C-3PA: Streaming Conformance, Confidence and Completeness in Prefix-Alignments

Kristo Raun[1], Max Nielsen[2], Andrea Burattin[2], and Ahmed Awad[1]

[1] University of Tartu, Tartu, Estonia {kristo.raun,ahmed.awad}@ut.ee
[2] Technical University of Denmark, Kgs. Lyngby, Denmark
s202785@student.dtu.dk,andbur@dtu.dk

Abstract. The aim of streaming conformance checking is to find discrepancies between process executions on streaming data and the reference process model. The state-of-the-art output from streaming conformance checking is a prefix-alignment. However, current techniques that output a prefix-alignment are unable to handle warm-starting scenarios. Further, no indication is given of how close the trace is to termination — a highly relevant measure in a streaming setting.
This paper introduces a novel approximate streaming conformance checking algorithm that enriches prefix-alignments with confidence and completeness measures. Empirical tests on synthetic and real-life datasets demonstrate that the new method outputs prefix-alignments that have a cost that is highly correlated with the output from the state-of-the-art optimal prefix-alignments. Furthermore, the method is able to handle warm-starting scenarios, and indicate the confidence level of the prefix-alignment. A stress test shows that the method is well-suited for fast-paced event streams.

Keywords: Streaming conformance checking · Prefix-alignments · Warm-starting · Confidence · Data streams

1 Introduction

Every organization has processes that need to be executed in order to achieve the organizational goals. Most contemporary processes are built on computer systems — every action leaves a footprint in a database or a log file. This kind of data paves way for process mining, allowing for the discovery of process models, process monitoring, and measurements of process executions based on organizational data. A key component of process mining is conformance checking, i.e. are processes executed as expected based on a given reference process model?

Traditionally, conformance checking is done in an offline setting: an event log is constructed by filtering on events that occurred within a specified time range. The event log cannot contain any events which occurred after the point of data extraction. These limitations hide several important caveats.

Firstly, process executions commonly exhibit overlap and parallelism. This indicates that some of the process executions from a specified time range are likely

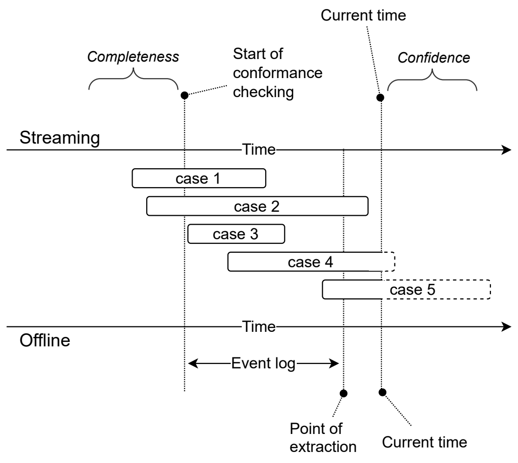


Fig. 1: Differences between streaming and offline conformance checking.

to have started before the time range, and some may not yet have concluded. In some cases, this can be remedied by domain knowledge and data preprocessing, e.g. filtering out cases that have not yet concluded. However, such preprocessing induces data loss due to the removal of incomplete process executions and consequently lowers the trustworthiness of the analysis.

Secondly, and more importantly, in an offline setting, the data used for analysis, and the results, are obsolete by design. The longer it takes from data extraction to analysis to decision-making, the less valuable the data becomes. In certain use cases, such as fraud detection, autonomous driving, or health monitoring, making decisions based on stagnant data is impractical, except for high-level trend analysis. Being unaware of discrepancies in individual ongoing process executions can have a broad impact on an organization. This is supported by recent studies, indicating the need for moving towards stream processing and real-time availability of data in process mining [9].

In streaming conformance checking, data is observed as an unbounded stream of events rather than a static event log [5]. The organization can receive indications of discrepancies in a continuous manner, as soon as these discrepancies occur. This allows the organization to potentially remedy the negative impact of observed discrepancies. In order to do that effectively, it is important to know the exact locations of both, the wrongly executed and skipped activities.

In long-running processes, an issue that might occur is the *warm-starting* scenario. That is, some process executions are ongoing before the conformance checker is initiated, and are thus not completely observable. Examples of such cases can be seen in Figure 1 with case 1 and 2. In most conformance checking techniques, this causes a false positive, i.e., a discrepancy is indicated, even if the process followed a conforming behavior. Completeness issues may lower the trustworthiness of the conformance checking method, and require a *grace* *period*, after which a conformance checking method can be expected to output the correct conformance.

Similarly, the ending of a process execution poses a challenge. Streaming conformance checking methods give equal weight to the conformance of traces that are at the beginning of traces, as they do for traces that are near the end of their lifecycle. It can be argued that a conforming trace that has seen only a few events has a higher probability of divergence than a conforming trace that has seen most of the expected events. For example, if case 4 and case 5 in Figure 1 are both conforming, then case 5 has a higher chance of seeing non-conforming behavior. This indicates the need for a confidence measure that would complement the conformance so that organizations can be more alert for traces that are just initializing, rather than the traces that are concluding.

The above challenges are especially relevant in the context of complex business processes. When complex processes are deployed, a high level of automation is needed from the computing side, while humans are typically in charge of supervision. In case of deviations, fast decision-making is crucial to keep the systems up and running. To support fast decision-making in such systems, we pose the following two research questions:

- RQ1: can we define a streaming conformance checking algorithm capable of computing prefix alignments with confidence and completeness measurements in warm-starting scenarios?
- RQ2: can the algorithm identified in RQ1 be used in real-world settings?

The paper is structured as follows. Section 2 describes the relevant background. Section 3 discusses the current state of the art. Section 4 introduces the approach and the C-3PA algorithm for computing prefix-alignments enriched with confidence and completeness measures. Section 5 shows the results of the empirical testing, a comparison to existing methods, and a discussion of the algorithm's properties and its matching to the research questions. Section 6 concludes the work and offers ideas for potential further research.

2 Background

In this section, we introduce the main components necessary for understanding the theoretical background of the introduced approach. For further background in process mining and conformance checking, we refer to [1] and [8], respectively.

In process mining terminology, a process model indicates the expected behavior. A process model can be depicted with various semantics. Most conformance checking methods assume the process model to be a Petri net. As is characteristic of real-life processes, a Petri net may exhibit behavior such as sequences, choices, parallelism, and loops [1]. The sequence of fired transitions in a Petri net is an execution sequence $\pi \in B$, where B is the set of behaviors allowed by the Petri net. B is infinite when the model has loops because a loop can unfold an unlimited number of times.

Once the expected behavior is known, the conformance checker also needs to know what is the actual observed behavior. In offline conformance checking, the actual behavior is commonly found in an event log. An event $e \in \mathbb{E}$ is a tuple $e = (c, a, t) \in \mathcal{C} \times \mathcal{A} \times \mathcal{T}$, referring to a case identifier $c \in \mathcal{C}$, an associated activity $a \in \mathcal{A}$, and a timestamp $t \in \mathcal{T}$. An event log $L \in \mathcal{B}(\mathcal{C} \times \mathcal{A} \times \mathcal{T})$ can be considered a container of traces. A trace σ is a finite sequence of events that can be grouped together based on the case id. $\hat{\sigma}$ refers to the prefix of a trace, e.g. activities seen so far in an ongoing trace.

Streaming conformance checking is done on an event stream $S \colon \mathbf{N}_{\geq 0} \longrightarrow \mathbb{E}$. Conceptually, a stream is not a collection of events but the natural occurrence of events — at any given time, a new event may be seen, having either a known or a previously unseen case id. An event stream is expected to be unbounded.

A process model and an event log/stream are inputs for a conformance checker. The state of the art output is an alignment between a trace and a model. More formally, an alignment $\gamma = \langle (x_1, y_1), \ldots, (x_n, y_n) \rangle$ is a sequence of pairs, where each pair $(x, y) \in (\mathcal{A} \cup \{\gg\}) \times (\mathcal{A} \cup \{\gg\})$ links an activity of the trace σ, or the skip symbol $\gg$, to an activity of the execution sequence π, or the skip symbol, whereas a step $(\gg, \gg)$ is illegal. Alignments allow for an intuitive understanding of the nonconforming activities: they indicate the places where activities match with the expected behavior of the process model (a synchronous move), the places where an activity diverged (a log move), and also the places where the activity expected by the model did not occur (a model move). We denote a conformance cost of an alignment as $\delta(\gamma)$. While different costs can be assigned to log and model moves (non-synchronous moves), in this paper we assume a cost of 1 for these non-synchronous activities. An alignment can be optimal or suboptimal. An optimal alignment has the minimal cost, i.e. commonly the least amount of non-synchronous moves between a trace and a model. One trace can have multiple optimal alignments. In Petri nets, a silent transition τ indicates a possible skip activity. In accordance with most other research in this area, this paper assigns model moves on τ transitions a cost of 0, i.e. if the model allows for skips and the skip is done, then it is not penalized.

As described in [17], in a streaming setting, an alignment overestimates the cost of divergence. Thus, the prefix-alignment ($\hat{\gamma}$) is preferred, as the alignment is not forced to complete the execution sequence on the model. In the rest of the paper, *complete alignment* is used to refer to the alignments from the previous paragraph and to distinguish them from *prefix-alignments*.

While various techniques exist for optimizing the calculation of an optimal alignment, it is still an exponentially hard problem to solve. Thus, approximation methods have been introduced to speed up the calculation. In this paper, the trie data structure is used. A trie $T = (N, E, \mathit{root}, l, \mathcal{F}, \mathit{min}, \mathit{max})$ where N is a finite set of nodes, $E \subset N \times N$ is a set of edges, $\mathit{root} \in N$ is the root of the trie, $l : N \to (\mathcal{A} \cup \{\bot\})$ is a labeling function for nodes, $\mathcal{F} \subset N$ indicates leaf nodes, and $\mathit{min}, \mathit{max} : E \to \mathbb{N}$ are relations assigned to an edge showing the minimum and maximum distance to reach a leaf node when traversing that edge.

A trie serves as the process model instead of, for example, a Petri net. Importantly, if a Petri net includes a loop, then the Petri net can generate infinite behavior, however a trie needs to be finite. Thus, the behavior B' allowed by the trie is a subset of the behavior in the Petri net $B' \subseteq B$. Similarly to a Petri net, a trie can be constructed by giving the model constructor an event log - i.e., a *proxy log*. Unlike the Petri net, a trie allows for all of the behavior in the proxy log, but no behavior that is outside of it.

A state $s \in S$ is a tuple $(n, \hat{\gamma}, \delta(\hat{\gamma}), \epsilon(\hat{\gamma}), \upsilon(\hat{\gamma}), dt)$, where n is the current node in the trie, $\hat{\gamma}$ is the current prefix-alignment, $\delta(\hat{\gamma})$ is the alignment cost of the current state, $\epsilon(\hat{\gamma}) \in \mathbb{N}$ is the completeness measure, quantifying the behavior not seen at the beginning of the trace,$\upsilon(\hat{\gamma}) \in \mathbb{N}$ is the confidence measure, indicating the amount of behavior not yet seen, and dt is the associated decay time of the state used for releasing states from memory. A state buffer $BS : \mathcal{C} \mapsto \mathcal{P}(S)$ is a mapping of case ids to the powerset of states. The state buffer is important for keeping track of the latest states of each seen case.

3 State of the art

One of the first methods for offline conformance checking was token-based replay [13]. The goal is to replay the traces on top of the Petri net. While the method is computationally efficient, it has some important shortcomings: it does not handle well τ transitions nor duplicate labels, and it is not highly informative in terms of the occurred discrepancies.

The alignment-based technique from [2] has been widely accepted as the state of the art for some years. The main benefits of alignments over token-based replay are the deterministic and concise representation of both conforming and non-conforming activities. An unfavorable quality of alignments is the computational complexity — in order to calculate optimal alignments, commonly a synchronous product net is built between the Petri net model and a model representation of the observed trace. Then, a search algorithm such as $A*$ is used for finding the shortest path that minimizes the conformance cost. While methods have been introduced to improve the computation time, it is generally still considered impractical to use alignments in large real-life logs and it remains an active research area to improve the computation complexity of alignments [8].

As mentioned in Section 2, complete alignments generally overestimate the cost in a streaming setting, because the methods force the moves on the model to a final state. Prefix-alignments, originally introduced in [3] and adapted to a streaming setting in [17], alleviate the problem by not penalizing traces that have not concluded according to the model. The method introduced in [17] calculates prefix alignments on top of event streams. Further, the method includes a window parameter, allowing for a trade-off between computational complexity and approximation error. A window size of 1 has the least computational complexity, but the highest possible error, as it only builds on top of the previously calculated prefix-alignment. An infinite window size allows for the computation of optimal alignments at the cost of increased computation time.

Further work in prefix-alignments has mostly attempted to improve the memory footprint of the calculation [14, 16]. Recently, [12] introduced an approximate streaming algorithm on top of the trie data structure for calculation of prefix-alignments that is able to outperform previous methods for prefix-alignment computation. However, similarly to previous prefix-alignment methods, it is unable to handle the warm-starting scenario and does not indicate the confidence of the prefix-alignment.

One of the first computationally efficient streaming conformance checking methods was introduced in [6]. While the method is efficient, it only indicates whether the trace is conforming to the model or not. Thus, an extension was introduced in [7] where the terms *completeness* and *confidence* were introduced for quantifying the warm-starting and trace conclusion, similarly to this paper. More recently, a Hidden Markov Model (HMM) based approach was introduced in [10] that supports warm-starting, but not quantifying the confidence.

To generalize, two main paths for streaming conformance checking can be observed. Metrics-based approaches, where additional information is embedded on top of the conformance. The main benefits of these methods are generally a fast computation time and a more holistic view of a trace via the completeness and confidence measures. The downside is the vagueness of the conformance metric, which is well-suited for alerting purposes, but lacks a clear indication of the discrepant activities. The other path is the prefix-alignment based approaches. Diametrically, these approaches benefit from a clear representation of both the unexpected and skipped activities. However, they generally exhibit a longer computation time, especially for finding the optimal prefix-alignments. More importantly, they are unable to handle warm-starting nor explain the confidence of the conformance cost.

This paper attempts to bridge this gap by next introducing a method for incorporating the completeness and confidence measures into prefix-alignments.

4 Conformance, confidence and completeness

In this section, we introduce the C-3PA algorithm that is able to consider conformance, confidence, and completeness, while outputting prefix-alignments. We first discuss the approach on a high level and introduce how confidence and completeness measures are contrived. Then, we look at the execution steps of the algorithm. Finally, we discuss the space and time complexity of the approach.

4.1 Approach

As discussed in Section 2, in order to do conformance checking it is needed to know the allowed behavior. In order to be computationally more efficient, the method introduced here uses the trie to represent the allowed behavior. The approach builds upon [4] and [12], with the contribution of this paper highlighted in Figure 2. Importantly, unlike any previous approach, the prefix-alignments are augmented with confidence and completeness measures. To the best of our

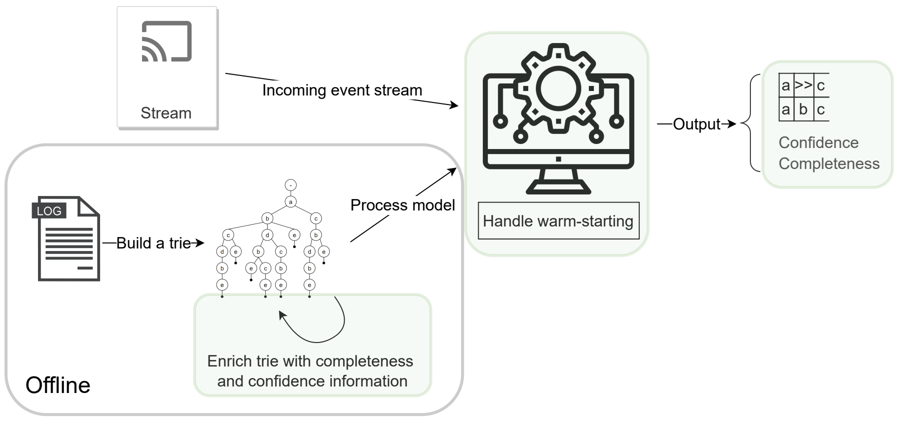


Fig. 2: Approach overview with contributions of this paper highlighted.

knowledge, this is the first method that outputs prefix-alignments while handling warm-starting scenarios and computing confidence and completeness.

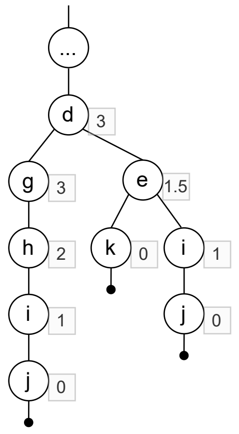
Confidence shows how much of the trace is expected to still arrive. For this purpose, the nodes of the trie have been supplemented with information about their confidence cost. To calculate the confidence measure the approach looks at all the paths that go from the current node until a leaf node and takes the average of the minimum paths. An example is shown in figure 3a. The calculation gives equal weight to all possible paths from the current node and is thus a good indicator for the likelihood of a trace's conclusion. Alternative methods are discussed at the end of Section 5 as they are out of scope for this paper.

Completeness quantifies the warm-starting of a seen trace. In other words, how much of the behavior of the trace occurred before the conformance checker was able to observe it. In order to achieve this, the construction of the trie is augmented by creating edges from the root node to every other node in the trie. Essentially, it is a mapping of activities to potential warm-starting nodes. Every known activity has a map of costs associated with warm-starting, which point to the set of nodes with that particular cost and activity label. An example is shown in Figure 3b.

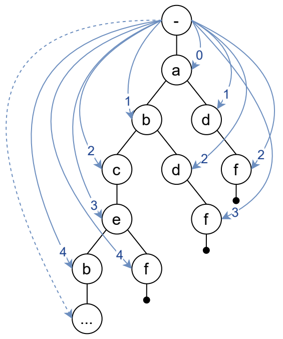
4.2 C-3PA algorithm

The pseudocode for the C-3PA algorithm is shown in Algorithm 1. The algorithm takes as input an event, the trie, and a state buffer. Based on the case ID of the arrived event, the algorithm checks from the state buffer whether the case has previously been seen or not. If a case has not been seen previously, then a new state is instantiated from the root node of the trie.

The algorithm iterates over each state associated with this case, and if possible then makes a synchronous move on the current activity. If a synchronous move is not possible, then a cost limit is instantiated — a new event cannot increase the cost of the trace by more than 1. Three types of moves are attempted:



(a) Confidence annotation.



(b) Completeness annotation.

Fig. 3: Trie enrichments.

a log move, model moves, and *warm-starting moves*. The warm-starting moves use the edge augmentation of completeness cost, as shown in Figure 3b, to try to *jump* to the currently seen event from the root node. Importantly, the guiding cost function then determines the best states, considering both the conformance and completeness. The completeness cost is stored separately from the conformance cost, allowing to quantify the impact of the warm-starting scenario and paving way for future work in terms of preferring either warm- or cold-starting.

The guiding cost function does not consider the confidence measure. Confidence is looking into the future and paths that conclude earlier have a higher confidence value. Thus, using confidence as part of the equation would guide the algorithm to favor the shortest paths through the trie. This may lead to suboptimal paths and is thus not a natural part of what should steer the algorithm. However, there may be use cases where confidence, with alterations, could be used as part of the cost function. This is discussed at the end of Section 5.

The most recent state of a case in the state buffer contains information about the prefix alignment, completeness, and confidence of the trace.

4.3 Complexity

The trie construction happens offline and thus mainly impacts the space complexity. Based on [12], the trie is commonly $O(log(|L|)$ to the size of the *proxy log* used to generate the trie. Adding the confidence measure requires a single traversal over all the nodes of the trie, $O(N)$. Warm-starting expects an edge from the root node to every node in the trie. Thus, adding the edges during the construction of the trie is also $O(N)$.

Synchronous moves and log moves can be done in $O(1)$. The biggest impact on the time complexity, thus, comes from handling model moves and warm-starting. Both of these depend on the branching factor of the trie. The branching factor is in the worst case the number of traces in the *proxy log* $O(|\sigma \in L|)$.

Algorithm 1 C-3PA

```
Input: $e(c, a), T, B$
 1: $S \leftarrow \emptyset$
 2: $S_{interim} \leftarrow \emptyset$
 3: if  $c \in B$ then
 4:     $S \leftarrow B(c)$
 5: else
 6:     $S \leftarrow \{s_{root}\}$                          ▷ Initialize state with root node
 7: for each $s \in S$ do
 8:     if $s.syncPossible(a)$ then        ▷ Attempt to make a sync move on activity a
 9:         $S_{interim} \leftarrow S_{interim} \cup \{s_{new}\}$   ▷ Store a new state with synchronous move
10: if $noSyncMovesDone$ then
11:     $cost_{lim} \leftarrow minCost(S) + 1$             ▷  Assign a cost limit for non-sync moves
12:     for each $s \in S$ do
13:         $s_{log} \leftarrow makeLogMove(s, a)$                 ▷ New state with log move
14:         $S_{model} \leftarrow makeModelMoves(s, a)$ ▷ A set of new states with model moves
15:         $S_{ws} \leftarrow makeWarmStart(s, a)$          ▷ A set of new warm-starting states
16:         for each $s_{temp} \in \{s_{log}\} \cup S_{model} \cup S_{ws}$ do
17:             if $s_{temp}.\delta(\gamma) + s_{temp}.\epsilon(\gamma) <= cost_{lim}$ then
18:                 $S_{Interim} \leftarrow S_{Interim} \cup \{s_{temp}\}$
19: $S \leftarrow manageStates(S)$               ▷ Update decay time, remove old states
20: for each $s_{temp} \in S_{interim}$ do
21:     $S \leftarrow S \cup s_{temp}$
22: $B.S \leftarrow S$
```

The depth of the search is dependent on the length of the currently seen prefix $O(|\hat{\sigma}|)$. The complexity is thus $O(|\sigma \in L| \times |\hat{\sigma}|)$. In a process model, this would indicate behavior that allows any activity to occur as the first activity, followed by an infinite loop of a single unique activity. Thus, the amortized complexity is more likely $O(log(|\sigma \in L|) \times log(|\hat{\sigma}|))$. L is finite and can thus be considered a constant, leaving the complexity as $O(log(|\hat{\sigma}|))$, i.e., increasing logarithmically with the size of the trace prefix.

In conclusion, this means that the computation should only be hindered if the trace lengths become very large, making it suitable for most streaming use cases. This answers RQ1 from Section 1. Next, we will look at the empirical evaluation to answer RQ2.

5 Experiments

In this section, we look at the empirical tests conducted to validate the algorithm's[3] output and to compare it to existing algorithms. First, we look at the experiments related to the warm-starting scenario. Specifically, the goal is to validate whether the algorithm is able to handle warm-starting and what are the possible implications of enabling warm-starting under different settings. Second, we investigate the conformance result of the algorithm, both in terms of the correctness of the prefix-alignment, and correlation with other methods. Then, we look at the computation speeds of various methods and conduct stress testing

[3] https://github.com/MaxTNielsen/ConformanceCheckingUsingTries/tree/current_branch

on the new algorithm to validate its applicability for streaming settings. Finally, we end with a discussion of the results obtained, the strengths and weaknesses of the introduced algorithm.

For running the experiments, the real-life event logs from BPI challenges in 2012[4] and 2017[5] were used. Additionally, synthetic datasets[6] were included as the datasets include a log and a pre-defined Petri net reference model. In total, 14 original logs were used. Event logs were used to validate the entire behavior of C-3PA and allow for equal comparison against other methods, without impact from networking or other outside factors.

For the BPI logs, the Inductive Miner [11] was used with a noise threshold of 0.95 to discover a Petri net model. To build the tries used by the C-3PA algorithm, logs were simulated on top of the Petri net models using the method from [15] with 2000 generated traces, random path simulation and a looping factor of 3. For warm-starting validation, the logs were pre-processed by filtering out 20% or 50% of the starting activities of each trace in each log, resulting in an additional 28 logs.

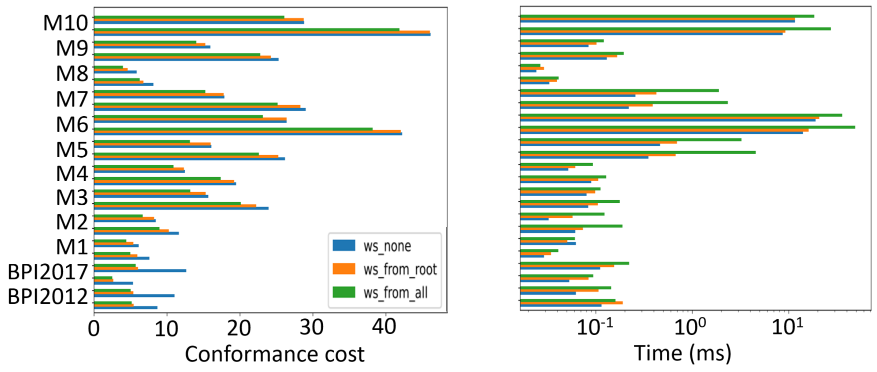
5.1 Warm-starting

To validate the warm-starting capability of the algorithm, the algorithm was initialized with the following three settings: warm-starting enabled from all states, warm-starting enabled only from the root state, warm-starting disabled. The three variations were executed on the 28 logs pre-processed for warm-starting. The average conformance costs of these executions are shown in Figure 4a.

The conformance cost improvements for warm-starting enabled only from the root state (*ws_from_root*) is moderate compared to the variation with no warm-starting (*ws_none*): across all the datasets, the improvement is 7.3%. An apparent reason for this is that the root state needs to be in memory, and thus once the root state is out of the buffer, warm-starting is no longer an option. An implication of this option is that any chosen warm-starting scenario will be equal to doing model moves on the unseen prefix.

For warm-starting from all states (*ws_from_all*), the change in the conformance cost is much more noticeable. Across all datasets, the improvement is 17.1%. This comes, though, at the cost of execution time. As shown in Figure 4b the warm-starting across all states is taking noticeably longer. This makes sense, because warm-starting is costly, and with the warm-starting enabled for all states, the warm-starting will be visited for each non-synchronous move. However, the benefit of warm-starting is one of the focal points of the algorithm, and thus in the following experiments the option with warm-starting enabled from all states will be used.

[4] https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
[5] https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b
[6] https://github.com/PADS-UPC/RL-align/tree/master/data/originals/M-models



(a) Alignment costs.

(b) Execution time (logarithmic scale).

Fig. 4: Warm-starting experiments.

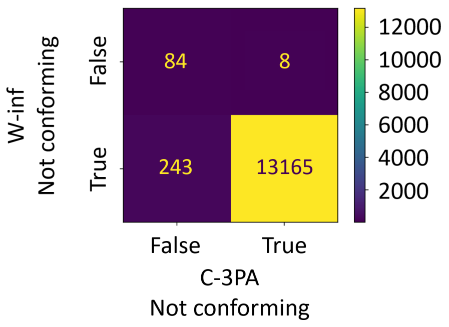
5.2 Comparison to existing methods

The algorithm introduced in this paper is an approximate algorithm. Thus, it is important to validate that the algorithm is actually outputting the correct conformance. In the following, we will investigate how precise the algorithm is for indicating conformance issues by building a confusion matrix with optimal prefix-alignments as the baseline, and then analyzing the Spearman correlation of non-conforming results.

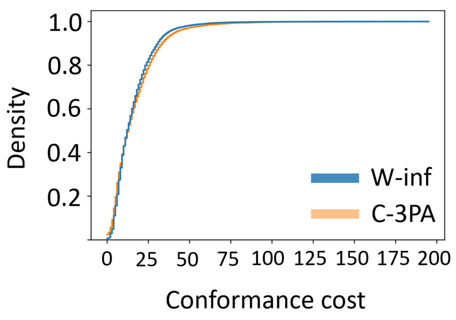
Confusion matrix To assess the correctness, the first step is to evaluate how often the algorithm reports conformance when actually non-conformance should have been reported and vice-versa. For this comparison, the optimal prefix-alignments from [17] are used as the ground truth. The derived confusion matrix across all 12 original datasets is shown in Figure 5a. The confusion matrix shows that almost all traces are correctly classified, with most traces being non-conforming to the process models. 243 traces are false positive — C-3PA indicates a compliant trace, while actually non-conformance is shown by optimal prefix-alignments. 8 traces are false negatives, indicating that C-3PA classified the trace as non-conforming while actually, it was conforming.

The interpretation is that generally, the algorithm can classify non-conformant traces well. As the algorithm is dependent on the trie data structure, a potential improvement for the classification could be achieved by increasing the size of the trie. For the purposes of this paper, such an investigation is out of scope. Another thing to note is that there is a high proportion of non-conformant traces present in the datasets. Still, as the ultimate goal for a conformance checker should be to detect non-conformant behavior, this skewness is considered acceptable.

Correlation Spearman correlation was used to validate that the output from C-3PA behaves similarly to the output from previously existing algorithms. Table



(a) Confusion matrix



(b) Cumulative distribution function

Fig. 5: Warm-starting experiments.

Correlations	**HMM**	**BP**	**OCC W-1**	**OCC W-inf**
Conformance	0.28	0.52	0.95	0.98
Completeness	0.66	0.35	-	-
Confidence	-	0.44	-	-

Table 1: Spearman correlations against other methods.

1 shows the correlations, with 1 indicating complete positive correlation, −1 indicating complete negative correlation, and 0 indicating no correlation.

As discussed in Section 3, the HMM [10] and BP [7] methods are able to output additional measures in addition to conformance, but they do not compute the prefix-alignments. Thus, the conformance correlation is moderate with these methods. Interestingly, the completeness correlation with HMM is relatively strong, while it is much weaker with the BP method. The confidence is also moderately correlated with the output from BP. All in all, it seems that C-3PA is giving output similar to these methods, but due to operational differences, the algorithms are not too strongly correlated.

In comparison with the prefix-alignments with window size 1 (OCC W-1) and optimal prefix-alignments (OCC W-inf) from [17], the conformance correlation is very strong. For further investigation, cumulative distribution functions were constructed as shown in Figure 5b. The resulting plots indicate a high similarity between the distributions, exhibiting almost identical curves. This indicates that despite the underlying approximations, the C-3PA algorithm is suitable for outputting prefix-alignments describing process deviations.

5.3 Stress test

Important characteristics of streaming conformance checking are event processing time and memory consumption. The events may arrive in a very fast manner, and it is important to calculate the conformance quickly. At the same time, the stream is unbounded, but the memory of the conformance checker is not. Thus, the method needs to have a good handling of memory.

	C-3PA	OCC W-1	OCC W-inf	HMM	BP
BPI2012	2.66	48.95	95.89	2.94	0.04
BPI2017	2.14	40.32	80.66	3.02	0.04
M1	0.51	8.61	13.48	5.35	0.03
M2	1.50	53.94	85.55	17.67	0.03
M3	7.39	-	-	-	-
M4	8.57	164.96	331.38	2.20	-
M5	47.60	-	-	-	-
M6	1871.24	-	-	-	-
M7	29.34	-	-	-	-
M8	1.21	5.67	8.68	1.10	0.02
M9	14.07	443.58	740.48	4.79	-
M10	1028.41	-	-	-	-

Table 2: Average processing time per event (ms).

The event processing time of the C-3PA algorithm and other methods is shown in Table 2. To be noted, the results need to be interpreted with some reservations: HMM implementation is in Python, while all other methods are implemented in Java. Further, such a direct comparison may be influenced by factors deriving from implementation, rather than an algorithm's actual potential. Regardless, it is currently the best indication available for showing the applicability of the various methods in a streaming setting.

Based on the results, C-3PA outperforms OCC, in some cases by an order of magnitude, while simultaneously being able to handle warm-starting and indicating the confidence of the prefix-alignments. The results are in most cases notably slower than that of BP, but this is expected as BP does not output the prefix-alignments, but rather just gives a trace-level measurement of the conformance. A dash (-) indicates that no response was received within 30 minutes. This includes the pre-processing time, which is the main factor for HMM and BP (building reachability graphs), and algorithm execution time, which is the main factor for OCC. In the worst case, for dataset M6, the trie generation took 949 ms and algorithm execution total time was 842 seconds for C-3PA.

The memory consumption of C-3PA is shown in Figure 6. The memory consumed per event does not increase as the stream progresses, as indicated by the red line. The total memory consumption does increase, as an increasing number of cases are kept in memory. In the current implementation, the user can define how many individual cases can be stored in the memory before the case together with its states is released. In general, the approach is memory efficient while permitting either a smaller or larger memory configuration depending on the organizational needs.

5.4 Discussion and limitations

The C-3PA is a conformance checking algorithm that outputs prefix-alignments, can handle warm-starting scenarios and presents a confidence level of the prefix-alignment. The results indicate that the algorithm is suitable for real-life situ-

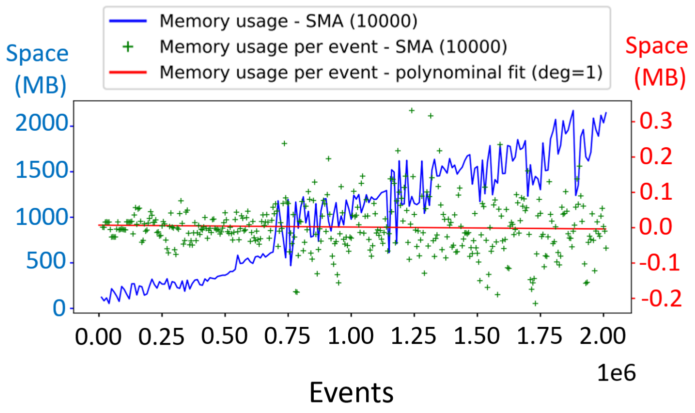


Fig. 6: Memory consumption across 2 million events.

ations, answering RQ2, handling fast-paced event streams and correlating well with optimal prefix-alignments.

One of the limitations of the algorithm comes from the use of the trie as the underlying process model — a trie may not be ideally suited for large models with a lot of concurrent behavior and several loop cycles. Also, for handling warm-starting, a grace period may still be required as currently, the algorithm would attempt to warm-start infinitely, which is not practical. Finally, in terms of confidence, further activities within the case could still theoretically occur, i.e., the quantification of confidence may be misleading in some instances. Ultimately, based on the scope of this paper they do not undermine the results achieved.

6 Conclusion

This paper introduced a novel approximate algorithm for streaming conformance checking. It is knowingly the first algorithm that fuses together the representability of prefix-alignments, allows for warm-starting scenarios, and is able to quantify the confidence of a prefix-alignment with regard to the conclusion of the trace. Extensive empirical testing was conducted to show the algorithm's ability to handle warm-start scenarios, show its correlation to existing streaming conformance checking methods, and to stress test the algorithm under latency and memory constraints. Future research directions would be to investigate whether confidence could also be integrated into the cost function, for example by utilizing a known stochastic distribution of branches in the trie. Furthermore, a more extensive study could be done in terms of datasets and trie construction, in order to achieve equality with optimal prefix-alignments.

Acknowledgement

This work was supported by the European Social Fund via "ICT programme" measure, the European Regional Development Fund, and the programme Mobilitas Pluss (2014-2020.4.01.16-0024).

References

1. van der Aalst, W.M.: Process mining: a 360 degree overview. In: Process Mining Handbook, pp. 3–34. Springer (2022)
2. Adriansyah, A.: Aligning observed and modeled behavior (2014)
3. Adriansyah, A., Van Dongen, B.F., Zannone, N.: Controlling break-the-glass through alignment. In: 2013 International Conference on Social Computing. pp. 606–611. IEEE (2013)
4. Awad, A., Raun, K., Weidlich, M.: Efficient approximate conformance checking using trie data structures. In: 2021 3rd International Conference on Process Mining (ICPM). pp. 1–8. IEEE (2021)
5. Burattin, A.: Streaming process mining. pp. 349–372. Springer (2022)
6. Burattin, A., Carmona, J.: A framework for online conformance checking. In: International Conference on Business Process Management. pp. 165–177. Springer (2017)
7. Burattin, A., Zelst, S.J.v., Armas-Cervantes, A., Dongen, B.F.v., Carmona, J.: Online conformance checking using behavioural patterns. In: International Conference on Business Process Management. pp. 250–267. Springer (2018)
8. Carmona, J., van Dongen, B., Weidlich, M.: Conformance checking: foundations, milestones and challenges. In: Process Mining Handbook, pp. 155–190. Springer (2022)
9. Kipping, G., Djurica, D., Franzoi, S., Grisold, T., Marcus, L., Schmid, S., Brocke, J.v., Mendling, J., Röglinger, M.: How to leverage process mining in organizations-towards process mining capabilities. In: International Conference on Business Process Management. pp. 40–46. Springer (2022)
10. Lee, W.L.J., Burattin, A., Munoz-Gama, J., Sepúlveda, M.: Orientation and conformance: A hmm-based approach to online conformance checking. Information Systems **102**, 101674 (2021)
11. Leemans, S.J., Fahland, D., Van Der Aalst, W.M.: Discovering block-structured process models from event logs containing infrequent behaviour. In: International conference on business process management. pp. 66–78. Springer (2013)
12. Raun, K., Awad, A.: I will survive: An online conformance checking algorithm using decay time. arXiv preprint (2022)
13. Rozinat, A.: Process mining: conformance and extension (2010)
14. Schuster, D., Zelst, S.J.v.: Online process monitoring using incremental state-space expansion: an exact algorithm. In: International Conference on Business Process Management. pp. 147–164. Springer (2020)
15. Vanden Broucke, S., De Weerdt, J., Vanthienen, J., Baesens, B.: An improved process event log artificial negative event generator. Available at SSRN 2165204 (2012)
16. Zaman, R., Hassani, M., Van Dongen, B.F.: Efficient memory utilization in conformance checking of process event streams. In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. pp. 437–440 (2022)
17. van Zelst, S.J., Bolt, A., Hassani, M., van Dongen, B.F., van der Aalst, W.M.: Online conformance checking: relating event streams to process models using prefix-alignments. International Journal of Data Science and Analytics **8**(3), 269–284 (2019)