STARDUST: a novel process mining approach to discover evolving models from trace streams

Vincenzo Pasquadibisceglie, and Annalisa Appice, and Giovanna Castellano, *Senior Member, IEEE* and Nicola Fiorentino and Donato Malerba, *Member, IEEE*

Abstract—In this paper we introduce STARDUST (event STream Analysis for pRocess Discovery Using Sampling sTragies), a process discovery approach that analyses a trace stream, in order to discover a process model that may change over time. The basic idea is to adopt a sampling technique to select the most representative trace variants to be considered for the process discovery, then to alert a concept drift as the trace variants to be sampled change over time and, finally, to trigger the discovery of a new process model as a drift is alerted. We formulate the proposed approach under the assumption that the trace distribution commonly follows the Pareto's principle (i.e., a few trace variants covers the majority of cases) which is commonly satisfied in several business processes. Experimental results on various benchmark event logs handled as streams show the effectiveness of the proposed approach also compared to a state-of-the- art concept drift detection approach.

Index Terms—Process discovery, Stream data mining, Concept drift, Sampling, Pareto's principle

1 Introduction

Many of today's process mining tools are predominantly based on the assumption that, when discovering a process model from event logs, the process is in steady state, i.e., the process at the beginning of the recorded period is the same as the process at the end of the recorded period. Conversely, real business processes are commonly characterized by complexity, variability, and lack of steady-state. Due to changing conditions, a business process may evolve and increase its variability during time. Hence, a business process model is a complex object that produces continuous event data which may change over time. In other words it can be seen as a data source that produces a continuous stream of events that evolve in time. Given this dynamic definition of business process, stream mining-based process discovery approach can be successfully applied to track how event data vary over time and change the process model to possibly adapt it to the emergence of new data.

Traditional machine learning methods, working in static contexts, where data are immediately available and stored, can not be directly applied in the context of stream mining, where data arrive as a continuous stream and can be volatile and transient [1]. Indeed, in a streaming scenario the data arrive online and the system does not have control over the order in which the data arrive and when data will be processed. In addition, data streams can have potentially unlimited size and, once processed, data are no longer available. The processed data can only be retrieved if they are explicitly stored in a memory whose size is smaller than the entire data stream. Another key aspect of stream mining is the change in distribution over time, better known as *Concept Drift* [2]. While in the static scenario the underlying distribution of the data does not change over time, guaranteeing a consistency between data features and their labels, in the streaming scenario data features have a strong time dependency. Any process discovery model operating in the streaming context needs solutions to identify and handle concept drifts so as to keep the performance of the model stable, by updating it with the arrival of new data or by completely replacing it when a drift occurs.

Despite the intrinsic dynamic nature of business processes, most of the literature about process discovery has seen the proliferation of solutions that fall into a static context by discovering a process model from a finite amount of traces (sequences of activities executed in a process case) recorded in an event log. Only recent studies have begun to explore the importance of repeating process discovery over time to keep a process model constantly updated, in order to capture the evolving nature of the behaviour of business processes and maintain good performance over time (i.e., high conformance[1] of upcoming traces to discovered process models). In this paper we present a stream mining approach to perform the process discovery task in a dynamic setting, i.e., in trace streams that may change over time. The proposed approach, called STARDUST (event STream Analysis for pRocess Discovery Using Sampling sTragies) analyses a trace stream and identifies possible changes (concept drifts) in the sequence of activities observed in the recorded traces. As soon as a drift is alerted, the discovery of a new process model is triggered.

The main goal of this study is the definition of an online

• *V. Pasquadibisceglie, A. Appice, G. Castellano and D. Malerba are with the Dipartimento di Informatica, Università degli Studi Aldo Moro di Bari, via Orabona 4 - 70125 Bari, Italy and also with the Consorzio Interuniversitario Nazionale per l'Informatica - CINI, via Orabona 4 - 70125 Bari, Italy. N. Fiorentino is with the Dipartimento di Informatica, Università degli Studi Aldo Moro di Bari, via Orabona 4 - 70125 Bari, Italy E-mail: vincenzo.pasquadibisceglie@uniba.it, annalisa.appice@uniba.it, giovanna.castellano@uniba.it, n.fiorentino3@studenti.uniba.it, donato.malerba@uniba.it*

1. Conformance checking relates the sequence of activities in a trace to the control-flow of activities in the process model and compares both to indicate where a trace and a model agree or disagree [3].

process discovery approach to update a process model over time, in order to improve conformance of upcoming traces to the discovered process model. Hence concept drift detection is explored as a means to understand when a process model needs to be updated so as to accurately conform to the control flow of activities of the upcoming traces.

The approach described in this study is formulated for business processes whose trace distribution roughly follows the Pareto's principle i.e., a small fraction of trace variants explains most of the traces observed. Actually, most business process traces recorded in event logs follow such a Pareto's distribution [3], also referred to as the "80/20 rule" (i.e., around 20% of all trace variants cover around 80% of all observed cases), although the values 80 and 20 may be arbitrary. For example, in the event log "Reference Alignment" published in the BPIC 2018 Challenge [4], 80% of traces recorded in the event log are covered by about 1% of distinct trace variants. Foundations of process discovery literature [3] account for the Pareto's distribution of event logs, by advocating the importance of creating process models based on the prevalent behaviors. Various activity-based filtering approaches have been integrated in process discovery algorithms to simplify activity traces and exclude exceptional trace behaviours that may proliferate if the distribution of traces over variants does not follow a Pareto's distribution [3]. On the other hand, sampling approaches are commonly used as a preprocessing step to identify important traces to be used for the process discovery [5].

Based upon considerations on the expected Pareto's distribution of traces recorded executing a business process, STARDUST adopts a frequency-based sampling technique to select the most representative (i.e., frequent) trace variants to be considered for the process discovery. It alerts a concept drift when selected variants change over time and updates the process model accordingly. In this way, STARDUST keeps a process model that changes over time preserving the ability to accurately describe the upcoming traces produced according to the drifted behaviours. Various approaches (e.g., random, frequency, length, similarity, structure, cluster-based) are formulated in process mining to identify representative traces for process discovery (see [5] for a recent survey). In this study, we chose to adopt frequency-based sampling leveraging results in [5], which show the higher conformance of process models discovered from traces sampled with frequency-based sampling in event logs with Pareto's distribution. In addition, [6] has recently shown that the process discovery performed with traces selected with frequency-based sampling outperforms the process discovery with traces generated with an abstraction-based approach in event logs following Pareto's distribution of traces. In this study, experimental results on various benchmark event logs handled as streams show the effectiveness of the proposed approach also compared to a state-of-the- art concept drift detection approach.

The paper is organized as follows. Section 2 overviews recent advances of literature in the stream learning of business processes. Section 3 reports preliminary concepts, while Section 4 describes the proposed STARDUST approach. In Section 5 we present the benchmark data collections considered for the empirical evaluation, describe the experimental setting and discuss the relevant results. Finally, Section 6 draws conclusions and sketches the future work.

2 Related work

The topic of concept drift has been widely investigated in the field of stream data mining [7], so that process mining research may, in principle, take advantage of amazing achievements in stream data mining to handle concept drifts in event data. However, the complexity of business processes working in real environments demands for specific methods to deal with concept drifts in process mining [8].

The topic of detecting concept drifts in event data was originally investigated in [8], where an event log is transformed into a stream of traces ordered on their arrival time, and hypothesis tests are performed on sliding windows to identify if a significant difference has occurred between a feature set extracted on consecutive windows. This study introduces a feature extraction step, thus the effectiveness of the concept drift detector may depend on the quality of the feature extraction. In addition, this study performs an offline analysis that needs a complete event log for consumption and only supports drift detection after the business process has been executed. However, the investigation of the use of hypothesis tests for concept drift detection is continued in [9], where tests are performed over distributions of behavioral relations between events observed in two adjacent sliding windows with adaptive size.

Inspired by these initial studies, a few recent studies have continued the exploration of concept drift detection methods operating in the process mining field [10]. In particular, the cluster analysis has recently emerged as a prominent approach for concept drift detection in event data. In [2], clusters discovered on the transformed feature space of an event stream are tracked over time to detect concept drifts occurring as anomalous cases. In [11], case distances are calculated by comparing streamed cases to a graph global model that represents the current state of the process. These distances are subsequently processed through clustering, while the emergence of new clusters is alerted as the occurrence of a concept drift. In [12], concepts of the theory of regions and state similarity are adopted to perform online conformance checking and quantify the non-conforming behaviour of running cases with a constant time complexity per analysed event. In [13], Sliding Window, Lossy Counting and Lossy Counting with Budget are investigated to cope with concept drift and realize online the discovery of a declarative process model, i.e., a set of constraints that should hold in conjunction during the process execution. In addition, the performance of several approaches (.g., hypothesis test-based, clustering-based) formulated for concept drift detection in process mining is experimentally compared in [14], while the root causes of concept drifts are initially investigated in [15], in order to react to change or anticipate future change.

Previous studies reported above mainly focus on detecting and visualizing concept drift points to perform conformance checking. The work in [13] is similar to the method proposed in our study, as it couples concept drift detection to declarative process model discovery. Our study also couples concept drift detection to process model discovery, but we focus on procedural process model discovery in place of

declarative process model discovery. We highlight that the attention of the current study to the discovery procedural process models is validated by the absence of a conclusive assessment of the superiority of either procedural or declarative process models [16]. Both these modelling languages continue engaging equal interest in process mining.

On the other hand, the discovery of procedural process models with online data has been recently investigated in [17]. However, this previous study mainly focuses on incrementally handling large amounts of data using finite memory within several procedural process discovery techniques without explicitly alerting concept drift events. In fact, the proposed architecture continuously updates a procedural process model as new events are recorded. In principle, concept drifts may be possibly recognized a-posteriori in the root-cause analysis of the changes observed in the process models updated as new events are processed. Our study works in a different perspective as it explicitly detects concept drifts, in order to restart the process model discovery in correspondence of the detected drifts only. In addition, we introduce a concept drift detector that is different from prior methods for concept drift detection based on either clustering or hypothesis tests. Our proposed method checks significant changes in the frequency of distinct behaviours observed in event data and identifies a few behaviours that gain relevance for the process discovery after a concept drift.

To complete this overview, we note that concept drift detection starts being a crucial topic also in predictive process monitoring. Despite that the state-of-the-art process monitoring approaches [18], [19], [20], [21] first construct a predictive model based on past process cases, and then use it to predict the unfolding (e.g., next activity, case outcome, completion time) of running cases, without the possibility of updating it with new cases when they complete their execution, this can make predictive process monitoring too rigid to deal with the variability of real processes that continuously evolve or exhibit new variant behaviors over time. In [22], an incremental learning strategy is explored to update a predictive process model whenever new cases become available so that the predictive model evolves over time to fit current circumstances. In [23] various incremental learning strategies (based on re-training or fine-tuning with full data or windowed data collected starting from the last drift point up to the current drift point) are analysed to update predictive process models trained as simple neural networks or dynamic Bayesian networks. In this study, concept drifts are detected with the approach based on hypothesis tests described in [8]. Similarly to our method, these studies do not limit to detect and visualize concept drifts, but they try to bridge the gap between concept drift detection and online discovery. However, the focus of these previous studies is on handling concept drifts in discovering models for predictive process monitoring, while the focus of this study is on discovering procedural process models.

3 PRELIMINARIES

In this section we introduce preliminary concepts referred to event streams and process discovery.

3.1 Event and trace streams

Let $\mathcal{A}$ be the set of all activity names, $\mathcal{C}$ be the set of all case identifiers and $\mathcal{T}$ be the set of all timestamps. Let $\perp$ denote the name of the completion activity (i.e., the activity that denotes the completion of a business case execution).

Definition 1 (Event). An event e is a triple $e = (c, a, t) \in \mathcal{C} \times \mathcal{A} \times \mathcal{T}$ that represents the occurrence of the activity a in the case c at timestamp t.

Let $\mathcal{E} = \mathcal{C} \times \mathcal{A} \times \mathcal{T}$ be the event universe. To identify case, activity and timestamp components of an event $e = (c, a, t) \in \mathcal{E}$, we introduce the functions: $\pi_{case}(e) = c$, $\pi_{activity}(e) = a$ and $\pi_{time}(e) = t$, respectively. An event e with $\pi_{case}(e) = c$ and $\pi_{activity}(e) = \perp$ denotes the completion of a case c. [2]

Definition 2 (Event stream). An event stream Σ is an unbounded sequence $\Sigma = e_1, e_2, \ldots$ of events defined on the event universe $\mathcal{E}$, such that $e_i \in \mathcal{E}$ and $\pi_{time}(e_i) \leq \pi_{time}(e_{i+1})$ for each $i \in \mathbb{N}^+$

By resorting to a case perspective, the case identifier can be used to group events already recorded in Σ into finite event sequences sorted by timestamps.

Definition 3 (Case). A case c is an event sequence $c = \langle e_1, e_2, \ldots, e_n \rangle$, such that $e_i \in \Sigma$ and $\pi_{case}(e_i) = c$ for each $i = 1, 2, \ldots, n$, and $\pi_{time}(e_i) \leq \pi_{time}(e_{i+1})$ for each $i = 1, 2, \ldots, n-1$.

Let $c(i)$ be the i-th event of a case c and $|c|$ be the number of events recorded in c. The case c is a full case if $\pi_{activity}(c(|c|)) = \perp$; a running case otherwise.

For simplicity, in the remaining of this paper, we consider a case through its activity trace.

Definition 4 (Activity trace). The activity trace σ_c of a case c is the activity sequence $\sigma_c = \langle a_1, \ldots, a_n \rangle$, such that $e_i = c(i)$ and $\pi_{activity}(e_i) = a_i$ for each $i = 1, 2, \ldots, n$ with $n = |c|$.

A trace stream is an unbounded sequence of activity traces associated with cases fully recorded in an event stream and sorted by the completion timestamp of cases.

Definition 5 (Trace stream). A trace stream Σ^* is a sequence of activity traces associated with full cases recorded in an event stream Σ and sorted by the timestamp of the completion event of each case, that is, $\Sigma^* = \sigma_1, \sigma_2, \ldots$, where, for each $i \in \mathbb{N}^+$, σ_i denotes the activity trace of the case c_i, $\pi_{activity}(c_i, |c_i|) = \perp$ and $\pi_{time}(c_i, |c_i|) \leq \pi_{time}(c_{i+1}, |c_{i+1}|)$.

Figure 1 shows an example of a trace stream Σ^* associated with an event stream Σ. Notice that Σ^* is populated with the activity traces σ_1, σ_2, σ_3 and σ_4 that correspond to the full cases 1, 2, 3 and 4, respectively. The trace σ_5 will be added to Σ^* as the completion event of the running case 5 will be recorded in Σ. Hence, as shown in this example, an event stream Σ can be mapped into a trace stream Σ^* and a list $\mathcal{R}$ of activity traces associated with running cases

2. The completion of a case is commonly declared in a multitude of business processes, e.g., ticket resolution in help desk processes. In any case, we are aware that business processes where the completion of a case may not be explicitly declared are not handled in this study and require further investigation in future works.

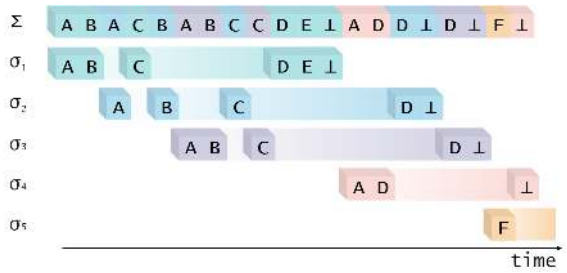


Figure 1: An example of trace stream

Table 1: An example of a multiset of activity traces, where each superscript number denotes the number of times (frequency) the distinct trace variant appears in the trace stream. The trace stream is shown in Figure 1.

timestamp	$\mathcal{B}_{\Sigma^*}$
2021-09-20 11:35:01	$\{\langle A,B,C,D,E\rangle^1\}$
2021-09-21 10:10:11	$\{\langle A,B,C,D,E\rangle^1, \langle A,B,C,D\rangle^1\}$
2021-09-21 11:30:43	$\{\langle A,B,C,D,E\rangle^1, \langle A,B,C,D\rangle^2\}$
2021-09-21 11:32:45	$\{\langle A,B,C,D,E\rangle^1, \langle A,B,C,D\rangle^2, \langle A,D\rangle^1\}$

whose completion event has not been recorded in Σ. The trace stream Σ^* is a multiset of full activity traces $\mathcal{B}_{\Sigma^*}$, since multiple traces recorded in Σ^* may share the same activity sequences. In Figure 1, both σ_2 and σ_3 share the same activity sequence $\langle A,B,C,D\rangle$. Distinct activity sequences are called trace variants. The number of times a distinct trace variant σ appears into Σ^* represents the frequency with which σ is repeated into $\mathcal{B}_{\Sigma^*}$. For example, the trace variant $\langle A,B,C,D\rangle$ occurs twice in Figure 1.

Definition 6 (Behavior of a multiset). The behavior of $\mathcal{B}_{\Sigma^*}$ is the set of distinct trace variants recorded in $\mathcal{B}_{\Sigma^*}$.

The function $\beta(\mathcal{B}_{\Sigma^*})$ selects the behavior of $\mathcal{B}_{\Sigma^*}$.

Due to the streaming setting, $\mathcal{B}_{\Sigma^*}$ changes each time a completion event is recorded in Σ. Let σ be the activity trace whose case has been just completed in Σ, σ is removed from $\mathcal{R}$ and added to $\mathcal{B}_{\Sigma^*}$. We distinguish two cases. If σ represents a trace variant never seen before in Σ^* (i.e., $\sigma \notin \beta(\mathcal{B}_{\Sigma^*})$) then a new set is put into $\mathcal{B}_{\Sigma^*}$ to represent σ with frequency equal to 1. Otherwise the frequency of σ is incremented by 1 into $\mathcal{B}_{\Sigma^*}$. Table 1 shows an example of how $\mathcal{B}_{\Sigma^*}$ may change over time as new traces are completed in Σ^*. At the timestamp 2021-09-21 11:32:45, $\mathcal{B}_{\Sigma^*}$ is a multiset that comprises: $\langle A,B,C,D,E\rangle$ with frequency equal to 1, $\langle A,B,C,D\rangle$ with frequency equal to 2, and $\langle A,D\rangle$ with frequency equal to 1. $\beta(\mathcal{B}_{\Sigma^*})$ is the set of the three distinct trace variants: $\langle A,B,C,D,E\rangle$, $\langle A,B,C,D\rangle$ and $\langle A,D\rangle$.

Notice that the size of $\mathcal{R}$ is expected to fluctuate over time since an activity trace stays in $\mathcal{R}$ as long as its case is running, and it is removed from $\mathcal{R}$ as the completion of its case is recorded in Σ. Instead, the size of $\mathcal{B}_{\Sigma^*}$ is expected to grow over time. So, a pruning strategy can be used to limit the size of $\mathcal{B}_{\Sigma^*}$ over time. To this aim, we introduce a forgetting mechanism that prunes trace variants that are unobserved for long time in Σ^* (inactive trace variants).

Definition 7 (Inactive trace variant). Let w be the forgetting size, a trace variant σ is marked as inactive in Σ^* if it has not occurred in the w newest traces observed in Σ^*.

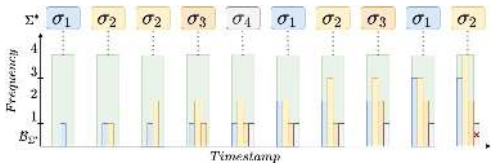


Figure 2: Example of forgetting mechanism with $w = 5$

Figure 2 shows an example of the forgetting-based pruning strategy. It removes σ_4 from $\mathcal{B}_{\Sigma^*}$ as it disappears for $w = 5$ consecutive activity traces recorded in Σ^*. Using this pruning strategy we record an approximation of Σ^* in $\mathcal{B}_{\Sigma^*}$, which forgets behaviours that have disappeared for a long time. This bases on the idea that trace variants becoming inactive can disappear from the online process model.

3.2 Process discovery

Starting from a multiset of activity traces, we can perform a process discovery task, in order to discover a process model, that is, a generative representation of the activity traces recorded in the processed multiset. As described in [14], this representation can generate multiple activity traces based on the optional paths it describes. Specifically, we can refer to the process model behavior as the set of trace variants that can be generated from the process model. So, a process discovery algorithm commonly constructs a process model from a multiset of activity traces.

Definition 8 (Process model discovery). Let us consider a multiset of activity traces $\mathcal{B}_{\Sigma^*}$, a generative representation M of $\mathcal{B}_{\Sigma^*}$ and a set of activity traces $\mathcal{B}_M$ that can be generated by the execution of the model M, a process discovery algorithm is a function $\delta\colon \mathcal{B}_{\Sigma^*} \mapsto \mathcal{B}_M$.

Various robust process discovery algorithms have been formulated in the recent literature to generate Petri nets (.g., Hybrid ILP Miner [24], Inductive Miner [25], Split Miner [26]). They focus on the discovery of procedural process models, that is, the process model perspective of this study. The ability of these algorithms to discover effective process models is commonly measured in terms of ability to (1) parse the traces in the multiset of real activity traces, (2) not parse other traces and (3) be as simple as possible. The first property is called *fitness*, the second *precision* and the last one *simplicity*. Most of these algorithms may suffer for the presence of many infrequent behaviors [27] that can lead to spaghetti-like models. A straightforward solution to overcome this problem is to down-size the amount of event data to be processed with a process discovery algorithm by possibly removing infrequent behaviors. To this aim, various filtering [28], [29] and sampling [5], [30] methods have been proposed in the recent literature as a preprocessing step for process discovery. In particular, as business processes often produce activity traces that follow the Pareto's distribution traces [3], frequency-based sampling is commonly used in several real processes to filter-out infrequent trace variants.

Definition 9 (Frequency-based sampling). Let $\mathcal{B}_{\Sigma^*}$ be a multiset of activity traces and μ be the sampling rate ($0 < \mu < 100\%$). The frequency-based sampling algorithm sorts the trace variants of $\mathcal{B}_{\Sigma^*}$ in a descending

order by the frequency of variants and generates $\mathcal{B}'_{\Sigma^*}$ that comprises the smallest multiset of the top-frequent trace variants of $\mathcal{B}_{\Sigma^*}$ so that $\frac{cardinality(\mathcal{B}'_{\Sigma^*})}{cardinality(\mathcal{B}_{\Sigma^*})} \geq \mu$.

The function *cardinality* counts the total number of trace occurrences recorded in a multiset of activity traces. For example, let us consider a multiset $\mathcal{B}_{\Sigma^*} = \{\langle A,B,C,D,E\rangle^7, \langle A,B,C,D\rangle^6, \langle A,D\rangle^3\}$ with cardinality equal to 16. Let us consider $\mu = 80\%$, the frequency-sampling algorithm generates the multiset $\mathcal{B}'_{\Sigma^*} = \{\langle A,B,C,D,E\rangle^7, \langle A,B,C,D\rangle^6\}$ that contains the two top-frequent trace variant of $\mathcal{B}_{\Sigma^*}$ so that $\frac{cardinality(\{\langle A,B,C,D,E\rangle^7, \langle A,B,C,D\rangle^6, \langle A,D\rangle^3\})}{cardinality(\{\langle A,B,C,D,E\rangle^7, \langle A,B,C,D\rangle^6\})}$=81.25. Notably [5] and [6] have recently proved that the use of the frequency-based sampling algorithm allows the discovery of simpler process models with higher conformance to traces compared to sampling techniques based on alternative extractive criteria (e.g., trace length, similarity or structure) or generative criteria (e.g., trace prototype abstraction).

3.3 Process discovery on streams

In principle, process discovery in a trace stream must be restarted as a new activity trace is recorded in the stream. Online process discovery studies how to record an event stream in a data synopsis that can be continuously queried to extract the process model at each timestamp. In [17], no concept drift detector has been integrated, so the query should be repeatedly processed each time a new case has been completed in the stream spending, in several cases, time to generate an unchanged process model. In this study, we adopt a different perspective that founds on the consideration that new cases do not necessarily lead to the discovery of new process models. So, well known process discovery algorithms, robustly designed for a static setting may be easily reused in the streaming setting by restarting them only in correspondence of concept drift events that potentially lead to the discovery of new process models. This perspective inspires the idea of exploring the feasibility of existing process discovery algorithms in a streaming setting by coupling them to an effective concept drift detection method. This detector should identify any *significant change* in the behaviour of the multiset of activity traces recorded with a trace stream and restart the process model discovery in correspondence of these drifts only. Specifically, we detect a concept drift each time a change occurs in the behaviour of the frequency-based sample extracted from the multiset of activity traces associated with the trace stream.

Definition 10 (Behavior concept drift). Let $\mathcal{B}_{\Sigma^*(t_i)}$ and $\mathcal{B}_{\Sigma^*(t_{i+1})}$ be the two multisets of activity traces recorded for the trace stream Σ^* at the timestamps t_i and t_{i+1}, respectively. If $\beta(\mathcal{B}'_{\Sigma^*(t_{i+1})}) \neq \beta(\mathcal{B}'_{\Sigma^*(t_i)})$, then a concept drift is alerted in Σ^* between t_i and t_{i+1}.

This definition of a behaviour concept drift bases on the common use of the frequency-based sampling as a preprocessing step to discard infrequent behaviors reducing the risk of discovering spaghetti-like process models especially in business processes whose traces follow the Pareto's distribution. We alert a concept drift as behaviors that were initially infrequent emerge as frequent or, vice-versa, behaviors that were frequent in the past tend to disappear in the stream. Table 2 shows an example of a behavior concept drift detected between t_i and t_{i+1} due to the emerging of an activity trace variant, $\langle A,D\rangle$ as a new frequent behavior.

Figure 3: Sudden and gradual behaviour concept drifts. The green line and the yellow line denote the beginning of the drift and the time the drift is detected with STARDUST. The sudden drift is detected with a delay of 42 traces, while the gradual drift is detected with a delay of 101 traces.

Behaviour concept drifts may manifest as sudden drifts (when behaviours change abruptly) or gradual drifts (when behaviours change gradually as their presence decreases initially in delimited contexts to finally apply to the entire stream). Figure 3 shows an example of these two types of behaviour concept drifts that can be both detected by STARDUST, although the detection of a gradual drift may exhibit a higher delay than the detection of an abrupt drift. We point out that in our approach behaviour concept drifts are detected to trigger the process model discovery update. Indeed, the main goal is keeping an updated online process model by improving the conformance of upcoming traces to the updated process model. We do not handle the task of recognizing the drift type, that would require specific techniques for each drift type [31]. Finally, a behaviour concept drift may also be recurrent (when behaviour changes repeat with recurring incidence over time). In this study, a recurrent drift may be recognized a-posteriori by comparing all the process models updated in correspondence of concept drifts. Process models that repeat seasonally may be recognized as the consequence of recurrent drifts.

Final concerns regard the evaluation of the effectiveness of a process model discovered over a stream. Prior online process discovery studies measure the effectiveness of the final process model with respect to all the processed activity traces since the final goal is to discover a process model with large amount of event data. In this study, we adopt the Prequential Evaluation schema [32] that is commonly adopted to assess accuracy of online classification models for data streams. According to this schema, the accuracy of a model trained on past data is measured on the upcoming data. Therefore we evaluate how the process model currently discovered along the trace stream satisfies the conformance of upcoming traces in the same stream.

4 Proposed Approach

The pseudo-code of STARDUST is reported in Algorithm 1. The algorithm consumes an event stream Σ to keep online an updated process model M and measure the conformance $Eval$ of each activity trace σ to M as the completion event of the full case associated with σ is recorded in Σ. The algorithm is composed of three steps: (1) **Initialization** that discovers an initial process model; (2) **Process model**

Table 2: An example of concept drift occurring between t_i and t_{i+1}

timestamp	$\mathcal{B}_{\Sigma^*}$	$\mathcal{B}'_{\Sigma^*}$ with $\mu = 0.8$	$\beta(\mathcal{B}'_{\Sigma^*})$
t_{i-1}	$\{\langle A,B,C,D,E\rangle^7, \langle A,B,C,D\rangle^6, \langle A,D\rangle^2\}$	$\{\langle A,B,C,D,E\rangle^7, \langle A,B,C,D\rangle^6\}$	$\{\langle A,B,C,D,E\rangle, \langle A,B,C,D\rangle\}$
t_i	$\{\langle A,B,C,D,E\rangle^7, \langle A,B,C,D\rangle^6, \langle A,D\rangle^3\}$	$\{\langle A,B,C,D,E\rangle^7, \langle A,B,C,D\rangle^6\}$	$\{\langle A,B,C,D,E\rangle, \langle A,B,C,D\rangle\}$
t_{i+1}	$\{\langle A,B,C,D,E\rangle^7, \langle A,B,C,D\rangle^6, \langle A,D\rangle^4\}$	$\{\langle A,B,C,D,E\rangle^7, \langle A,B,C,D\rangle^6, \langle A,D\rangle^4\}$	$\{\langle A,B,C,D,E\rangle, \langle A,B,C,D\rangle, \langle \mathbf{A},\mathbf{D}\rangle\}$

discovery that monitors the stream to re-train the process model every time a concept drift is detected; (3) **Evaluation** that measures the conformance of the current process model at the completion of each new activity trace. Both step (1) and (2) wrap a process discovery algorithm δ and integrate the frequency-based sampling algorithm with sampling rate μ to extract the top-frequent trace variants. During both steps, the event stream Σ is processed at the trace level and recorded in two data synopses: $\mathcal{R}$ (i.e., the list of activity traces of current running cases) and $\mathcal{B}_{\Sigma^*}$ (i.e., the multi-set of active trace variants of full cases). These synopses are updated each time a new event is recorded in Σ.

4.1 Data synopses

The number of activity traces of full cases already observed in the stream is registered into the trace enumerator i.

The list of activity traces associated with observed running cases is recorded into a data synopsis $\mathcal{R}$ composed of an header table $\mathcal{H}$ and a prefix tree $\mathcal{T}$. This synopsis is inspired to the data structure introduced in [33]. $\mathcal{H}$ is a Hash table, where each cell represents the identifier c of a running case. $\mathcal{H}[c]$ contains the link to a node of $\mathcal{T}$ so that the tree path from the root to the pointed node records the activity trace of c. $\mathcal{T}$ is a tree that consists of one root labeled as *null* (denoted as $root$) and a set of prefix activity sub-trees as the children of $root$. Each node n in the activity prefix sub-tree consists of three fields: *activity-name* that contains the activity registered in the event that n represents, *father-link* that contains the link to the father node of n (*null* in $root$) and *child-link* that collects the links to the child nodes of n sorted by *activity-name* (*null* in leaf nodes). The size of $\mathcal{T}$ is $\sum_{j=1}^{m}(c+\uparrow N) + \sum_{j=1}^{\#N} a + \sum_{j=1}^{\#N-1}(2*\uparrow N)$, where m is the number of cases registered in $\mathcal{H}$, c is the memory used to record a case identifier, $\#N$ is the number of nodes in $\mathcal{T}$, a is the memory used to register an activity in a node, $\uparrow N$ is the memory used to register the link (pointer) to a node.

The multi-set of active trace variants associated with observed full cases is recorded into a data synopsis $\mathcal{B}_{\Sigma^*}$ that is a Hash table, where each cell registers a trace variant σ. $\mathcal{B}_{\Sigma^*}[\sigma]$ denotes the tabular entry which stores the couple $(frequency, enum)$ with $frequency$ that represents the frequency of σ in the stream and $enum$ the value of trace enumerator index at which the last occurrence of σ has been observed. The one-to-one association between the trace variant (key) and the table cell is made by means of a Hash function. The size of $\mathcal{B}_{\Sigma^*}$ is $\sum_{j=1}^{b}(l_j a + f + t)$, where b is the number of trace variants, l_j is the length of the j trace variant recorded in $\mathcal{B}_{\Sigma^*}$, a is the memory used to register an activity, f is the memory used to register the frequency of a trace variant and t is the memory used to register the trace enumerator value associated to the trace variant.

Initially, $i = 0$, $\mathcal{H}$ is an empty Hash table and $\mathcal{T}$ contains the root node $root$. As an event e with $\pi_{case}(e) = c$ and $\pi_{activity}(e) = a$ is observed in Σ, we distinguish three cases.

1) c is a new running case. It is registered in $\mathcal{H}$. If there exists a child of $root$, denoted as n, such that n.*activity-name*=a, the link to n is recorded into $\mathcal{H}[c]$. Otherwise a new child node, denoted as n, is added to $root$ in $\mathcal{T}$ with n.*activity-name*=a, n.*father-link*=$root$ and n.*child-link*=$null$. The link to n is added to $root$.*child-link*.
2) c exists in $\mathcal{H}$ and $a = \perp$. Let n be the node pointed by $\mathcal{H}[c]$, the activity trace σ of c is extracted traversing $\mathcal{T}$ from n to $root$. Notice that σ, that is read in the reverse order, is ready to be added to $\mathcal{B}_{\Sigma^*}$. The cell $\mathcal{H}[c]$ is dropped from $\mathcal{H}$. If n is a leaf and there is no further cell of $\mathcal{H}$ pointing to n, then n is dropped from $\mathcal{T}$. This node dropping operation is recursively applied to $\mathcal{T}$ by traversing bottom-up all ancestors of n in $\mathcal{T}$.
3) c exists in $\mathcal{H}$ and $a \neq \perp$. Let n be the node pointed by $\mathcal{H}[c]$. If there exists n' that is a child of n such that n'*activity-name*$= a$, the link to n' is recorded into $\mathcal{H}[c]$. Otherwise a new child n' of of n is added to $\mathcal{T}$ so that n'.*activity-name*=a, n'.*father-link*=n and n'.*children-link*=$null$, while the link to n' is added to n.*child-link*.

As the completion event of a case c is recorded in Σ, i is incremented by 1, the trace σ associated with c is removed from $\mathcal{R}$, and σ is added to $\mathcal{B}_{\Sigma^*}$. We distinguish two cases:

1) σ corresponds to a trace variant already registered in a cell of $\mathcal{B}_{\Sigma^*}$. In this case, $\mathcal{B}_{\Sigma^*}[\sigma].frequency$ is incremented by 1 and $\mathcal{B}_{\Sigma^*}[\sigma].enum$ is set equal to i
2) σ is a new trace variant. In this case, $\mathcal{B}_{\Sigma^*}[\sigma]$ is registered into $\mathcal{B}_{\Sigma^*}$ with $\mathcal{B}_{\Sigma^*}[\sigma].frequency = 1$ and $\mathcal{B}_{\Sigma^*}[\sigma].enum$ is set equal to i.

Finally, the pruning mechanism with forgetting size w is performed. So, trace variants with $\mathcal{B}_{\Sigma^*}[\sigma].enum + w \leq i$ are labeled as inactive and dropped from $\mathcal{B}_{\Sigma^*}$.

4.2 Initialization step

In the initialization step, STARDUST sets $i = 0$ (i.e., trace counter) and starts monitoring the event stream Σ to discover an initial process model M. When a new case c starts its execution, its activity trace is recorded in $\mathcal{R}$ where it stays until the completion event of the case has been recorded in Σ. When c has been completed, the trace σ associated with c feeds Σ^*, so i is incremented by one and σ is removed from $\mathcal{R}$ to be recorded into $\mathcal{B}_{\Sigma^*}$. As η activity traces have been recorded in $\mathcal{B}_{\Sigma^*}$ (i.e., $i = \eta$), the frequency-based sampling

Algorithm 1: STARDUST algorithm

Data: Σ (event stream), η(number of traces for the initialization), μ (sampling rate), δ (process discovery algorithm), w (forgetting size)
Result: M (evolving process model), $Eval$ (measurements of the conformance of streamed traces to the online process model)

begin
  /* Initialization */
  $i = 0$
  $\mathcal{R} = (\{\}, \{\text{root}\})$
  $\mathcal{B}_{\Sigma^*} = \{\}$
  for $i < \eta$ **do**
    e=observe(Σ)
    if $\pi_{activity}(e) = \perp$ **then**
      $i = i + 1$
      Let σ be the activity trace associate with case $c = \pi_{case}(e)$. Remove σ from $\mathcal{R}$
      Record σ in $\mathcal{B}_{\Sigma^*}$
    else
      Record e in $\mathcal{R}$
  $B = \beta(\mathcal{B}_{\Sigma^*})$
  $\mathcal{B}'_{\Sigma^*}$ =frequentSampling($\beta(\mathcal{B}_{\Sigma^*}), \mu$)
  M=processdiscovery($\mathcal{B}'_{\Sigma^*}, \delta$)
  /* Process discovery and evaluation */
  for *e=observe*(Σ) **do**
    if $\pi_{activity}(e) = \perp$ **then**
      $i = i + 1$
      Let σ be the activity trace associate with case $c = \pi_{case}(e)$. Remove σ from $\mathcal{R}$
      Add conformance(M, σ) to $Eval$
      Record σ in $\mathcal{B}_{\Sigma^*}$
      $\mathcal{B}'_{\Sigma^*}$ =frequentSampling($\beta(\mathcal{B}_{\Sigma^*}), \mu$)
      if $B \neq \beta(\mathcal{B}'_{\Sigma^*})$ **then**
        $B = \beta(\mathcal{B}'_{\Sigma^*})$
        M=processDiscovery($\mathcal{B}'_{\Sigma^*}, \delta$)
    else
      Record e in $\mathcal{R}$
  return $M, Eval$

algorithm is run with sampling rate μ, in order to extract $\mathcal{B}'_{\Sigma^*}$ from $\mathcal{B}_{\Sigma^*}$. Subsequently, STARDUST both runs δ to discover the process model M as a generative representation of $\mathcal{B}'_{\Sigma^*}$ and saves a copy of $B = \beta(\mathcal{B}'_{\Sigma^*})$. M will be used in the evaluation step, while B will be used in the behaviour concept drift detection of the process model discovery step.

4.3 Process model discovery step

As the initialization step has been completed, the process model discovery step continues monitoring the event stream Σ and triggering the behaviour concept drift detection and possibly the process discovery as each new incoming activity trace has been completed. This step is performed continuously to discover behavior concept drifts and re-run δ to train a new process model M as a concept drift is alerted on the monitored traces. Let σ be the new activity trace registered in $\mathcal{B}_{\Sigma^*}$. The frequency-based sampling algorithm is performed with sampling rate μ, in order to extract $\mathcal{B}'_{\Sigma^*}$ from $\mathcal{B}_{\Sigma^*}$. If $B \neq \beta(\mathcal{B}'_{\Sigma^*})$, then STARDUST updates $B = \beta(\mathcal{B}'_{\Sigma^*})$ and runs δ to discover a new process model M from the current $\mathcal{B}'_{\Sigma^*}$. Otherwise, the old M is kept without triggering any re-training step for the process discovery.

4.4 Evaluation step

The evaluation step is performed online in parallel to the process discovery step to monitor the conformance of upcoming traces to the online process model. Let σ be an activity trace recorded in $\mathcal{B}'_{\Sigma^*}$, the conformance of σ to the current process model M is evaluated in terms of Fmeasure (i.e., harmonic mean) of fitness and precision [3]. According to the foundations of conformance checking [3], fitness is derived based on the number of satisfied and violated process model rules. Precision is quantified by the number of possible continuations in the process model not observed in the trace.[3] Notice that the evaluation of the conformance of σ is done with the process model M that was kept (or discovered) after completing the process discovery step before the observation of σ.

5 Experiments

We evaluated the performance of STARDUST by performing experiments on seven event logs handled as streams.

5.1 Implementation details

STARDUST was implemented in Python 3.9 – 64 bit version. This implementation is available via the public GitHub repository.[4] We used three state-of-the-art process discovery algorithms: Hybrid ILP Miner [24] (ver 6-10.154) and Inductive Miner [25] imported from PM4PY, Split Miner [26] imported from Apromore[5]. We integrated the implementation of the Cost-based fitness [34] and the Align-ETConformance [35] from PM4PY. Both metrics are measured with alignments computed with the algorithm A^* [36]. In addition, we implemented the Extedend Cardoso Metrics module [37] to compute the extended Cardoso index that measures the complexity of a process model by its complex structures, i.e., Xor, Or and And components. The subprocess module[6] present in Python 3.6.9 was adopted to run the Java plug-in of Hybrid ILP Miner and Split Miner through the Python code by creating new processes. A pre-processing step was performed to simplify the self-loop events that may appear in an activity trace. A self-loop is a sub-sequence of a trace where an activity is repeated on n consecutive events with $n \geq 2$. A self-loop with n repetitions is called a n-repetition. We removed every n-repetition of the same event with a 2-repetition by performing an activity dropping operation. This simplification is introduced to reduce the risk that led to handle a huge number of similar trace variants showing the same loop repeated on a different number of times. This simplification of repetitions was implemented when an event was registered in the prefix tree data synopsis. A node is not added to the tree if it registers the same activity already registered in both its father and grand-father node.

5.2 Event logs and experimental setting

We processed trace streams generated from seven benchmark event logs provided by the 4TU Centre for Research.[7] These event logs record executions of business processes in healthcare, assistance, tourism and traffic management.

3. Any other conformance metric can be used in the evaluation step.
4. https://github.com/vinspdb/STARDUST
5. https://apromore.org/platform/tools/
6. https://docs.python.org/3/library/subprocess.html
7. https://data.4tu.nl/portal

Table 3: Event stream description: number of activities ($\#a$), events ($\#e$), traces ($\#t$), trace variants ($\#v$), trace variants (and percentage of trace variants) that cover the 80% of all the traces recorded in the event log ($\#t(80\%)$), average trace length ($avg.length(\sigma)$), event log size ($size(\Sigma)$ in KBytes with 2 bytes used to record an activity name and 4 bytes used to record a case identifier, stream duration in days ($time(\Sigma)$), average trace duration in days ($avg.time(\sigma)$)

event stream (Σ)	$\#a$	$\#\sigma$	$\#e$	$\#v$	$\#v$ (80%)	$avg.length(\sigma)$	$size(\Sigma)$	$time(\Sigma)$	$avg.time(\sigma)$
BPIC13Incidents (BPIC13I)	13	7554	65533	2278	770 (33.8%)	9	438.5	783	12.1
BPIC18ReferenceAlignment (BPIC18R)	6	43802	128554	515	5 (0.97%)	3	1034.1	909	78.5
BPIC20DomesticDeclarations (BPIC20D)	17	10500	56437	99	3 (3.03%)	5	401.6	889	11.5
BPIC20PrepaiedTravelCost (BPIC20P)	29	2099	18246	202	26 (12.8%)	9	122.1	772	36.8
BPIC20RequestForPayments (BPIC20R)	19	6886	36796	89	4 (4.49%)	5	262.1	941	12
Road	11	150370	561470	231	4 (1.73%)	4	4271.1	4917	3416
Hospital	18	100000	451359	1020	10 (0.98%)	5	3308.2	1131	130.9

Table 4: Experimental process discovery configurations. Dynamic configurations (d) are referred to STARDUST, while static configurations (s) are referred to the static counterpart of STARDUST. **fl** refers to running a process discovery algorithm by using the embedded filtering mechanism. **fr** refers to running a process discovery algorithm my accounting for the frequency of trace variants.

conf	fl	fr	mode	conf	fl	fr	mode
nfl_nfr_d	no	no	dynamic	nfl_nfr_s	no	no	static
nfl_ufr_d	no	yes	dynamic	nfl_ufr_s	no	yes	static
ufl_nfr_d	yes	no	dynamic	ufl_nfr_s	yes	no	static
ufl_ufr_d	yes	yes	dynamic	ufl_ufr_s	yes	yes	static

Table 3 reports the characteristics of these event logs. Unfortunately there is no description of concept drifts occurring in these benchmark real event logs. However, starting from [38], various studies (also discussed in Section 2) have repeatedly assessed the lack of steady-state in real business processes and the need of discovering evolving process models. The main goal of this experimentation is showing that the proposed approach can actually improve the conformance of upcoming traces to a process model by updating the model over time. This ability is explored orthogonally to both the time spent processing the stream and the memory used to keep a summary of the stream in memory during the processing time, in order to account for the data stream processing constraints.

For each event stream, we conducted experiments by using 10% of the total number of activity traces recorded in the original event log for the initialized step ($\eta = 10\%$). We set the sampling rate of the frequency-based sampling algorithm equal to 0.8 ($\mu = 0.8$). This sampling rate set-up was decided according to the Pareto's principle. In addition, we set the forgetting size equal to the size of the initialization window ($w = \eta$). We ran Hybrid ILP Miner (ILP), Inductive Miner (IMi) and Split Miner (SM) as process discovery algorithms. Each process discovery algorithm was run by either enabling the embedded filtering technique of the algorithm (ufl) or disabling filtering (nfl). Filtering techniques are commonly used in process discovery to drop infrequent activities, infrequent variants and infrequent arcs, in order to remove infrequent behaviours and discover simpler process models [3]. They are integrated in all the process discovery algorithms used in this study. In particular, IMi and SM integrate a filtering technique that removes infrequent nodes and edges of the created direct-follow-graphs. ILP integrates a sequence encoding filtering technique that filters exceptional behaviours within the integer linear programming formulation of the discovery process [39]. In this experimentation, filtering techniques were run with the default parameter set-up in the corresponding process discovery algorithms. So, the process discovery was completed by processing either the multi-set of sampled activity traces (repeated according to their frequency) (ufr) or the set of distinct sampled trace variants (considered without their frequency) (nfr). In ufr, the process discovery algorithm δ takes the multi-set $\mathcal{B}'_{\Sigma^*}$ as input for the process discovery. In nfr, δ takes the set $\beta(\mathcal{B}'_{\Sigma^*})$ as input for the process discovery. Examples of both $\mathcal{B}'_{\Sigma^*}$ and $\beta(\mathcal{B}'_{\Sigma^*})$ are shown in Table 2.

For all these configurations of the process discovery algorithm, we compared the performance of the dynamic setting (d) realized by STARDUST to the performance of the static counterpart (s) that keeps the process model discovered in the initialization step without updating it on concept drifts. A summary of the experimented configurations is reported in Table 4. In addition, we compared the performance of STARDUST to the performance of the related method detecting concept drifts with the cluster-based approach presented in [40].[8]

To measure the performance of discovered process models, we adopted the Prequential Evaluation strategy commonly used in stream data mining [32]. We measured the conformance of upcoming traces to the newer discovered process model. The conformance was measured in terms of Fmeasure of fitness and precision. This metric was computed on all the activity traces acquired as the initialization step was completed. The average metric measured on all the test activity traces is reported in this study. In addition, we measured the complexity of discovered process models in terms of Extended Cardoso index. We measured the memory used (in KBytes) to keep streamed data in memory during the online process. Finally, we measured the computation time spent in seconds to process the entire trace stream. The computation times were collected running the experiments on Intel(R) Core(TM) i7-9700 CPU, GeForce RTX 2080 GPU, 32GB Ram Memory, Windows 10 Home.

5.3 Results and discussion

5.3.1 Dynamic versus static setting analysis

We start this analysis comparing the performance of various configurations of STARDUST (see Table 4) to that of their static counterparts. This analysis is conducted to explore how the use of the behavior concept drift detection approach integrated in STARDUST may be an opportunity to

8. We used the implementation of the cluster-based approach available at https://github.com/gbrltv/cdesf2

Table 5: Performance of process models discovered by running ILP, IMi and SM in both STARDUST (d) and its static counterpart (s): number of drifts (#d) average Fmeasure (of fitness and precision) and total computation time (in seconds). The best results are in bold.

stream	conf	#d	Fmeasure			time (s)		
			ILP	**IMi**	**SM**	**ILP**	**IMi**	**SM**
BPIC13I	nfl_nfr_d	63	0.50	0.57	**0.58**	73	87	76
	nfl_ufr_d		0.50	0.57	**0.58**	83	88	80
	ufl_nfr_d		**0.52**	**0.58**	0.52	71	77	69
	ufl_ufr_d		**0.52**	**0.58**	0.52	62	96	73
	nfl_nfr_s	0	0.44	0.50	0.50	**46**	**50**	**49**
	nfl_ufr_s		0.44	0.50	0.50	**46**	**50**	**49**
	ufl_nfr_s		0.44	0.50	0.50	**46**	**50**	50
	ufl_ufr_s		0.44	0.50	0.50	**46**	**50**	**49**
BPIC18R	nfl_nfr_d	11	**0.96**	**0.96**	**0.96**	133	123	127
	nfl_ufr_d		**0.96**	**0.96**	**0.96**	136	124	128
	ufl_nfr_d		**0.96**	**0.96**	0.95	133	123	122
	ufl_ufr_d		0.94	**0.96**	**0.96**	132	130	126
	nfl_nfr_s	0	0.94	0.94	0.94	129	110	111
	nfl_ufr_s		0.94	0.94	0.94	129	**109**	111
	ufl_nfr_s		0.94	0.94	0.94	129	110	**109**
	ufl_ufr_s		0.94	0.94	0.94	128	**109**	110
BPIC20D	nfl_nfr_d	4	0.80	**0.83**	**0.83**	48	47	48
	nfl_ufr_d		0.80	**0.83**	**0.83**	49	47	48
	ufl_nfr_d		0.80	**0.83**	**0.83**	49	47	48
	ufl_ufr_d		**0.83**	**0.83**	**0.83**	47	47	48
	nfl_nfr_s	0	0.51	0.51	0.51	44	**43**	**44**
	nfl_ufr_s		0.51	0.51	0.51	46	**43**	**44**
	ufl_nfr_s		0.51	0.51	0.51	47	**43**	**44**
	ufl_ufr_s		0.64	0.51	0.51	**43**	**43**	**44**
BPIC20P	nfl_nfr_d	68	0.64	0.53	**0.70**	20	**24**	25
	nfl_ufr_d		0.64	0.53	**0.70**	21	**24**	25
	ufl_nfr_d		0.68	0.53	0.69	20	**24**	22
	ufl_ufr_d		**0.69**	**0.62**	0.66	18	25	**21**
	nfl_nfr_s	0	0.29	0.21	0.29	20	26	25
	nfl_ufr_s		0.29	0.21	0.29	19	26	25
	ufl_nfr_s		0.25	0.29	0.29	19	**24**	25
	ufl_ufr_s		0.50	0.26	0.29	**16**	**24**	25
BPIC20R	nfl_nfr_d	5	0.78	**0.82**	**0.82**	32	31	31
	nfl_ufr_d		0.78	**0.82**	**0.82**	32	31	31
	ufl_nfr_d		0.80	**0.82**	**0.82**	33	31	31
	ufl_ufr_d		**0.81**	**0.82**	**0.82**	30	31	31
	nfl_nfr_s	0	0.48	0.48	0.48	29	**28**	**28**
	nfl_ufr_s		0.48	0.48	0.48	29	**28**	**28**
	ufl_nfr_s		0.48	0.48	0.48	31	**28**	**28**
	ufl_ufr_s		0.61	0.48	0.48	**27**	**28**	**28**
Road	nfl_nfr_d	7	0.73	0.63	0.80	744	740	740
	nfl_ufr_d		0.73	0.63	0.80	745	741	747
	ufl_nfr_d		0.78	0.63	**0.81**	734	742	715
	ufl_ufr_d		0.65	**0.65**	0.79	**659**	**727**	719
	nfl_nfr_s	0	0.53	0.50	0.74	969	1053	871
	nfl_ufr_s		0.53	0.50	0.74	817	1051	875
	ufl_nfr_s		0.72	0.50	0.74	837	1058	880
	ufl_ufr_s		**0.79**	0.50	0.79	704	1062	**672**
Hospital	nfl_nfr_d	4	**0.83**	**0.61**	**0.85**	591	661	597
	nfl_ufr_d		**0.83**	**0.61**	**0.85**	596	**659**	599
	ufl_nfr_d		**0.83**	**0.61**	**0.85**	612	661	597
	ufl_ufr_d		0.67	**0.61**	0.72	570	674	**544**
	nfl_nfr_s	0	0.78	0.60	0.82	638	666	607
	nfl_ufr_s		0.78	0.60	0.82	627	665	603
	ufl_nfr_s		0.78	0.60	0.82	630	666	608
	ufl_ufr_s		0.60	0.60	0.76	**434**	705	574

improve the performance of a traditional process discovery algorithm independently of the configuration adopted to run the algorithm (i.e., possible use of filtering mechanisms, as well as possible consideration of trace variants with their frequency). Table 5 reports the number of concept drifts detected with STARDUST, as well as the conformance and time metrics measured for all the tested configurations of both STARDUST and its static counterpart. Figure 4 shows how the complexity of process models discovered with the tested configurations of STARDUST changes over time in correspondence of concept drifts when the process model discovery step is repeated.

Results on the conformance (Fmeasure of fitness and precision) show that the behavior concept drift detector approach integrated in STARDUST is decisive for discovering process models that allow us to measure higher conformance of upcoming activity traces to the lastly discovered process model. This is confirmed by the results of statistical tests performed to verify whether the improvement of conformance of the process models discovered with STARDUST configurations is significant over the various trace streams. To this aim, we used Friedman's test that is a non-parametric test commonly used to compare multiple methods over multiple data collections. The null-hypothesis states that all the methods are equivalent. Under this hypothesis, the ranks of compared methods should be equal. In this study, we rejected the null hypothesis with $p\text{-}value \leq 0.05$. As the null-hypothesis was rejected, that is, no method was singled out, we used a post-hoc test—the Nemenyi test—for pairwise comparisons.

Results of Nemenyi tests reported in Figure 5 confirm that all the configurations of the static counterpart of STARDUST rank lower than the dynamic configurations of STARDUST independently of the process discovery algorithm considered. This analysis also provides specific insights about the behavior of the compared process discovery algorithms in terms of conformance of the discovered process models with respect to the use of filtering mechanisms and frequency information in the process discovery. For example, both ILP and IMi take advantage of their embedded filtering mechanism, while SM performs better when the filtering mechanism has been disabled. Interestingly, this behaviour of filtering on ILP and IMi is shown independently of the process discovery setting tested (dynamic or static). On the other hand, exploring the effect of the frequency information, we note that SM and ILP perform better by neglecting the frequency information (i.e., by considering only one repetition of each sampled trace variant in the process discovery) in the dynamic setting, while they take advantage of accounting for repetitions of trace variants in the process discovery performed in the static setting. Differently, in IMi, the frequency information, that is neglected into the top-ranked static configuration (ufl_nfr_s), is relevant into the top-ranked dynamic configuration (ufl_ufr_d).

Results on the total computation time show that dynamic configurations of STARDUST were commonly slower than configurations of the static counterpart. The more computation time of dynamic configurations was spent monitoring concept drifts and repeating the process discovery as concept drifts were detected. In any case, the gap between the computation time spent in both the dynamic setting and the static setting is small in all the tested streams independently of the process discovery algorithms considered.

Interestingly, in Road, the static setting commonly spent more computation time than the dynamic setting. In fact, the only exception of this behavior is observed with ufl_ufr with SM. This can be explained by analysing the complexity (extended Cardoso index) of the process models discovered in the dynamic setting (Figure 4). In fact, the process models of Road discovered in correspondence of concept drifts are commonly simpler than the process model discovered in the initialization step. This is also shown in the Petri net representation of these process models. As an example, Figure 6 shows the Petri nets of the process models of Road

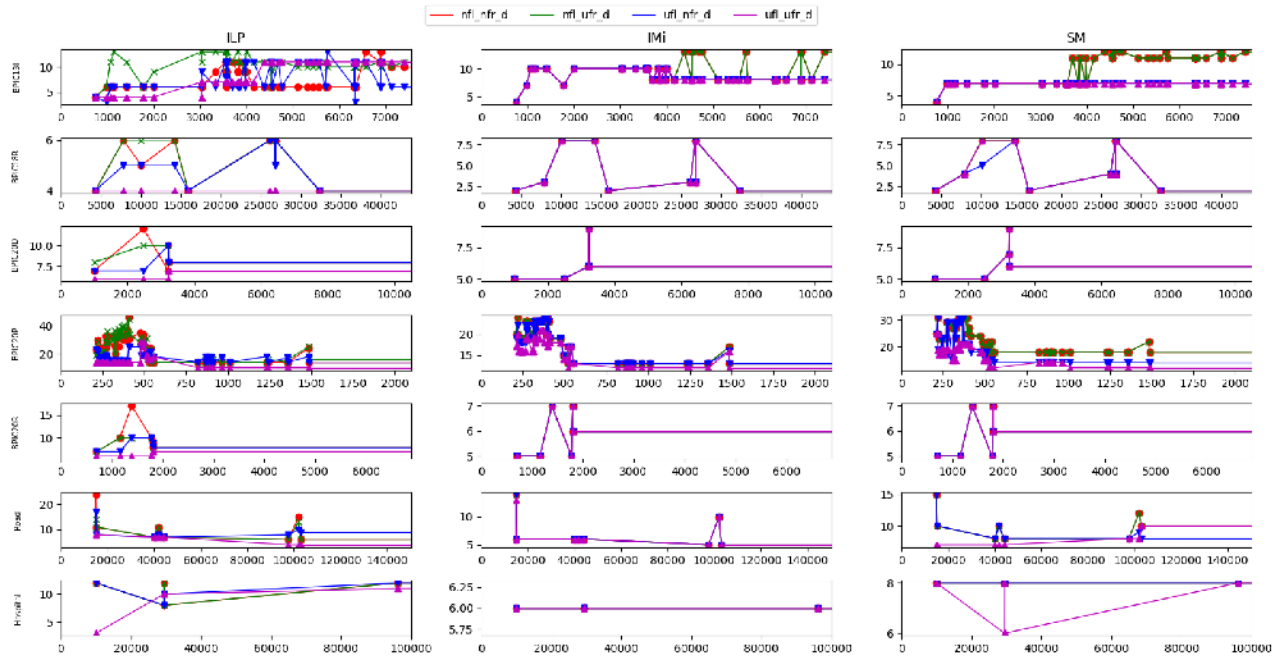
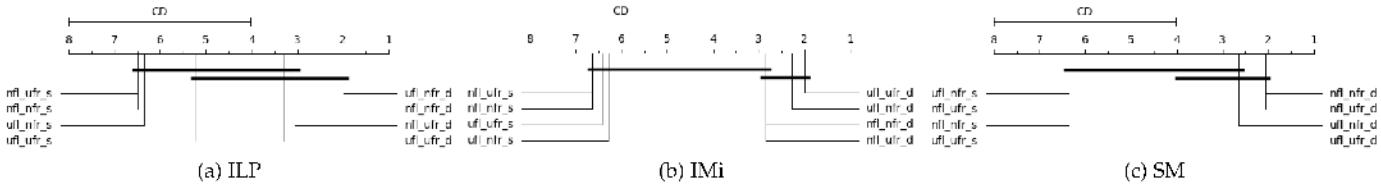


Figure 4: Extended Cardoso index (axis Y) of process models kept over time (axis X) by running STARDUST with ILP, IMi and SM



(a) ILP (b) IMi (c) SM

Figure 5: Nemenyi test of Fmeasure of process models discovered with ILP (a), IMi (b) and SM (c) with both STARDUST (d) and its static counterpart (d). Groups of methods that are not significantly different (at $p \leq 0.05$) are connected.

discovered by STARDUST with IMi in the configuration nfl_nfr. We note that the cycle on *Payment*, that appeared in the initial process model, was removed in the subsequent process models. In addition, a multitude of behaviours described in the initial process model (e.g., concerning various control-flow on activities *Create fine*, *Insert Notification*, *Add penalty*) were removed in subsequent process models. This simplification of the process models allows us to increase the conformance measured over the current process model. On the other hand, as the time spent verifying the conformance of an upcoming activity trace to the lastly discovered process model depends on the complexity of the process model, the simpler the process model, the lower the time spent in the conformance evaluation. Another aspect that deserves attention is that the process models discovered during the Road stream reveals the presence of a recurrent concept drift (see the process model repeated on the concept drifts 1 and 3, as well as the process model repeated on the concept drifts 2 and 4 in Figure 6) disclosing the existence of a temporary periodicity in the appearance of a specific process model.

Additional considerations concern the variability of the complexity of the process models with respect to the process discovery algorithm. In a few cases, the complexity of the process model changes with the tested configuration of the process discovery algorithm. In these cases, the use of frequency without filtering (nfl_ufr) may lead to the discovery of more complex process models (e.g., in BPIC13I and BPIC20P). In general, independently of the process discovery algorithm and its configuration, the complexity of the process model may change over time, although it commonly stays stable over long time. For some streams (e.g., BPIC13I) the process models become more complex, as the number of frequent behaviours increases over time. For other streams (e.g., BPIC20P and Road) the complexity of the process model is simplified over time, as the number of frequent behaviours decreases over time. For example, the process models discovered in BPIC20P became commonly simpler after that 750 traces were fully recorded in the stream. The variability in the complexity of the process model over time is coupled to an improvement of the conformance of the online process model to the upcoming traces in the stream (as shown in Table 6). The only exception is observed in Road with ILP(ufl_ufr), where the process model becomes simpler over time in the dynamic setting, but this higher simplicity is at the cost of a loss of conformance (Fmeasure decreases from 0.79 of in ufl_ufr_s vs to 0.65 of ufl_ufr_d).

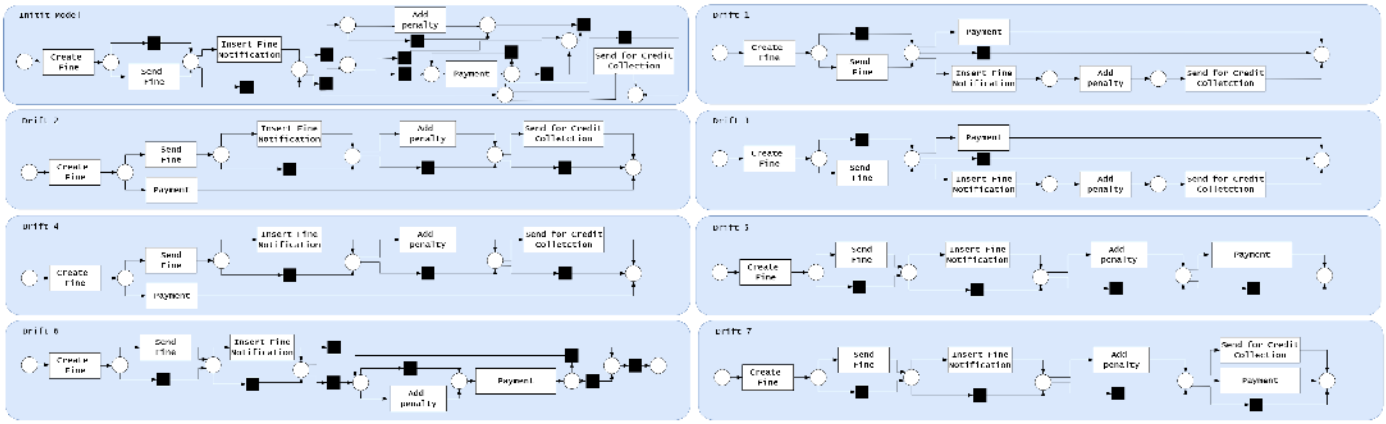


Figure 6: Process models discovered with STARDUST (IMi, nfl_nfr) in the initialization step, as well as in correspondence of each concept drift alerted in Road

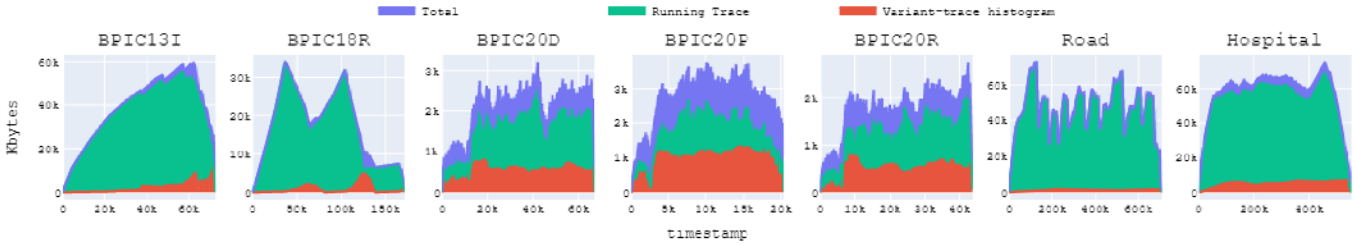


Figure 7: Memory usage in Kbytes (axis Y) over time (axis X). **Running trace** denotes the memory used keeping running traces recorded in the stream, while **trace variant histogram** denotes the memory used to keep the histogram of active trace variants with **Total=Running trace+trace variant histogram**.

5.3.2 *Memory usage analysis*

We proceed this analysis exploring the memory used in the dynamic setting of STARDUST. Figure 7 shows the amount of memory (in Kbytes) used to keep every stream in memory over the time.[9] In all the considered streams, most of the memory was used to record running cases. Although the total amount of the used memory fluctuates over time, in general, the overall trend of the memory usage stays roughly constant over time. This behavior is achieved as all the cases that were completed in the stream were removed from the data synopsis $\mathcal{R}$ and recorded in the data synopsis $\mathcal{B}_{\Sigma^*}$, while the size of $\mathcal{B}_{\Sigma^*}$ was kept under control over time thanks to the forgetting mechanism that discarded inactive trace variants. We also observe that the amount of used memory varies with the stream. In general, more memory is consumed in streams that record the higher number of trace variants (i.e., BPIC13I), as well as in streams that register high number of cases run in parallel for long time (e.g., Road and Hospital). On the other hand, BPIC13I is the stream that records the highest number of trace variants with a trace distribution that is slightly compliant with the Pareto's principle (80% of traces covered by 33.8% of trace variants as reported in Table 3). Despite Road and Hospital have a high number of cases that stay incomplete in the stream over long time, the adopted data synopsis used to keep the running cases in memory required about 60 KBytes memory to keep streams that finally register 4271.8 KBytes and 2208.2 KBytes (as reported in Table 3) of information, respectively.

9. To measure the memory usage we assume that 2 bytes were used to record an activity name, while 4 bytes were used to record a case identifier, the frequency of a trace variant, the value of the trace enumerator and the pointer to a tree node.

5.3.3 *Forgetting mechanism analysis*

In this experimentation, we commonly set the forgetting size equal to the number of activity traces processed in the initialization step (i.e., $w = \eta$ by default). However, we also explored the effect of w on the performance of STARDUST. We conducted this study using BPIC20R and Road. Figure 8 shows Fmeasure, computation time (s), total memory usage (Kbytes) and number of drifts when STARDUST is run with IMi by varying the forgetting size w among $\frac{\eta}{2}$, η (default) and $\frac{3\eta}{2}$. As a baseline of this experiment, we considered the case that no forgetting mechanism was adopted in STARDUST (None). Results show that the number of detected drifts changes with w. However, the effect of the forgetting mechanism depends on the stream. It is beneficial in BPIC20R, where we observe a gain in the Fmeasure of the process model discovered online to the upcoming traces and a reduction of the amount of memory used to keep the streamed data over time. This behaviour is at the cost of a slight increase in the computation time spent processing the stream. On the other hand, the use of the forgetting mechanism decreases the Fmeasure of the process model discovered online in Road causing no significant decrease in the total memory used keeping the streamed data online and an increase of the computation time spent processing the stream. This is due to the higher number of concept

drifts detected. As shown in Figure 6, the process models discovered in Road suggest the existence of recurrent drifts. A sensitive forgetting mechanism leads to a local overfitting phenomenon in the process model adaptation to short-term local drifts since the model is updated frequently missing the global behaviour of the traces. In general, the experiment shows that the forgetting mechanism may save memory and contribute to improve Fmeasure, although the set-up of (a possible adaptive) w deserves further investigation in future works especially in presence of frequent recurrent drifts.

5.3.4 Frequency-based sampling analysis

We explored the performance of STARDUST by varying the sampling rate μ among 0.7, 0.8 (default), 0.9 and 1.0. Note that frequency-based sampling is disabled when $\mu = 1.0$. Figure 9 shows Fmeasure, computation time (s), total memory usage (Kbytes) and number of drifts of STARDUST run with IMi in both BPIC20R and Road. Results show that the worst performance of STARDUST (the lowest Fmeasure, the highest computation time and the most memory used) is achieved when $\mu = 0.9$ and $\mu = 1.0$, and the number of detected drifts increases. In these cases, less frequent trace variants were taken into account for both the concept drift detection and the process discovery by leading to overfit the process model kept online to local drifts occurring on infrequent behaviours. In general, this experimentation shows the advantage of neglecting infrequent trace behaviours through sampling, although the set-up of the sampling rate may depend on specific characteristic of streams. Investigating characteristics of streams that may affect the set-up of μ deserves further investigations in the future.

5.3.5 Concept drift analysis

We complete this analysis by comparing the performance of the behavior concept drift detector integrated in the default dynamic configuration of STARDUST to the related concept drift detector based on the cluster analysis, called CDESF, and introduced in [40]. CDESF is run with the default parameter setup suggested by the authors in the github repository (i.e., time horizion=259200, λ=0.05, $\epsilon = 0.2$, $\mu = 4$, stream speed=100). Results of the conformance of the process models updated on the concept drifts discovered with the two approaches are reported in Table 6. These results show that, except for BPIC20D and BPIC20R, the behavior concept drift detector identifies a higher number of concept drifts than the cluster-based approach. CDESF does not identify any concept drift in Hospital, while STARDUST detects four concept drifts. On the other hand, updating the process model in correspondence of the detected concept drifts allows STARDUST to keep an updated process model of Hospital that outperforms the one maintained by CDESF in terms of conformance to the upcoming traces. In general, the use of the proposed behavior concept drift detector provides process models that keep higher conformance over time than the counterpart process models discovered integrating the cluster-based concept drift detector. This conclusion can be commonly drawn independently of the process discovery algorithm, as well as its configuration. The only exceptions are observed in various configurations of BPIC13I and Road, where the CDESF allows us to achieve, in some cases, higher conformance in process discovery than the behaviour concept drift detector proposed in this study. In any case, the best conformance is still achieved with STARDUST in the experiments on Road when SM is used as process discovery algorithm.

Table 6: Fmeasure of process models discovered with ILP, IMi and SM using both the behavior concept drift detector of STARDUST (S) and the cluster-based approach CDESF [40] (C). The best results are in bold.

stream	conf	#drift		Fmeasure					
				ILP		IMi		SM	
		S	C	S	C	S	C	S	C
BPIC13I	nfl_nfr	63	7	0.50	**0.51**	0.57	**0.60**	0.58	**0.60**
	nfl_ufr			0.50	**0.51**	0.57	**0.57**	0.58	**0.60**
	ufl_nfr			**0.52**	0.51	0.58	**0.60**	0.52	**0.53**
	ufl_ufr			**0.52**	0.49	0.58	**0.60**	0.52	**0.53**
BPIC18R	nfl_nfr	11	2	**0.96**	0.94	**0.96**	0.94	**0.96**	0.94
	nfl_ufr			**0.96**	0.94	**0.96**	0.94	**0.96**	0.94
	ufl_nfr			**0.96**	0.94	**0.96**	0.94	**0.95**	0.94
	ufl_ufr			**0.94**	**0.94**	**0.96**	0.94	**0.96**	0.94
BPIC20D	nfl_nfr	4	5	**0.80**	0.60	**0.83**	0.68	**0.83**	0.68
	nfl_ufr			**0.80**	0.60	**0.83**	0.68	**0.83**	0.68
	ufl_nfr			**0.80**	0.60	**0.83**	0.68	**0.83**	0.68
	ufl_ufr			**0.83**	0.64	**0.83**	0.68	**0.83**	0.68
BPIC20P	nfl_nfr	68	3	**0.64**	0.26	**0.53**	0.24	**0.70**	0.33
	nfl_ufr			**0.64**	0.26	**0.53**	0.24	**0.70**	0.33
	ufl_nfr			**0.68**	0.50	**0.53**	0.24	**0.69**	0.33
	ufl_ufr			**0.69**	0.50	**0.62**	0.21	**0.66**	0.33
BPIC20R	nfl_nfr	5	6	**0.78**	**0.78**	**0.82**	0.81	**0.82**	0.81
	nfl_ufr			**0.78**	**0.78**	**0.82**	0.81	**0.82**	0.81
	ufl_nfr			**0.80**	0.79	**0.82**	0.81	**0.82**	0.81
	ufl_ufr			**0.81**	0.81	**0.82**	0.81	**0.82**	0.81
Road	nfl_nfr	7	1	**0.73**	0.64	0.63	**0.66**	**0.80**	0.77
	nfl_ufr			**0.73**	0.64	0.63	**0.66**	**0.80**	0.77
	ufl_nfr			0.78	**0.79**	0.63	**0.66**	**0.81**	0.77
	ufl_ufr			0.65	**0.73**	0.65	**0.65**	**0.79**	**0.79**
Hospital	nfl_nfr	4	0	**0.83**	0.78	**0.61**	0.60	**0.85**	0.82
	nfl_ufr			**0.83**	0.78	**0.61**	0.60	**0.85**	0.82
	ufl_nfr			**0.83**	0.78	**0.61**	0.60	**0.85**	0.82
	ufl_ufr			**0.67**	0.60	**0.61**	0.60	0.72	**0.82**

6 Conclusion

In this paper, we have presented a process discovery approach, called STARDUST, that deploys traditional process mining algorithms in a dynamic (streaming) setting. It discovers an initial process model from an initial batch of activity traces, integrates a concept drift detection approach that monitors relevant changes in the behavior of acquired activity traces, in order to alert concept drifts, and discovers a new process model as a concept drift is detected in the stream. The experiments show the effectiveness of the proposed approach also compared to a state-of-the- art concept drift detection approach.

One limitation of the proposed approach is the lack of root-cause analysis to identify sources of concept drifts that may condition performance of process models (e.g., root cause of periodicity of specific concept drifts). To overcome this limitation, we plan to integrate the proposed approach in a pipeline able to explore process deviations along their root causes. On the other hand, STARDUST does not provide any information on the drift type (sudden or gradual). We plan to explore the possible integration of techniques for drift categorization [31]. Another limitation is that STARDUST is formulated for business processes that declare the completion activity of their full cases. This is expected in several business processes, e.g., ticket resolution in help desk processes. However, a future research direction

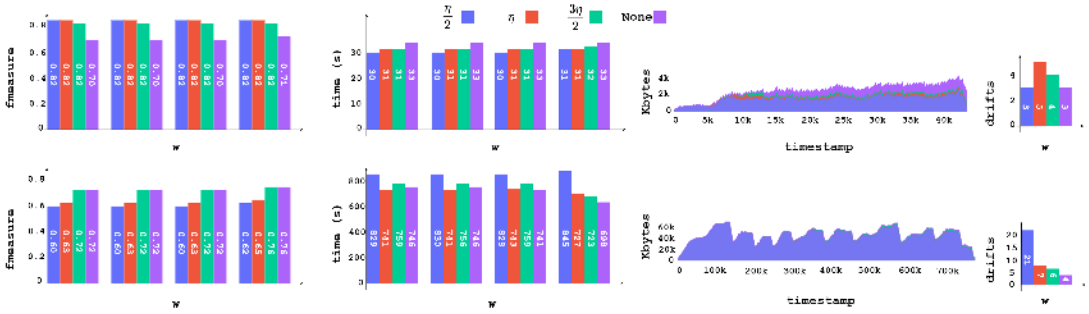


Figure 8: Fmeasure, computation time (s), total memory usage (Kbytes), number of drifts of STARDUST run with IMi in BPIC20R and Road by varying the forgetting size w among $\frac{\eta}{2}$, η (default) and $\frac{3\eta}{2}$. None denote the case no forgetting mechanism is enabled.

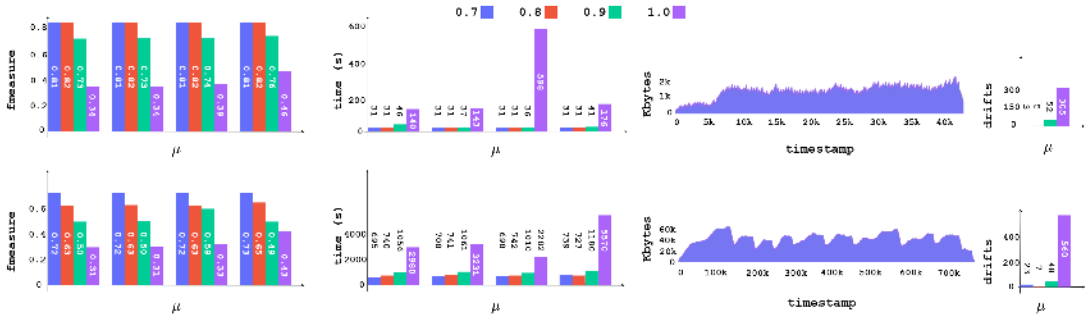


Figure 9: Fmeasure, computation time (s), total memory usage (Kbytes), number of drifts of STARDUST run with IMi in BPIC20R and Road by varying the sampling rate μ among 0.7, 0.8 (default), 0.9 and 1.0 (no sampling).

includes extending the proposed approach to handle the possibility of traces recorded "without the final activity".

In addition, we plan to explore alternatives to the used frequency-based sampling (e.g., active learning strategies) to identify representative traces of the process behaviours. As the proposed approach is formulated to handle business processes whose traces are distributed according to the Pareto's distribution, we intend to continue the research started in [6] and explore effectiveness of abstraction-based strategies to identify representative traces of streamed process behaviours that violate the Pareto's principle. Finally, we intend to explore some adaptive forgetting mechanism to automatically adapt both the forgetting size and the sampling rate over time to the stream characteristics.

REFERENCES

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," 06 2002, pp. 1–16.

[2] S. Barbon Junior, G. M. Tavares, V. G. T. da Costa, P. Ceravolo, and E. Damiani, "A framework for human-in-the-loop monitoring of concept-drift detection in event log stream," in *Companion of the Web Conference 2018, WWW 2018*, P. Champin, F. Gandon, M. Lalmas, and P. G. Ipeirotis, Eds. ACM, 2018, pp. 319–326.

[3] W. M. P. van der Aalst and J. Carmona, Eds., *Process Mining Handbook*, ser. Lecture Notes in Business Information Processing. Springer, 2022, vol. 448.

[4] B. van Dongen and F. F. Borchert, "BPI challenge 2018," 4TU.ResearchData. Dataset., Mar 2018.

[5] M. Fani Sani, S. van Zelst, and W. van der Aalst, "The impact of biased sampling of event logs on the performance of process discovery," *Computing*, pp. 1–20, 02 2021.

[6] V. Pasquadibisceglie, A. Appice, G. Castellano, and W. M. P. van der Aalst, "PROMISE: coupling predictive process mining to process discovery," *Inf. Sci.*, vol. 606, pp. 250–271, 2022.

[7] M. Bahri, A. Bifet, J. Gama, H. M. Gomes, and S. Maniu, "Data stream analysis: Foundations, major tasks and tools," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 11, no. 3, 2021.

[8] R. P. J. C. Bose, W. M. P. van der Aalst, I. Zliobaite, and M. Pechenizkiy, "Dealing with concept drifts in process mining," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 25, no. 1, pp. 154–171, 2014.

[9] A. Ostovar, A. Maaradji, M. Rosa, A. T. Hofstede, and B. van. Dongen, "Detecting drift from event streams of unpredictable business processes," in *ER*, 2016.

[10] N. J. Omori, C. M. Tavares, P. Ceravolo, and S. Barbon Junior, "Comparing concept drift detection with process mining tools," in *Proceedings of the XV Brazilian Symposium on Information Systems*, F. G. Rocha *et al.*, Eds. ACM, 2019, pp. 31:1–31:8.

[11] G. M. Tavares, P. Ceravolo, V. G. T. da Costa, E. Damiani, and S. Barbon Junior, "Overlapping analytic stages in online process mining," in *2019 IEEE International Conference on Services Computing, SCC 2019*, E. Bertino, C. K. Chang, P. Chen, E. Damiani, M. Goul, and K. Oyama, Eds. IEEE, 2019, pp. 167–175.

[12] A. Burattin and J. Carmona, "A framework for online conformance checking," in *Business Process Management Workshops*, E. Teniente and M. Weidlich, Eds. Springer Int. Publishing, 2018, pp. 165–177.

[13] A. Burattin, M. Cimitile, F. M. Maggi, and A. Sperduti, "Online discovery of declarative process models from event streams," *IEEE Trans. on Services Computing*, vol. 8, no. 06, pp. 833–846, 11 2015.

[14] P. Ceravolo, G. Marques Tavares, S. Barbon Junior, and E. Damiani, "Evaluation goals for online process mining: a concept drift perspective," *IEEE Transactions on Services Computing*, pp. 1–1, 2020.

[15] J. N. Adams, S. J. van Zelst, L. Quack, K. Hausmann, W. M. P. van der Aalst, and T. Rose, "A framework for explainable concept drift detection in process mining," in *Business Process Management - 19th International Conference, BPM 2021*, ser. Lecture Notes in Computer Science, A. Polyvyanyy, M. T. Wynn, A. V. Looy, and M. Reichert, Eds., vol. 12875. Springer, 2021, pp. 400–416.

[16] D. Fahland, J. Mendling, H. A. Reijers, B. Weber, M. Weidlich, and S. Zugal, "Declarative versus imperative process modeling languages: The issue of maintainability," in *Business Process Management Workshops*, S. Rinderle-Ma, S. Sadiq, and F. Leymann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 477–488.

[17] S. J. van Zelst, B. F. van Dongen, and W. M. P. van der Aalst, "Event stream-based process discovery using abstract representations," *Knowl. Inf. Syst.*, vol. 54, no. 2, pp. 407–435, 2018.
[18] N. Tax, I. Verenich, M. La Rosa, and M. Dumas, "Predictive business process monitoring with LSTM neural networks," in *Int. Conference on Advanced Information Systems Engineering, CAISE 2017*, E. Dubois and K. Pohl, Eds. Springer, 2017, pp. 477–492.
[19] M. Camargo, M. Dumas, and O. G. Rojas, "Learning accurate LSTM models of business processes," in *International Conference on Business Process Management, BPM 2019*, ser. LNCS, T. T. Hildebrandt, B. F. van Dongen, M. Röglinger, and J. Mendling, Eds., vol. 11675. Springer, 2019, pp. 286–302.
[20] V. Pasquadibisceglie, A. Appice, G. Castellano, and D. Malerba, "Predictive process mining meets computer vision," in *Business Process Management Forum, BPM 2020*, D. Fahland, C. Ghidini, J. Becker, and M. Dumas, Eds. Cham: Springer International Publishing, 2020, pp. 176–192.
[21] ——, "A multi-view deep learning approach for predictive business process monitoring," *IEEE Transactions on Services Computing*, vol. 15, no. 4, pp. 2382–2395, 2022.
[22] C. Di Francescomarino, C. Ghidini, F. M. Maggi, W. Rizzi, and C. D. Persia, "Incremental predictive process monitoring: How to deal with the variability of real environments," *CoRR*, vol. abs/1804.03967, pp. 1–16, 2018.
[23] S. Pauwels and T. Calders, "Incremental predictive process monitoring: The next activity case," in *Business Process Management*, A. Polyvyanyy, M. T. Wynn, A. Van Looy, and M. Reichert, Eds. Cham: Springer International Publishing, 2021, pp. 123–140.
[24] S. J. van Zelst, B. F. van Dongen, W. van der Aalst, and H. M. W. Verbeek, "Discovering workflow nets using integer linear programming," *Computing*, vol. 100, no. 5, pp. 529–556, 2018.
[25] S. J. J. Leemans, D. Fahland, and W. van der Aalst, "Discovering block-structured process models from event logs - a constructive approach," in *Application and Theory of Petri Nets and Concurrency*, J.-M. Colom and J. Desel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 311–329.
[26] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and A. Polyvyanyy, "Split miner: automated discovery of accurate and simple business process models from event logs," *Knowledge and Information Systems*, vol. 59, pp. 251–284, 2019.
[27] S. Suriadi, R. Andrews, A. ter Hofstede, and M. Wynn, "Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs," *Information Systems*, vol. 64, pp. 132–150, 2017.
[28] M. F. Sani, S. J. van Zelst, and W. van der Aalst, "Improving process discovery results by filtering outliers using conditional behavioural probabilities," in *Business Process Management Workshops - BPM 2017 International Workshops, Revised Papers*, ser. Lecture Notes in Business Information Processing, E. Teniente and M. Weidlich, Eds., vol. 308. Springer, 2017, pp. 216–229.
[29] N. Tax, N. Sidorova, and W. van der Aalst, "Discovering more precise process models from event logs by filtering out chaotic activities," *J. Intell. Inf. Syst.*, vol. 52, no. 1, pp. 107–139, 2019.
[30] M. F. Sani, S. J. van Zelst, and W. van der Aalst, "Improving the performance of process discovery algorithms by instance selection," *Comput. Sci. Inf. Syst.*, vol. 17, no. 3, pp. 927–958, 2020.
[31] A. Maaradji, M. Dumas, M. L. Rosa, and A. Ostovar, "Detecting sudden and gradual drifts in business processes from execution traces," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 10, pp. 2140–2154, 2017.
[32] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Mach. Learn.*, vol. 90, no. 3, pp. 317–346, 2013.
[33] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *SIGMOD Rec.*, vol. 29, no. 2, p. 1–12, 2000.
[34] A. Adriansyah, B. van Dongen, and W. van der Aalst, "Conformance checking using cost-based fitness analysis," in *15th IEEE Int. Enterprise Distributed Object Computing Conf.*, 2011, pp. 55–64.
[35] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. van Dongen, and W. van der Aalst, "Measuring precision of modeled behavior," *Inf. Syst. E Bus. Manag.*, vol. 13, no. 1, pp. 37–67, 2015.
[36] S. van Zelst, A. Bolt Iriondo, and B. van Dongen, "Tuning alignment computation : an experimental evaluation," in *International Workshop on Algorithms & Theories for the Analysis of Event Data ATAED 2017*, ser. CEUR Workshop Proceedings, W. van der Aalst, R. Bergenthum, and J. Carmona, Eds., 2017, pp. 6–20.
[37] K. B. Lassen and W. van der Aalst, "Complexity metrics for workflow nets," *Information and Software Technology*, vol. 51, no. 3, pp. 610–626, 2009.
[38] W. v. d. Aalst *et al.*, "Process mining manifesto," in *International Conference on Business Process Management, BPM 2011, Proceedings*. Springer, 2011, pp. 169–194.
[39] S. J. van Zelst, B. F. van Dongen, and W. van der Aalst, "Avoiding over-fitting in ILP-based process discovery," in *Business Process Management*, H. R. Motahari-Nezhad et al., Ed. Cham: Springer International Publishing, 2015, pp. 163–171.
[40] G. M. Tavares, P. Ceravolo, V. G. Turrisi Da Costa, E. Damiani, and S. Barbon Junior, "Overlapping analytic stages in online process mining," in *IEEE International Conference on Services Computing, SCC 2019*, 2019, pp. 167–175.

Vincenzo Pasquadibisceglie is a short-term researcher at CINI. He received a PhD in Computer Science in the University of Bari Aldo Moro. His research activity concerns process mining, deep learning and computational intelligence. He was involved in a regional research project on process mining. He received 2019 IET's Vision and Imaging Award ICDP 2019. He is member of the IEEE Task Force on Process Mining.

Annalisa Appice is an Associate Professor at the Department of Computer Science, University of Bari Aldo Moro, Italy. She received a Ph.D. in Computer Science in the University of Bari Aldo Moro. Her research interests include machine learning, process mining and cybersecurity. She has been responsible for the local unit of various research projects. She is a member of the editorial board of several international journals. She is member of the IEEE Task Force on Process Mining.

Giovanna Castellano is an Associate Professor at the Department of Computer Science, University of Bari Aldo Moro, Italy, where she is the coordinator of the Computational Intelligence Lab. She is member of IEEE Computational Intelligence Society, the EUSFLAT society and the INDAM-GNCS society. Her research interests are in the area of Computational Intelligence and Computer Vision. She is Associate Editor of several international journals.

Nicola Fiorentino is graduated in both Computer Science and Philosophy from the University of Bari Aldo Moro, Italy. Currently, he is a student in the master degree in Computer Science of the University of Bari Aldo Moro.

Donato Malerba is a Full Professor in the Department of Computer Science, University of Bari Aldo Moro, Italy. He is IEEE member. He has been responsible for the local unit of several research projects. He received an IBM Faculty Award in 2004. He is in the editorial board of several international journals. His research interests include machine learning, data mining, big data analytic and their applications. He is member of the IEEE Task Force on Process Mining.