Reviving Token-based Replay: Increasing Speed While Improving Diagnostics

Alessandro Berti[0000−0003−1830−4013] and Wil van der Aalst[0000−0002−0955−6940]

Process and Data Science group, Lehrstuhl für Informatik 9 52074 Aachen, RWTH Aachen University, Germany

Abstract. Token-based replay used to be the standard way to conduct conformance checking. With the uptake of more advanced techniques (e.g., alignment based), token-based replay got abandoned. However, despite decomposition approaches and heuristics to speed-up computation, the more advanced conformance checking techniques have limited scalability, especially when traces get longer and process models more complex. This paper presents an improved token-based replay approach that is much faster and scalable. Moreover, the approach provides more accurate diagnostics that avoid known problems (e.g., "token flooding") and help to pinpoint compliance problems. The novel token-based replay technique has been implemented in the PM4Py process mining library. We will show that the replay technique outperforms state-of-the-art techniques in terms of speed and/or diagnostics.

Keywords: Log-Model Replay · Process Diagnostics · Localized Conformance Checking

1 Introduction

The importance of conformance checking is growing as is illustrated by the new book on conformance checking [8] and the Gartner report which states "we see a significant trend toward more focus on conformance and enhancement process mining types" [9]. Conformance checking aims to compare an event log and a process model in order to discover deviations and obtain diagnostics information [15]. Deviations are related to process executions not following the process model (for example, the execution of some activities may be missing, or the activities are not happening in the correct order), and are usually associated with higher throughput times and lower quality levels. Hence, it is important to detect them, understand their causes and re-engineer the process in order to avoid such deviations. A prerequisite for both conformance checking and performance analysis is a replay technique, that relates and compares the behavior observed in the log with the behavior observed in the model. Different replay techniques have been proposed, like *token-based replay* [17] and *alignments* [8, 6]. In recent years, alignments have become the standard-de-facto technique since they are able to find an optimal match between the process model and a process execution contained in the event log. Unfortunately, their performance on complex process models and large event logs is poor.

Token-based replay used to be the default technique, but has been almost abandoned in recent years, because the handling of invisible transitions, that are contained in the output models of algorithms like the heuristics miner or the inductive miner, is based on heuristics and the technique suffer of several know drawbacks. For example, models may get flooded with tokens in highly non-conforming executions, enabling unwanted parts of the process model and hampering the overall fitness evaluation. Moreover, detailed diagnostics have been introduced only for alignments.

In this paper, a revival of token-based replay is proposed by addressing some of the weaknesses of traditional token-based replay techniques. The new approach is supported by the PM4Py process mining library[1].

The remainder of the paper is organized as follows: in Section 2 an introduction to token-based replay and alignments is provided. Section 3 presents the novel approach which modifies the original technique and uses a different implementation strategy. Section 4 proposes different ways to localize conformance checking both prior (simplifying the model, reducing the complexity and the time required to do token-based replay) and after the replay (evaluating which elements of the Petri net are used and/or have encountered problems during the replay operation). In Section 5, additional diagnostics are introduced based on the localized replay output. Section 6 concludes the paper.

2 Background and Related Work

Petri nets are the most widely used process model in process mining frameworks: popular discovery algorithms like the alpha miner and the inductive miner (through conversion of the resulting process tree) can produce Petri nets. An accepting Petri net is a Petri net along with a final marking.

Definition 1 (Accepting Petri nets). *A (labeled, marked) accepting Petri net is a net of the form* $PN = (P, T, F, W, M_0, M_F, l)$*, which extends the elementary net so that:*

- (P, T, F) *is a net (*P *and* T *are disjoint finite sets of places and transitions;* $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs).*
- $W : F \to \mathbb{N}$ *is an arc multiset, so that the count (or weight) for each arc is a measure of the arc multiplicity.*
- $M_0 : P \to \mathbb{N}$ *is the initial marking*[2]*.*
- $M_F : P \to \mathbb{N}$ *is the final marking.*
- $l : T \to \sum \cup \{\tau\}$ *is a labeling function that assigns to each transition* $t \in T$ *either a symbol from* $\sum$ *(the set of labels) or the empty string* τ*.*

The *preset* of a place, $\bullet p$, is the set of all transitions $t \in T$ such that $(t, p) \in F$. The *postset* of a place, $p\bullet$, is the set of all transitions $t \in T$ such that $(p, t) \in F$. The preset and postset of a transition could be defined in a similar way. A

[1] The official website of the library is http://www.pm4py.org

[2] A marking $M : P \to \mathbb{N}$ is a place multiset.

transition t is said to be *visible* if $l(t) \in \sum$; is said to be *hidden* if $l(t) = \tau$. If for all $t \in T$ such that $l(t) \neq \tau$, $|\{t' \in T | l(t') = l(t)\}| = 1$, then the Petri net contains *unique visible* transitions; otherwise, it contains *duplicate* transitions. The initial marking is corresponding the initial state of a process execution. Process discovery algorithms may associate also a final marking to the Petri net, that is the state in which the process execution should end. The execution semantics of a Petri net is the following:

- A transition $t \in T$ is *enabled* (it may *fire*) in M if there are enough tokens in its input places for the consumptions to be possible, i.e. iff $\forall s \in \bullet t : M(s) \geq W(s,t)$.
- Firing a transition $t \in T$ in marking M consumes $W(s,t)$ tokens from each of its input places s, and produces $W(t,s)$ tokens in each of its output places s.

For a process supported by an information system, an event log is a set of cases, each one corresponding to a different execution of the process. A case contains the list of events that are executed (in the information system) in order to complete the case. To each case and event, some attributes can be assigned (e.g. the activity and the timestamp at the event level). A classification of the event is a string describing the event (e.g. the activity is a classification of the event). For each case, given a classification function, the corresponding trace is the list of classifications associated with the events of the case.

The application of token-based replay is done on a trace of an event log and an accepting Petri net. The output of the replay operation is a list of transitions enabled during the replay, along with some numbers: c is the number of consumed tokens (during the replay), p is the number of produced tokens, m is the number of missing tokens, r is the number of remaining tokens. At the start of the replay, it is assumed that the tokens in the initial marking are inserted by the environment, increasing p accordingly (for example, if the initial marking consists of one token in one place, then the replay starts with $p = 1$). The replay operation considers, in order, the activities of the trace. In each step, the set of enabled transitions in the current marking is retrieved. If there is a transition corresponding to the current activity, then it is fired, a number of tokens equal to the sum of the weight of input arcs is added to c, and a number of tokens equal to the sum of the weight of output arcs is added to p. If there is not a transition corresponding to the current activity enabled in the current marking, then a transition in the model corresponding to the activity is searched (if there are duplicate corresponding transitions, then [17] provides an algorithm to choose between them). Since the transition could not fire in the current marking, the marking is modified by inserting the token(s) needed to enable it, and m is increased accordingly. 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, and c is increased accordingly. If the marking reached after the replay of the trace is different from the final marking, then missing tokens are inserted and remaining tokens r are set accordingly.

The following relations hold during the replay: $c \leq p + m$ and $m \leq c$. The relation $p + m = c + r$ holds at the end of the replay. A fitness value could be calculated for the trace as:

$$f_\sigma = \frac{1}{2}\left(1 - \frac{m}{c}\right) + \frac{1}{2}\left(1 - \frac{r}{p}\right)$$

For each case L_i of the event log L, let c_i be the number of consumed tokens, p_i the number of produced tokens, m_i the number of missing tokens and r_i the number of remaining tokens. Then, the following formula calculates the fitness at the log level

$$f_L = \frac{1}{2}\left(1 - \frac{\sum_{L_i \in L} m_i}{\sum_{L_i \in L} c_i}\right) + \frac{1}{2}\left(1 - \frac{\sum_{L_i \in L} r_i}{\sum_{L_i \in L} p_i}\right)$$

This quantity is different from the average of fitness values at trace level. When, during the replay, a transition corresponding to the activity could not be enabled, and invisible transitions are present in the model, a technique is deployed to traverse the state space (see [17]) and possibly reach a marking in which the given transition is enabled. A heuristic (see [17]) that uses the shortest sequence of invisible that enables a visible task is proposed. This heuristic tries to minimize the possibility that the execution of an invisible transition interferes with the future firing of another activity.

A well-known problem for token-based replay is the *token flooding problem* [8]. Indeed, when the case differs much from the model, and a lot of missing tokens are inserted during the replay, it happens that also a lot of tokens remain unused and many transitions are enabled. This leads to misleading diagnostics because unwanted parts of the model may be activated, and so the fitness value for highly problematic executions may be too high. To illustrate the token-flooding problem consider a process model without concurrency (only loops, sequences, and choices) represented as a Petri net. At any stage, there should be at most one token in the Petri net. However, each time there is a deviation, a token may be added resulting in a state which was never reachable from the initial state.

The original token-based replay implementation [17] was only implemented in earlier versions of the ProM framework (ProM4 and ProM5) and proposes localized metrics on places of the Petri net that help to understand which parts of the model are more problematic. To improve performance in the original implementation, a preprocessing step could be used to group cases having the same trace. In this way, the replay of a unique trace is done once by the token-based replay. Alternatively, more ad-hoc token-based replay approaches were used by the heuristic miner and the genetic miner. In the latter approach, the qualities of candidate models are derived. These techniques tend to put multiple dimensions (replay fitness, precision, etc.) into a single fitness measure.

Currently, the standard replay technique on Petri nets is the computation of alignments. There are different approaches on alignments [8, 6]. In the assessment, we are considering the approach described in [6]. Execution speed of

alignments on process models containing a lot of different states may be problematic, although some techniques have been proposed, such as decomposing alignments [2] and recomposing them [10]. Moreover, the approach described in [18] is also helping to handle bigger instances, making the user decide about the granularity of the alignment steps.

3 Improved Token-Based Replay

3.1 Changes to the Approach

The approach proposed in [17] is relatively fast when there are no duplicate or silent transitions. However, in comparison to the alignments, managing invisible transitions may be time-consuming due to the necessary state-space explorations.

The idea proposed in this paper is to perform a pre-processing step in order to store a map of the shortest paths between places, and then use this map when hidden transitions need to be traversed. This saves the time necessary to perform the state-space explorations. Therefore, the proposed approach works with accepting Petri nets that have no invisible transitions with empty preset or postset, since they would not belong to any shortest path between places.

3.2 Preprocessing Step: Shortest Paths Between Places

Given an accepting Petri net $PN = (P, T, F, W, M_0, M_F, l)$, it is possible to define a directed graph $G = (V, A)$ such that the vertices V are the places P of the Petri net, and $A \subseteq P \times P$ is such that $(p_1, p_2) \in A$ if and only if at least one invisible transition connects p_1 to p_2. Then, to each arc $(p_1, p_2) \in A$, a transition $\tau(p_1, p_2)$ could be associated picking one of the invisible transitions connecting p_1 to p_2.

Using an informed search algorithm for traversing the graph G, the shortest paths between nodes are found. These are a sequence of edges $\langle a_1, \dots a_n \rangle$ of minimal length, that correspond to a sequence of transitions $\langle t_1, \dots, t_n \rangle$ using the mapping provided by τ.

Given a marking M such that $M(p_1) > 0$ and $M(p_2) = 0$, a marking M' where $M'(p_2) > 0$ could be reached by firing the sequence $\langle t_1, \dots, t_n \rangle$ that is the shortest path in G between p_1 and p_2. The following subsection will explain how to apply the shortest paths to traverse invisible transitions and reach a marking where a transition is enabled.

3.3 Enabling Transitions

The approach described in this subsection helps to enable a transition t through the traversal of invisible transitions. This helps in avoiding the insertion of missing tokens when an activity needs to be replayed on the model, but no corresponding transition is enabled in the current marking M. Moreover, it helps to avoid time-consuming state-space explorations that are required by the approach proposed in [17].

For a marking M and a transition t, it is possible to define the following sets:

- $\Delta(M,t) = \{p \in {\bullet t} \mid M(p) < W(p,t)\}$ is the set of places that miss some tokens to enable transition t. If the set $\Delta(M,t)$ is not empty, then the transition t could not be enabled in the marking M.
- $\Lambda(M,t) = \{p \in P \mid W(p,t) = 0 \wedge M(p) > 0\}$ is the set of places for which the marking has at least one token and t does not require any of these places to be enabled.

When t is not enabled, the set $\Delta(M,t)$ is not empty. The idea is about using places in $\Lambda(M,t)$ (that are not useful to enable t) and, through the shortest paths, reach a marking M' where t is enabled.

Given a place $p_1 \in \Lambda(M,t)$ and a place $p_2 \in \Delta(M,t)$, if a path exists between p_1 and p_2 in G, then it is useful to see if the corresponding shortest path $\langle t_1, \ldots, t_n \rangle$ could fire in marking M. If that is the case, a marking M' could be reached having at least one token in p_2. However, the path may not be not realizable, or may require a token from one of the input places of t. So, the set $\Delta(M',t)$ may be smaller than $\Delta(M,t)$, since p_2 gets at least one token. The approach is about considering all the combinations of places $(p_1,p_2) \in \Lambda(M,t) \times \Delta(M,t)$ such that a path exists between p_1 and p_2 in G. These combinations, namely $\{(p_1,p_2),(p_1',p_2'),(p_1'',p_2'')\ldots\}$, are corresponding to some shortest paths $S = \{\langle t_1,\ldots,t_m\rangle, \langle t_1',\ldots,t_n'\rangle, \langle t_1'',\ldots,t_o''\rangle\}$ in G.

The algorithm to enable transition t through the traversal of invisible transitions considers the sequences of transitions in S, ordered by length, and tries to fire them. If the path can be executed, a marking M' is reached, and the set $\Delta(M',t)$ may be smaller than $\Delta(M,t)$, since a place in $\Delta(M,t)$ gets at least one token in M'. However, one of the following situations could happen: 1) no shortest path between combinations of places $(p_1,p_2) \in \Lambda(M,t) \times \Delta(M,t)$ could fire: in that case, we are "stuck" in the marking M, and the token-based replay is forced to insert the missing tokens; 2) a marking M' is reached, but $\Delta(M',t)$ is not empty, hence t is still not enabled in marking M'. In that case, the approach is iterated on the marking M'; 3) a marking M' is reached, and $\Delta(M',t)$ is empty, so t is enabled in marking M'. When situation (2) happens, the approach is iterated. A limit on the number of iterations may be set, and if it is exceeded then the token-based replay proceeds to insert the missing tokens in marking M.

The approach is straightforward when sound workflow nets without concurrency (only loops, sequences, and choices) are considered, since in the considered setting (M marking where transition t is not enabled) both sets $\Lambda(M,t)$ and $\Delta(M,t)$ have a single element, a single combination $(p_1,p_2) \in \Lambda(M,t) \times \Delta(M,t)$ exists and, if a path exists between p_1 and p_2 in G, and the shortest path could fire in marking M, a marking M' will be reached such that $\Delta(M',t) = \emptyset$ and transition t is enabled. Moreover, it performs particularly well on models that are output of popular process discovery algorithms, e.g., inductive miner, heuristics miner, etc., where potentially long chains of invisible (skip, loop) transitions needs to be traversed in order to enable a transition. The approach described in this subsection can also manage duplicate transitions corresponding to the activity that needs to be replayed. In that case, we are looking to enable any one of the

transitions belonging to the set $T_C \subseteq T$ that contains all the transitions corresponding to the activity in the trace. The approach is then applied on the shortest paths between places $(p_1, p_2) \in \cup_{t \in T_C} \Lambda(M,t) \times \Delta(M,t)$. A similar approach can be applied to reach the final marking when, at the end of the replay of a trace, a marking M is reached that is not corresponding to the final marking. In that case, $\Delta = \{p \in P \mid M(p) < M_F(p)\}$ and $\Lambda = \{p \in P \mid M_F(p) = 0 \wedge M(p) > 0\}$. This does not cover the case where the reached marking contains the final marking but has too many tokens.

3.4 Token Flooding Problem

To address the token flooding problem, which is one of the most severe problems when using token-based replay, we propose several strategies. The final goal of these strategies is to avoid the activation of transitions that shall not be enabled, keeping the fitness value low for problematic parts of the model. The common pattern behind these strategies is to determine *superfluous tokens*, that are tokens that cannot be used anymore. During the replay, f (initially set to 0) is an additional variable that stores the number of "frozen" tokens. When a token is detected as superfluous, it is "frozen": that means, it is removed from the marking and f is increased. Frozen tokens, like remaining tokens, are tokens that are produced in the replay but never consumed. Hence, at the end of the replay $p+m = c+r+f$. To each token in the marking, an *age* (number of iterations of the replay for which the token has been in the marking without being consumed) is assigned. The tokens with the highest age are the best candidates for removal. The techniques to detect superfluous tokens are deployed when a transition required the insertion of missing tokens to fire, since the marking would then possibly contain more tokens. One of the following strategies can be used:

1. Using a decomposition of the Petri net in semi-positive invariants [11] or S-components [1] to restrict the set of allowed markings. Considering S-components, each S-component should hold at most 1 token, so it is safe to remove the oldest tokens if they belong to a common S-component.
2. Using place bounds [12]: if a place is bounded to N tokens and during the replay operation the marking contains $M > N$ tokens for the place, the "oldest" tokens according to the age are removed.

3.5 Changes to the Implementation to Improve Performance

The implementation of the approach proposed in [17] has been made more efficient thanks to ideas adopted from the alignments implementation in ProM6 [5]:

1. *Post-fix caching*: a post-fix is the final part of a case. During the replay of a case, the couple marking+post-fix is saved in a dictionary along with the list of transitions enabled from that point to reach the final marking of the

model. For the next replayed cases, if one of them reaches exactly a marking + post-fix setting saved in the dictionary, the final part of the replay could be retrieved from the dictionary.
2. *Activity caching*: activity caching means saving in a dictionary, during the replay of a case, the list of hidden transitions enabled from a given marking to reach a marking where a particular transition is enabled. For the next replayed cases, if one of them reaches a marking + target transition setting saved in the dictionary, then the corresponding hidden transitions are fired accordingly to enable the target transition.

3.6 Evaluation

In this section, the token-based replay (as implemented in the PM4Py library) is assessed, looking at the speed and the output of the replay, against the alignments approach (as implemented in the "Replay a Log on Petri Net for Conformance Analysis" plug-in of ProM6). Alignments produce results that differ from token-based replay, so results are not directly comparable. Both are replay techniques, so the goal of both techniques is to provide information about how much a process execution is fit according to the process model (albeit the fitness measures are defined in a different way, and so are intrinsically different). This is valid in particular for the comparison of execution times: a trace may be judged fitting according to a process model in a significantly lower amount of time using token-based replay in comparison to alignments. If an execution is unfit according to the model, it can also be judged unfit in a significantly lower amount of time. For a comparison between the two approaches, read Section 8.4 of book [8] or consult [16, 3].

Table 1: Performance of PM4Py token-based replayer on real-life logs in comparison to the alignments approach implemented in ProM6 on models extracted by the inductive miner implementation in PM4Py.

Log	**Cases**	**Variants**	**T.I.P4Pys**	**A.I.P6s**	**Speedup**
repairEx	1104	77	0.06	0.2	3.3
reviewing	100	96	0.10	0.4	4.0
bpic2017	42995	16	0.30	1.5	5.0
receipt	1434	116	0.09	0.8	8.9
roadtraffic	150370	231	1.03	5.5	5.3
Billing	100000	1020	1.36	8.0	5.9

In Table 1, an evaluation of the performance of the token-based replayer on real-life logs respectively is provided. Tests have been done on a Intel I7-5500U powered computer with 16 GB DDR4 RAM. The logs can be retrieved from the 4TU log repository[3]. The *T.I.P4Pys* column shows the execution time (in seconds) of the token-based replay implementation in PM4Py on a model extracted by the inductive miner approach on the given log, the *A.I.P6s* column shows the

[3] The logs are available at the URL https://data.4tu.nl/repository/collection:event_logs

execution time of the alignment-based implementation in ProM6 on the same log and model. The Speedup column shows how many times the token-based replay is faster than the alignment-based implementation. For real-life logs and models extracted by the inductive miner, the token-based replay implementation in PM4Py is 5 times faster on average. Even for large logs, the replay time is less than a few seconds.

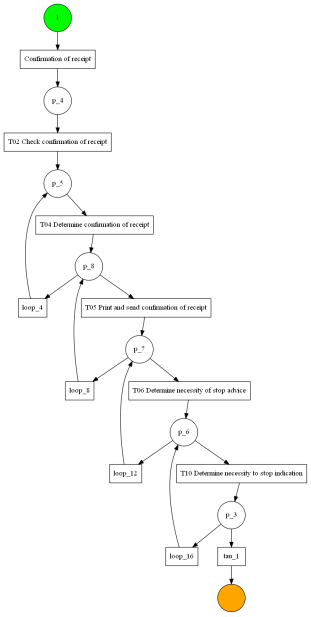

Fig. 1: Model extracted by the inductive miner implementation in PM4Py on a filtered version of the "Receipt phase of an environmental permit application process" event log. Excluding the activities of the log that are not in the model, only 53% of cases of the original log are fitting according to this model.

Table 2: Comparison in token-based replay execution times on models extracted by inductive miner on the given logs with or without postfix and activity caching.

Log	**No caching(s)**	**PC(s)**	**AC(s)**	**PC + AC(s)**
repairEx	0.10	0.08	0.08	*0.06*
reviewing	0.33	0.42	0.14	*0.10*
bpic2017	0.37	0.42	0.30	*0.30*
receipt	0.17	0.15	0.12	*0.09*
roadtraffic	1.58	2.08	1.18	*1.03*
Billing	2.23	1.91	1.45	*1.36*

In Table 2, the effectiveness of the implementation is evaluated in order to understand how the improvements in the implementation contribute to the overall efficiency of the approach. Columns in the table represent the execution time of the replay approach when no caching, only post-fix caching, only activity

caching and the sum of post-fix caching and activity caching is deployed. In the vast majority of logs, the combination of post-fix caching and activity caching provides the best efficiency.

Table 3: Fitness evaluation comparison between the PM4Py token-based replayer (without the token flood cleaning procedure), the token-based replayer in ProM5 and the alignments approach implemented in ProM6 on models extracted by the alpha miner and the inductive miner implementations in PM4Py. Since inductive miner returns a Petri net with perfect fitness, it is expected that the token-based replayer is able to replay the log returning fitness 1.0 for all such combinations. On models extracted by the alpha miner, that do not generally provide perfect fitness, it is expected that the implementation in PM4Py (without the token flood cleaning procedure) is equivalent to the token-based replay implementation in ProM5.

Log	**F.I.PM4Py**	**F.I.P5**	**F.I.P6**	**F.A.PM4Py**	**F.A.P5**
repairEx	1.0	1.0	1.0	0.88	0.88
reviewing	1.0	1.0	1.0	1.0	1.0
bpic2017	1.0		1.0	0.72	
receipt	1.0	1.0	1.0	0.39	0.39
roadtraffic	1.0		1.0	0.62	
Billing	1.0		1.0	0.69	

In Table 3, a comparison between the fitness values recorded by the token-based replay implementation in PM4Py, the token-based replay implementation in ProM5 and the alignments implementation in ProM6 is provided, for both alpha miner and inductive miner models. The meaning of the columns is the following: *F.I.PM4Py* is the fitness value achieved by the token-based replay implementation in PM4Py on a model extracted by the inductive miner approach on the given log, *F.I.P5* is the fitness value achieved by the token-based replay implementation in ProM5 on a model extracted by the inductive miner approach on the given log, *F.I.P6* is the fitness value achieved by the alignments implementation in ProM6 on a model extracted by the inductive miner approach on the given log, *F.A.PM4Py* is the fitness value achieved by the token-based replay implementation in PM4Py on a model extracted by the alpha miner approach on the given log, *F.A.P5* is the fitness value achieved by the token-based replay implementation in ProM5 on a model extracted by the alpha miner approach on the given log. For some real-life logs (bpic2017, roadtraffic, Billing) the token-based replay implementation in ProM5 did not succeed in the replay in 10 minutes (an empty space has been reported in the corresponding columns). Alignments have not been evaluated on the models extracted by alpha miner since it is not assured to have a sound workflow net to start with. The fitness values obtained in Table 3 show that the token-based replay implementation in PM4Py (without the token flood cleaning procedure), on these logs and the models extracted from them by the inductive miner, is as effective in exploring hidden transitions as the token-based replay implementation in ProM5 and the alignments implementation in ProM6.

Table 4: Comparison between the output of the token-based and alignments applied on some logs and the models extracted by the inductive miner implementation in PM4Py on a filtered version of these logs (using the auto filter method of PM4Py). The set of transitions activated in the model by the token-based replay and the alignments for each case has been considered (the middle columns report the overall number of transitions activated in the model by both approaches). Then, a similarity score has been calculated for each case considering the size of the intersection between the two sets and the size of the union. The minimum, maximum, average and median similarity score for the cases in the log has been reported in the right columns of the table, along with the fitness values provided by alignments and token-based replay.

Log	**Tot.T.Al.**	**Tot.T.TR.**	**Min.s.**	**Max.s.**	**Avg.s.**	**Med.s.**	**Fit.al.**	**Fit.tr.**
repairEx	18879	18459	0.538	1.0	0.977	1.0	0.977	0.986
reviewing	2658	2621	0.88	1.0	0.935	0.928	0.900	0.946
bpic2017	171980	171980	1.0	1.0	1.0	1.0	1.0	1.0
roadtraffic	1368414	815326	0.333	1.0	0.591	0.667	0.667	0.758

In order to compare token-based replay and alignments, a comparison between the output of the two approaches has been proposed in Table 4. Some popular logs, that are taken into account also for previous evaluations, are being filtered in order to discover a model (using inductive miner) that is not perfectly fit against the original log. Instead of comparing the fitness values, the comparison is done on the similarity between the set of transitions that were activated in the model during the alignments and the set of transitions that were activated in the model during the token-based replay. The more similar are the two sets, the higher should be the value of similarity. The similarity is calculated as the ratio of the size of the intersection of the two sets and the size of the union of the two sets. This is a simple approach, with some limitations: 1) transitions are counted once during the replay 2) the order in which transitions are activated is not important 3) the number of transitions activated by the alignments is intrinsically higher: while token-based replay could just insert missing tokens and proceed, alignments have to find a path in the model from the initial marking to the final marking, so a higher number of transitions is expected. In Table 4, the meaning of the columns is the following: *Tot.T.Al.* is the number of transitions activated by the alignments approach (a path leading from the initial to the final marking); *Tot.T.TR.* is the number of transititions activated by the token-based replay approach (that is not necessarily a path from the initial to the final marking); *Min.sim.* is the minimum similarity score between the alignments and the token-based replay approach on a case; *Max.sim.* is the maximum similarity score; *Avg.sim.* is the average similarity score; *Med.sim.* is the median similarity score; *Fit.al.* is the fitness value provided by alignments, *Fit.tr.* is the fitness value provided by token-based replay. This comparison, aside fitness values, confirm that the result of the two replay operations, represented as a set

of transitions activated in the model, is very similar, with the exception of the "Road Traffic Fine Management Process" log. For this log, the auto-filtering procedure of PM4Py produces an overly simple model, where token-based replay could survive by inserting missing tokens, but alignments cannot, hence the significantly larger number of transitions activated in the model to explain the behavior observed in the log. Table 4 provides some evidence, aside from fitness values, that the output of the two replay techniques is comparable.

To illustrate the importance of handling the token flooding problem, we consider the "Receipt phase of an environmental permit application process" event log. On this log, a sound workflow net has been extracted which is represented in Figure 1. For this log and model, token flooding occurs because the order of activities is interchanged in some variants of the log. As missing tokens are inserted multiple activities become enabled due to the surplus of tokens. As a result, token-based replay using the original approach yields diagnostics very different from the alignment-based approaches. The original values of average trace fitness and log fitness are 0.92 and 0.93 respectively. Applying the token flooding cleaning procedure, the values go down to 0.86 and 0.87 respectively, because the activation of unwanted parts of the process model is avoided. Albeit the underlying concepts/fitness formula are different (see Section 8.4 of [8]), it may be useful to see that the fitness value provided by alignments is 0.82, so with the token flooding cleaning procedure a more similar value of fitness is obtained.

3.7 Problems Not Addressed

The pre-processing step that stores a map of shortest paths between places is sensible to the presence in the model of implicit/redundant places. Indeed, two models with the same behavior can give different values. However, implicit places can be removed as a pre-processing step on the model. Token-based replay can return a list of transitions that have been activated in the model to replay the trace. However, this does not imply that a path through the model, from the initial to the final marking, is provided, since the insertion of missing tokens can happen if a transition needs to be enabled.

4 Approach: Localization of Conformance Checking Results

Next to providing an overall measure for conformance, conformance checking should also provide diagnostics pinpointing compliance problems. Therefore, we propose two localization approaches:

- The simplification of the original Petri net, in order to make the replay execution speed faster considering only the most problematic parts of a process model.
- The localization of problems encountered during the replay, that permits to understand where deviations happened and their effects.

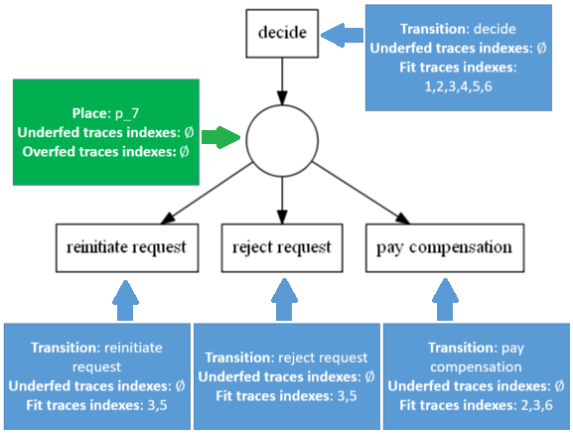

Fig. 2: Petri net, obtained from the "Running example" log, projected on a specific place. This kind of simplification helps to reduce the execution time of the replay operation, and to avoid the token flooding problem. The diagnostics obtained by applying our improved token-based replay are represented.

4.1 Simplification of the Original Petri Net

Replay operations on large models may take too much time. However, it is possible to simplify the model, keeping only parts that are problematic, in order to reduce the execution time of the replay operation.

The decomposition techniques presented in [2, 13, 14, 7] have been used to decompose a Petri net in several subnets for performance reasons. However, for diagnostic purposes an automated decomposition driven only by the model's structure is undesirable. Therefore, we provide the possibility to specify a list of activities in the log and corresponding transitions in the model to check. This is particularly useful when the user knows already which parts of the process are or could be problematic. We also add the possibility to get detailed information about a single element (place or transition) of the Petri net. This information is valuable when comparing fitting executions versus non-fitting executions.

With token-based replay, we propose two simplification approaches to focus attention:

- *Projection on a specific place*: when the preset and the postset of the place are not empty and contain only unique visible transitions, then it is possible to obtain a Petri net containing only the place and the transitions belonging to the preset and the postset. This is particularly useful to detect instances where some tokens are missing / are remaining on the specific place, while not being affected by problems like token flooding. A representation of a Petri net projected on a specific place, obtained from the "Running example" log, is shown in Figure 2.
- *Projection on a set of activities*: it is possible to make selected transitions invisible and retain only the transitions that have a label belonging to a specified set of activities as visible. Then reduction rules are applied to simplify the model with respect to the invisible transitions [4]. This guarantees to get

a Petri net that, for the specific set of activities, has the same language as the original Petri net.

```
from pm4py.objects.log.importer.xes import factory as xes_importer
from pm4py.algo.discovery.inductive import factory as inductive_miner
from pm4py.algo.conformance.tokenreplay import factory
as token_based_replay
from pm4py.evaluation.replay_fitness import factory
as replay_fitness_factory

log = xes_importer.apply("C:\\running-example.xes")
net, im, fm = inductive_miner.apply(log)
aligned_traces = token_based_replay.apply(log, net, im, fm)
fitness = replay_fitness_factory.apply(log, net, im, fm)
```

Fig. 3: Example PM4Py code to apply token-based replay to a log and an accepting Petri net.

4.2 Localization of the Replay Results

Localizing fitness issues in the process model is an essential step in the provision of more detailed diagnostics. The approach described in [17] already provided some diagnostics aimed at localizing the problem:

- *Place underfedness*: when missing tokens are inserted in the place during the replay operation of a case, the place is signed as underfed (it has fewer tokens than needed at some stage) for the specific case.
- *Place overfedness*: when remaining tokens are in the place after the end of the replay of a case, the place is signed as overfed (it has more tokens than needed) for the specific case.

Table 5: Localization of the replay result at place level on the filtered model, represented in Figure 1, obtained from the "Receipt phase" log (only places with problems have been reported).

Place	**# Cases Underfed**	**# Cases Overfed**
p_8	1	0
p_4	35	0
p_7	521	0

To introduce additional localized diagnostics at the transition level, it is important to notice that, when the transition is fired during the replay of a case, is possible to register the *current case status*, for example recording all values of the attributes of the current and of the previous events of the case. The easiest option is to keep a single value for each attribute, that is corresponding to the value of the last occurrence of the given attribute. So, the following localized information could be introduced at the transition level:

- *Transition underfedness*: some tokens needed to fire the transition are missing. It is possible to flag a transition as underfed for the specific case, saving also the status of the case when the transition has been fired.

– *Transition fitness*: the transition could be fired regularly. In this case, it is possible to save the status of the case when the transition has been fired.

It is important also the save information for events with an activity that is not corresponding to any transition in the model. This could be done saving the current case status when such activities happen.

Table 6: Localization of the replay result at the transition level on the filtered model, represented in Figure 1, obtained from the "Receipt phase" log (only transitions with problems have been reported).

Transition	**# Cases Underfed**	**# Cases Fit**
$T05$	1	1299
$T02$	35	1316
$T06$	521	830

The result of localization on a filtered version of the "Receipt phase of an environmental permit application process" event log, and the model represented in Figure 1, is shown in Table 5 (for places with problems) and Table 6 (for transitions with problems). Moreover, in Figure 2 the fitness information has been projected visually on the elements of the Petri net.

5 Advanced Diagnostics

The localized information is useful to compare, for each problematic entity, the set of cases of the log that are fit according to the given entity and the set of cases of the log that are not fit according to the given entity (called "unfit"). In particular, the following questions can be answered:

1. If a given transition is executed in an unfit way, what is the effect on the throughput time?
2. If a given activity that is not contained in the process model is executed, what is the effect on the throughput time?

These questions can be answered by throughput time analysis. Essentially, an aggregation (for example, the median) of the throughput times of fit and unfit cases is taken into account, and the results compared. Usually, transitions executed in an unfit way are corresponding to higher throughput times.

The comparison between the throughput time in non-fitting cases and fitting cases permits to understand, for each kind of deviation, whether it is important or not important for the throughput time. For evaluating this, the "Receipt phase of an environmental permit application process" log is taken. After some filtering operations, the model represented in Figure 1 is obtained. Several activities that are in the log are missing according to the model, while some transitions have fitness issues. After doing the token-based replay enabling the local information retrieval, and applying the *duration_diagnostics.diagnose_from_trans_fitness* function to the log and the transitions fitness object, it can be seen that transition

T06 Determine necessity of stop advice is executed in an unfit way in 521 cases. For the cases where this transition is enabled according to the model the median throughput time is around 20 minutes, while in the cases where this transition is executed in an unfit way the median throughput time is 1.2 days. So, the throughput time of unfit cases is 146 times higher in median than the throughput time of fit cases. Considering activities of the log that are not in the model, that are likely to make the throughput time of the process higher since they are executed rarely, applying the *duration_diagnostics.diagnose_from_not_existing_activities* method it is possible to retrieve the median execution of cases containing these activities, and compare it with the median execution time of cases that do not contain them (that is 20 minutes). Taking into account activity *T12 Check document X request unlicensed*, it is contained in 44 cases, which median throughput time is 6.9 days (505 times higher than standard).

6 Conclusion

In this paper, an improved token-based replay approach has been proposed and has been implemented in the Python process mining library PM4Py[4]. A set of process discovery, conformance checking and enhancement algorithms are provided in the library. An example script, that loads a log, calculates a model, and does conformance checking, is shown in Figure 3. This illustrates that the conformance checking technique presented in this paper can be combined easily with many other process mining and machine learning approaches.

The approach has shown to be more scalable than existing approaches. Due to a better handling of invisible transitions and improved intermediate storage techniques, the approach outperforms the original token-based approaches, and proves to be faster than alignment-based approaches also for models with invisible transitions.

Next to an increase is speed, the problem of token flooding is addressed by "freezing" superfluous tokens (see Section 3.4). This way replay does not lead to markings with many more tokens than what would be possible according to the model, avoiding the activation of unwanted parts of the process models and leading to lower values of fitness for problematic parts of the model.

Localization of conformance checking using token-based replay can be used to simplify the model prior to replay and help to better diagnose where the deviation happened. Moreover, we showed that we are able to diagnose the effects of deviations on the case throughput time.

The approach has been fully implemented in the PM4Py process mining library. We hope that this will trigger a revival of token-based replay, a technique that seemed abandoned in recent years. Especially when dealing with large logs, complex models, and real-time applications, the flexible tradeoff between quality and speed provided by our implementation is beneficial.

[4] It can be installed in Python $\geq$ 3.6 through the command *pip install pm4py*. See http://pm4py.pads.rwth-aachen.de/installation/ for details.

References

1. van der Aalst, W.: Structural characterizations of sound workflow nets. Computing Science Reports **96**(23), 18–22 (1996)
2. van der Aalst, W.: Decomposing Petri nets for process mining: A generic approach. Distributed and Parallel Databases **31**(4), 471–507 (2013)
3. van der Aalst, W., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **2**(2), 182–192 (2012)
4. van der Aalst, W., van Hee, K.M., ter Hofstede, A.H., Sidorova, N., Verbeek, H., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. Formal Aspects of Computing **23**(3), 333–363 (2011)
5. Adriansyah, A.: Aligning observed and modeled behavior (2014)
6. Adriansyah, A., Sidorova, N., van Dongen, B.: Cost-based fitness in conformance checking. In: Application of Concurrency to System Design (ACSD), 2011 11th International Conference on. pp. 57–66. IEEE (2011)
7. van den Broucke, S.K., Munoz-Gama, J., Carmona, J., Baesens, B., Vanthienen, J.: Event-based real-time decomposed conformance analysis. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". pp. 345–363. Springer (2014)
8. Carmona, J., Dongen, B., Solti, A., Weidlich, M.: Conformance Checking: Relating Processes and Models. Springer (2018)
9. Kerremans, M.: Gartner Market Guide for Process Mining, Research Note G00353970 (2018), www.gartner.com
10. Lee, W.L.J., Verbeek, H., Munoz-Gama, J., van der Aalst, W., Sepúlveda, M.: Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining. Information Sciences **466**, 55–91 (2018)
11. Martínez, J., Silva, M.: A simple and fast algorithm to obtain all invariants of a generalised Petri net. In: Application and Theory of Petri nets, pp. 301–310. Springer (1982)
12. Miyamoto, T., Kumagai, S.: Calculating place capacity for Petri nets using unfoldings. In: Application of Concurrency to System Design, 1998. Proceedings., 1998 International Conference on. pp. 143–151. IEEE (1998)
13. Munoz-Gama, J., Carmona, J., van der Aalst, W.: Conformance checking in the large: Partitioning and topology. In: Business Process Management, pp. 130–145. Springer (2013)
14. Munoz-Gama, J., Carmona, J., van der Aalst, W.: Single-entry single-exit decomposed conformance checking. Information Systems **46**, 102–122 (2014)
15. Rogge-Solti, A., Senderovich, A., Weidlich, M., Mendling, J., Gal, A.: In log and model we trust? a generalized conformance checking framework. In: International Conference on Business Process Management. pp. 179–196. Springer (2016)
16. Rozinat, A., van der Aalst, W.: Conformance testing: measuring the alignment between event logs and process models. Citeseer (2005)
17. Rozinat, A., van der Aalst, W.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**(1), 64–95 (2008)
18. Taymouri, F., Carmona, J.: A recursive paradigm for aligning observed behavior of large structured process models. In: International Conference on Business Process Management. pp. 197–214. Springer (2016)