Improving Parallel Pattern Discovery from Directly Follows Graph Model

Indra Waspada
Department of Informatics
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
Department of Computer Science
Universitas Diponegoro
Semarang, Indonesia
indrawaspada@lecturer.undip.ac.id

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

Endang Siti Astuti
Department of Business Administration
Universitas Brawijaya
Malang, Indonesia
endangsiti@ub.ac.id

Hanung Nindito Prasetyo
Department of Informatics
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
Department of Information System Diploma
Telkom University
Bandung, Indonesia
hanungnp@telkomuniversity.ac.id

Raden Budiraharjo
Department of Informatics
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
Department of Information System
Institut Teknologi Nasional
Bandung, Indonesia
budiraharjo@itenas.ac.id

2022 5th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI) | 978-1-6654-5512-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISRITI56927.2022.10053029

***Abstract*— Process model discovery is an important part of process mining. State-of-the-art algorithms still have trouble detecting long parallel patterns with incomplete concurrency patterns. This paper proposes a method to exploit the characteristics of parallel block patterns to discover an accurate parallel block. The experiment compared the proposed method with state-of-the-art algorithms, Inductive Miner and Split Miner. The quality of the discovery results is measured using fitness, precision, and f-score. The experimental results show that the proposed method outperforms the state-of-the-art algorithm in discovering long parallel block patterns with incomplete concurrency patterns.**

Keywords—process mining, process model discovery, directly follows graph, parallel pattern, concurrency pattern

I. INTRODUCTION

Advancements in process mining science have brought business process analysis capabilities to a new level. Process model discovery is one of the main stages of process mining [1]. With appropriate process discovery techniques, business process models can be obtained as a representation of actual business activities [2]. Business developments, competition, market changes, new policies, and various other effects can cause business processes to change constantly. These changes are often not able to be followed manually. Therefore the role of process mining is important because it can get an overview of the latest process models more easily and quickly. Insight based on real data from the field will become an accurate foundation for organizations to develop strategies, make decisions and policies.

Currently, many algorithms have been developed to find process models. Most classical algorithms have difficulty with concurrency, for example, transition systems, finite state machines, Markov chains, and Hidden Markov models [3]. Several algorithms are starting to take concurrency into account, including the α-algorithm and its various variants. However, the α-algorithm-based technique produces a model with low fitness quality [4]. The Inductive miner algorithm works through a block structure so that the model obtained is guaranteed to be sound by construction. Inductive miners can produce models with high fitness quality but not good in terms of precision. The Split Miner (SM) algorithm tries to cover these weaknesses. Instead of working with block structures, SM prefers to detect the split and join points separately in determining the type of branching structure (XOR, AND, OR).

This paper raises the problem of detecting long parallel patterns with incomplete concurrency patterns. The definition of incomplete here is when not all concurrency traces are available in the event log. The incomplete ones make it difficult for the discovery algorithm to deduce the correct pattern.

As a solution to the problem we improve the Graph-based process discovery (GPD) [5]. This paper proposes contributions as follows:

1. Introduce some terminologies for the parallel block pattern characteristics.
2. An algorithm to detect parallel blocks with long patterns.
3. An approach to recognize an incomplete concurrency to be removed from a parallel pattern block.

An experiment has been conducted by comparing the ability of parallel pattern discovery using the IM and SM algorithms. The experimental scenario is done using two types of event logs. The first type is an event log from artificial data to test the algorithm's performance under complete event log conditions. Then the second data is real-life data from the process of unloading containers for import activities at container terminal companies in Indonesia

Experimental results on ideal data show that all algorithms can work well in detecting parallel blocks. This is indicated by the same fitness, precision, and f-score with a score of 1.00.

Meanwhile, experiments with incomplete event log data gave different results between IM, SM, and GPD+. The model produced by SM is not sound, so it cannot work with alignment-based quality measurement [6], [7]. Therefore we also use a Markovian-based measurement tool [8]. The experiment results show that GPD+ is superior to the state-of-the-art methods in detecting long parallel patterns with incomplete event logs.

The remainder of this paper is organized as follows. Section II describes some of the state-of-the-art methods of process discovery. In Section III, we present our proposed method. The result and discussion are explained in Section IV. Section V provides a conclusion and overview of future works.

II. Literature Review

Inductive Miner (IM) [9], [10], and Split Miner (SM) [11] algorithms are state-of-the-art process discovery algorithms. IM attempts to split event logs using sequence slicers, XOR, AND, and loops. If no perfect cut is found, this algorithm compares the value of the best cut to be selected as a decision to choose a cutter. The process model obtained tends to produce high fitness values but often sacrifices precision values. Whereas SM starts the discovery process by building a directly follows graph (DFG) as an intermediate model. Based on DFG, SM detects the parts which are split and joined. The next stage is to analyze the characteristics of each split and join to decide on the appropriate branching structure. With this approach, the SM technique produces a process model with a higher precision value than IM but slightly lower in fitness value.

IM and SM need to load all logs into memory, causing limitations in working with large data. Sarno et al. [12], [13] proposed the Graph-based Invisible Task (GIT) method to detect invisible tasks by utilizing a graph database. Furthermore, Waspada et al. [5] developed a Graph-based Process Discovery (GPD) to improve GIT so that it can work more broadly by paying attention to the frequency on edge and taking into account concurrency and frequency in the detection of exclusive OR (XOR), parallel (AND), and inclusive OR (OR) patterns.

The following will be discussed the GPD algorithms used in [5]. These algorithms are executed in 9 steps:

1. *Load the event log as graph nodes*. At this stage, the event log data is loaded into the graph database as two types of nodes, i.e., traces and a process model.
2. *Create Directly Follows Graph and Its Frequency*. By referring to the definition of a directly-follows graph (DFG)[3] as depicted in definition 1. DFG is created in the 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 a Sequence Relationship.

Definition 1 (Directly-Follows Graph): With an event log L where $L \in \mathbb{B}(\mathcal{A}^*)$, the directly-follows graph of L is written as $G(L) = \left(A_L, \mapsto_L, A_L^{start}, A_L^{end}\right)$ with:

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

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

Here two things are processed. First, create DFG for process and traces. This trace with only sequence pattern will be a benefit for pattern matching purposes, i.e., in concurrent relationship detection. Second, each time a relationship is merged in the graph model, the frequency counter of the relationship also increases. This value is termed in [11] as directly-follows frequency (DFF) as defined in definition 2.

Definition 2 (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 $\left|\left\{(e_i, e_j) \in \mathcal{E} \; x \; \mathcal{E} \mid e_i^l = l_1 \wedge e_j^l = l_2 \wedge \exists_t \in \mathcal{E} \left[\exists e_x \in t\left[e_x = e_i \wedge e_{x+1} = e_j\right]\right]\right\}\right|$

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).
4. *Identify a concurrent relationship*. Concurrent relations are detected when there is a direct relationship between node A to node B and vice versa. This condition can arise in models because, in reality, both of them 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 short-loop and concurrent relationships definition as defined in [11].

Definition 3 (Short-loop): Given two activities, a and b, a short-loop ($a \circlearrowright b$) exist iff meets the requirement in (1) and (2) [11].

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

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

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 4 (Concurrent relationship): Given activities a and b, they are concurrent $a \| b$ iff meet conditions in (3), (4), and (5) [11].

$$|\boldsymbol{a \rightarrow b}| > \boldsymbol{0} \;\wedge\; |\boldsymbol{b \rightarrow a}| > \boldsymbol{0} \qquad (3)$$

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

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

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 a specific similar (threshold, ε) value.

5. *Identify Split and Join relationships.* After the concurrent relationship detection, the rule for Split and Join can be designed. These Split and Join are the foundation for the complex pattern detection algorithm.
6. *XOR pattern identification and creation*. This algorithm is designed with the requirement that branch nodes in the XOR relation cannot have a concurrent relationship.
7. *AND pattern identification and creation*. The AND pattern is characterized by the condition when the relationship frequency (ifr or ofr) of the gateway node is the same as the frequency of its branch nodes.
8. *OR pattern identification and creation*. The OR relationships have a pattern where the gateway node has a higher relationship frequency than each branch node.
9. *Remove all Concurrent relationships*. All concurrent relationships were removed from the process model.

III. The Proposed Method

A. Parallel Pattern Block Characteristics

Incomplete concurrency patterns in a long parallel block cannot be well recognized using IM, SM, and conventional GPD. Therefore in this paper, we propose a technique to improve the GPD to handle the challenge.

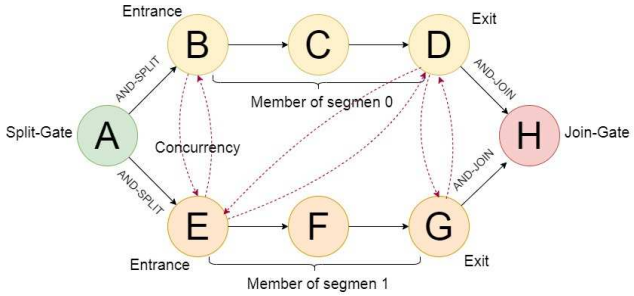


Fig. 1. An illustration of a parallel block pattern.

The proposed method exploits the characteristics of parallel pattern blocks. An illustration of parallel blocks and the terminology used is presented in Fig. 1. The characteristics introduced are:

1. A parallel pattern block has a front-side gate (AND-split) which consists of one Split-Gate, a pair of Entrances nodes, and concurrent relations between entrance nodes.
2. The Entrance node ends at an Exit node. Exit nodes are connected to a Join-Gate node
3. The Join-Gate node must meet the requirements as the node farthest from the Entrance, which is the final estuary of the path of each branch (segment 0 and segment 1)
4. The output frequency of the Split-Gate node has the same value as the input frequency of each Entrance
5. The type of a back-side gate of the parallel pattern block may vary. The relationship going to the Join-Gate is only possible in the form of AND-JOIN (if an AND-join pattern is found) or OR-JOIN.
6. All nodes along the Entrance and Exit nodes are in the same parallel segment. Every edge that are connected across segments will be concluded as concurrency.

B. Graph-based Process Discovery Plus (GPD+)

Based on the DFG model, which already has concurrency labels, split and join labels, and by utilizing the characteristics of the parallel block pattern as shown in Fig. 1, a more reliable parallel block detection algorithm can be designed.

The following are the steps proposed to detect parallel blocks:

1. *Detect and mark the front side of a parallel pattern block (AND-split)*

The algorithm to detect AND-split in this step is similar to that used in GPD. However, it is equipped with a Split-Gate and Entrances identification as the front side of the parallel pattern block. Our improved algorithm also performs (parallel) block detection starting from the block with the split gate node closest to the starting point of the process. The algorithm to detect the closest AND split pattern is described in Algorithm 1.

Algorithm 1 Algorithm To Detect ANDsplit

```
Input: DFG model with Split_Gate label, startNode
1: function discoverTheClosestANDSplit (DFG, startNode)
2:     allSplitGates ← getAllSplitGates(DFG)
3:     closestSG ← MIN(DFG, startNode, allSplitGates)
4:     if closestSG has an ANDsplit pattern:
5:         entrances ← detectParallelPattern(DFG, closestSG)
6:         setANDsplit(DFG, closestSG, entrances)
```

2. *Find and mark the back side of the parallel pattern block*

Detection of the back side gate is done by first identifying the farthest concurrency pair node with the Entrance node, e.g., the D node in Fig. 1. From this node, then our algorithm will find the closest Join-Gate. Based on the Join-Gate found, then the Exit nodes can be concluded. The algorithm to detect the ANDjoin pattern is described in Algorithm 2.

Algorithm 2 Algorithm To Detect ANDjoin

```
Input: DFG model with ANDsplit attributes
1: function discoverANDjoin (DFG)
2:     for ent in entrances:
3:         concurrentNs← getAllConPairNodes(ent, entrances)
4:         conInSegmens.append(concurrentNs)
5:     farthestConNodes←getFarthestConInSegmen
6:     allJG← getAllJoinGates(DFG)
7:     closestJG← MIN(DFG, farthestConNodes, allJG)
8:     exitNodes← detectExitNodes(DFG , entrances, closestJG)
9:     setANDjoin(DFG , exitNodes, closestJG)
```

3. *Remove all relationship across parallel activities*

After parallel blocks are detected, then all relations that cross between segments inside the block can be ascertained as concurrent relations so that these relations are removed from the model.

C. Data set and preprocessing

This study uses three data. The two data are artificial data to test the basic ability of the algorithm to find parallel patterns from complete event logs. The way to get the artificial data is to start by making a Petri net model using the Apromore web-based application (https://academic-eu.apromore.org/). Then through the model, the event log is

generated via ProM using the log generator plugin [14]. By adjusting the characteristics of the generated cases and their number, an event log that meets the completeness characteristics will be obtained.

The third data is real-life data from import container unloading activities at one of the container terminals in Indonesia. The data was taken over a period of three months, namely from July 2021 – September 2021. The focus of the study is the green line, namely the flow of containers that do not require physical inspection. This flow can be seen in Fig. 2. When the ship is on its way, permit documents can be processed online (BAPLIE). Container ships docked at the wharf (VESSEL_ATB). After the ship has docked, the work of unloading the containers from the ship to the internal truck (DISCHARGE) is carried out. The containers are then transported by truck to the stacking yard (STACK). In parallel, the required documents are checked by customs to obtain an SPPB document as a release order (CUSTOMS_DEL). After the SPPB is released, the customer must complete all administrative requirements to obtain permission to take the container (JOB_DEL). Then the customer sends a truck to pick up the container (TRUCK_IN). The TPS officer moves the container from the pile in the field to the customer's truck (UNSTACK_TO_TRUCK), and the truck exits through the gate (TRUCK_OUT).

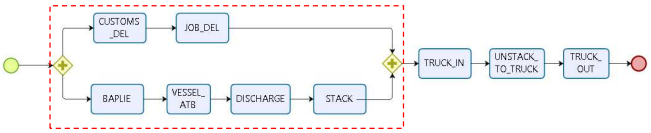


Fig. 2. Green line container unloading business processes in import activities at Container Terminals.

Real-life data is obtained from the Terminal Operations System (TOS) by querying it. The results obtained do not meet the required event log structure. Therefore, we do preprocess to comply with the Extensible Event Stream (XES) format standard. In addition, we also filtered out variants that did not comply with the SOP rules. The final event log results are presented in Fig. 3. This data is stored in CSV format so that it can be loaded into the Neo4j graph database.

Our concern focuses on parallel sections. Therefore, a modification is made by removing the TRUCK_IN, UNSTACK_TO_TRUCK, and TRUCK_OUT activities which are sequence patterns outside the parallel block. We also added START and END activities at the beginning and end of the process. The result of the discovered DFG model in the Neo4j graph database is presented in Fig. 4.

Fig. 4 shows a potential long parallel block with an incomplete event log. This is indicated by incomplete concurrency relations from VESSEL_ATB to JOB_Del. As a result of this incompleteness, the relationship between CUSTOMS_DEL, VESSEL_ATB, and JOB_DEL will be detected as an AND-Join pattern, and the relationship between JOB_DEL, VESSEL_ATB, DISCHARGE will also be detected as an AND-Split pattern. These anomalies cause the state-of-the-art algorithms to fail to get good-quality of discovery results.

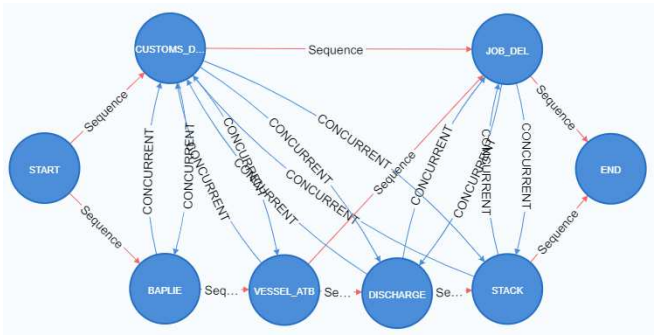
D. *Experimental Setting*

We compare the capabilities of GPD+ with state-of-the-art process discovery, i.e., IM and SM. The experiments carried out focused on parallel structures.

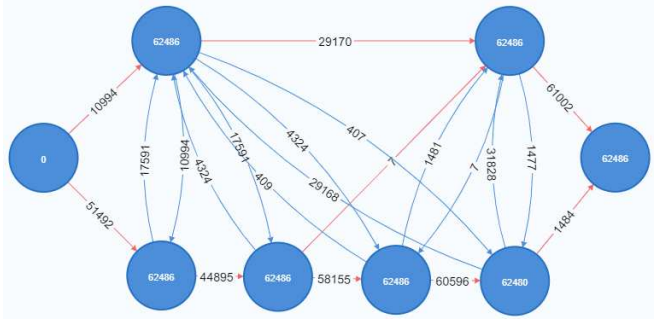
First, the three algorithms will be tested with ideal data to ensure the basic ability to detect parallel structures. We create two artificial data sets, namely a short parallel structure and a long parallel structure, which contain a complete concurrency relationship.

index	CTR_TYPE	container key	time:timestamp	lifecycle:transition	event	CTR_SIZE	frequency	case
0	TNK	AAMU2609009	2021-09-25 10:40:48+01:00	complete	BAPLIE	20.0	29150	AAMU2609009-2021-09-26
1	NaN	AAMU2609009	2021-09-25 22:00:00+01:00	complete	VESSEL_ATB	NaN	29150	AAMU2609009-2021-09-26
2	NaN	AAMU2609009	2021-09-26 08:12:10+01:00	complete	DISCHARGE	NaN	29150	AAMU2609009-2021-09-26
3	NaN	AAMU2609009	2021-09-26 08:33:37+01:00	complete	STACK	NaN	29150	AAMU2609009-2021-09-26
4	NaN	AAMU2609009	2021-09-27 18:00:00+01:00	complete	CUSTOMS_DEL	NaN	29150	AAMU2609009-2021-09-26
...	...	...	...	...	...	...	...	...
262	NaN	TGCU2014471	2021-09-05 18:00:00+01:00	complete	CUSTOMS_DEL	NaN	1	TGCU2014471-2021-09-05
263	NaN	TGCU2014471	2021-09-05 18:16:33+01:00	complete	STACK	NaN	1	TGCU2014471-2021-09-05
264	NaN	TGCU2014471	2021-09-06 08:17:52+01:00	complete	JOB_DEL	NaN	1	TGCU2014471-2021-09-05
265	NaN	TGCU2014471	2021-09-06 11:08:35+01:00	complete	TRUCK_IN	NaN	1	TGCU2014471-2021-09-05
266	NaN	TGCU2014471	2021-09-06 12:01:26+01:00	complete	TRUCK_OUT	NaN	1	TGCU2014471-2021-09-05

Fig. 3. Event log of real-life data for experiments



(a)



(b)

Fig. 4. The discovery result of container import activities in the DFG model. (a) displaying the node labels and relationship type, (b) displaying node input frequencies and relationship frequencies.

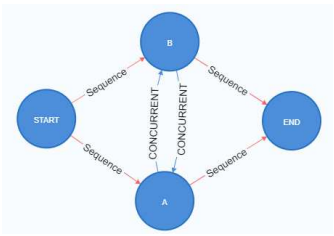
Next, the main part of the experiment is the detection of long parallel structures with incomplete concurrency relationships. We use real-life data from import activities at one of the container terminals in Indonesia.

Each algorithm will be tested to perform process discovery. The measurement of the quality results uses two measurement tools, i.e., the alignment-based tool and the Markovian tool.

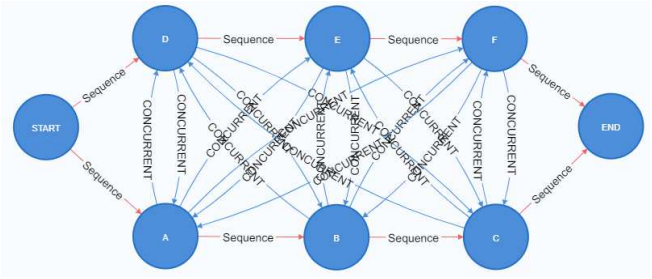
IV. Result and Discussion

A. Scenario using artificial event log

Experiments with artificial event logs aim to obtain a DFG model with a complete concurrency structure so that it can prove the capability of the discovery technique under ideal conditions. Fig. 5 presents the intermediate discovery result in DFF model. Fig. 5(a) shows the DFF model for short parallel block. Meanwhile, Fig. 5(b) is the DFF model for long parallel block. The final result need to be in Petri net model so it can be measured using the quality measurement tools. The quality measurement of the final model discovered using IM, SM, and GPD+ is presented in Table 1.



(a)



(b)

Fig. 5. The visualization of DFF model with complete concurrency relationship in parallel block pattern: (a) short block, (b) long block

Table 1. The Results Of Parallel Block Detection Experiments Under Complete Event Log Conditions.

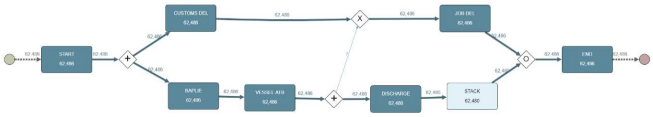
Scenario	Algoritma	Fitness	Precision	fscore
Short Parallel Pattern	IM	1.00	1.00	1.00
	SM	1.00	1.00	1.00
	GPD+	1.00	1.00	1.00
Long parallel pattern	IM	1.00	1.00	1.00
	SM	1.00	1.00	1.00
	GPD+	1.00	1.00	1.00

B. Scenario using real-life event log

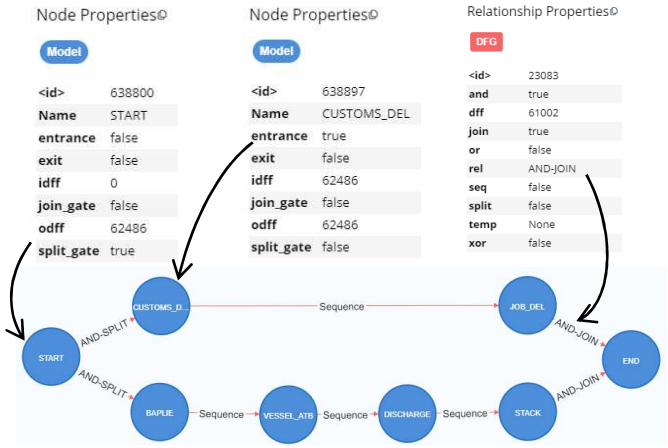
Next, the test is carried out on real-life data with the DFG representation, as shown in Fig. 4. Process discovery using IM results in a Petri Net model in Fig. 7(a). SM produces a BPMN model in Fig. 6(a). The BPMN model can be changed to Petri Net using the ProM Plugin, as shown in Fig. 7(b).

GPD+ works on the Neo4j graph database to provide discovery results in the graph model shown in Fig. 6(b). It shows some information about the semantics found using GPD+ in parallel blocks. It appears that the "START" node is Split_Gate, the "CUSTOMS_DEL" node is Entrance, and the relationship between "JOB_DEL" and "END" has "and" and "join" attributes with values 'true' . This graph model can be converted into a Petri Net model using the algorithm from [15] to obtain the model in Fig. 7 (c).

Fig. 4 (a) shows that CUSTOMS_DEL has the farthest concurrency pair with STACK. This pattern gives GPD+ the knowledge that Join_Gate must be after STACK so an early Join_Gate trap (i.e., JOB_DEL) can be avoided. The comparison of quality results is presented in Table 2.

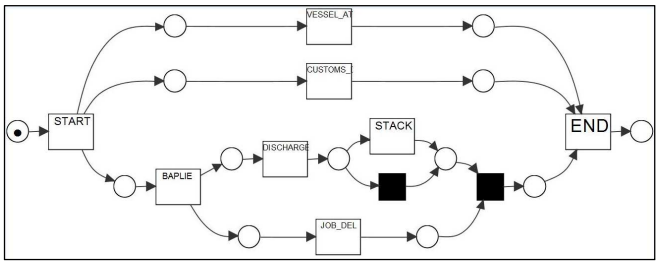


(a)

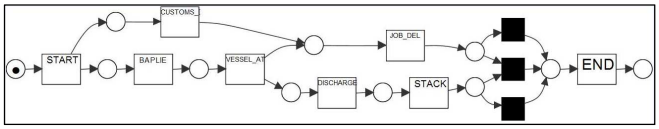


(b)

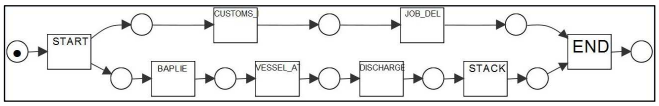
Fig. 6. Result of process model discovery: (a) BPMN model discovered using SM, (b) Graph-based model discovered using GPD+.



(a) Model discovered using IM



(b) Model discovered using SM



(c) Model discovered using GPD+

Fig. 7. The Petri net model results from long parallel block detection from the DFF model with incomplete concurrency structures.

C. Discussion

The success of all the algorithms compared in detecting parallel blocks with complete event logs is basically due to all algorithms assuming that the event log is complete. Meanwhile, conditions in the field are rarely ideal.

Experiments using real-life data containing incomplete concurrency data cause the IM and SM algorithms to interpret the captured behavior incorrectly. The IM algorithm tends to produce models with high fitness but low precision. The low

precision values indicate that the model will accept more behavior never present in the log history. The SM algorithm can interpret a model with higher precision values than IM, but lower in fitness. SM also produces a model that is not sound.

The proposed GPD+ method discovers the best quality model for a parallel pattern block. It utilizes a recognizable chain of concurrency relationships to help find the closest join gate position. In this way, GPD+ is successful in escaping from the trap of incomplete concurrent relationships.

TABLE 2. RESULTS OF LONG PARALLEL BLOCK DETECTION EXPERIMENTS WITH INCOMPLETE EVENT LOGS USING REAL-LIFE DATA.

Algorithm	Quality measurement	Fitness	Precision	fscore
IM [9]	Alignment-based	**1.00**	0.703	0.827
	Markovian-based	**1.00**	0.167	0.287
SM [11]	Alignment-based	-	-	-
	Markovian-based	0.686	0.303	0.420
GPD+	Alignment-based	0.999	**0.933**	**0.961**
	Markovian-based	0.943	**0.673**	**0.786**

V. CONCLUSION

This research proposes an improved method to detect parallel patterns. The experimental results on ideal data show that all algorithms can work well in detecting parallel blocks. Meanwhile, experiments with incomplete concurrency patterns data gave different results between IM, SM, and GPD+.

IM and SM algorithms cannot recognize incomplete concurrency, so the resulting model is not as expected. IM gives interpretations that tend to produce models with high fitness but very low precision values. In contrast, SM produces better precision than IM but gets a lower fitness value. Besides that, the model produced by SM does not sound. GPD+ can detect parallel blocks without being stuck with incomplete concurrency relationships so that the results' quality succeeds in outperforming the state-of-the-art algorithm.

REFERENCES

[1] W. M. P. van der Aalst, "Process mining: discovering and improving Spaghetti and Lasagna processes," in *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, 2011, hal. 1–7.

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

[3] W. van der Aalst, *Process Mining: Data Science in Action*. Springer, 2016.

[4] J. She dan D. Yang, "Process Mining: Algorithm for S-Coverable Workflow Nets," in *2009 Second International Workshop on Knowledge Discovery and Data Mining*, 2009, hal. 239–244.

[5] I. Waspada, R. Sarno, dan 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, hal. 127–139, 2020.

[6] A. Adriansyah, "Aligning Observed and Modeled Behavior," Technische Universiteit Eindhoven, 2014.

[7] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. Van Dongen, dan W. M. P. Van Der Aalst, "Alignment based precision checking," *Lecture Notes in Business Information Processing*, vol. 132 LNBIP, no. 2, hal. 137–149, 2013.

[8] A. Augusto, A. Armas-Cervantes, R. Conforti, M. Dumas, dan M. La Rosa, "Measuring Fitness and Precision of Automatically Discovered Process Models: A Principled and Scalable Approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 4, hal. 1870–1888, 2022.

[9] S. J. J. Leemans, D. Fahland, dan W. M. P. van der Aalst, "Discovering Block-Structured Process Models From Event Logs - A Constructive Approach," in *Applications and Theory of Petri Nets 2013*, vol. 7927, Springer Berlin, 2013, hal. 311–329.

[10] S. J. J. Leemans, D. Fahland, dan W. M. P. van der Aalst, "Scalable process discovery and conformance checking," *Software & Systems Modeling*, vol. 17, no. 2, hal. 599–631, Jul 2016.

[11] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, dan A. Polyvyanyy, "Split miner: automated discovery of accurate and simple business process models from event logs," *Knowledge and Information Systems*, vol. 59, no. 2, hal. 251–284, Mei 2019.

[12] R. Sarno dan K. R. Sungkono, "A survey of graph-based algorithms for discovering business processes," *International Journal of Advances in Intelligent Informatics*, vol. 5, no. 2, hal. 137, 2019.

[13] R. Sarno, K. R. Sungkono, M. Taufiqulsa'di, H. Darmawan, A. Fahmi, dan K. Triyana, "Improving efficiency for discovering business processes containing invisible tasks in non-free choice," *Journal of Big Data*, vol. 8, no. 1, 2021.

[14] S. K. L. M. Vanden Broucke, J. De Weerdt, J. Vanthienen, dan B. Baesens, "An Improved Process Event Log Artificial Negative Event Generator," *SSRN Electronic Journal*, 2012.

[15] I. Waspada, R. Sarno, E. S. Astuti, H. N. Prasetyo, dan R. Budiraharjo, "Graph-Based Token Replay for Online Conformance Checking," *IEEE Access*, vol. 10, no. September, hal. 102737–102752, 2022.