Applications and Theory of Petri Nets 2013*, Vol. 7927, pp. 311–329, 2013.

[5] C. W. Günther and W. M. P. Van Der Aalst, "Fuzzy mining - Adaptive process simplification based on multi-perspective metrics", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 4714 LNCS, pp. 328–343, 2007.

[6] A. Augusto, R. Conforti, M. Dumas, and M. La Rosa, "Split Miner: Discovering Accurate and Simple Business Process Models from Event Logs", In: *Proc. of 2017 IEEE International Conference on Data Mining (ICDM)*, 2017.

[7] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and A. Polyvyanyy, "Split miner: automated discovery of accurate and simple business process models from event logs", *Knowledge and Information Systems*, Vol. 59, No. 2, pp. 251–284, 2019.

[8] R. Sarno, K. R. Sungkono, and R. Septiarakhman, "Graph-Based Approach for Modeling and Matching Parallel Business Processes", *Information*, Vol. 21, No. 5, pp. 1603–1614.

[9] 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.

[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, p. 137, 2019.

[11] H. Darmawan, R. Sarno, and A. S. Ahmadiyah, "Anomaly Detection Based on Control-flow Pattern of Parallel Business Processes", *Telkomnika*, Vol. 16, No. 6, pp. 2808–2815, 2018.

[12] C. Stephanie and R. Sarno, "Detecting Business Process Anomaly Using Graph Similarity Based on Dice Coefficient, Vertex Ranking and Spearman Method", In: *Proc. of 2018 International Seminar on Application for Technology of Information and Communication*, pp. 171–176, 2018.

[13] R. S. A. Wiratmo and K. R. Sungkono, "Graph-based Algorithm for Checking Wrong Indirect Relationships of Process Model Containing Non-Free Choice", *Telkomnika*, Vol. 13, No. 2, pp. 281–289, 2015.

[14] R. Sarno, K. R. Sungkono, A. Y. Hadiwijaya, and C. Fatichah, "An Algorithm for Discovering Process Models Containing Non-Free Choice Using Graph Database", *International Journal of Intelligent Engineering and Systems*, Vol. 9, No. 3, pp. 1–8, 2019.

[15] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*. O'Reilly, 2015.

[16] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management 2nd Edition*. Springer, 2018.

[17] W. Van Der Aalst, *Process Mining: Data Science in Action*. Springer, 2016.

[18] H. M. W. Verbeek, T. Basten, and W. M. P. Van Der Aalst, "Diagnosing workflow processes using Woflan", *The Computer Journal*, Vol. 44, No. 4, pp. 246–279, 2001.

[19] F. Mannhardt, M. de Leoni, H. A. Reijers, and W. M. P. van der Aalst, "Balanced multi-perspective checking of process conformance", *Computing*, Vol. 98, No. 4, pp. 407–437, 2016.