IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Graph-Based Token Replay for Online Conformance Checking

Indra Waspada[1,3], Riyanarto Sarno[1] (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, Indonesia
[2]Department of Business Administration, Brawijaya University, Malang, Indonesia
[3]Department of Informatics, Faculty of Science and Mathematics, Diponegoro University, Semarang, Indonesia
[4]Department of Information System Diploma, School of Applied Science, Telkom University, Bandung, Indonesia
[5]Department of Information System, Faculty of Industrial Technology, Institut Teknologi Nasional, Bandung, Indonesia,

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

This work was supported in part by the Directorate of Research and Community Service; in part by the Directorate General of Higher Education, Research, and Technology; in part by the Indonesian Ministry of Education, Culture, Research, and Technology through the Penelitian Terapan Unggulan Perguruan Tinggi (PTUPT) Program; in part by the Institut Teknologi Sepuluh Nopember (ITS) through the Project Scheme of the Publication Writing and IPR Incentive Program (PPHKI); and in part by the Indonesian Minitry of Education, Culture, Research, and Technology through the Matching Fund (MF) Kedaireka Program. The work of Indra Waspada was supported by Diponegoro University Scholarship Program.

ABSTRACT Conformance checking is able to detect deviations in business process execution. An online detection capability is required to anticipate and respond immediately to possible impacts. The state-of-the-art online conformance checking is a prefix-alignment (PA) technique. However, this technique has an important weakness of maintaining all the state data of running processes in memory. In an online environment, the last event of a case is unknown, whereas a PA requires this information to free up memory space for other cases. Consequently, the PA does not meet the requirements of online conformance checking to process infinite data (event stream) without memory constraints. PA also has a conplex state space search computation especially for large and complex reference process model. In this paper, a Graph-Based Online Token Replay (GO-TR) method is proposed. This method takes benefit of 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 developed a cypher based algorithm to simulate token replay on the RI to handle the event stream. Finally, we propose a cypher-based algorithm to identify and replay invisible path. The experiment result show that GO-TR has succeeded in adapting TBR and solved the problem of wrong-place tokens in TBR. GO-TR outperform the replay throughput for relatively low amount of data against state of the art online conformance checking. In terms of memory usage, GO-TR shows its superiority over state of the art because it is safe against memory limitations problems.

INDEX TERMS Conformance Checking, Event Stream, Graph Database, Token-Based Replay, Memory limitation

I. INTRODUCTION

Conformance checking is able to detect deviations in business process execution. There are two important replay techniques, namely Token-Based Replay (TBR) and Alignment, which work in an offline environment. TBR was first introduced by Rozinat et al. [1] as a replay technique that is based on Petri net. While alignment [2] currently is the de facto standard because of its ability to provide optimal alignment information.

In real life, there are many conditions that require immediate inspection. Therefore, an online conformance checking is required. With this online detection capability, actions to anticipate possible impacts can be taken as soon as poosible.

Prefix-alignment [3]–[5] is a replay-based state-of-the-art online conformance checking, which is a modification of the conventional alignment. However, this technique has an important weakness of maintaining all the case administrations of running processes in memory. In an online environment, the last event of a case is unknown, whereas a prefix alignment requires the information to release it from memory. Consequently, the prefix alignment does not meet the requirements and the needs of online conformance checking, namely: (1) the ability to process infinite data (event stream) without memory constraints, (2) the ability to respond and maintain all cases (of the event stream) concurrently and (3) the robustness against unknown future behavior. Several studies have tried to overcome these weaknesses by providing an approach to predict the end of the case, but the solution cannot be used in general and has another impact that needs to be resolved [3], [6].

In this study, a Graph-Based Online Token Replay (GO-TR) method is proposed, which adapts TBR to an online environment through the support of a graph database. This proposal is expected to solve problems in memory usage. The TBR technique was chosen because of its characteristics, namely: (1) simple replay computation (compared to the alignment) and (2) only need to maintain the current marking (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 applying a graph-based process discovery model technique. 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 the online conformance checking progress as a Replay Image in the database. In addition, the graph database has a constant node traversal that suitable to support the proposed invisible path identification and replay especially for complex model which produces 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.

Several contributions to build GO-TR are made :

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

The experimental results show that GO-TR has succeeded in adapting TBR to the graph database and at the same time providing improvements to overcome the problem of wrong-token place on the TBR. We also found that, for relatively small amount of data, GO-TR works with the highest 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 superior over state of the art because it is safe against memory limitations.

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

II. RELATED WORK

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

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 expressly describes a series of activities resulting from 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, et al. [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] advances 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 et al. [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 from the list the-invisible-path candidate with the shortest route and then running an invisible replay. This solution makes ITBR faster than alignment. However, there is a weakness in ITBR because this algorithm requires an invisible replay which if it fails it leaves a wrong-place-token problem

Our research is inspired by TBR algorithm and modify it to work on the graph database. A cypher-based algorithm is also built to identify invisible path accurately and to replay the invisible path efficiently.

B. Online Conformance Checking

The increased attention to online process discovery research [12]–[14] was followed by the growth of online conformance checking research. 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 handle processes that are already running without knowing previous information. However, the abstraction approach in the form of behavioral patterns is less expressive in explaining the occurrence of deviations. It also has weakness to recognize new deviation pattern. Zelst et al. [3] proposed prefix-alignments that modify and develop conventional alignments to work with incomplete cases so that they can respond to every event stream that comes online. Schuster et al. [4] improved the computation of prefix alignment using state space expansion so that it can be done in increments without repeating the previous computations.

Current proposed online conformance checking techniques generally assume infinite memory. 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 stored is still not limited. In reality, memory has limitations that affect the ability to accommodate incoming data streams. [16] gave the 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 occurs if the case is still active so that the remaining cases that come later 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 general (because some business process domains may have very long active cases up to several months). In addition, the space to accommodate forgotten cases for recovery purposes also has the potential to require infinite memory.

Graph-based Online Token Replay (GO-TR) is proposed as a graph-based online conformance checking method. The method utilizes a graph database to store the case administrations as a replay image (RI) to avoid the problem of memory constraint in responding to unlimited event streams.

C. Graph-Based Process Mining

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

In line with Graph-based process discovery [7], the graph database is used to propose online conformance checking that avoids memory constraints and is robust against unknown future behavior.

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 the process. Event logs are generally stored using the XES (eXtensible Event Stream) standard. 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 example in Table 1 is presented in Table 2.

Table 1. An 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

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3208098

...	...	...	...
152	Pay compensation (f)	Andy	31-12-2020:16.01
151	Pay compensation (f)	Jack	31-12-2020:12.00
...	...	...	...

Table 2. A 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 an online environment, the data analyzed is not in the form of an event log but in the 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 an 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

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

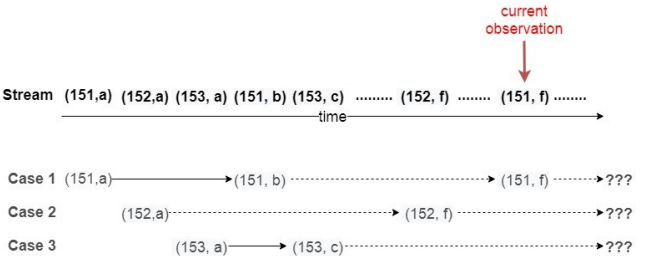


Figure 1. Example of Event Stream

Event stream processing systems **do not have knowledge about future events**. For example, Figure 1, at the point marked as "current observation", it appears that case 1 and case 2 have reached f, which is assumed to be the end of the case, but the event stream processing systems will not know it. There is always a possibility of new activities coming, including those that should not be possible, for example, due to deviations/ anomalies. While case 3 has not been completed, and it is not known when the next event will come, it is also not known when 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 was found and needed 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.

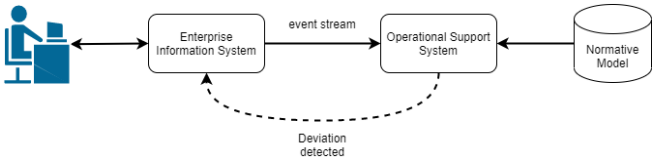


Figure 2. Online Conformance Checking for Deviation Detection

One of the important activities in operational support is deviation detection in the 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. Figure 2 illustrates an online conformance checking system for detecting deviations.

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

Therefore, with reference to [1] regarding assumptions about data streams and [18] regarding the unique characteristics of online conformance checking, the differences in requirements between offline and online conformance checking are summarized in Table 3.

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

The process model describes how a process should be executed. In this paper, the process model is represented in a Petri net.

Definition 1 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,* $\alpha : T \to A$ *is the transition mapping function to labels.*

A marking, ie 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 2 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* $m' = (m \backslash \bullet t) \cup t\bullet$.

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

Definition 3 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 the proposed method. As for the theory of alignment can be traced in [2], [18].

The TBR discussed here is a method that works in an offline environment. Basically TBR works based on trace logs and accepting Petri net. 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 3 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 the place which is the initial marking. The replay operation considers, sequentially, the activity on the trace. At each step, the set enabled transition in current marking is fetched. If there is a transition corresponding to the current activity, the transition can be activated, a number of tokens equal to the number of input places added to c, and a number of tokens equal to the number of output places 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 transitions cannot be activated in current marking, 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, 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, the missing tokens will be inserted and the last one is calculating 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

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3208098

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 display can be seen in Figure 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.

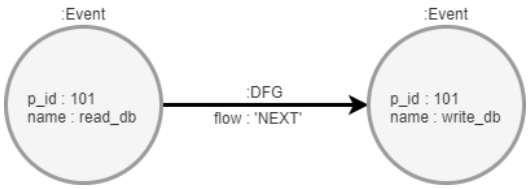


Figure 3. Simple Graph

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, it is necessary to exploit, so that replay in GO-TR involves lightweight write operations and query operations that are easy to find and trace.

Petri Net model representation in the graph database is the foundation for GO-TR. Petri net consists of places and transitions. Figure 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, namely Transition and Place. While an example of the realization of the schema model can be seen in Figure 5. There is an initial marking, namely a node of type Place with the name of the source, and the final marking with the name sink. There are two nodes of type Transition with labels A and B; between A and B, there is a Place with the name p_3. All relationships are identical, pointing in one direction and of type Arc. Based on the Petri Net model, a cypher algorithm can then be developed to run replay.

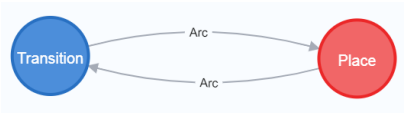


Figure 4 Schema model for Petri Net in Neo4j

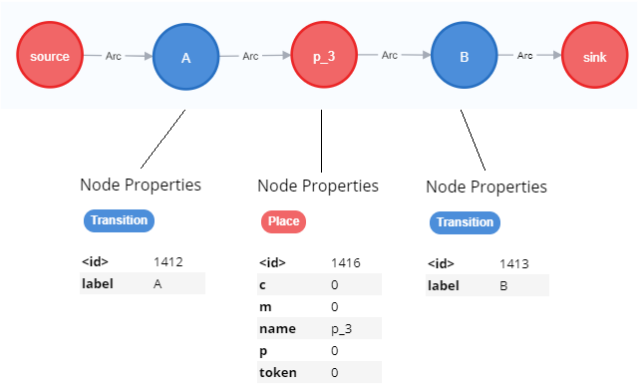


Figure 5. Example of Model Instantiation of Petri Net in Neo4j with Additional Attributes (c, p, m) in Place for GO-TR.

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

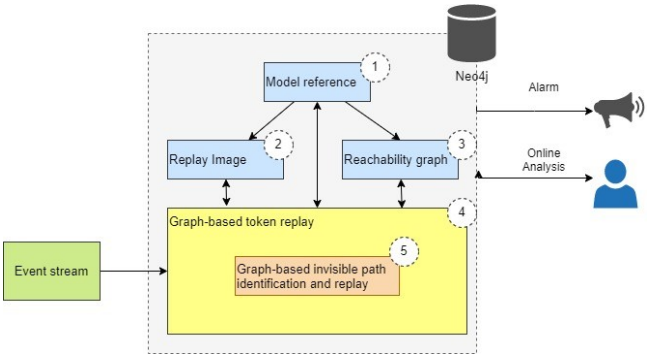


Figure 6. Architecture of GO-TR

A. Model reference

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

1. If an event log is available in a (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]–[22]. The results obtained are still in the directly followed graph (DFG)

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3208098

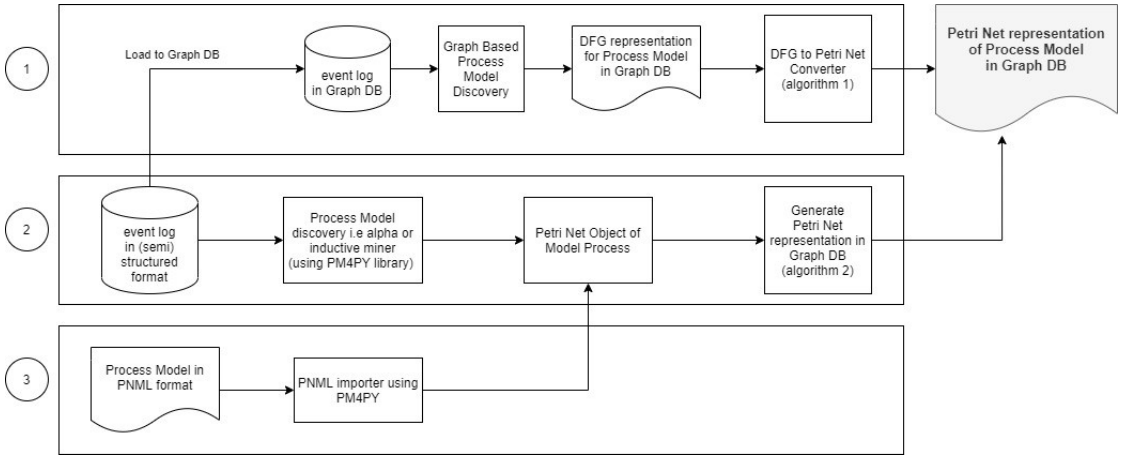


Figure 7 Techniques for Obtaining the Petri Net Representation of Process Model in a Graph Database from several Data Source Formats

representation, so the next step is to convert to Petri net using algorithm 1.

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 carried out directly from the event log in a (semi) structured format such as XES or CSV. In this study, the process discovery uses the PM4Py library, for example, 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 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 it into the graph database and get a Petri net object. Based on the petrinet object, then a representation of the process model is generated in the graph database using algorithm 2.

ALGORITHM 2: 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)
```

B. *Replay Image*

A Replay Image (RI) is proposed to store case administration.

Definition 4 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.*

ALGORITHM 3: CREATE REPLAY IMAGE

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

	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.cloneSubgraphFromPaths(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*

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3208098

The replay behavior in RI follows Definition 2 and stores the replay data, including token consumed (c), token produced (p), and token missing (m), according to Definition 4. As an illustration, the comparison between the reference model and the Replay Image is presented in Figure 8.

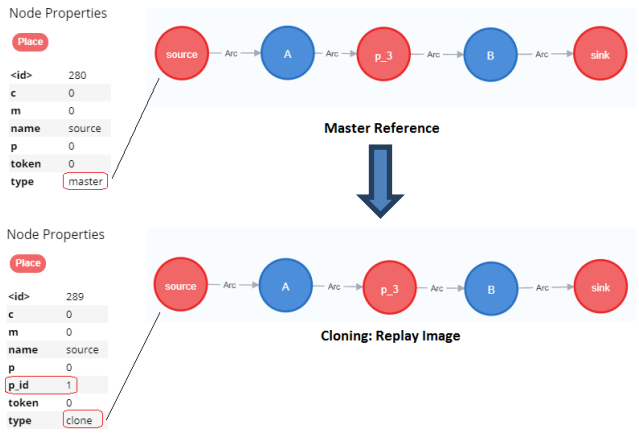


Figure 8. A Master Reference and Its Replay Image

C. *Reachability Graph*

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

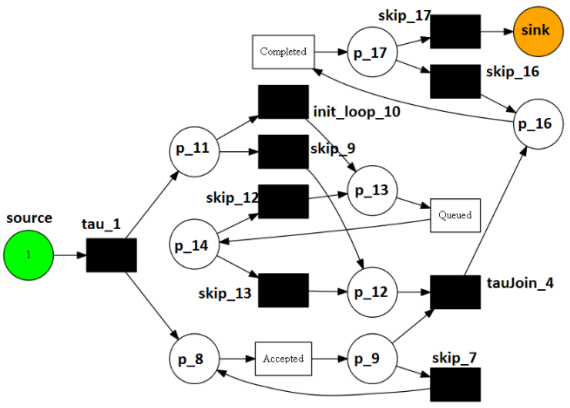


Figure 9. Example of a process model generated from the BPIC13 data set

Based on the resulting reachability graph object, it is then loaded into the graph database using algorithm 4. First, all states which are nodes in neo4j are created (lines 1-3). Then connect between states with transitions in the form of relationships on neo4j (lines 4-12). Giving the name of the invisible label to the relationship will greatly help the invisible path identification algorithm.

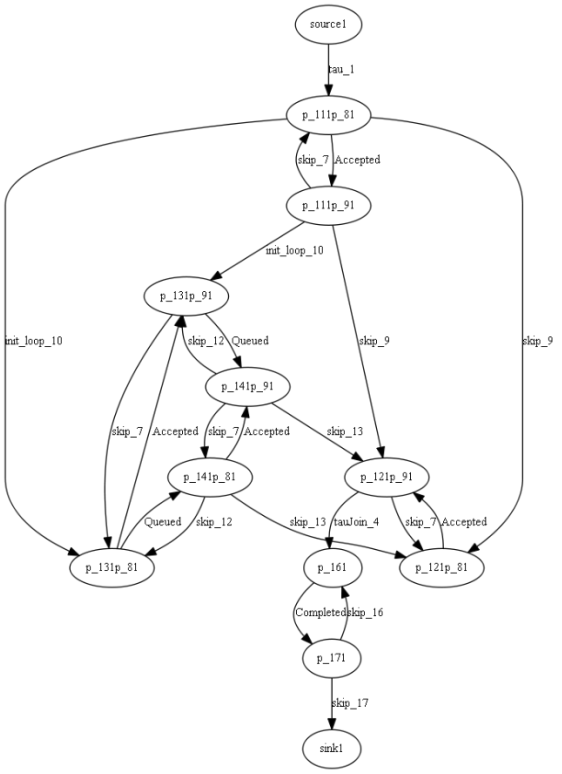


Figure 10. An example of the reachability graph obtained from the process model in Figure 21

ALGORITHM 4: GENERATE REACHABILITY GRAPH IN NEO4J

```
    Input: rg , net        # reachability graph object
    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: GRAPH-BASED ONLINE TOKEN REPLAY

```
    Input: N = (P, T, F, α, m_i, m_f ), RG(N), S ∈ (CxA)
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)
```

D. Graph-Based Token Replay for Online Conformance Checking

The core 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 5 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 Figure 1. Basically, GO-TR accepts input data in the 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 duplicate process model from the reference master will be duplicated as a Replay Image (RI). An RI is a state representation of the replay progress shown through the position of the last token (current marking), as well as data related to tokens during the replay process, including produce (p), consume (c), missing (m), and remain (r). Each RI is stored persistently in the graph database so that it can be accessed (by query cipher) at any time.

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 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, detect whether there are empty input places in the activity to be replayed as a condition for enabling the activity (line 12).

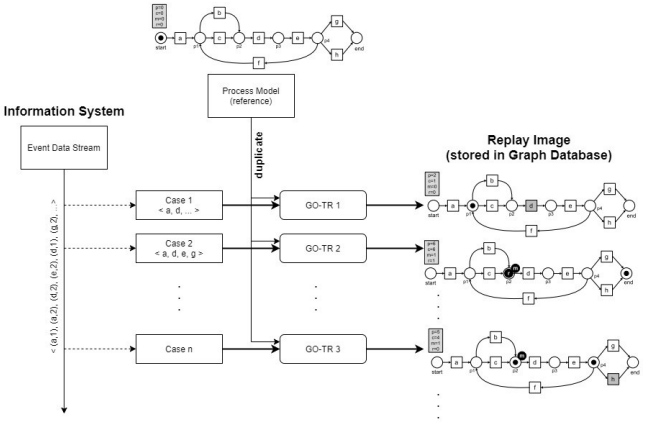


Figure 11. Proposed Online Token-Based Replay on Graph Database

If there are empty input places (line 13), and if all the empty input places are 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 done normally (line 16), but if the invisible path fails to get, then the replay must be done by inserting the missing token (line 18). The insertion of a missing token is a sign that there is a deviation so that a warning (in real-time) is needed, which is equipped with case id information and activity name (line 19).

If there is an empty input place (line 13), but there is an empty input place that is not connected to the invisible task, the replay can only be done by inserting a missing token (line 21) so that this replay is also declared a deviation.

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

The explanation of the cipher 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	*Cipher*
1	*Find all input places of matched activity*	*MATCH (ip: Place {p_id: $c })-[r]->(t:Transition {label:$a})*
2	*Consume tokens from each input place and update the relationship attributes. All input places are set as a non-final marking*	*SET ip.token = ip.token - 1 , ip.c = ip.c + 1, ip. fm = 0, 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. All output places are set as final marking places*	*SET op.token = op.token + 1, op.p = op.p + 1 , op.fm = 1, r.p = r.p +1, r.f = r.f + 1*

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*

E. Graph-Based Invisible Path Replay

The core algorithm of Token-Based Replay cannot replay the invisible task, so it will bring up missing tokens. Our proposed method takes advantage of the graph database's ability to store graph data natively and its fast node traversal capabilities. The Reachability graph (RG) is generated from the reference process model. This RG is stored in the graph database.

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 if in algorithm 5 (lines 12-14) there are input places that do not contain tokens, and each empty input place is connected to an invisible task. To detect an invisible path, it is necessary to first find the state position in the reachability graph as *source_state* (line 1). If *source_state* exists (line 4), then need to check the existence of candidate target_state (lines 5). From all the *target_state_candidates*, look for the one with the shortest distance from source_state (line 6).

If a reachable and a shortest distance target_state is found, it means that the invisible path has been found (line

7-8). To simulate replay on an invisible path, the invisibleReplay function (algorithm 9) will update the attributes of all nodes and edges along the invisible path found (line 9). Returns is True if the invisible path is found (line 11). On the other hand, if the spf_target_state is not obtained, the invisible path will not be obtained, so the return value given is False (lines 11-12).

If current marking is not one of the states in the reachability graph (line 13), then invisible path detection requires token replay to simulate the invisible task candidate can be executed correctly. It is first necessary to store the current state, including information about current marking, current edge frequency, and current edge produced (line 2). Then the invisible token replay simulation is run (line 14). If the token succeeds in reaching the target, the return value is True (line 15), while if the target fails to reach the state, it needs to roll back to the initial state, which is the current_state, and return False (lines 17-18).

The invisible replay algorithm performs replay without requiring a replay token simulation but instead updates the node and edge attribute values along the invisible path found. The cipher algorithm to perform invisible replay is presented in Algorithm 9. Line 1 prepares all invisible_task that will be replayed. Line 2 performs

updates related to token-consuming activities and updates the values of c, f, and fm. Lines 3-4 perform updates related to token-producing activities and update the values of p, f, and fm.

ALGORITHM 9: INVISIBLE REPLAY

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

	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*

F. Experiment Set Up

The experiment was carried out on a computer with an Intel Core i7-3632QM 16GB RAM processor. Python 3.6 and the pm4py 2.24 library is used to perform process discovery and generate reachability graphs. The following are the experimental scenarios carried out:

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),        // temporary storage for current states
        current_edge_produced(c)
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
```

1. The correctness of the Token-based Replay adaptation

GO-TR adapts TBR. To prove that GO-TR is working properly, conformance checking experiments are carried out with scenarios using normal cases, cases containing deviations. and when cases with models containing multi-input invisible tasks.

The experimental dataset uses public reallife data from BPIC 2013 Incident. To obtain the process model, an inductive miner is used with a noise threshold of 0.2. The result obtained is a process model that contains many invisible tasks and an invisible task with multiple input invisible tasks. The case used for testing is a modification of one of the cases in the 2013 Incident BPIC data.

Token-based Replay testing uses the ITBR program provided by PM4PY. This program only works for offline conformance checking, so it requires input in the form of an event log..

2. Throughput

This experiment was conducted to compare the average speed in replaying each event arrival (stream). The events data are sent to the conformance checking machine on a streaming basis based on the order of arrival (not following the arrival timestamp). The data set used is CCC19 which is a public data set. Comparison of performance is done by duplicating the number of case id as much as 20, 100, and 200. To get the number of 100 and 200, it is done by replicating the 20 available case variants.

The variable to be observed is the duration required to complete the inspection of the event stream data. Based on the known duration and number of events, the throughput value (T) and replay speed per event (v) can be obtained using the formula T=n/d , v=d/n , where n = number of completed events, d = duration of completion, T= throughput, and v=average replay speed per event.

3. Memory consumption

This experiment aims to compare memory usage between prefix alignment and GO-TR. The dataset used is CCC19 public data. The variable observed in this experiment is 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 have been carried out.

A. The correctness of the Token-based Replay adaptation

In this section, observations are made on several test scenarios to ensure the correctness of the GO-TR replay results by comparing them with TBR. The dataset used is BPIC13 incident management. Figure 31 presents the process model of the BPIC13 Incident. The model was generated with the help of the pm4py library using Inductive Miner with a noise threshold of 0.2.

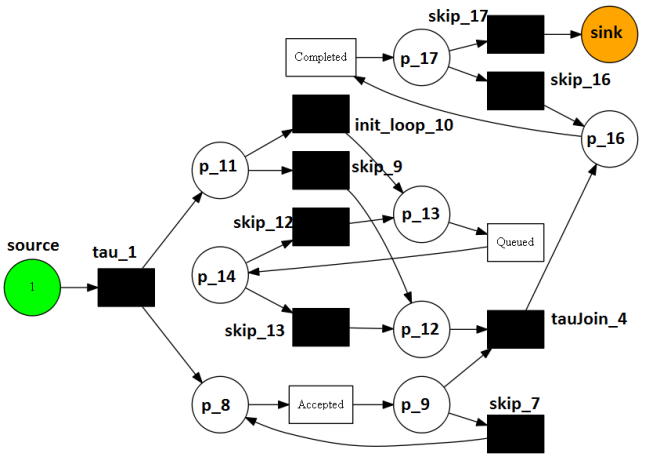


Figure 12. The results of the discovery of the process model with the BPIC13 dataset

There are AND branches (eg AND-Split in tau_1) and loops (eg Accepted). There is also an invisible task tauJoin_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.

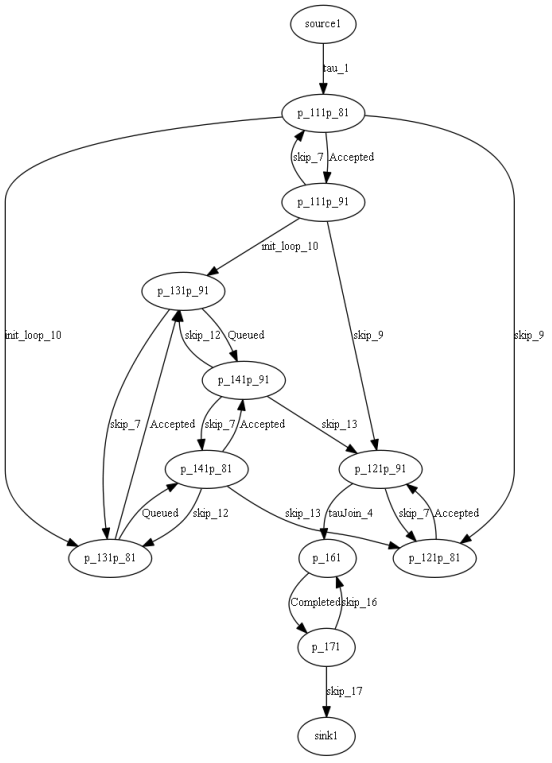


Figure 13. Reachability graph generated from the model in Figure 12

The first experiment is to use a normal case example containing Queued, Accepted, Completed activities. This test is to prove that the TBR algorithm used in GO-TR can recognize invisible paths. The experimental results are

presented in Table 18. The next experiment uses a case containing a loop, namely Queued, Accepted, Queued, Completed, Completed. The experimental results are presented in Table 19.

The results in Table 18 and Table 19 show that GO-TR and ITBR can work well in all normal cases and give the same results for all statistics.

The third experiment using case examples contains the following activities: Queued, Completed, Completed. A good TBR algorithm will know that it is necessary to add a missing token to p_16. This is because by marking on p_8 and p_14, enabling taujoin_4 requires a replay of Accepted. Interestingly here, the algorithm that works on the invisible path between GO-TR and ITBR is different. The results of marking with this experimental case for GO-TR and ITBR are presented in Table 20.

Table 4. The results of the experiment with the input case Queued, Accepted, Completed

Dataset BPIC13	ITBR	GO-TR
Trace	Queued, Accepted, Completed	
Consumed token	14	10
Produced token	14	10
Missing token	0	0
Remaining token	0	0
Fitness	1.00	1.00
Wrong-place token problem	-	-

Table 5. The results of the experiment with the input case Queued, Accepted, Queued, Completed, Completed

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	-	-

The marking movement presented in Table 20 can be explained as follows. First of all the system provides initial marking on a place named Source. Then with the arrival of the first event, Queued, the TBR algorithm starts to work. With the reference model in Figure 26, it can be seen that the initial marking position on Source causes all input places in Queued not to have tokens so that the TBR algorithm will try to find the shortest invisible path possible from source to p_13. In this case, an invisible path is found, namely source (p_8, p_11) (p_8, p13) so that Queued can be replayed normally to produce state (p_8, p_14).

Table 6. Marking Result Comparison

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

The second event is Completed. To enable the Completed activity, a token at position p_16 is required.

a) The ITBR algorithm finds there is an invisible path that is p_14 p_12 p_16 then tries to run a replay. But when it comes to p_12 the replay stops because p_9 has no token and is not connected to the invisible task so that tauJoin_4 cannot be activated. That way the replay attempt via invisible path failed to reach p_16. However, ITBR is already running tokens from p_14 to p_12. So the marking becomes (p_12, p_8). The final position of the token at p_12 is the wrong-place token. The token position at p_12 will have an impact on analysis errors because apart from being able to be achieved through Queued activation, it can also be directly reached via skip_9.

b) Meanwhile, GO-TBR, 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). GO-TR is safe from the wrong-place token problem.

Table 7. The duration comparison experiment completed a number of event stream arrivals from the CCC19 data set

Teknik Online conformance checking	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**

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

The last event is Completed again. This time TBR found route p_17 p_16 as invisible path. Meanwhile, GO-TR because the previous activity was a deviation, no states (p_14, p_8, p_17) were found in the reachability graph, so we had to run algorithm 2, namely through a simulation of invisible path tracing and rolling back all states to the initial condition if the search failed to reach the target (missing token place point). Both ITBR and GO-TR found the invisible path exactly to reach p_16. After successful replay, the marking position will return to (p_14, p_8, p_17) for ITBR and (p_14, p_8, p_17) for GO-TR. Table 21 which shows the statistics of replay results.

B. Throughput

This section describes the experimental results to compare the execution performance required by prefix-alignment (Schuster & van Zelst, 2020) and GO-TR to complete a number of event data streams. Comparisons are made in a single processing environment to observe throughput, namely the number of activities that can be completed per second, the average replay speed per event

Table 8. Comparison of throughput between prefix alignment and GO-TR

Teknik Online conformance checking	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

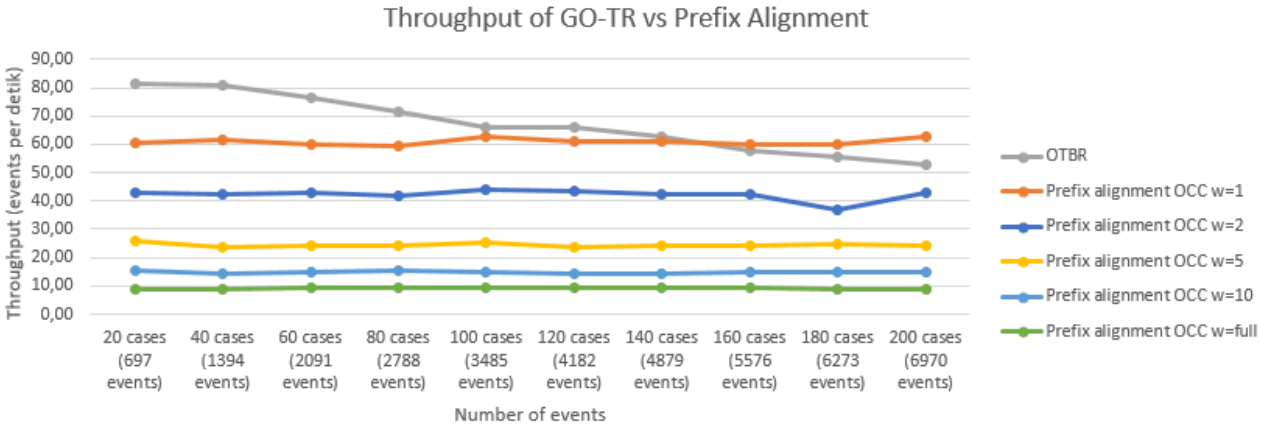


Figure 14. Throughput comparison between state of the art OCC and GO-TR

handled, and the effect of increasing the number of cases handled on the replay execution speed.

This experiment uses the public dataset CCC19 (Munoz-Gama et al., 2019) which provides log data along with its reference model. The log data is in event-log format (XES) while the reference model is in pnml format. Experiments using prefix-alignment refer to the source code of the IAS prefix-alignment program. Meanwhile, to simulate GO-TR, CCC19 log data requires conversion of event log to event stream representation by sorting each event based on arrival time.

Next, to load the reference model into the graph database, algorithm 2. Reachability graph (RG) is also needed by GO-TR to identify invisible paths. This RG model is generated from the Petri Net object with the help of the pm4py library. The resulting reachability graph model is loaded into the graph database using algorithm 4.

The experimental results are presented in Table 22. The OCC alignment prefix with w=full requires the highest computation because it performs a complete optimal alignment computation from the beginning of each new event arrival. The number of 100 and 200 cases requires execution time of more than 2 hours so that the table is marked with '-'.

Prefix alignment with a small window for example w=2 is very fast but has an impact on reducing the guarantee of getting optimal alignment, besides that, what is more important is that it still has bounded memory problems. Meanwhile, IAS with heuristics and without recalculation outperformed OCC without window (w=full) even though it was still inferior to OCC with window. IASR guarantees optimal alignment while IAS does not guarantee optimal alignment. GO-TR has a result close to OCC w=2, and is much faster than IAS.

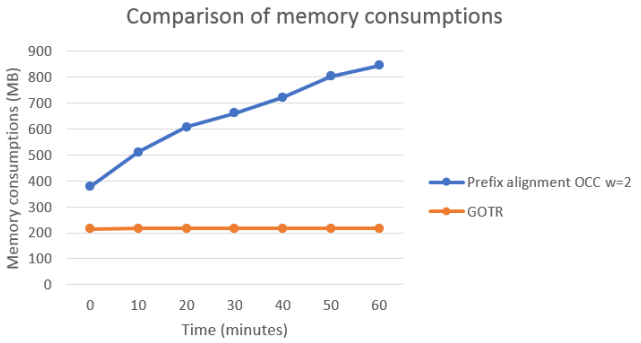
Based on the results of this experiment, it appears that GO-TR with a low number of cases shows the highest throughput. This is related to simple computing so that it can execute replays quickly. While PA computing is very influential on replay speed, the faster the replay computation, the greater the throughput. At w=1 PA throughput is close to GO-TR.

The data in Table 23 shows that GO-TR experienced a decrease in throughput as the number of cases handled increased. This event is because in GO-TR each case has its own representation of the replay image in the graph database. The more cases that are handled, it will affect the query time required in the replay process.

C. Memory consumption

This experiment is the most important part to prove the reliability of the online conformance checking technique against memory limitations. The test is carried out by observing memory consumption by the program (python) that executes PA and GO-TR. Testing on state of the art online conformance checking using PA OCC w=2. The results using PA OCC w=2 for 60 minutes are presented in Figure 34. These results indicate that memory consumption increases linearly with the number of events that come. This is because the PA needs space 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 that if the arrival of the case is declared to be infinite, it will requires infinite memory space to accommodate it. This makes PA vulnerable to memory limitations.

The results of the GO-TR experiment in Figure 34 indicate that the arrival of the event stream has no effect on memory consumption. This can happen because the administration of all cases are stored in the graph database. The running program simply replays the replay-image of the case based on the event id that comes. The replay results immediately update the replay-image. Based on these results, it is proven that GO-TR is safe from memory limitation problems.



Gambar 1. Konsumsi memori oleh prefix alignment w=2 menggunakan dataset CCC19

VI. CONCLUSION

We propose Graph-based online token replay (GO-TR) as a replay-based online conformance checking that is free of the problem of memory limitations. Our proposed solution adapts the token replay technique on a graph database. To build the GO-TR we made several contributions: proposing replay image as the representation of the Petri Net model in the graph database, adapting Token-based Replay on a graph database that can receive event stream data for online conformance checking, and proposing a cypher-based invisible path identification and replay algorithm.

Based on observations and analysis of the experiments carried out, it is proven that GO-TR has succeeded in adapting TBR and is safe against the wrong-place token problem. For small amounts of data, GO-TR works with the highest throughput compared to prefix-alignment. However,

GO-TR's throughput performance decreases as the amount of data increases. In terms of memory usage, GO-TR shows its advantages over state of the art because it is safe against 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 the response to high-speed data.

REFERENCES

[1] A. Rozinat dan W. Van Der Aalst, "Conformance checking of processes based on monitoring real behavior," no. March, 2008.

[2] A. Adriansyah, *Aligning Observed and Modeled Behavior*, no. 2014. 2014.

[3] S. J. van Zelst, A. Bolt, M. Hassani, B. F. van Dongen, dan 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, hal. 269–284, 2019.

[4] D. Schuster dan S. J. van Zelst, "Online process monitoring using incremental state-space expansion: An exact algorithm," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12168 LNCS, no. i, hal. 147–164, 2020.

[5] D. Schuster dan G. J. Kolhof, "Scalable Online Conformance Checking Using Incremental Prefix-Alignment Computation," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12632 LNCS, hal. 379–394, 2021.

[6] R. Zaman, M. Hassani, dan B. F. Van Dongen, "Prefix Imputation of Orphan Events in Event Stream Processing," *Front. Big Data*, vol. 4, no. October, hal. 1–21, 2021.

[7] 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," *J. Big Data*, vol. 8, no. 1, 2021.

[8] K. R. Sungkono *dkk.*, "Graph-based Process Discovery containing Invisible Non-Prime Task in Procurement of Animal-Based Ingredient of Halal Restaurants," *Proc. - 2021 IEEE Asia Pacific Conf. Wirel. Mobile, APWiMob 2021*, no. 978, hal. 134–140, 2021.

[9] A. S. Ahmadiyah *dkk.*, "Business Processes Discovery in Halal Restaurant Kitchen Using Graph-Based Algorithm," *Proc. - 2021 IEEE Asia Pacific Conf. Wirel. Mobile, APWiMob 2021*, hal. 128–133, 2021.

[10] K. R. Sungkono, E. O. Putri, H. Azkiyah, dan R. Sarno, "Checking Wrong Decision and Wrong Pattern by Using A Graph-based Method," *Proc. 2021 13th Int. Conf. Inf. Commun. Technol. Syst. ICTS 2021*, hal. 184–189, 2021.

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

[12] A. Burattin, M. Cimitile, F. M. Maggi, dan A. Sperduti, "Online Discovery of Declarative Process Models from Event Streams," *IEEE Trans. Serv. Comput.*, vol. 8, no. 6, hal. 833–846, 2015.

[13] S. J. van Zelst, B. F. van Dongen, dan W. M. P. van der Aalst, "Event stream-based process discovery using abstract representations," *Knowl. Inf. Syst.*, vol. 54, no. 2, hal. 407–435, Mei 2017.

[14] V. Leno, A. Armas-Cervantes, M. Dumas, M. La Rosa, dan F. M. Maggi, "Discovering Process Maps from Event Streams," 2018.

[15] A. Burattin dan J. Carmona, "A framework for online conformance checking," *Lect. Notes Bus. Inf. Process.*, vol. 308, hal. 165–177, 2018.

[16] A. Burattin, S. J. van Zelst, A. Armas-Cervantes, B. F. van Dongen, dan J. Carmona, *Online conformance checking using behavioural patterns*, vol. 11080 LNCS. Springer International Publishing, 2018.

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

[18] J. Carmona, B. van Dongen, A. Solti, dan M. Weidlich, *Conformance Checking*. Cham: Springer International Publishing, 2018.

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

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

[21] R. Sarno, H. Darmawan, dan K. R. Sungkono, "Graph-based Algorithms for Discovering Non-free-choice in Invisible Tasks of Business Process," *Int. J. Electr. Comput. Eng. (IJECE). Accept.*, 2019.

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

INDRA WASPADA is currently pursuing the Doctoral degree in Computer Science, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia. He is a Lecturer and a Researcher at Department of Computer Science, Diponegoro University. His current research mainly focuses on process mining. He is also interested in data mining and business process management.

RIYANARTO SARNO (Member, IEEE) received the Doctor (Ph.D.) degree in 1992. He is currently a Professor with the Informatics Department, Institut Teknologi Sepuluh Nopember (ITS). He is interested in research projects about machine learning, the Internet of Things, Knowledge engineering, Enterprise Computing, and Information Management. He has researched Process mining for a period of five years, Being an author of more than five books and over 300 scientific articles led him incorporated in the top 2% world ranking scientist, in 2020 by Standford University.

Endang Siti Astuti holds a Bachelor of Business Administration from Fakultas Ketatanegaraan dan Ketataniagaan (FKK) Brawijaya University (1976), Bachelor of Business Administration (1979), Master (2000) and Doctoral (2006). She is currently a member of the PPIKID TEAM and UB Lecturer Certification Assessor.

HANUNG NINDITO PRASETYO obtained his B.Sc. in Mathematics study program from Universitas Pendidikan Indonesia (UPI) in 2003 and Master of Engineering in Informatics from Institut Teknologi Bandung in 2013 and is currently undergoing a Ph.D. in the field of Computer Science at Institut Teknologi Sepuluh Nopember Surabaya with interest in Process Mining. His research interests include databases, business processes, process mining processes, and the development of applications and information systems.

RADEN BUDIRAHARJO is a Ph.D student at the Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia. He is also a faculty member at the Department of Information Systems, Institut Teknologi Nasional, Bandung, Indonesia. His current researches include topics such as, but not limited to, process mining, data mining, machine learning, and other topics related to Information Systems and Computer Science fields of study