Received 19 July 2022, accepted 7 September 2022, date of publication 20 September 2022, date of current version 3 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3208098

RESEARCH ARTICLE

Graph-Based Token Replay for Online Conformance Checking

INDRA WASPADA[1,3], RIYANARTO SARNO[1], (Senior Member, IEEE), ENDANG SITI ASTUTI[2], HANUNG NINDITO PRASETYO[1,4], AND RADEN BUDIRAHARJO[1,5]

[1]Department of Informatics, Faculty of Intelligent Electrical and Informatics Technology, Institut Teknologi Sepuluh Nopember, Surabaya 60111, Indonesia
[2]Department of Business Administration, Brawijaya University, Malang 65145, Indonesia
[3]Department of Informatics, Faculty of Science and Mathematics, Diponegoro University, Semarang 50275, Indonesia
[4]Department of Information System Diploma, School of Applied Science, Telkom University, Bandung 40257, Indonesia
[5]Department of Information System, Faculty of Industrial Technology, Institut Teknologi Nasional, Bandung 40124, Indonesia

Corresponding author: Riyanarto Sarno (riyanarto@if.its.ac.id)

This research was funded by Lembaga Pengelola Dana Pendidikan (LPDP) under Riset Inovatif-Produktif (RISPRO) Program; the Indonesian Ministry of Education and Culture under Penelitian Terapan Unggulan Perguruan Tinggi (PTUPT) Program; and Institut Teknologi Sepuluh Nopember (ITS) under project scheme of the Publication Writing and Intellectual Property Rights Incentive Program (Pusat Publikasi dan Hak Kekayaan Intelektual); and also supported by Diponegoro University Scholarship Program.

ABSTRACT Conformance checking detects deviations in business process executions. An online detection method is needed to give immediate response to anticipate possible impacts. The state-of-the-art online conformance checking is the Prefix-Alignment (PA) technique. However, this technique has a limitation of maintaining all of the administration data of cases in memory. In an online environment, the last event of a case is never known, whereas a PA requires last event information to release the case from memory to free up space for other cases. Hence, the PA does not meet the requirements of online conformance checking in processing infinite data of event stream without memory constraints. PA also has a complex state space search computation especially for large and complex process model references. In this paper, a Graph-Based Online Token Replay (GO-TR) method is proposed. This method takes benefit from Graph Database to adapts the Token-Based Replay (TBR) technique which has simple replay computation. We propose a Replay Image (RI) to store the case administration and develop a cypher based algorithm to simulate token replay on the RI to handle the event stream. We also propose a cypher-based algorithm to identify and replay invisible paths. The experiment results show that GO-TR has been successful in adapting TBR and solving the problem of wrong-placed tokens in TBR. GO-TR outperforms PA in yielding replay throughputs of relatively small amount of data in online conformance checking. In terms of memory usage, GO-TR shows its superiority over PA because it does not have memory limitations problems.

INDEX TERMS Conformance checking, event stream, graph database, token-based replay, memory limitation.

I. INTRODUCTION

Conformance checking detects deviations in business process executions. There are two important replay techniques, which are Token-Based Replay (TBR) and Alignment. Both techniques work in offline environments. TBR was first introduced by Rozinat and Aalst [1] as a replay technique based on Petri net. While Alignment [2] is currently the de facto standard for offline conformance checking because of its ability to provide optimal alignment information.

In real life, there are many conditions that require immediate inspections. Therefore, an online conformance checking technique is required. Online detection capability enables anticipative action of possible impacts as soon as possible.

Prefix-Alignment (PA) [3], [4], [5] is a state-of-the-art replay-based online conformance checking technique. It is a modification of the conventional Alignment technique. However, PA has a limitation of maintaining all of the

The associate editor coordinating the review of this manuscript and approving it for publication was Zhangbing Zhou.

administration data of cases in memory. In an online environment, the last event of a case is never known, whereas a PA requires last event information to release the case from memory to free up space for other cases. Hence, the PA does not meet the requirements of online conformance checking which is the ability to process infinite data of event stream without memory constraints.

Several studies have attempted to overcome the PA weakness by suggesting approaches to predict the end of a case, but these solutions cannot be used in general environments. Aside from that, these solutions cause other impacts that need to be resolved [3], [6].

In this study, a Graph-Based Online Token Replay (GO-TR) method is proposed. GO-TR adapts TBR for online environments through the support of a graph database. This proposal is expected to solve problems in memory usage of the PA. The TBR technique was chosen because of its characteristics, namely: (1) simple replay computation (as compared to the Alignment); and (2) maintaining only the running marking (as compared to the Alignment technique that must maintain all candidates marking).

The graph database was chosen because of its ability to store graph data natively. Previous study by Sarno *et al.* [7] had succeeded in integrating an ERP system with a graph database to store event logs directly and apply a graph-based process discovery model technique in it. Several developments and applications also show that the graph-based methods can provide high fitness and precision scores [8], [9]. Sungkono *et al.* [10] also used a graph database to check wrong decisions and wrong patterns.

The method proposed in this study utilizes the persistence nature of graph database to store the last state of an online conformance checking progress as a Replay Image in the graph database. In addition, the graph database has a constant node traversal that is suitable to support the proposed invisible path identification and invisible path replay especially for complex models which produce large reachability graph. The combination of the persistence and speed of node tracing makes the graph database a reliable solution to support GO-TR implementation without memory constraint.

This paper proposes contributions as follows:

1. Petri Net model representation in the graph database as a replay image. Replay image is a reference model that stores the running marking and administrative data related to replay.
2. Token-based Replay adaptation on a graph database that can receive event stream data for online conformance checking.
3. Algorithm for identification and replay a Graph-based invisible path. We take advantage of the graph database to identify invisible paths accurately and efficiently.

The experimental results show that GO-TR has been successful in adapting TBR to the graph database and at the same time providing solution to the wrong-placed token problem on the TBR. We also found that, for relatively small amount of data, GO-TR resulted in higher throughput compared to prefix-alignment. However, GO-TR's throughput decreases as the amount of data increases. In terms of memory usage, GO-TR shows its superiority over PA because it does not have memory limitations problems

The next section of the paper will be presented as follows. Section 2 describes related works. Section 3 explains the definitions and concepts that underlie our proposals. Section 4 discusses the fundamentals of our proposed method. Section 5 presents the experiment results and discusses the findings from the experiment. Section 6 provides conclusions and overview of future research opportunities.

II. RELATED WORK

In this section, researches related to this work are described, from conventional conformance checking, online conformance checking, to the development of graph-based process mining researches.

A. CONVENTIONAL CONFORMANCE CHECKING

At the beginning of its growth, process mining was oriented to extracting event log data in an offline environment. Likewise, conformance checking techniques, such as the Token-Based Replay (TBR) [1] and Alignment [2] techniques, can only work in an offline environment.

TBR is a conformance checking technique with replay output that describes a series of activities resulting from a replay. Basically, the TBR algorithm is very simple, but when the reference model contains an invisible task, the TBR requires additional efforts to detect it. Rozinat and Aalst [1] proposed the detection of invisible paths by building a local reachability graph and then tracing the entire state space. This method requires complex computations so that it slows down TBR execution.

The alignment technique [2] improved TBR by building a synchronous product between the reference model and the execution log to choose the best replay route. The resulting series of activities is referred to as alignment. Meanwhile, the computational result to get the alignment with the smallest cost is known as optimal alignment.

Berti and Aalst [11] proposed an Improved TBR (ITBR) by adding preprocessing to detect all invisible path lists at the beginning. The invisible path search is done by selecting the shortest route from a list of invisible-path candidates and then running the algorithm for invisible replay. This solution makes ITBR faster than the Alignment technique for Petri nets also for models with invisible transitions. However, ITBR requires an invisible tasks replay check which when it fails will leave a wrong-placed token problem.

Our work was inspired by TBR algorithm and we modified the algorithm in order to work on the graph database. A cypher-based algorithm was also built to identify invisible path accurately, to replay the invisible path efficiently, and to avoid wrong-placed token problem.

B. ONLINE CONFORMANCE CHECKING

The increased attentions to online process discovery researches [12], [13], [14] were followed by the growth of online conformance checking researches. Burattin *et al.* [15] proposed online conformance checking using an extended transition system by pre-computing the deviation and then adding it to the transition system. Burattin *et al.* [16] also developed a behavioral patterns technique that can detect deviations more flexibly and reliably to handle processes that are already running without knowing their previous information. However, the abstraction approach in the form of behavioral patterns is less expressive in explaining the occurrence of deviations. It also has a weakness in recognizing new deviation patterns. Zelst *et al.* [3] proposed prefix-alignments which had modified and improved conventional alignments so it can work with incomplete cases and can respond to every event stream that comes in online environment. Schuster and Zelst [4] improved the computation of prefix alignment using state space expansion so the computation can be done in increments without repeating the previous computations.

Most online conformance checking techniques that are available generally assume that the memory is infinite. For example, [3] limits the number of past activities that are taken into account for alignment calculations based on the number of windows, but the number of traces that are stored in memory is still not limited. In reality, memory has limitations that can affect the ability to accommodate incoming data streams. Burattin *et al.* [16] gave an idea of limiting the number of cases in memory by forgetting cases that were thought to be inactive. However, this method creates a missing-prefix problem that will occur if the case is still active. As a result, the cases that come afterwards will be detected as deviations. Zaman *et al.* [6] proposed a prefix imputation approach using a two-step approach to limit memory usage by selectively removing cases from memory and overcoming the missing-prefix problem by trying to recover it by connecting the missing-prefix parts. However, this approach cannot be used in all cases. For example, some business process domains may have very long active cases up to several months period. In addition, the space to accommodate forgotten cases for recovery purposes also potentially requires infinite memory.

In this paper, we propose Graph-based Online Token Replay (GO-TR) as a graph-based online conformance checking method. The method uses a graph database to store case administrations as a replay image (RI) to avoid the memory constraint problem for unlimited event streams.

C. GRAPH-BASED PROCESS MINING

An event stream is part of a business process that contains activities that have unique behavior for each case. A graph model can represent the relationship between these activities well. Sarno *et al.* [17] pioneered the implementation of process mining in a graph database environment. This study proposes a Graph-based invisible task (GIT) to perform a process model discovery that contains invisible tasks. Next, Sarno *et al.* [7] designed an event log storage directly from ERP to the graph database so that there is no need to convert the event log format to perform process discovery. The graph-based process discovery algorithm is superior in time complexity and computational time to other discovery methods [7].

In this paper, our work is aligned with the Graph-Based Process Discovery technique in [7] and [17]. We use graph database to propose graph-based online conformance checking which avoids memory contsraint problem and is robust againts unknow future behaviour.

III. FUNDAMENTALS

In this section, some of the definitions and concepts that underlie the proposed method are described.

A. EVENT LOG AND EVENT STREAM

An event log is data that is generated as a record of activities in an information system. As an example can be seen in Table 1. Each row is an event that describes an instance of a process. Event logs are generally stored in XES format (eXtensible Event Stream). XES groups each event in a single trace sequentially according to its case id. A simple illustration for the XES format of the event log in Table 1 is described in Table 2.

TABLE 1. Example of recorded event log.

Case	Activity	Resource	Timestamp
...	...	...	...
151	Register request (a)	Jack	30-12-2020:10.02
152	Register request (a)	Andy	30-12-2020:11.32
153	Register request (a)	Rob	30-12-2020:11.45
151	Examine thoroughly (b)	Anne	30-12-2020:15.06
153	Check ticket (c)	Rob	30-12-2020:16.10
...	...	...	...
152	Pay compensation (f)	Andy	31-12-2020:16.01
151	Pay compensation (f)	Jack	31-12-2020:12.00
...	...	...	...

TABLE 2. Simplified event log structure in XES format.

Case	Activity	Resource	Timestamp
151	Register request (a)	Jack	30-12-2020:10.02
151	Examine thoroughly (b)	Anne	30-12-2020:15.06
...	...	...	...
151	Pay compensation (f)	Jack	31-12-2020:12.00
152	Register request (a)	Andy	30-12-2020:11.32
...	...	...	...
152	Pay compensation (f)	Andy	31-12-2020:16.01
153	Register request (a)	Rob	30-12-2020:11.45
153	Check ticket (c)	Rob	30-12-2020:16.10
...	...	...	...

For online environment, the analyzed data is not in form of an event log but in form of an event stream. In theory, event streams are infinite sequences. According to [18], an event stream is a data stream of events which the following assumptions are made: (1) each item is assumed to contain just a small and fixed number of attributes; (2) algorithms processing data streams should be able to process unlimited amount of data without exceeding memory limits; (3) the amount of memory available to an algorithm is considered finite, and typically much smaller than the data observed in a reasonable span of time; (4) there is a small upper bound on the time allowed to process an item, e.g. algorithms have to scale linearly with the number of processed items: often the algorithms work with one pass of the data; (5) stream sources are assumed to be stationary or evolving.

Definition 1 (Event Stream): Let C denote the universe of case identifiers, and $\mathbb{A}$ denote the universe of activities. An event stream S is an infinite sequence over $C \times \mathbb{A}$, i.e., $S \in (C \times \mathbb{A})$. A pair $(c, a) \in C \times \mathbb{A}$ represents an event, i.e., activity a was executed in the context of case c. $S(1)$ denotes the first event that is received, whereas $S(i)$ denotes the i-th event

Fig. 1 illustrates the sequence of the event stream based on Fig. 1. Each event is marked with a case id and the name of the activity, for example: (**151**,a), (*152*,a), (<u>153</u>,a), (**151**,b), (<u>153</u>,c), ..., (*152*,f),..., (**151**,f),.... The sequence of these event streams contains three distinct cases. In each case there are activities that have dependency relationships.

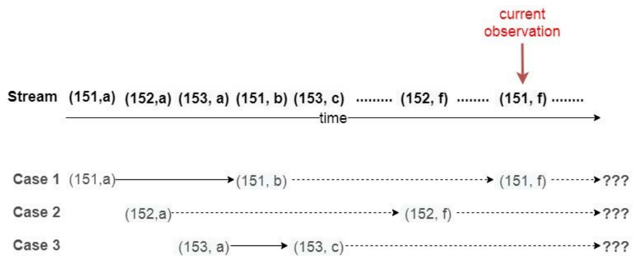

current observation
Stream (151,a) (152,a) (153, a) (151, b) (153, c) .... (152, f) .... (151, f) ....
time
Case 1 (151,a) (151, b) (151, f) ???
Case 2 (152,a) (152, f) ???
Case 3 (153, a) (153, c) ???

FIGURE 1. Example of event streams.

Event stream processing systems do not have knowledge about future events. For example, at the point marked as "current observation" in Fig. 1, it appears that Case 1 and Case 2 have reached f, which is assumed to be the end of the case. However, an event stream processing system will not recognize it as the end of the case. There are always possibilities of new activities coming, including those that should not be possible after the last node. For example, unlikely activities coming due to deviations or anomalies. In Fig. 1, Case 3 is not completed yet. It is not known when the next event will come nor the case will end.

B. ONLINE CONFORMANCE CHECKING

Traditional process mining works in an offline environment using "post mortem" data, which means it focuses on data cases that have been completed [19]. For operational support purposes, a "pre mortem" event stream data needs to be handled online. The incoming event stream is a partial trace that completes the puzzle of the event data series in a case. Each event arrival adds to the completeness of a case bound by a behavioral relationship so that event streams containing several events of alternate case ids must be handled separately and concurrently.

One of the important activities in operational support is deviation detection in form of online conformance checking. In contrast to the offline environment, the online conformance checking system has the following unique characteristics [19]: (a) it cannot see the complete case, so it focuses more on the event stream as a partial case of a particular case, (b) when there is a deviation then a fast response is required. Fig. 2 illustrates an online conformance checking system for detecting deviations.

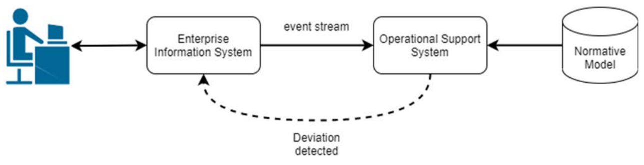


FIGURE 2. Online Conformance Checking for deviation detection.

Due to these uniqueness, the methods used in the offline environment cannot be directly applied to the online environment. Further modification and improvement are needed so that the techniques and algorithms used can respond to the data flow in real-time.

The differences of requirement between offline and online conformance checking are summarized in Table 3 with reference to [18] related to the assumptions of data streams and [19] related to the unique characteristics of online conformance checking.

TABLE 3. Comparison of requirements between offline and online conformance checking.

Requirement	Offline CC	Online CC
Algorithm	The event log data is static, so the algorithm can be executed on a case-by-case basis and requires relatively little memory.	Event streams come infinitely, so the algorithm must be able to process infinite data without being constrained by memory
Memory	No memory capacity constraint because cc can be executed sequentially per case	Must assume that memory availability is limited or less than the amount of data processed

C. PETRI NET-BASED PROCESS MODEL

In this paper, process models are represented in Petri net. A process model describes how a process should be executed.

Definition 2 (Petri Net): A Petri net is a tuple $N = (P, T, F, \alpha, m_i, m_f)$ where P is a finite set of places, T is a finite set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs as flow relation, $\alpha : T \rightarrow A$ is the transition mapping function to labels.

A marking, i.e. the state of the Petri net is a multi-set places. $m_i, m_f \in \mathbb{B}(P)$ is the initial marking dan final marking of Petri net N. A firing mechanism is required on the transition as a replay rule

Definition 3 (Firing Transition): Firing transition in Petri net can be done if there are tokens available in all input places to be consumed by transition $t \in T$, with $\bullet t \leq m$. Firing a transition denoted by $(N, m)[t\rangle$ on transition $t \in T$ with marking m wil result in a new marking $m' = (m \setminus \bullet t) \cup t\bullet$.

When the replay is firing, the token position as a marking will change. The flow of changes in marking tokens can be described in a reachability graph.

Defition 4 (Reachability Graph): If m_i is the initial marking of a Petri net N, then a set of reachable markings of N can be expressed as RS(N). The reachibility graph of N is expressed as RG(N), which is a graph where the nodes are each set of marking RS(N), while edge is a firing transition, so that an edge $m_1, t, m_2 \in RS(N) \times T \times RS(N)$ exist, if and only if $m_1[t\rangle m_2$.

D. TOKEN-BASED REPLAY

The replay technique in conformance checking performs a replay for each trace of execution on the process model by executing tasks according to the sequence of events. Among the well-known replay techniques are Token-Based Replay (TBR) and Alignment, both of which work on Petri net. This section focuses more on discussing TBR that will be adopted in our proposed method. The theory of Alignment is described in [2] and [18].

The TBR discussed in this paper is a method that works in an offline environment. Basically TBR works on trace logs and an Accepting Petri net. The Accepting Petri net is a Petri net along with a final marking. The output of the replay operation is a list of transitions activated during replay, along with some values (c, p, m and r) defined as follows:

Definition 5 (Consumed, Produced, Missed, and Remaining Tokens): Let L be the event log, and σ is the trace of L. Then c is the number of tokens consumed during replay σ. p is the number of tokens produced during replay σ. m is the number of tokens lacking during replay σ., and r is the number of tokens remaining during replay σ.

As a first step before starting the replay, it is assumed that the environment puts a token into a place where the initial marking is. The replay operation considers the activity on the trace sequentially. In every step of the process, this operation fetches the set enabled transition in current marking. If there is a transition corresponds to the current activity, the transition can be activated. A number of tokens equal to the number of input places are then added to c, and a number of tokens equal to the number of output places are then added to p.

If there are no transitions that match the current activity enabled in the current marking, then the transitions in the model that match the activity will be searched. Since the transitions cannot be activated in current marking, a marking is modified by inserting the required token to activate it, thus increasing the value of m.

At the end of the replay, if the final marking is reached, then it is assumed that the environment consumes the tokens from the final marking, so the value of c is increased. If the marking achieved after completing the replay trace is different from the final marking, then the missing tokens will be inserted and the last one calculates the number of remaining r tokens. The following formula applies during the replay: $c \leq p + t$ and $m \leq c$ so that the relation $p + m = c + r$ applies at the end of the replay.

E. GRAPH DATABASE

A graph database is a database management system that is based on graph theory. The graph theory uses nodes for storing entities and edges for relationships among them. Graph databases emphasize the relationship between data points. The implementation of the graph in this study uses Neo4j GDBMS and the graph query language Cipher [9].

The main elements of a graph are nodes and relationships. A **node** in Cipher is symbolized by brackets "()". The node that gets the additional label name "(: Label)", will limit the selection of the node designation in question based on that label. In addition, a variable can also be used on the "(variableName: Label)" node so that the next variableName can be used to access nodes labeled Label.

While a relationship is symbolized by a string such as an arrow "–>", which implicitly indicates the direction of the relationship since each relationship is associated with an ordered set of nodes, i.e., a source (from) and a destination (to) node. Cipher annotations always require two nodes, even if no specific node is declared. So a minimal example of defining a relationship in Cipher is: "()–>()" i.e., the relationship can never be without source and destination nodes. Similar to nodes, relationship groups can be delimited by specifying labels in square brackets "()-[:Label]->()" and variables such as "()-[variablename:Label]->()". Node and relationship variable declarations can be freely combined.

A simple graph can be seen in Fig. 3. For example the cipher syntax `(x:Event {p_id: 102})-[r:DFG {Flow:'NEXT'}]->(y)` will declare the variable x for the node with the label:Event and variable r for relations labeled:DFG. In addition, the p_id attribute is also declared for node x and the Flow attribute for relationship r.

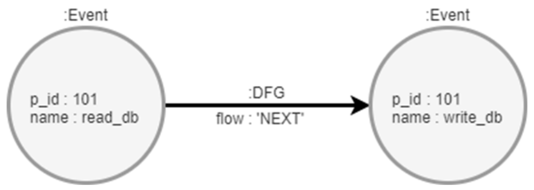


FIGURE 3. Example of a simple graph.

IV. THE PROPOSED METHOD

In this study, Graph-based Online Token-Based Replay (GO-TR) is proposed. GO-TR stores state replays in the graph database. Therefore, GO-TR requires an effective replay algorithm to accommodate lightweight write and query operations that are easy to find and trace.

Petri Net model representation in the graph database is the foundation of GO-TR. Petri net consists of places and transitions. Fig. 4 presents the proposed schema model for Petri net on the Neo4j graph database. This graphical display can be obtained using the apoc library via the `call apoc.meta.graph()` command. It appears that there are two types of nodes used in the model, namely Transition and Place. An example of the realization of the schema model can be seen in Fig. 5. There is an initial marking, a node of Place type named "source", and the final marking named "sink". There are two nodes of Transition type with labels A and B. Between A and B, there is a Place named "p_3". All relationships are identical, pointing in one direction and of Arc type. Based on the Petri Net model, a cypher algorithm can then be developed to run a replay.

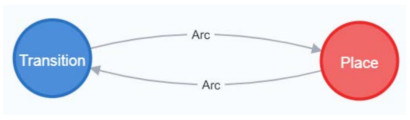
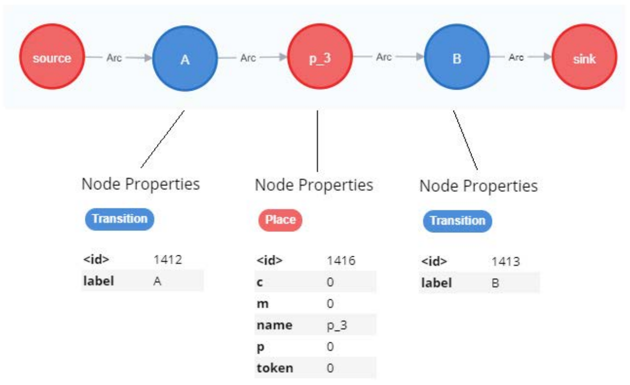


FIGURE 4. Schema model for petri net in Neo4j.



<id>	1412
label	A

<id>	1416
c	0
m	0
name	p_3
p	0
token	0

<id>	1413
label	B

FIGURE 5. Example of model instantiation of Petri net in Neo4j with additional attributs (c, p, m) in place for GO-TR.

Based on the Petri Net model in the graph database, GO-TR was built with the architecture presented in Fig. 6. GO-TR consists of several components, they are: a reference model, replay image, reachability graph, graph-based token replay, and also graph-based invisible path identification and replay.

A. MODEL REFERENCE

The initial component that is required for conformance checking is the reference process model. We propose the process model representation in the form of Petri net in a graph database. Fig. 7 illustrates several scenarios based on the availability of data to obtain the representation of the Petri net. The scenarios are:

1. If an event log is available in a structured or semi structured format that can be imported into the graph database (or it could be that the event log is already available natively in the graph database), then a graph-based process model discovery is made [7], [20], [21]. The results obtained are still in the directly followed graph (DFG) representation. The next step is to convert the DFG to Petri net using algorithm 1.

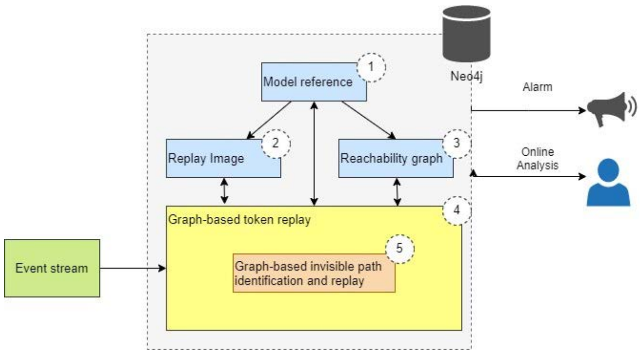


FIGURE 6. Architecture of GO-TR.

Algorithm 1: Algorithm To Convert Directly Followed Graph (DFG) Representation To Petri Net Representation

```
Input: DFG model with relationship type
 1: function DFGtoPetrinet (dfgModelId)
 2:     detects the relationship type between two nodes:
 3:         (a.) if Sequence or AND:
 4:             add Place and relation between the two
                nodes
 5:         (b.) if Xor_split:
 6:             add Place and relation at split point
 7:             merge relation on input side to Place
 8:         (c.) if Xor_join:
 9:             add Place and relation at join point
10:             merge relation on the output side of Place
11:     delete all DFG relationships
12: obtained the process model in the representation of Petri Net
```

2. Process model discovery is executed directly from the event log in a structured or semi structured format such as XES or CSV. In this study, the process discovery uses the PM4Py library, namely the alpha algorithm or inductive miner. The result of discovery is a process model with Petri net object format. By using algorithm 2, the process model in Petri net object

Algorithm 2: Algorithm to Generate Petri Net Representation of Model Process on Graph Db

```
Input: net                    // a Petri net object of Process Model
 1: function generatePetrinetForGraphDB (net)
 2:   (2.a.) createTransition(tx, activity)
 3:   (2.b.) createPlace(tx, place)
 4:   (2.c.) createRelationship_placeToTrans (tx, pName, tLabel)
 5:   (2.d.) createRelationship_transToPlace (tx, tLabel, pName)
```

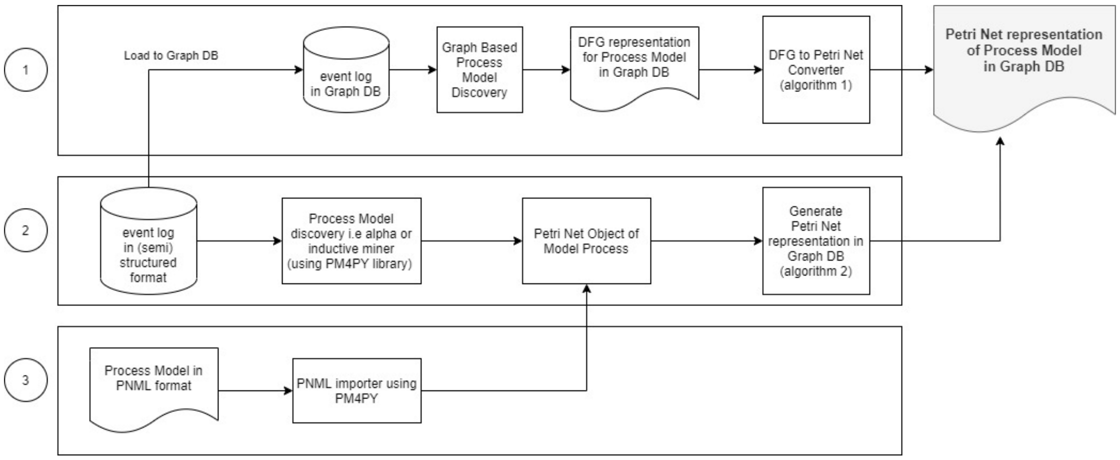

1
Load to Graph DB
event log in Graph DB
Graph Based Process Model Discovery
DFG representation for Process Model in Graph DB
DFG to Petri Net Converter (algorithm 1)
Petri Net representation of Process Model in Graph DB
2
event log in (semi) structured format
Process Model discovery i.e alpha or inductive miner (using PM4PY library)
Petri Net Object of Model Process
Generate Petri Net representation in Graph DB (algorithm 2)
3
Process Model in PNML format
PNML importer using PM4PY

FIGURE 7. Techniques for obtaining the Petri net representation of process model in a Graph database from several data source formats.

format can be used to generate its representation in the graph database.

3. The process model is available in PNML format. PNML importer from the PM4PY library is used to load the process model into the graph database to get a Petri net object. Based on the petrinet object, a representation of the process model is generated in the graph database using algorithm 2.

B. REPLAY IMAGE

A Replay Image (RI) is proposed to store case administration. Algorithm 3 explain algorithm to create an RI from its master reference Petri net model.

Definition 6 (Replay Image): Replay image is Petri net and its updated marking for each unique case. A Replay Image is a tuple $N_{RI} = (N, \beta, m_c, i)$ where β is a function that maps each place P with the property set {c, p, m} where $c \in \mathbb{Z}$ is the number of tokens consumed, $p \in \mathbb{Z}$ is the number of tokens produced, and $m \in \mathbb{Z}$ is the number of missing tokens inserted. m_c is the current marking which shows the marking of the last update. Where i is the unique identity of the replay image based on the case id of the checked event stream.

The replay behavior in RI follows firing transition mechanism in Def. 3. As Def. 5, RI stores the replay data, including token consumed (c), token produced (p), and token missing (m). As an illustration, the comparison between the reference model and the Replay Image is presented in Fig. 8.

Algorithm 3: Create Replay Image

Input: $N = \left(P, T, F, \alpha, m_i, m_f\right), c \in C$

	Pseudocode	***Cipher***
1	*Find rootA as the initial marking node of model reference*	*MATCH (rootA:Place {type:'master', im:True})*
2	*Clone the rootA to rootB as the root node of replay image*	*WITH distinct rootA CALL apoc.refactor.cloneNodes([rootA]) YIELD input, output WITH rootA, input, output AS rootB SET rootB.type='clone', rootB.p_id = $c*
3	*Continue to clone the rest sub graph of model reference to replay image*	*WITH rootA, rootB MATCH path = (rootA)-[∗]->(node) WITH rootA, rootB, collect(distinct path) as paths CALL apoc.refactor.cloneSubgraph FromPaths(paths, { standinNodes:[[rootA, rootB]] }) YIELD input, output, error*
4	*Update the replay image with new attributes*	*WITH collect(DISTINCT output) AS nodes UNWIND nodes as node SET node.type = 'clone', node.p_id = $p_id*

C. REACHABILITY GRAPH

The reference model in the petrinet model can be brought to its reachability graph by using PM4Py which provides libraries for generating reachability graphs. Fig. 9 is an example of a process model that will produce a reachability graph presented in Fig. 10.

The resulting reachability graph object is then loaded into the graph database using algorithm 4. First, all states which are nodes in neo4j are created (lines 1-3). Then the states are connected using transitions in form of relationships in neo4j (lines 4-12). Next, all the relationships are labeled invisible. This is important for designing the invisible task identification algorithm.

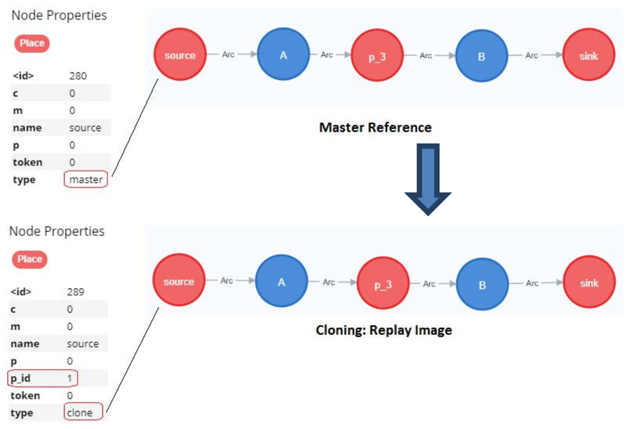


FIGURE 8. Master reference and its replay image.

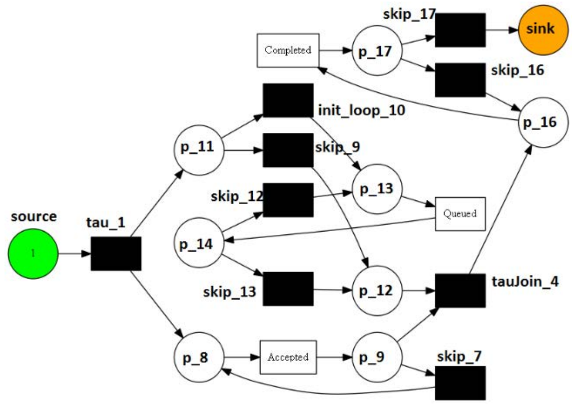


FIGURE 9. Example of a process model generated from BPIC13 data.

D. GRAPH-BASED TOKEN REPLAY FOR ONLINE CONFORMANCE CHECKING

The main algorithm in the Graph-based online Token Replay (GOTR) technique is to replay each event arrival (stream) to a reference model in the form of a replay image.

Definition 7: Graph-based online token replay (GO-TR) Given a Petri net $N = (P, T, F, \alpha, m_i, m_f)$*, a replay action* $(N, m)[t\rangle$*, and a replay image* $N_{RI} = (N, \beta, m_c, i)$*. Graph-based Token replay is obtained by reading the transition conditions* $t \in T$ *to be replayed along with the token requirements at the input place nya* $p \in P$ *through the replay image* N_{RI} *in the graph database. Based on the information obtained, the necessary handling is carried out to ensure that the replay can be carried out, such as tracing the invisible path and inserting the missing token. Followed by running a replay on the appropriate transition while updating it (write) on the replay image.*

The GO-TR schema can be seen in Fig. 11. Basically, GO-TR accepts input data in form of event streams. Each event that comes is accompanied by its respective case id as a replay reference. When a case comes with a new id, a replication of process model from the reference master will be created as a Replay Image (RI).

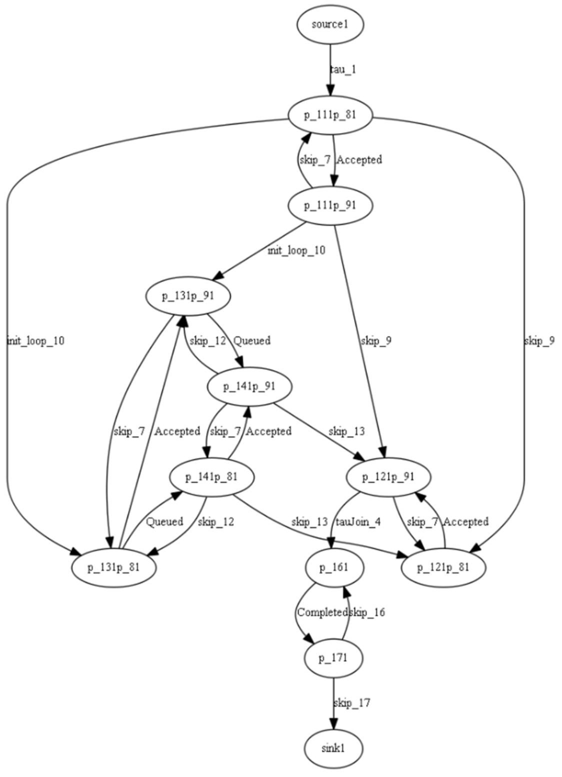


FIGURE 10. Example of the reachability graph obtained from the process model in Fig. 9.

Algorithm 4: Generate Reachability Graph in neo4j

```
    Input: rg, net    # reachability graph object, Petri net
    function generateRGinNeo4j (rg, net)
1       for s in rg.states:
2           sname = s.name
3           createState(session, sname)
4       for t in net.transitions:
5           tname = t.name
6           if t.name not in trans_name:
7               tlabel = 'invisible'
8           else:
9               Tlabel = t.name
10          source = t.from_state
11          target = t.to_state
12          createRelationship(session,
              source.name, t.name, tlabel,
              target.name)
```

Algorithm 5 explains in detail the algorithm for replaying the GO-TR. The GO-TR technique begins by detecting the identity of the event that comes. If this event is an event with a case id that has never been detected, it will be recognized that a new process is in progress (line 6). Therefore, it is necessary to prepare a new Replay Image (RI) in the graph database that can be recognized through the case id identity (line 7). Next, the program will make sure the activity name

Algorithm 5: Graph-Based Online Token Replay

```
Input: $N = (P, T, F, \alpha, m_i, m_f)$, $RG(N)$, $S \in (C \times \mathbb{A})$
1  states, places ← getReachabilityGraphProperties(RG)
2  while true do
3      event ← $S(i)$                                        // get i-th event of event streams
4      c ← event[0]                                          // extract case-id from current event
5      a ← event[1]                                          // extract activity label from current event
6      if c not in id_list:
7      createReplayImage (c, model_ref)                      // algorithm 3
8      if a not in activity_name:
9      unknownActivities ← event
10     Send an alert that the activity on event $(c, a)$ is not recognized
11     continue
12     eip ← getAllEmptyInputPlaces$(c, a)$                  // detect if empty input places are exist
13     if eip is exist:
14         if isAllEipConnectedToInvTask(c, eip):            // all input places are connected to inv task
15         if invisiblePathReplay (c, currentMarkingName, eip, states, places):   // algorithm 8
16             replay_info ← normalReplay (c, a)             // algorithm 6
17             else:
18             replay_info ← replayWithInsertToken(c, $m_f$, a)   // algorithm 7
19             Send an alert that a deviation occured in event (c, a)
20         else:
21         replay_info ← replayWithInsertToken (c, $m_f$, a)
22         Send an alert that a deviation occured in event (c, a)
23     else:                                                 // empty Input Place is NOT exist
24     replay_info ← normalReplay (c, a)
```

is recognized in the reference model; if it is not recognized, it will be skipped and recorded in the unknownActivities list as a deviation (lines 8-11). Next, the algorithm will detect whether there are empty input places in the activity. If empty input places are detected then the activity cannot be enabled (line 12).

If there are empty input places (line 13) which are all connected to an invisible task, it is necessary to identify an invisible path using algorithm 8 (lines 15). If the invisible path is found, then the replay can be executed normally (line 16). However, if the invisible path fails to be identified, then the replay must be executed by inserting the missing token (line 18). The insertion of a missing token is a sign that there is a deviation. In that case, a warning (in real-time) containing case id information and activity name is then issued (line 19).

If there are empty input places (line 13) which are not connected to an invisible task, the replay can only be executed by inserting a missing token (line 21) so it will also be declared as a deviation.

If there is no empty input place (line 13) which means all of the input places are filled with tokens, then the replay can be executed normally (line 24).

The explanation of the Cypher algorithm for the normal replay is presented in algorithm 6, while the replay that requires the insertion of missing tokens is described in algorithm 7.

Algorithm 6: Normal Replay

Input: $N = (P, T, F, \alpha, m_i, m_f)$, $(c, a) \in CxA$

	Pseudocode	***Cypher***
1	*Find all input places of matched activity*	*MATCH (ip: Place {p_id: $c })-[r]->(t:Transition {label:$a})*
2	*Consume token from each input place and update the relationship attributes.*	*SET ip.token = ip.token - 1, ip.c = ip.c + 1, r.c = r.c + 1, r.f = r.f + 1*
3	*Find all output places of matched activity*	*WITH distinct t AS t, ip, collect(ip) as ips MATCH (t)-[r]->(op)*
4	*Produce token to each output place and update the relationship attributes.*	*SET op.token = op.token + 1, op.p = op.p + 1, r.p = r.p +1, r.f = r.f + 1*

E. GRAPH-BASED INVISIBLE PATH REPLAY

The basic algorithm of Token-Based Replay cannot replay an invisible task. The algorithm needs missing token insertion,

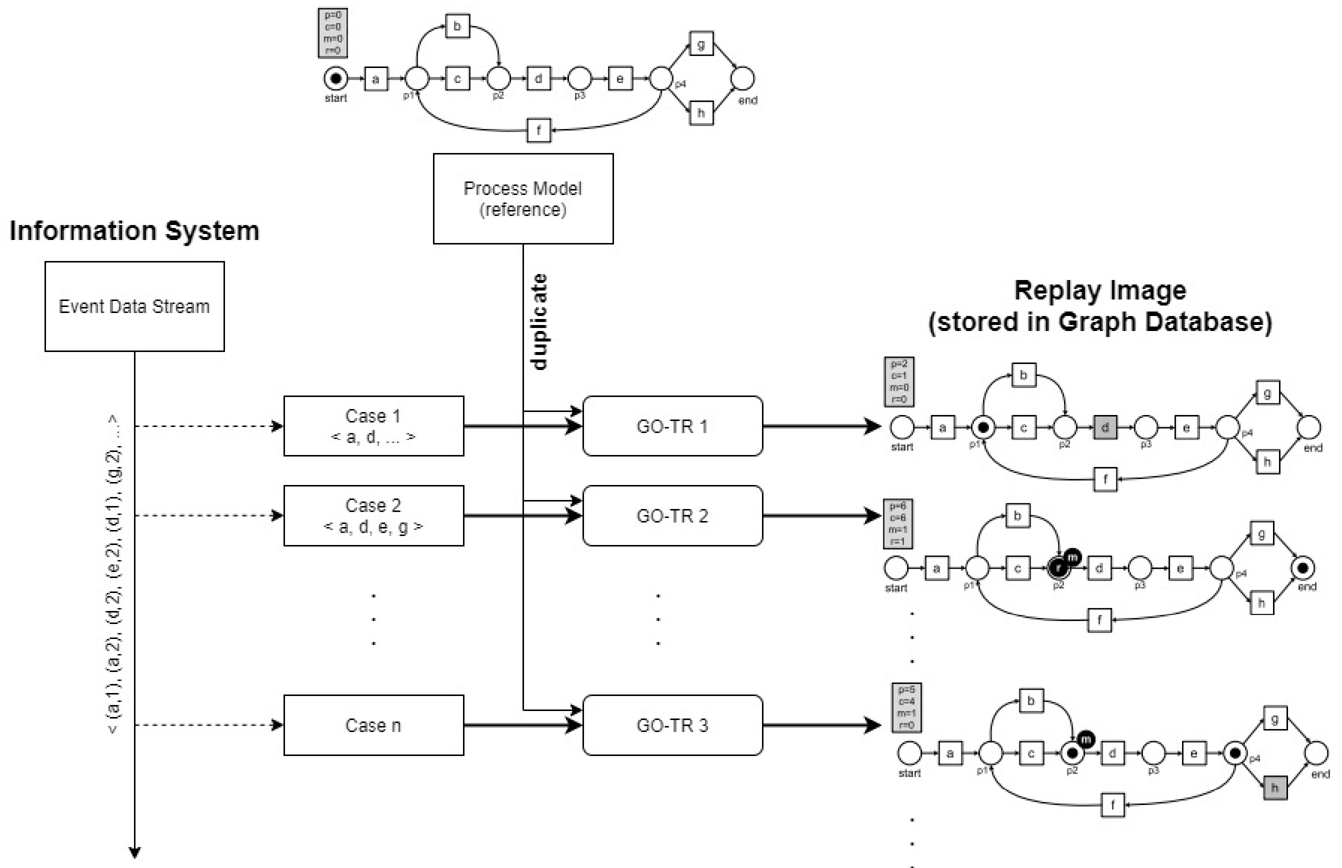


FIGURE 11. **Proposed graph-based online token replay.**

which will be detected as a deviation, to replay a visible task which preceded by an invisible task. We proposed graph-based method to handle the invisible task replay. Our proposed method takes advantage of the graph database's ability to store graph data natively and its fast node traversal capabilities.

In every iteration of the replay event that comes (in algorithm 4), it always begins with checking whether all input places of the activity to be replayed have all tokens filled (line 13). If there is an input place that is not filled with a token, it is necessary to check whether there is an invisible path that can be executed so that this input place can then be filled with a token.

The algorithm for identifying and replaying invisible paths is presented in Algorithm 8. This algorithm will start when Algorithm 5 (lines 12-14) runs input places that do not contain tokens which are all connected to an invisible task. To detect an invisible path, it is necessary to first find the state position in the reachability graph as the *source_state* (line 1). If a *source_state* exists (line 4), then the algorithm needs to check the existence of *target_state_candidates* (lines 5). From all the *target_state_candidates*, the algorithm looks for the one with the shortest distance from *source_state* (line 6).

An invisible path is found when a reachable and the shortest distance *target_state* is found (line 7-8). The invisibleReplay function (algorithm 9) will update the attributes of all nodes and edges along the invisible path that is found to simulate replay on an invisible path (line 9). Returns is True if the invisible path is found (line 11). On the other hand, return is False if the spf_target_state is not obtained, hence the invisible path will not be obtained (lines 11-12).

F. EXPERIMENT SET UP

The experiment was carried out on a computer with an Intel Core i7-3632QM processor with 16GB of RAM, and Python 3.6. The pm4py 2.2.4[1] library was used to perform process discovery and to generate the reachability graphs.

The following are the scenarios of the experiment that was carried out:

1) THE CORRECTNESS OF THE TOKEN-BASED REPLAY ADAPTATION

GO-TR adapts TBR. Conformance checking experiments were carried out with scenarios using normal cases, cases

[1]https://github.com/pm4py/pm4py-core/releases/tag/2.2.4

Algorithm 7: Replay With Insert Token (Abnormal Replay)

Input: $N = (P, T, F, \alpha, m_i, m_f)$, $(c, a) \in CxA$

	Pseudocode	***Cipher***
1	*Check if exist input place with a missing token, then insert an artificial token and update the missing token status as true (1)*	*OPTIONAL MATCH (ip: Place {p_id: $p_id})–>(e:Transition {label:$activity}) WHERE ip.token = 0 SET ip.token = ip.token + 1, ip.m = ip.m + 1*
2	*Count number of missing token*	*WITH ip_mt, count(ip_mt) as num_of_missing_token*
3	*Consume token from each input places and update the relationship frequency*	*MATCH (ip: Place {p_id: $p_id})-[r]->(t:Transition {label:$activity}) SET ip.token = ip.token - 1, ip.c = ip.c + 1, r.c = r.c + 1, r.f = r.f + 1*
4	*Produce token to each output place and update the relationship frequency*	*WITH distinct t AS t, ip_mt MATCH (t)-[r]->(op) SET op.token = op.token + 1, op.p = op.p + 1, op., r.p = r.p +1, r.f = r.f + 1*
5	*Return the number of input places with missing token*	*RETURN count(ip_mt) as num_of_missing_token*

with deviations. and cases with multi-input invisible tasks, to prove the GO-TR technique was working properly.

The experimental dataset used public real-life data from BPIC 2013 Incident.[2] The inductive miner algorithm was used with a noise threshold of 0.2 to obtain a process model that contained many invisible tasks and one invisible task with multi visibility input tasks. The case used for testing was a modification from one of the 2013 BPIC Incident data cases.

We used the ITBR program provided by PM4PY to test the TBR technique. This program only works for offline conformance checking, so it requires input in form of an event log.

2) THROUGHPUT

This experiment was conducted to compare GO-TBR against PA in average speed for replaying each event arrival (stream). The event data was sent to the conformance checking machine on streams based on the order of arrival (not following the arrival timestamp). The data set used was CCC19[3] which is a public data set. We duplicate the number of case ids as much as five and ten folds of their original number of 20 available case variants to compare the throughput of both techniques.

3) MEMORY CONSUMPTION

This experiment was aimed to compare memory usage between PA and GO-TR. The dataset used was CCC19 public data. The variable observed in this experiment was the amount of memory consumed along the arrival of the event.

V. RESULT AND DISCUSSION

This section presents and discusses the results of the experiments that had been carried out.

A. THE CORRECTNESS OF THE TOKEN-BASED REPLAY ADAPTATION

In this section, observations were made on several test scenarios to ensure the correctness of the GO-TR replay results by comparing them with TBR results. The dataset used was BPIC13 incident management. Fig. 12 presents the process model of the BPIC13 Incident. The model was generated with the help of PM4PY using Inductive Miner with a noise threshold of 0.2. The PM4PY was also used to generate the reachability graph in Fig. 13 from its Petri net object model.

There are AND branches (e.g. AND-Split in tau_1) and loops (e.g. Accepted). There is also an invisible task tau-Join_4 which has two inputs. The first input, p_12, is linked to the invisible tasks skip_13 and skip_9. While the second input, p_9, is connected to the visible task Accepted. This condition is hereinafter referred to as invisible task with multi visibility input tasks.

The first experiment used a normal case Queued $\rightarrow$ Accepted $\rightarrow$ Completed. This experiment was to prove that the TBR algorithm used in GO-TR could recognize invisible paths. The experimental results are presented in Table 4. The next experiment used a case containing a loop, i.e. Queued $\rightarrow$ Accepted $\rightarrow$ Queued $\rightarrow$ Completed $\rightarrow$ Completed. The experimental results are presented in Table 5. The marking movement presented in Table 6 can be explained as follows. First of all the system provides initial marking on a place labeled as "Source". Then with the arrival of the first event, labeled as "Queued", the TBR algorithm starts to work. With the reference model in Fig. 12, it can be seen that the initial marking position on "Source" causes all input places in "Queued" to not have tokens. Therefore, the TBR algorithm will try to (p_8, p_11) $\rightarrow$ (p_8, p13), can be found so that "Queued" can be replayed normally to produce state (p_8, p_14).

The results in Table 4 and Table 5 show that GO-TR and ITBR can work well in all normal cases. They also give the same results for all statistics.

The third experiment with "wrong-placed token" problem contains the following activities: Queued $\rightarrow$ Completed $\rightarrow$ Completed. A good TBR algorithm will recognize that it is necessary to add a missing token to p_16. With marking

[2]https://data.4tu.nl/articles/dataset/BPI_Challenge_2013_incidents/12693914/1

[3]https://data.4tu.nl/articles/dataset/Conformance_Checking_Challenge_2019_CCC19_/12714932

Algorithm 8: Invisible Path Replay

```
1  source_state ← findStatename(states, $m_c$)     // find a state of RG that matches the current marking $m_c$
2  current_states ← current_marking(c), current_edge_freq(c), current_edge_produced(c)     // temporary storage for current states
3  function InvisiblePathReplay(c, $m_c$, eip, states, places)
4  if source_state is exist:
5      target_state_candidates ← findTargetStatesCandidates (states, eip)
6      spf_target_state, spf_trans ← findInvisiblePath(source_state, target_state_candidates)
7      If spf_target_state is exist:
8      target_marking ← extractTokenPlace(spf_target_state, places)
9      invisibleReplay(c, spf_trans)     // algorithm 9, invisible replay with no need to simulate token replay
10     Return True
11     else
12     Return False
13 else
14     If invisibleTokenReplay(c, $m_c$, eip) is succeed     // invisible replay with token replay
15     Return True
16     else
17     rollback(current_states)
18     Return False
```

Algorithm 9: Invisible Replay

Input: $N = (P, T, F, \alpha, m_i, m_f)$, $(c, a) \in C \times \mathbb{A}$

	Pseudocode	*Cipher*
1	*Prepare a pair of a list of activities a*	*WITH $a AS as WITH [i in range(0, size(ts)-1) | {a:as[i]}] AS pairs UNWIND pairs as pair*
2	*Match each activity in model reference with each activity in a, then set its* ***input place and relationship*** **attributes**	*MATCH (ip {p_id:$c})-[r]->(tr:Transition) WHERE tr.name = pair.t SET r.f = r.f + 1, ip.token = ip.token - 1, ip.c = ip.c + 1*
3	*Match all output places of activated activities*	*WITH tr, pair MATCH (tr)-[s]->(op)*
4	*Set its* ***output place and relationship*** **attributes**	*WITH distinct s AS s, op SET s.f = s.f + 1, op.token = op.token + 1, op.p = op.p + 1*

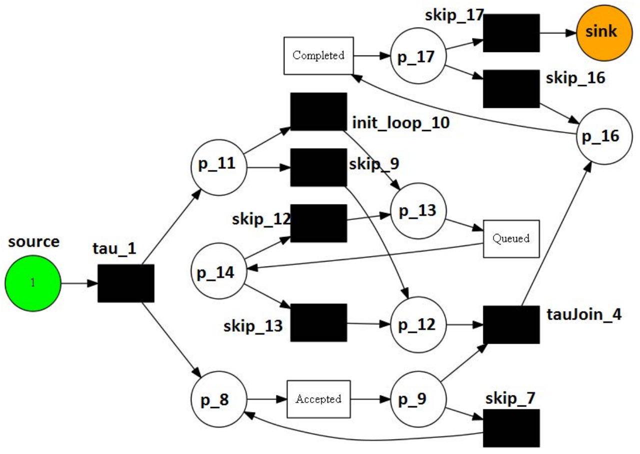


FIGURE 12. Discovery result from the process model with BPIC13 dataset.

TABLE 4. Results of the experiment that use a normal case.

Dataset BPIC13	ITBR	GO-TR
Trace	Queued, Accepted, Completed	
Consumed token	10	10
Produced token	10	10
Missing token	0	0
Remaining token	0	0
Fitness	1.00	1.00
Wrong-place token problem	-	-

position on p_8 and p_14, the necessity to add a missing token is because a replay is required on Accepted activity to enable taujoin_4. The GO-TR and ITBR algorithms that work in "wrong-placed token" problem is different. It is interesting. The results from the marking in the third experiment for GO-TR and ITBR are presented in Table 6.

The marking movement presented in Table 6 can be explained as follows. First of all the system provides initial marking on a place labeled as "Source". Then with the arrival of the first event, labeled as "Queued", the TBR algorithm

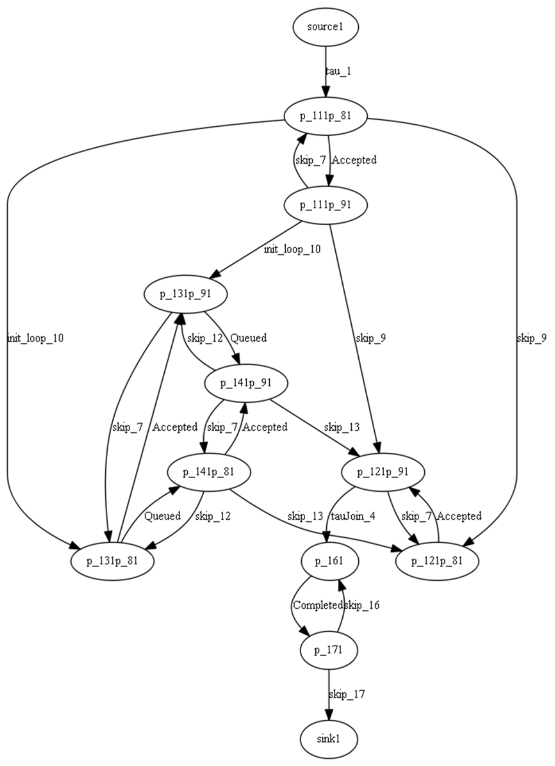


FIGURE 13. Reachability graph generated from the model in Fig 12.

TABLE 5. Results of the experiment that use a case containing loop.

Dataset BPIC13	ITBR	GO-TR
Trace	Queued, Accepted, Queued, Completed, Completed	
Consumed token	14	14
Produced token	14	14
Missing token	0	0
Remaining token	0	0
Fitness	1.00	1.00
Wrong-place token problem	-	-

starts to work. With the reference model in Fig. 12, it can be seen that the initial marking position on "Source" causes all input places in "Queued" to not have tokens. Therefore, the TBR algorithm will try to find the shortest possible invisible path from "source" to p_13. In this case, an invisible path, source → (p_8, p_11) → (p_8, p13), can be found so that "Queued" can be replayed normally to produce state (p_8, p_14).

The next event is "Completed". A token at position p_16 is required to enable the "Completed" activity. A different work between GO-TR and ITBR algorithms can be explained as follows.

a. The ITBR algorithm finds an invisible path p_14 → p_12 → p_16. It then tries to run a replay. But when it comes to p_12 the replay stops as p_9 has no token and it is not connected to the invisible task. Therefore, tauJoin_4 cannot be activated. As a result, the replay attempt via invisible path failed to reach p_16. However, ITBR is already running tokens from p_14 to p_12 and so the marking becomes (p_12, p_8). The current position of the token at p_12 is the "wrong-placed token". The token position at p_12 will cause errors on analysis. Because, apart from being able to be achieved through "Queued" activation, p_12 can also be directly reached via skip_9.
b. Meanwhile, GO-TR, by using algorithm 1, will find the invisible path from (p_14, p_8) to p_16 through the reachability graph. In this case the invisible path is not found so the marking does not change i.e. it stays at (p_14, p_8). As a result, GO-TR is safe from the "wrong-placed token" problem.

TABLE 6. Marking result comparison between ITBR and GO-TR.

Seq	Events	Marking (ITBR)	Marking (GO-TR)
1	(system produce an initial marking token)	Source	Source
2	'1-717918204', 'Queued'	P_14, p_8	P_14, p_8
3	'1-717918204', 'Completed'	**P_12**, p_8, p_17	**P_14**, p_8, p_17
4	'1-717918204', 'Completed'	**P_12**, p_8, p_17	**P_14**, p_8, p_17
5	(system consume final marking token)	**P_12**, p_8	**P_14**, p_8

Next step, both ITBR and GO-TBR algorithm require to insert a missing token on p_16 to enable "Completed". The replay results are the (p_12, p_8, p_17) marking for ITBR and the (p_14, p_8, p_17) marking for GO-TR.

The last event to be replayed is the second activity with "Completed" label. This time, ITBR finds the route p_17→p_16 as an invisible path. On the other hand, in GO-TR, because the previous activity was a deviation, no state of "(p_14, p_8, p_17)" is found in the reachability graph. In that case, we had to run the algorithm 2 through a simulation by tracing the invisible paths. If the search failed to reach the target (the missing token place point), the algorithm rolls back all states of the invisible paths to the initial condition. As a result, both ITBR and GO-TR can find the invisible path that reaches p_16. After a successful replay of "Completed", the marking positions will return to (p_12, p_8, p_17) for ITBR and (p_14, p_8, p_17) for GO-TR.

B. THROUGHPUT

This section describes the experimental results of the comparison between the execution performance of PA and GO-TR by completing a number of event data streams. Comparisons are made in a single processing environment to observe the

TABLE 7. Comparison of duration to complete event streams arrival using CCC19 dataset.

Online conformance checking technique	duration (seconds)									
	20 cases (697 events)	40 cases (1394 events)	60 cases (2091 events)	80 cases (2788 events)	100 cases (3485 events)	120 cases (4182 events)	140 cases (4879 events)	160 cases (5576 events)	180 cases (6273 events)	200 cases (6970 events)
Prefix Alignment OCC w=full	78,73	155,24	226,8	306,27	377,61	438,88	523,34	601,95	733,34	776,03
Prefix Alignment OCC w=10	45,72	98,15	141,45	183,01	231,50	287,00	337,20	370,41	426,50	474,51
Prefix Alignment OCC w=5	27,22	59,6	86,72	116,11	138,00	175,19	201,25	228,29	255,78	286,65
Prefix Alignment OCC w=2	16,34	32,72	48,85	66,24	79,38	95,61	115,69	131,54	170,76	162,93
Prefix Alignment OCC w=1	11,48	22,59	34,88	47,06	55,59	68,57	79,85	93,00	104,89	111,53
GO-TR	**8,54**	**17,22**	**27,32**	**38,97**	**52,60**	**63,48**	**77,86**	**96,77**	**113,24**	**132,17**

TABLE 8. Comparison of throughput between PA and GO-TR.

Online conformance checking technique	Throughput (event per seconds)									
	20 cases (697 events)	40 cases (1394 events)	60 cases (2091 events)	80 cases (2788 events)	100 cases (3485 events)	120 cases (4182 events)	140 cases (4879 events)	160 cases (5576 events)	180 cases (6273 events)	200 cases (6970 events)
Prefix Alignment OCC w=full	8,85	8,98	9,22	9,10	9,23	9,53	9,32	9,26	8,55	8,98
Prefix Alignment OCC w=10	15,24	14,20	14,78	15,23	15,05	14,57	14,47	15,05	14,71	14,69
Prefix Alignment OCC w=5	25,61	23,39	24,11	24,01	25,25	23,87	24,24	24,43	24,52	24,32
Prefix Alignment OCC w=2	42,66	42,60	42,80	42,09	43,90	43,74	42,17	42,39	36,74	42,78
Prefix Alignment OCC w=1	60,71	61,71	59,95	59,24	62,69	60,99	61,10	59,96	59,81	62,49
GO-TR	81,62	80,95	76,54	71,54	66,25	65,88	62,66	57,62	55,40	52,74

throughputs, which is the number of events that can be completed per second.

This experiment uses the CCC19 public dataset that provides log data and its reference model. The log data is in an event-log format (XES), while the reference model is in pnml format. In this paper, experiments that uses PA takes reference to the source code of the IAS prefix-alignment program.[4]

In the GO-TR simulation, the CCC19 log data needs to be converted from an event log to an event stream representation by sorting each event based on arrival time.

We need to load the reference model into the graph database using Algorithm 2 as the first step of the GO-TR experiment. The algorithm will generate a petri net representation of the model process from its pnml format on the graph database. In the next step, a reachability graph (RG) is also needed by GO-TR to identify the invisible paths. We use the PM4PY library to generate the RG model from the Petri net object. The resulting reachability graph model is loaded into the graph database using Algorithm 4.

The variables to be observed were the duration required to complete the inspection of the event stream data and the number of events. Based on the known duration and number of events, the throughput value (T) can be obtained using the formula T=n/d, where n = number of completed events, d = duration of completion, and T= throughput. The experimental results are presented in Table 7 and the throughput results are shown in Table 8 and depicted in Fig. 14.

The PA with "w=full" requires the highest computation because it performs a complete optimal alignment computation from the beginning of each new event arrival. A PA with a small window, for example "w=1", is very fast. However, it reduces the guarantee of getting optimal alignment. Moreover, it still has memory limitation problems.

Based on the results of the experiment, it appears that GO-TR with a small number of cases has the highest throughput. This is due to the simple computing that it can execute the replays in a short time. On the other hand, the PA computation is very influential on replay speed. The faster the computation, the greater the throughput. At "w=1", the PA throughput is close to GO-TR throughput.

The data in Table 8 shows that GO-TR experiences a decrease in throughput as the number of handled cases increase. It is because that in GO-TR, each case has its own representation of the RI. The more cases that are handled, the longer the query time required in the replay process.

[4]https://github.com/fit-daniel-schuster/online_process_monitoring_using_incremental_state-space_expansion_an_exact_algorithm

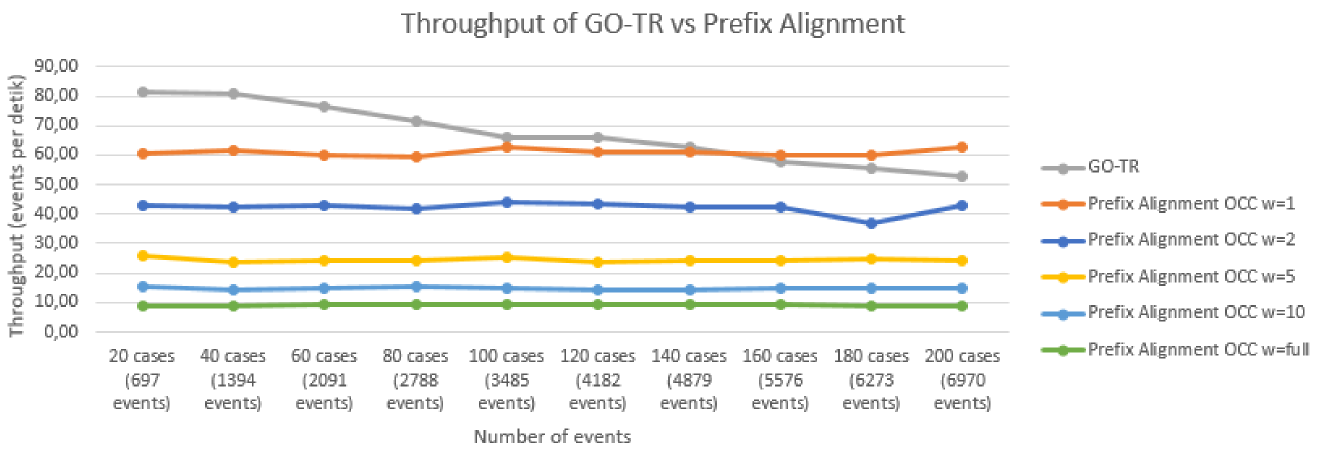


FIGURE 14. Throughput comparison between GO-TR and PA.

C. MEMORY CONSUMPTION

This third experiment is the most important to prove the reliability of the online conformance checking technique against memory limitations. The test is carried out by observing the program's (python's) memory consumption when executing PA and GO-TR.

Fig. 15 presents the result of the experiment when using PA with "w=2" for 60 minutes. The results in Fig. 15 indicate that memory consumption increases linearly with the number of coming events. This is due to the space needed by the PA to accommodate the administration of each case. This space continues to stay in memory as long as the observed case is still active. The more cases received, the greater the space required. So, if the arrival of the case is declared to be infinite, it will require an infinite memory space to accommodate. This makes PA vulnerable to memory limitations.

The results from the GO-TR experiment in Fig. 15 indicate that the arrival of event streams has no effect on memory consumption. This is possible because the administration of all cases are stored in the graph database. The running program simply replays the RI of a case based on the event id that comes. The results of the replay immediately update the RI. Based on the experiment, it is proven that GO-TR is invulnerable from memory limitation problems.

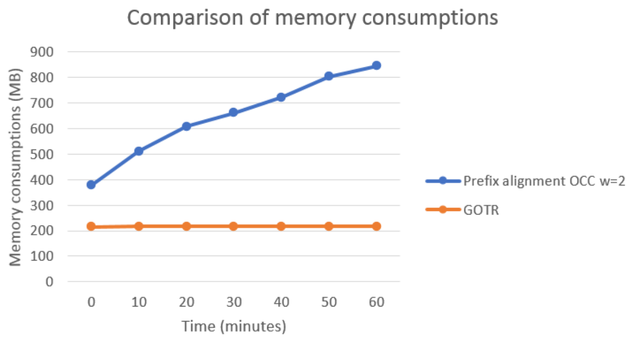


FIGURE 15. Memory consumption using PA with w=2 and GO-TR.

VI. CONCLUSION

In this paper, we propose the Graph-based online token replay (GO-TR) as a replay-based online conformance checking which is invulnerable to memory limitations. Our proposed solution adapts the token replay technique on a graph database. By building the GO-TR, we made several contributions, which are: proposing replay images as the representations of the Petri Net models in a graph database, adapting Token-based Replay on a graph database for online conformance checking that receives event stream data, and proposing a cypher-based invisible path identification and an invisible path replay algorithm.

Based on observations and analysis from the experiments, it is proven that GO-TR has been successful in adapting TBR and is invulnerable to the wrong-placed token problem. For small amounts of data, GO-TR works with the highest throughput when compared to PA. However, GO-TR's throughput performance decreases as the amount of data increases. In terms of memory usage, GO-TR shows its advantages over PA as it is invulnerable to memory limitations.

In future work, a study will be conducted to maintain the query performance along with the data growth. In addition, it is also necessary to observe the performance of its response to high-speed data.

REFERENCES

[1] A. Rozinat and W. M. P. Van Der Aalst, "Conformance checking of processes based on monitoring real behavior," *Inf. Syst.*, vol. 33, no. 1, pp. 64–95, Mar. 2008.
[2] A. Adriansyah, "Aligning observed and modeled behavior," Technische Universiteit Eindhoven, Eindhoven, The Netherlands, SIKS Diss. Ser. 2014-07, 2014.
[3] S. J. Van Zelst, A. Bolt, M. Hassani, B. F. Van Dongen, and W. M. P. Van Der Aalst, "Online conformance checking: Relating event streams to process models using prefix-alignments," *Int. J. Data Sci. Anal.*, vol. 8, no. 3, pp. 269–284, Oct. 2019.

[4] D. Schuster and D. S. J. Van Zelst, "Online process monitoring using incremental state-space expansion: An exact algorithm," in *Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics* (Lecture Notes in Computer Science), vol. 12168. Cham, Switzerland: Springer, 2020, pp. 147–164.

[5] D. Schuster and G. J. Kolhof, "Scalable online conformance checking using incremental prefix-alignment computation," in *Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics* (Lecture Notes in Computer Science), vol. 12632. Cham, Switzerland: Springer, 2021, pp. 379–394.

[6] R. Zaman, M. Hassani, and B. F. Van Dongen, "Prefix imputation of orphan events in event stream processing," *Frontiers Big Data*, vol. 4, Oct. 2021, Art. no. 705243.

[7] R. Sarno, K. R. Sungkono, M. Taufiqulsa'di, H. Darmawan, A. Fahmi, and K. D. Triyana, "Improving efficiency for discovering business processes containing invisible tasks in non-free choice," *J. Big Data*, vol. 8, no. 1, pp. 1–17, 2021.

[8] K. R. Sungkono, A. S. Ahmadiyah, R. Sarno, M. F. Haykal, M. R. Hakim, B. J. Priambodo, M. A. Fauzan, and M. K. Farhan, "Graph-based process discovery containing invisible non-prime task in procurement of animal-based ingredient of halal restaurants," in *Proc. IEEE Asia Pacific Conf. Wireless Mobile (APWiMob)*, Apr. 2021, pp. 134–140.

[9] A. S. Ahmadiyah, I. N. Hazimi, D. M. Rozi, R. A. Wibisono, D. Fitrado, M. A. Rifqi, R. Sarno, and K. R. Sungkono, "Business processes discovery in halal restaurant kitchen using graph-based algorithm," in *Proc. IEEE Asia Pacific Conf. Wireless Mobile (APWiMob)*, Apr. 2021, pp. 128–133.

[10] K. R. Sungkono, E. O. Putri, H. Azkiyah, and R. Sarno, "Checking wrong decision and wrong pattern by using a graph-based method," in *Proc. 13th Int. Conf. Inf. Commun. Technol. Syst. (ICTS)*, Oct. 2021, pp. 184–189.

[11] A. Berti and W. M. Van Der Aalst, "Reviving token-based replay: Increasing speed while improving diagnostics," in *Proc. CEUR Workshop*, vol. 2371, 2019, pp. 87–103.

[12] A. Burattin, M. Cimitile, F. M. Maggi, and A. Sperduti, "Online discovery of declarative process models from event streams," *IEEE Trans. Services Comput.*, vol. 8, no. 6, pp. 833–846, Nov. 2015.

[13] S. J. Van Zelst, B. F. Van Dongen, and W. M. P. Van Der Aalst, "Event stream-based process discovery using abstract representations," *Knowl. Inf. Syst.*, vol. 54, no. 2, pp. 407–435, Feb. 2018.

[14] V. Leno, A. Armas-Cervantes, M. Dumas, M. La Rosa, and F. M. Maggi, "Discovering process maps from event streams," in *Proc. Int. Conf. Softw. Syst. Process*, May 2018, pp. 86–95.

[15] A. J. Burattin dan Carmona, "A framework for online conformance checking," in *Proc. Int. Conf. Bus. Process Manag.* (Lecture Notes in Business Information Processing), vol. 308. Cham, Switzerland: Springer, 2018, pp. 165–177.

[16] A. Burattin, S. J. Van Zelst, A. Armas-Cervantes, B. F. Van Dongen, and J. D. Carmona, "Online conformance checking using behavioural patterns," in *Proc. Int. Conf. Bus. Process Manag.* (Lecture Notes in Computer Science), vol. 11080. Cham, Switzerland: Springer, 2018, pp. 250–267.

[17] R. Sarno, I. Teknologi Sepuluh Nopember, K. Sungkono, R. Johanes, D. Sunaryono, I. Teknologi Sepuluh Nopember, I. Teknologi Sepuluh Nopember, and I. Teknologi Sepuluh Nopember, "Graph-based algorithms for discovering a process model containing invisible tasks," *Int. J. Intell. Eng. Syst.*, vol. 12, no. 2, pp. 85–94, Apr. 2019.

[18] J. Carmona, B. Van Dongen, A. Solti, and D. M. Weidlich, *Conformance Checking*. Cham, Switzerland: Springer, 2018.

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

[20] R. Sarno and K. R. Sungkono, "A survey of graph-based algorithms for discovering business processes," *Int. J. Adv. Intell. Informat.*, vol. 5, no. 2, p. 137, Jul. 2019.

[21] I. Waspada, R. Sarno, and D. K. R. Sungkono, "An improved method of parallel model detection for graph-based process model discovery," *Int. J. Intell. Eng. Syst.*, vol. 13, no. 2, pp. 127–139, Apr. 2020.

INDRA WASPADA is currently pursuing the Ph.D. degree in computer science with the Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia. He is also a Lecturer and a Researcher with the Department of Computer Science, Universitas Diponegoro, Semarang. His current research interest includes process mining. He is also interested in data mining and business process management.

RIYANARTO SARNO (Senior Member, IEEE) received the Ph.D. degree, in 1992. He is currently a Professor with the Informatics Department, Institut Teknologi Sepuluh Nopember (ITS). He was the author of more than five books and over 300 scientific articles led him incorporated in the top 2% world ranking scientist by Standford University, in 2020. He has researched process mining for a period of five years. His research interests include machine learning, the Internet of Things, knowledge engineering, enterprise computing, and information management.

ENDANG SITI ASTUTI is currently a Full Professor with Brawijaya University. She is also a member of the PPIKID Team and a UB Lecturer Certification Assessor. Her research interests include business administration, information system management, perceptions of digital technology, leadership, entrepreneurship, e-commerce, and knowledge management.

HANUNG NINDITO PRASETYO received the B.Sc. degree in mathematics study program from Universitas Pendidikan Indonesia (UPI), in 2003, and the M.E. degree in informatics from the Institut Teknologi Bandung, in 2013. He is currently pursuing the Ph.D. degree in computer science with the Institut Teknologi Sepuluh Nopember, Surabaya. He is also a Faculty Member with Telkom University. His research interests include databases, business processes, process mining, and development of applications and information systems.

RADEN BUDIRAHARJO is currently pursuing the Ph.D. degree with the Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia. He is also a Faculty Member with the Department of Information Systems, Institut Teknologi Nasional (Itenas), Bandung, Indonesia. His current research interests include process mining, data mining, machine learning, IT governance, and other topics related to information systems and computer science fields of study.