## Queries to the Author

**When you submit your corrections, please either annotate the IEEE Proof PDF or send a list of corrections. Do not send new source files as we do not reconvert them at this production stage.**

**Authors: Carefully check the page proofs (and coordinate with all authors); additional changes or updates WILL NOT be accepted after the article is published online/print in its final form. Please check author names and affiliations, funding, as well as the overall article for any errors prior to sending in your author proof corrections. Your article has been peer reviewed, accepted as final, and sent in to IEEE. No text changes have been made to the main part of the article as dictated by the editorial level of service for your publication.**

**Per IEEE policy, one complimentary proof will be sent to only the Corresponding Author. The Corresponding Author is responsible for uploading one set of corrected proofs to the IEEE Author Gateway**

Q1. Please confirm or add details for any funding or financial support for the research of this article.
Q2. Please provide complete bibliographic details for Refs. [38] and [66].

# Evaluation Goals for Online Process Mining: A Concept Drift Perspective

Paolo Ceravolo, Gabriel Marques Tavares, Sylvio Barbon Junior, and Ernesto Damiani

**Abstract**—Online process mining refers to a class of techniques for analyzing in real-time event streams generated by the execution of business processes. These techniques are crucial in the reactive monitoring of business processes, timely resource allocation and detection/prevention of dysfunctional behavior. Many interesting advances have been made by the research community in recent years, but there is no consensus on the exact set of properties these techniques have to achieve. This article fills the gap by identifying a set of evaluation goals for online process mining and examining their fulfillment in the state of the art. We discuss parameters and techniques regulating the balance between conflicting goals and outline research needed for their improvement. Concept drift detection is crucial in this sense but, as demonstrated by our experiments, it is only partially supported by current solutions.

**Index Terms**—Online process mining, event stream, requirements and goals, concept drift



## 1 Introduction

Process Mining (PM) is a set of data science techniques focused on the analysis of *event logs* [1]. Events are recorded when executing a Business Process and collected into *cases*, i.e., end to end sequences of events relevant to the same process instance. Traditional PM algorithms were designed to work *offline*, analyzing historical batches of logs gathering the complete course of cases, if necessary with multiple passes of analysis. This is, however, insufficient, from a business standpoint, when the real-time assessment of processes is crucial to timely manage resources and quickly react to dysfunctional behaviors [2]. Today's fast-changing market requires systematic adjustments of processes in response to changes in the organization's operating system or to trends emerging from the environment [3].

Recently, the notion of *online* PM has emerged in reference to analytics capable of handling real-time event streams [4], [5]. An *event stream* differs from an event log because it is an unbounded sequence of events ingested one-by-one and allowing for limited actions in terms of iteration and memory or time consumption [6].

Traditional (*offline*) PM techniques cover three main tasks: *process discovery* where a new model is inferred based on the information contained in the event log; *conformance checking* where a model is compared with the event log, to analyze possible deviations; *process enhancement* where the model is updated to reach better performance results [1]. In recent years, researchers have achieved significant results in proposing adaptations to classic offline techniques to handle online processing, mainly for process discovery [5], [7], [8], [9], [10], [11], [12] and conformance checking [13], [14], [15], [16], [17], [18].

An assumption of several works is that *online* PM algorithms have to control time and space complexity, avoiding to exceed memory capacity, even dealing with logs that potentially tend to infinite size. In contrast, lower memory consumption is, in general, associated with lower accuracy; thus, the trade-off between these two dimensions should be controlled by algorithms, but little work has addressed this issue [10].

A major goal related to *online* PM is to get a real-time response over executed activities, minimizing the latency of reaction to deviant behavior. This requires inspecting incoming events quickly and incrementally, ideally event by event in one pass, still, a few incremental algorithms are available in the literature [10], [19]. In fact, the *offline* PM algorithms presuppose *complete cases* and it may be hard to convert them into incremental procedures [20].

Another crucial goal is Concept Drift Detection (CDD). Event streams are often non-stationary. The quality of a discovered model may change over time, and, by consequence, the validity of the conformance tests formerly executed is jeopardized. Techniques for detecting quality drifts in PM have been proposed [5], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35] but seldom applied to drive online updates or only partially able to fit the real-time constraint [20]. Also, there is no consensus on the criteria used to detect concept drift. Some approaches drive concept updates using a constant criterion [5], [26], [30], while others apply a variety of statistical tests to trigger it [24], [28], [30], [35]. Moreover, data streams are typically assumed as accurate and free of noise but this assumption is generally wrong. A cleansing stage, filtering out spurious events, may be required to improve the quality of the analysis [25], [36], [37]

- *P. Ceravolo and G. M. Tavares are with the Università degli Studi di Milano, 20122 Milano, Italy. E-mail: {paolo.ceravolo, gabriel.tavares}@unimi.it.*
- *S. Barbon Junior is with Londrina State University (UEL), Londrina 86057-970, Brazil. E-mail: barbon@uel.br.*
- *E. Damiani is with the Cyber-Physical Systems Center, Khalifa University, Abu Dhabi, UAE. E-mail: ernesto.damiani@ku.ac.ae.*

*Manuscript received 17 Oct. 2019; revised 15 June 2020; accepted 21 June 2020. Date of publication 0 . 0000; date of current version 0 . 0000. (Corresponding author: Paolo Ceravolo.) Recommended for acceptance by M. Shyu. Digital Object Identifier no. 10.1109/TSC.2020.3004532*

or pre-processing is required to ingest event data into the right level of abstraction [38].

These aforementioned goals are typically addressed in isolation thus the translation into functional and non-functional goals of the *online* PM problem is not uniform in the literature. This lack of common ground in terms of requirements and evaluation goals harms the assessment and benchmarking of *online* PM techniques. This paper aims at filling the gap by identifying a set of requirements pertinent to *online* PM. In our work, the state of the art is reviewed by a literature-based analysis of the requirements different approaches can issue. More specifically, we identify two design dimensions that require a balance between their conflicting goals. One dimension is represented by the relationship between *memory consumption* and *accuracy*, and the other by the relationship between *response latency* and *frequency of runs*. We also observe that handling multiple goals and addressing their conflicts requires integrating concept drift techniques. Progressing in this direction is essential to a better understanding of the topic. For this reason our work aims at:

- Identifying a set of *goals* of *online* PM to clarify the conflicting implications of different design approaches and the role of CDD in conciliating them.
- Proposing a *benchmark dataset* of event streams incorporating concept drift and incomplete cases.
- Performing an initial *assessment* of online PM techniques for CDD using quantitative measures for *accuracy* and scalability in *memory consumption*.

More specifically, the paper is organized as follows. Section 2 sets the foundations by presenting standard concepts used in PM and stream research. Section 3 proposes a set of requirements and goals for online PM algorithms. The section also reviews the approaches currently proposed in the literature investigating the requirements they support. Section 4 introduces a set of synthetic event streams created to simulate various online scenarios, thus, providing researchers with ways to compare CDD support in different online PM algorithms. Section 4 performs then experiments to compare current CDD techniques and analyzes their implications regarding the proposed requirements, with particular attention to the relationship between accuracy and memory consumption. Lastly, Section 5 concludes the paper and discusses subsequent steps for online PM.

# 2 Preliminaries

## 2.1 Process Mining Definitions

This section provides the basic concepts we are going to use throughout the paper.

An *Event Log* is a collection of *events* generated in temporal sequence and stored as *tuples*, i.e., recorded values from a set of *attributes*. Events are aggregated by *case*, i.e., the end to end execution of a business process. For the sake of classification, all cases performing the same sequence can be considered equal. A unique end to end sequence is therefore referred to as a *trace*.

**Definition 1 (Event, Attribute).** *Let* $\Sigma$ *be the* event universe, *i.e., the set of all possible event identifiers;* $\Sigma^*$ *denotes the set of all sequences over* $\Sigma$*. Events may have various* attributes, *such as* `timestamp`, `activity`, `resource`, `associated cost`, *and others. Let* $\mathcal{AN}$ *be the set of attribute names. For any event* $e \in \Sigma$ *and an attribute* $n \in \mathcal{AN}$*, then* $\#_n(e)$ *is the value of attribute* n *for event* e. *Typically values are restricted to a domain. For example,* $\#_{activity} \in A$*, where* $A$ *is the universe of the legal activities of a business process, e.g.,* $\{a, b, c, d, e\}$.

**Definition 2 (Trace, Subtrace).** *A* trace *is a non-empty sequence of events* $t \in \Sigma^*$ *where each event appears only once and time is non-decreasing, i.e., for* $1 \leq i < j \leq |t| : t(i) \neq t(j)$*. With abuse of notation we refer at the activity name of an event* $\#_{activity}(e)$ *as the event itself. Thus* $\langle a, b, d\rangle$ *denotes a trace of three subsequent events. An event can also be denoted by its position in the sequence as* $e_i$ *with* $e_n$ *the last event of a trace. A trace can also be denoted as a function generating the corresponding event for each position of its sequence:* $t(i \rightarrow n) \mapsto \langle e_i, \ldots, e_n\rangle$*. A subtrace is a sequence* $t(i \rightarrow j)$ *where* $0 < i \leq j < n$.

**Definition 3 (Case, Event Log).** *Let* $C$ *be the* case universe, *that is, the set of all possible identifiers of a business case execution.* $C$ *is the domain of an attribute* $case \in \mathcal{AN}$*. We denote a case* $c_i \in C$ *as* $\langle a, b, d\rangle_{c_i}$*, meaning that all events share the same case. For example, for* $c_i$ *we have* $\#_{case}(e_1) = \#_{case}(e_2) = \#_{case}(e_3)$*. An* event log $L$ *is a set of cases* $L \subseteq \Sigma^*$ *where each event appears only once in the log, i.e., for any two different cases the intersection of their events is empty.*

Given an Event Log $L$, we refer to its *behavior* as the set of traces that are required to represent all the cases in $L$.

**Definition 4 (Event Log behavior).** *An event log* $L$ *can be viewed as the multiset of traces induced by the cases in* $L$*. Formally,* $\overline{L} := \{t | \exists c_i \in L, c_i(i \rightarrow n) = t(i \rightarrow n)\}$*. The behavior of* $L$ *can be viewed as the set of the distinct elements of* $\overline{L}$*, formally* $\mathcal{B}_{\mathcal{L}} = support(\overline{L})$.

An event log $L$ is then a multiset because multiple cases can generate the same trace, while its behavior $\mathcal{B}_{\mathcal{L}}$ is the set of distinct traces induced by the cases.

Given a Model $M$, we refer to its *behavior* as the set of traces that can be generated from the model. In the presence of iterations, this set can be potentially infinite.

**Definition 5 (Process Model behavior).** *Given a process model* $M$*, we refer to its behavior* $\mathcal{B}_{\mathcal{M}} \subseteq \Sigma^*$ *as the set of traces that can be generated by its execution.*

Several quality measures can be defined in order to assess the accuracy of a model. These measures assess the appropriateness of a model against an event log considering their behavior.

**Definition 6 (Appropriateness).** *Appropriateness is a function* $a(\mathcal{B}_{\mathcal{L}}, \mathcal{B}_{\mathcal{M}})$ *or* $a(\mathcal{B}_{\mathcal{M}}, \mathcal{B}_{\mathcal{L}})$ *that measures aptness of ensuring that the* $\mathcal{B}_{\mathcal{L}}$ *is present in the* $\mathcal{B}_{\mathcal{M}}$ *versus ensuring that the* $\mathcal{B}_{\mathcal{M}}$ *is restrained to what observed in the* $\mathcal{B}_{\mathcal{L}}$.

Our definitions of behavior and appropriateness are abstract enough to be valid regardless of the specific implementations adopted in algorithms.

## 2.2 Process Mining Tasks

Discovering a model $M$ from $L$ implies to identify an appropriate generative representation of $L$, as a model can generate multiple traces based on the optional paths it describes. Many algorithms have been proposed, differing in terms of their underlying computational schema and data structure, and their resulting process modeling formalism. Most algorithms address the control-flow perspective, i.e., the model is expected to generate the behaviors observed in $L$. More recently, researchers have started targeting other perspectives such as data, resources, and time. We refer to [39] the reader interested in a detailed overview of process discovery algorithms. In the online setting, research approaches have principally focused on the control-flow perspective, with algorithms generating Petri nets [5], [7], [8], [10] as well as Declare models [11], [40]. This is then the perspective we consider in our definitions.

**Definition 7 (Process Discovery).** *A process discovery algorithm construct a process model from an event log and can thus be seen as a function* $\delta : \mathcal{B}_{\mathcal{L}} \mapsto \mathcal{B}_{\mathcal{M}}$.

When discovering a process model, different criteria can set the appropriateness of a representation. More specifically, *Fitness* and *Precision* have been largely used in the literature. The notion of *fitness* is aimed at capturing the extent of the behavior in $L$ that can be generated using $M$. If we trust on $M$, it can be used to detect anomalous traces in $L$. The notion of *precision* is aimed at capturing the extent of the behavior in $M$ that is not observed in $L$. A precise model does not generate more behavior than the observed.

**Definition 8 (Fitness).** *Fitness is a function* $f(\mathcal{B}_{\mathcal{L}}, \mathcal{B}_{\mathcal{M}})$ *that quantifies which part of the behavior observed in L can be reproduced in* $M$. *In abstract terms it can be defined as* $f(\mathcal{B}_{\mathcal{L}}, \mathcal{B}_{\mathcal{M}}) = \frac{\mathcal{B}_{\mathcal{L}} \cap \mathcal{B}_{\mathcal{M}}}{\mathcal{B}_{\mathcal{L}}}$.

**Definition 9 (Precision).** *Precision is a function* $p(\mathcal{B}_{\mathcal{M}}, \mathcal{B}_{\mathcal{L}})$ *that quantifies which part of the behavior that can be produced in* $M$ *cannot be observed in* $L$. *In abstract terms it can be defined as* $p(\mathcal{B}_{\mathcal{M}}, \mathcal{B}_{\mathcal{L}}) = \frac{\mathcal{B}_{\mathcal{M}} \cap \mathcal{B}_{\mathcal{L}}}{\mathcal{B}_{\mathcal{M}}}$.

Given that the set of traces characterizing a process model behavior may be infinite, the metrics proposed in the literature for fitness and precision work by approximations. Our definition is abstract as it does not specify how the comparison between the behavior in $L$ and $M$ is implemented. Indeed, defining an effective procedure requires addressing complex aspects, such as accounting the partial alignment between a trace and a model or confronting the finite behavior recorded on traces with the infinite behavior of the model [41]. These tasks are typically addressed using multipass analysis, meaning the offline PM measures of appropriateness cannot match the *event stream* criteria.

### 2.3 Stream Mining Definitions

Formally, a *data stream* is an ordered pair $(s, \Delta)$ where: $s$ is a sequence of tuples and $\Delta$ is a sequence of positive real time-intervals. Unfortunately, data stream analytic techniques [42], [43] cannot be readily applied in detecting business process behavior [43] due to a mismatch at the representation level. While stream analysis typically works at the event level, PM operates at the case level, where multiple events compose a trace. Nevertheless, in an *event stream* two subsequent events may belong to different cases then *online* PM algorithms are assumed to analyze events in two distinct stages. During the *ingestion stage*, a stream is read one event per time. During the *processing stage*, cases and traces are reconstructed and PM analytics are run. Also, in common to data stream analysis, *online* PM has to assume that the incoming flow of data is continuous and fast, i.e., the amount of memory that can be used during data analysis is much smaller than the entire series [42]. For this reason, a limited span of the stream is considered during analysis. Whatever this span is defined using memory space, time, or other conditions, we can refer to it as a window of analysis $W$. The behavior of the event stream can then be captured by comparing two distinct windows $Wa$ and $Wb$.

**Definition 10 (Window of Analysis).** *A window of analysis* $W$ *can be defined using its start time* $W_s$ *and its end time* $W_e$. *In comparing two windows* $Wa$ *and* $Wb$ *we can say that* $Wa$ *precedes* $Wb$, *formally* $Wa \prec Wb$, *if* $Wa_e < Wb_e$. $L^W$ *denotes the projection of an event log* $L$ *to a window* $W$.

The ability to use a window of analysis is crucial to *online* PM and can be used together with metrics measuring the appropriateness of a model to assess the conditions for triggering updates. This identifies a set of properties that must apply to any online solution.

#### 2.3.1 Properties of Online Process Mining

Given that analysis is executed on two different windows $(Wa \prec Wb \mid Wb \setminus Wa \neq \emptyset) \wedge |Wa| = |Wb|$, i.e., $Wb$ includes or excludes at least one behavior but the total number of observed behaviors does not change, the following properties should hold.

**Axiom 1.** $({\mathcal{B}_{\mathcal{L}}}^{Wb} \cap \mathcal{B}_{\mathcal{M}}) \setminus ({\mathcal{B}_{\mathcal{L}}}^{Wa} \cap \mathcal{B}_{\mathcal{M}}) > \emptyset \Longrightarrow f({\mathcal{B}_{\mathcal{L}}}^{Wb}, \mathcal{B}_{\mathcal{M}}) > f({\mathcal{B}_{\mathcal{L}}}^{Wa}, \mathcal{B}_{\mathcal{M}}), \quad p(\mathcal{B}_{\mathcal{M}}, {\mathcal{B}_{\mathcal{L}}}^{Wb}) > p(\mathcal{B}_{\mathcal{M}}, {\mathcal{B}_{\mathcal{L}}}^{Wa})$.

**Axiom 2:** $({\mathcal{B}_{\mathcal{L}}}^{Wb} \cap \mathcal{B}_{\mathcal{M}}) \setminus ({\mathcal{B}_{\mathcal{L}}}^{Wa} \cap \mathcal{B}_{\mathcal{M}}) = \emptyset \Longrightarrow f({\mathcal{B}_{\mathcal{L}}}^{Wb}, \mathcal{B}_{\mathcal{M}}) = f({\mathcal{B}_{\mathcal{L}}}^{Wa}, \mathcal{B}_{\mathcal{M}}), \quad p(\mathcal{B}_{\mathcal{M}}, {\mathcal{B}_{\mathcal{L}}}^{Wb}) = p(\mathcal{B}_{\mathcal{M}}, {\mathcal{B}_{\mathcal{L}}}^{Wa})$.

**Axiom 3:** $({\mathcal{B}_{\mathcal{L}}}^{Wb} \cap \mathcal{B}_{\mathcal{M}}) \setminus ({\mathcal{B}_{\mathcal{L}}}^{Wa} \cap \mathcal{B}_{\mathcal{M}}) < \emptyset \Longrightarrow f({\mathcal{B}_{\mathcal{L}}}^{Wb}, \mathcal{B}_{\mathcal{M}}) < f({\mathcal{B}_{\mathcal{L}}}^{Wa}, \mathcal{B}_{\mathcal{M}}), \quad p(\mathcal{B}_{\mathcal{M}}, {\mathcal{B}_{\mathcal{L}}}^{Wb}) < p(\mathcal{B}_{\mathcal{M}}, {\mathcal{B}_{\mathcal{L}}}^{Wa})$.

These axioms support important indications in terms of constraints applying to online PM tasks. First of all, process discovery must be rerun only if the process loses quality (Axiom 3). Conformance checking must be replayed each time the balance between $L$ and $M$ changes (Axioms 1 and 3). Finally, we have conditions where no update of the analysis is required (Axiom 2). This tells us that CDD is a general requirement for *online* PM.

#### 2.3.2 Types of Concept Drift

Static approaches have access to the complete data set. Thus, after a discovery procedure, the appropriateness of traces in front of the model is completely determined. In event stream processing, the appropriateness of traces changes over time, creating an additional challenge. In traditional data mining applications, concept drift is identified when in two separate points in time a *concept*, i.e., the true relation between a tuple, a feature vector, and its associated class, changes [43]. In *online* PM, drifts occur when the appropriateness between the model and the event stream changes creating, over time, the need for a model update. Otherwise, the model loses its

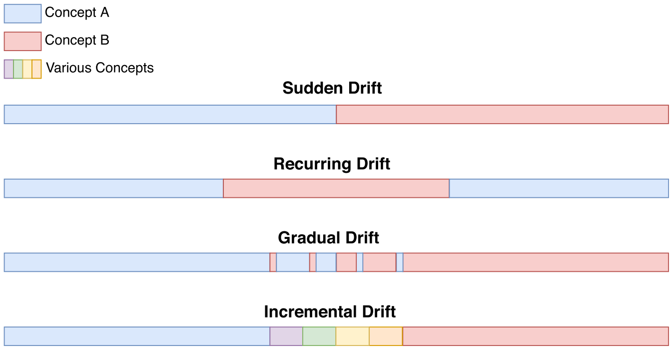


Fig. 1. Distributions of concepts for different drift types. Each drift type has its own transition period and characteristic. The image shows an initial concept $A$ that after a transition is replaced by concept $B$. As an example, the incremental drift has a transition composed of several different behaviors that change at a fast rate until it stabilizes.

representational power. This phenomenon manifests itself in different forms, Fig. 1 shows the four main types of concept drift identified in the literature [44]

- Sudden: concepts change abruptly.
- Recurring: changes appear seasonally over time, i.e., with recurring incidence.
- Gradual: concepts change by a gradual degradation, their quality decreases initially in delimited contexts to finally apply to the entire stream.
- Incremental: many small-scale intermediate changes are observed, i.e., an initial concept suffers several mutations until it becomes a different concept.

As observed in [45], current process discovery algorithms behave poorly when logs incorporate drifts: causal relations between events may appear and disappear, or even reverse, and therefore cannot be resolved. Experiments have demonstrated that concept drift produces a significant drop in the accuracy of PM algorithms [27]. Approaches to detect concept drift in event logs have been proposed [5], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35]. Nonetheless, they are not strongly linked to *online* PM approaches and mainly take aim at a single evaluation goal. In Sections 3.2.3 and 4.4, we propose a detailed evaluation of the CDD techniques today available, highlighting their limitations and discussing directions for progressing the field.

## 3 Framing the Online Process Mining State of the Art

In this section, we discuss the goals motivating organizations in introducing *online* PM and their relationship to solutions proposed in the literature.

### 3.1 Requirements of Online Process Mining

Different from *offline* PM, focused on observing a static log, *online* PM aims at leveraging insights about cases during their execution [46]. We come, then, by a base requirement discriminating what enters in our discussion.

*R0: Analysis Must Process Data Streams.* The solutions relevant to *online* PM must provide algorithms designed to ingest data streams. Data samples in a stream, potentially unbounded in size, demand for one-pass reading of not arranged flow of data [47], [48]. An architecture ingesting data streams and accumulating events, to feed algorithms working in batch mode, is for us out of scope. Also, we do not drive our attention to approaches focusing on improving the scalability of PM algorithms by exploiting task decomposition and parallelization [49], [50].

In the following, we show that the goals motivating the adoption of *online* PM are different and conflict with one another. Depending on the levels of satisfaction to be achieved for each goal, a trade-off can be identified. However, two conflicting goals cannot achieve their individual maximum levels of satisfaction at the same time. After listing these goals, we review the impact of current *online* PM approaches on them.

*G1: Minimize Memory Consumption.* As data streams are potentially infinite, and often characterized by high generation rates, an online analysis must minimize the amount of memory required during processing. A basic approach to address this goal is removing data out of the capacity of the available memory. This implies that the window of analysis $Wi$ is periodically updated. However, reducing the size of the data sample can seriously affect accuracy [5] (**G4**). Strategies for removing the less relevant recorded tuples are then studied, taking inspiration from the memory-friendly algorithms for stream mining used in machine learning [51]. CDD is proposed as a way of selecting relevant or irrelevant tuples to be maintained in memory [52]. The need for a trade-off between memory consumption and accuracy typically brings to a relaxed version of this goal, i.e., *it must be possible to bound the amount of memory to a given capacity*.

*G2: Minimize Response Latency.* One of the reasons for performing online analysis is to quickly react to events, e.g., for anomaly or fraud detection. If an executing business process is deviating from the expected behavior, an alert must be generated in time to analyze the case and coordinate a response. It follows that online analysis have to run on incomplete cases $c_i(i \rightarrow j)$, continuously assessing their appropriateness to a model: $a(\mathcal{B_L}, \mathcal{B_M})$. It is, however, evident that this affects the consumption of computational resources (**G3**). Lightweight representations for models and cases, together with single-pass evaluation of appropriateness metrics are then crucial challenges to be faced in order to achieve this goal.

*G3: Minimize the Number of Runs.* Online analysis should consume computational resources only when its execution can change the inferences arising from data. In general, the achievement of this goal is in opposition to **G2**, which requires the analysis is constantly running. However, it is reasonable to equip algorithms to find a trade-off. Imposing a constant scheme for updating the model or not supporting updates at all is not appropriate in online settings. As observed in Section 2.3.1, the flow of an event stream can bring to conditions either requiring to update the analysis or not. Some memory-friendly approaches are grounded on this idea [43]. It follows, CDD is crucial to identify the appropriate moment for updating the analysis.

*G4: Optimize Accuracy.* Even when implementing an online approach, it should be required to achieve levels of accuracy comparable to that of offline methods. In PM, accuracy is expressed by the appropriateness $a(\mathcal{B_M}, \mathcal{B_L})$. In this sense, accuracy is, in general, positively affected by the number of events considered during the analysis, the more

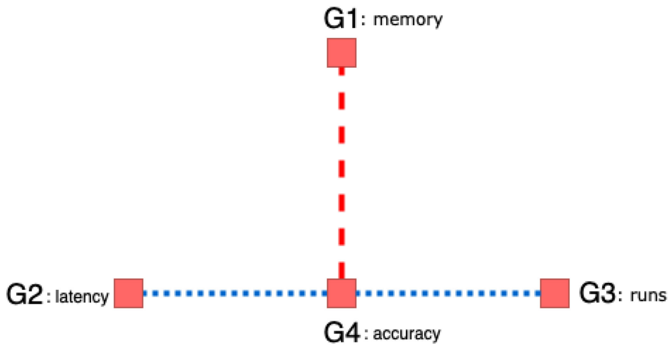


Fig. 2. Conflicting dimensions in *online* PM: **G1** (minimize memory consumption), **G2** (minimize response latency), **G3** (minimize runs) and **G4** (optimize accuracy).

events we use to discover $M$ the more overlap with $L$ we are likely to represent. That is, needless to say, the inverse of what **G1** requires. In addition, accuracy is positively affected by updates of the analysis [27]. As previously discussed, updates can arise from continuous incremental algorithms [53] (**G2**) or can be handled when the model is losing precision (**G3**), that implies being able to react to different types of concept drift [52]. A challenge to be faced when conciliating real-time response (**G2**) and accuracy is to have available lightweight representations resulting in accurate analysis even with fast access to data structures [20].

### 3.2 Requirement Satisfaction in Online Process Mining

We will now describe two design dimensions that require a balance between conflicting goals. Each dimension is represented by the two apogean goals connected by inverse relationships. One dimension is defined by the couple **G1-G4** and the other by **G2-G3**. However, as stated previously, **G4** is also dependent on **G2-G3**. A schematic representation of the conflicting dimensions in *online* PM is illustrated in Fig. 2.

In the following paragraphs, we illustrate different approaches and discuss how they support and control the achievement of previously set goals. Often the relationship between conflicting goals is not made explicitly, and rarely parameters for controlling the trade-off between goals are made available by current PM solutions. To review the literature, we use a three-level scale.

- *Full support* (✓). The method is designed to address the goal in all its aspects.
- *Partial support* (–). The method addresses the goal but does not satisfy all its aspects.
- *No support* (×). The method ignores the goal or cannot address it.

If the literature does not provide sufficient information to estimate the impact on a goal, we do not include it in the discussion. Fig. 3 offers a summary of our review using radar charts.

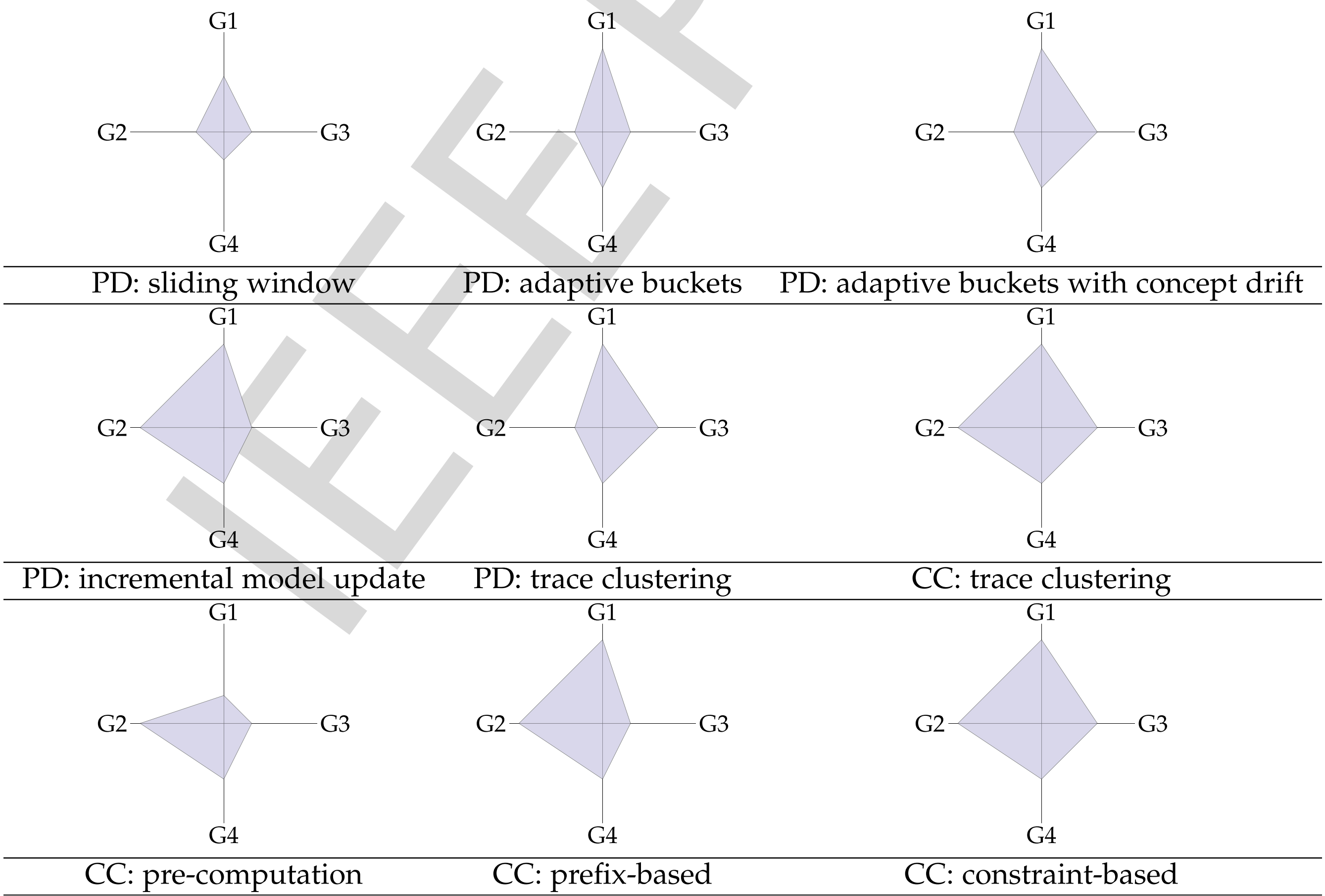


Fig. 3. Levels of control over goals, **G1** (minimize memory consumption), **G2** (minimize response latency), **G3** (minimize runs) and **G4** (optimize accuracy) by Process Discovery (**PD**) and Conformance Checking (**CC**) approaches.

#### 3.2.1 Approaches to Online Process Discovery

The most studied online procedures relate to Process Discovery (PD). They mostly focus on bounding memory consumption, though other goals are also discussed in the literature.

*Events Accumulation.* The first works addressing *online* PM are [45], [54]. They both focus on process discovery and identify that concept drift has to drive updates. This implicitly addresses **G3** as a computational task will run only if events follow new distributions. However, they rely on *offline* process discovery analysis, and for this reason, do not meet **×R0**. In [54], the events are accumulated capturing all the behavior observed in the event stream, then, using a sliding window, a statistical test verifies if significant changes are recorded and eventually calls an *offline* procedure to update the model. In [45] the events accumulated are used to generate an abstract interpretation of their behavior using a convex polyhedron that offers an upper approximation. When new cases are acquired, online analysis matches them with the polyhedron to estimate their divergence to the previously observed behavior, supporting the online assessment of concept drift. If a drift is detected, the polyhedron can be recalculated using an *offline* procedure. The problem of bounding the memory usage is not addressed (**×G1**), moreover, the time of accumulation implies a delay in the response (**×G2**).

*Sliding Window.* The sliding window method offers the simplest approach to bound the memory capacity based on maintaining only the last $n$ events of the stream. However, this approach is very limited. It does not guaranty the new behavior is captured (**×G4**) nor the memory capacity is met (**–G1**) as both these dimensions are dynamically evolving. In principle, extending the span of the window positively impacts accuracy but, at the same time, negatively affects memory usage and response time, which is by definition dependent on the dimension of the window (**×G2**). Moreover, the runs of analysis are predefined and cannot be controlled (**×G3**). Therefore, the validity of the approach is restrained to domains characterized by constant periodicity of updates and monotonic behavior. Actually, the approach is mentioned by different authors [10], [11], [45], [54] but not proposed as a solution.

*Adaptive Buckets.* The natural evolution of the sliding window approach is developing memory-friendly strategies, e.g., maintaining buckets of memory based on selective counting of events. *Sticky Sampling* and *Lossy Counting* are examples of, respectively probabilistic and deterministic, algorithms for computing item frequency over a data stream [55].

*Lossy Counting with Budget* is an adaptive version of the Lossy Counting algorithm that was successfully adapted to PM. The idea is to store events in buckets and count how many times an event is observed. When the maximum available memory (the budget) is reached, infrequent events are removed from memory (**√G1**). The accuracy of the analysis is controlled by an error margin $\epsilon$ that sizes the memory budget to be used. However, it is not possible to define a deterministic function mapping memory usage and accuracy (**–G4**) [10]. The approach requires, in any case, a separate algorithm to run process updates with an intrinsic delay in generating up-to-date responses (**×G2**), and nothing guarantees the updates will run consistently to axioms introduced in Section 2.3.1 (**×G3**). Moreover, these approaches do not offer support for all the drift types presented in Section 2.3.2.

The idea was originally proposed in [7] and improved in terms of memory consumption in [8]. The latter work introduces heuristics for memory pruning based on a decay factor applied both on events and cases. The approach was also applied to declarative process discovery [11], [40], where the model is expressed in terms of a set of constraints that cannot be violated during the process execution. Each constraint can be associated with an independent learner, reducing the response time required to update the model. In [36], the authors proposed a general framework to feed the abstract representations adopted by existing process discovery algorithms with event streams. All the algorithms investigated in the framework adapt their memory usage based on the current capacity.

Different authors put forward the idea that concept drift can drive the updates of memory buckets. However, some of the CDD techniques are unable to identify specific classes of drift such as incremental and recurring [5], [26], [30], while the advanced techniques that were proven to detect them adopt temporally extended tests that work with windows of significant size [24], [30], [32], [35] (**–G3**). In general, concept drift positively affects accuracy since, when updating the model, better conformance with incoming events is obtained [27] but, again, it is unclear which classes of drift can be effectively detected with memory-friendly approaches (**–G4**). We will go back to this issue in Section 4.4.

*Incremental Model Updates.* The most efficient way of limiting memory usage in stream processing is adopting incremental algorithms that consume events in a single step. This implies that no backtracking is possible and events can be deleted after being consumed (**√G1**). The incremental approach is also the best solution to minimize response latency because the analysis can be executed in real-time (**√G2**). However, the consumption of computational resources is continuous (**×G3**), and accuracy may be unsatisfactory in non-stationary environments where concept drift is recurring, because past behavior cannot be used to shape the model (**–G4**). Accuracy can be improved by keeping the results of incremental updates in buckets of memory and constructing a synoptic representation of the observed data stream. This is, however, a trade-off solution where both **G1** and **G2** are partially satisfied, due to the need of introducing auxiliary procedures besides the incremental model updates.

To the best of our knowledge, the only approaches using this strategy in *online* PM are [10], [20]. Barbon *et al.* [20] incrementally update a process model graph (PMG) to obtain a reliable process model. The PMG is maintained throughout stream processing with specific checkpoints triggering refresh and release statements of allocated resources. Similarly, Leno *et al*. [10] idea is to incrementally update an abstract representation bounding its size by the memory budget. The abstract representation is constructed by a *directly follow graph* with nodes representing activities and arcs representing direct follow relations between activities. The last event of every executing case is kept in memory, this way, any incoming event can be attached to the graph by a direct follow relation with its preceding event. Moreover, arcs and nodes are annotated with their observed frequencies thus, if the size of the graph exceeds the available memory, the less representative elements can be removed. The authors compared different deletion strategies with the *Lossy Counting with Budget* algorithm presented in [7], concluding that their approach outperforms it in terms of the amount of memory required to get high levels of accuracy. This confirms that the trade-off between memory and accuracy is a key parameter of configuration for *online* PM. Nonetheless, the relationship between memory consumption and accuracy and the impact that the emergence of concept drifts may have is not clarified in the literature.

*Trace Clustering.* In [31], various trace clustering techniques are tested for detecting concept drift. If a drift is detected, a model update can be executed, keeping the model aligned to the event log. However, the connection with event streams processing is not studied. In [19] process discovery is addressed using a sliding window approach (✓**G1**). The authors boost this approach by interlaying a density-based clustering procedure that interconnects multiple *online* PM tasks. A lightweight abstract representation of cases, supporting incomplete cases, is adopted to group cases in clusters constructed using density-based boundaries. Each time a new event is ingested, clusters are updated. Periodically, based on the dimension of the sliding window, the process model is updated (×**G2**). This clustering procedure allows, however, to identify anomalous cases and to capture concept drift with positive impacts on accuracy (–**G4**) and resource consumption (–**G3**), as dysfunctional cases are pruned from the model update procedure. The approach, however, requires to express drifts in terms of a distance between traces and cannot cope with PM appropriateness measures. For this reason, it does not offer full coverage of the axioms in Section 2.3.1.

### 3.2.2 Approaches to Online Conformance Checking

Approaches to *online* Conformance Checking (CC) essentially focus on supporting real-time analysis (**G2**). Memory consumption and accuracy are left in their natural inversional relationship or are managed using solutions discussed in Section 3.2.1.

*Event Accumulation.* Traditional offline CC uses the token-based replay technique, where previously executed cases are replayed over the model. Each executed activity corresponds to firing a token. Consumed, missed, and remaining tokens are counted to create conformance statistics. The most critical aspect of these replay techniques is that multiple alignments between a case and a model (corresponding to different starting points for log replay) are possible, and computing the optimal alignment is challenging from a computational point of view [56]. Even if, recently, general approximation schemata for alignment have been proposed [57], these approaches require to backtrack cases, and, for this reason, make memory bounding difficult (×**G1**). Moreover, they do not support the analysis of incomplete cases (×**G2**). Namely, these approaches rely on offline procedures and, thus, do not enter our comparative review (×**R0**).

*Pre-Computation.* Another strategy proposed to support online CC is based on the pre-computation of all the possible deviations on top of a model. In particular, in [13] a Petri net is converted into a transition system decorated with arcs describing the possible deviations from the model. Transitions representing deviations are associated with a cost, while transitions allowed by the model have no cost. By this approach, it is possible to compute conformance in real-time (✓**G2**). However, the requirements imposed in terms of memory consumption are high and difficult to be parameterized (×**G1**). The impact on model update and accuracy is not discussed by the authors, except for references to papers adopting the *Lossy Counting with Budget* approach (–**G4**). It can be, however, generally remarked that the approach imposes significant effort on model update (×**G3**), making hard to use CDD with a negative impact on accuracy for non-stationary environments.

*Prefix-Based.* In order to address the problem of computing the conformance of incomplete cases, in [17] an approach for assessing the optimal alignment of sub-traces is proposed. This goes in the direction of supporting conformance checking in real-time (✓**G2**), as, in principle, each new event can trigger the analysis. However, the approach is intrinsically related to the backtracking procedures required for alignment. The authors are aware of this problem and propose either *Random Sampling with Reservoir* [58] or *Decay-based data structures* [59], similarly to the solutions provided respectively in [7], [10] for process discovery. This way it is possible to manage the trade-off between memory consumption (✓**G1**) and accuracy (–**G4**). It is clear that CDD in non-stationary environments is a precondition to not lose accuracy. Despite that, no specific effort was dedicated to the interconnection of these prefix-based approaches and CDD; thus, there is no guarantee that the axioms introduced in Section 2.3.1 can be matched (×**G3**).

*Constraint-Based.* One pass conformance checking can be achieved using declarative constraints, i.e., relationships between the sequential order of activities that must be respected during the execution (typically expressed using linear temporal logic). In [60], a set of finite-state automata are proposed to validate declarative constraints. The authors show that online validation of these constraints is possible at the cost of clearly identifying the validity of the inferred conditions that can pass from different states given by the combination of violated/fulfilled, permanent/temporary conditions. The approach is designed to support real-time conformance checking at an event level (✓**G2**), i.e., supporting incomplete cases. Resource consumption is less significant than in other approaches because the analysis can be localized to the set of constraints that are activated by the case under analysis. This positively impacts memory consumption during analysis (✓**G1**), moreover, analysis run only if a constraint is matched (–**G3**). Accuracy is guaranteed if the set of constraints used is updated, which can be achieved using an approach based on adaptive buckets (–**G4**) [11]. However, this is clearly in counterbalance with **G3**, and no specific method for managing this balance is proposed. The experimental analysis we run shows that this approach, in practical terms, does not scale, due to the relevant number of cross-checks it imposes.

*Trace Clustering.* The approach presented in [19] uses density-based clustering to calculate, event by event, how similar a case is to the process model. This is a simplified conformance checking measure that can be computed in real-time (✓**G2**). The accuracy is, however, non-optimal as the adopted metrics do not have the same potential of methods that use backtracking procedures because they do not exploit the generalization power of model-aware replay procedures (–**G4**). Memory consumption is controlled by the sliding window, adopted to buffer incoming events (✓**G1**). Resource consumption is partially limited by the identification in real-time of dysfunctional cases that are pruned from conformance checking procedures (–**G3**).

TABLE 1
Requirement (**R0**) and Goals Met by the Concept Drift Detection Techniques Analyzed

| Technique | R0 | G1 | G2 | G3 | G4 |
|---|---|---|---|---|---|
| Bose *et al.* [21] | × | ✓ | × | × | – |
| Ostovar *et al.* [28] | ✓ | ✓ | – | × | – |
| Tavares *et al.* [19] | ✓ | ✓ | – | – | – |
| Yeshchenko *et al.* [32] | × | ✓ | × | × | – |
| Zheng *et al.* [62] | × | ✓ | × | × | – |

***G1*** *(minimize memory consumption),* ***G2*** *(minimize response latency),* ***G3*** *(minimize runs) and* ***G4*** *(optimize accuracy). ✓, – and × represent full, partial and no support, respectively.*

### 3.2.3 Concept Drift Detection

CDD was identified as a central issue already in the first works addressing *online* PM [45], [54]. The sliding window approach is, in general, adopted to track the latest process behavior. This makes it challenging to manage the balance between accuracy and memory management (**G1**-**G4**). Static window size is sometimes used [26], [30], the bias associated with the selected window size is then reflected on the results obtained by these solutions. To improve the accuracy of change-point detection, statistical tests were introduced to set the optimal size of the window of analysis [24], [61], but disregarding then memory management. As not all the proposed procedures can effectively identify specific classes of drift, such as incremental and recurring, statistical tests were also exploited to create robust detection techniques [24], [30]. Other approaches have improved change detection by isolating the behavioral properties impacted by changes [32], [35]. This allows going further change-point detection offering an explanation and a description of the detected changes. Advanced techniques, however, come at the cost of higher memory consumption. The control flow perspective is the most adopted, even if approaches focusing on temporal drifts are available [33], [34].

A general critical point to highlight is that CDD techniques are not specifically integrated with *online* PM tasks. Indeed, when two techniques adopt a different representation of the process behavior, their integration implies a higher consumption of resources (**G3**). Another critical aspect is that noise may be confused with concept drift if not appropriately filtered out [36], [37] (**G4**). Also, most CDD cannot cope with incomplete cases, an important requirement for implementing real-time response (**G2**).

In Section 4, we experimentally compare the performances of different CDD approaches. We then limit our review to solutions that were implemented in open source software and are available for execution.

The first drift detection PM approach implemented in open-source software is by Bose *et al.* [21]. The authors proposed an offline analysis of the event log, meaning that the log is consumed in a batch procedure. From the event log the relationships between activities, such as the *follows* or *precedes* relations are extracted. Then, two non-overlapping windows go through the event log. Finally, a statistical test is applied to compare the populations of both windows. Concept drift is found when the distributions of the populations are different.

Using a clustering algorithm as a kernel, Zheng *et al.* [62] introduced a three-stage approach to detect concept drifts from event logs. First, the log is converted into a relation matrix by extracting direct succession and weak order relations from traces. Then, each relation is checked for variation trends to obtain candidate change points. Lastly, the candidate change points are clustered using the DBSCAN algorithm [63] and combined into real change points.

In Yeshchenko *et al.* [32], the authors propose a technique for drift detection and visualization using Declare constraints and time series analysis [64]. Declare is a declarative process specification used to represent process behavior based on temporal rules, e.g., in which conditions activities may (or may not) be executed [65]. The approach starts by splitting the event log into sub-logs, where a set of Declare constraints are computed. Then, multi-variate time series representing constraints confidence are extracted and clustered. Each cluster is analyzed for change detection in the relation between constraints, highlighting the overall and behavior-specific drifts. The last step of the approach creates charts for a visual analysis of the detected drifts.

Differently from previous techniques, Ostovar *et al.* [28] perform natively over a stream of events. The authors describe process behavior using $\alpha+$ relations. An $\alpha+$ relation is characterized by a set of rules capturing the relation insisting between two activities, where the *follows* or *precedes* relations are the most representative [66]. Then, statistical tests over two consecutive sliding windows are performed. Moreover, the approach proposes a trade-off between accuracy and drift detection latency. For that, windows with adaptive sizes are adopted.

Tavares *et al.* [19] introduce a framework supporting multiple *online* PM tasks, including concept drift and anomaly detection. The framework models the business process as a graph and extracts case features based on graph distances. The features consider trace and inter-activity time between events as case descriptors. With the use of DenStream [67], case descriptors are clustered in an online fashion. Drifts are found once new clusters, which represent core behavior, are discovered in the stream.

Table 1 presents the requirements and goals supported by currently available drift detection techniques. Only Ostovar *et al.* [28] and Tavares *et al.* [19] satisfy the online processing premises (✓**R0**), ingesting event streams in a native way. Bose *et al.* [21], Yeshchenko *et al.* [32] and Zheng *et al.* [62] pre-process the event log and group events into cases. Then, they simulate a stream of traces (×**R0**).

Bose *et al.* [21], Ostovar *et al.* [28], Yeshchenko *et al.* [32] and Zheng *et al.* [62] approaches impose a boundary to the window of analysis. As stated in Section 3.2.1, these techniques minimize the amount of memory used as only the last $n$ events in the event stream are maintained (✓**G1**). At the same time, these methods present limitations regarding

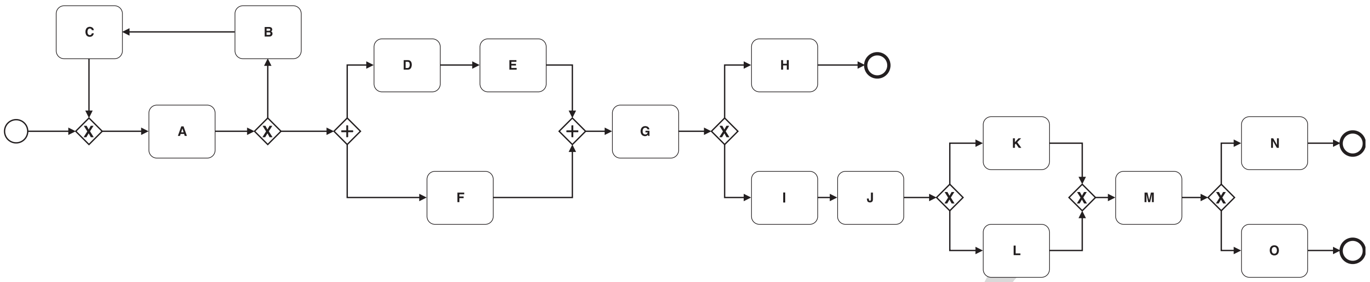


Fig. 4. BPMN model that represents the common behavior. Drifts are applied to this base model.

accuracy since there is a bias associated with the arbitrary dimension selected for windows. This drawback can be counterbalanced with window size tuning, leveraging the accuracy in specific scenarios (–**G4**).

Additionally, the response latency is associated with the window size. Since there is a minimum window size that yields acceptable results, the response time has a boundary. Bose *et al.* [21], Yeshchenko *et al.* [32] and Zheng *et al.* [62] approaches lack fast response to new events as they do not deal incomplete cases (×**G2**). For Ostovar *et al.* [28], the boundary can be reached by hyperparameter tuning (–**G2**). Moreover, the windows slide according to new events in the stream. This means that a single event stays in the window for at least $s$ iterations, where $s$ is the window size. Hence, $s$ runs over each event are performed (×**G3**).

The solution presented in Tavares *et al.* [19] consumes events in a single step as they arrive in the stream. Hence, events are deleted after consumption, saving memory (✓**G1**). Concerning CC the approach has a real-time response because each received event is clustered in the feature space (✓**G2**). Contrary PD has a latency that depends on the window size adopted (×**G2**). The technique maintains case descriptors in memory for some time, consequently running over cases more than once. However, these latter aspects can be partially controlled with the hyperparameters configurations, either minimizing the number of runs (–**G3**) or leveraging accuracy (–**G4**).

On a general note, it is clear that all methods reviewed here prioritize **G1**, as memory consumption is a key requirement when dealing with potentially infinite data streams. The same is observed in traditional data streams literature [43]. However, as seen in Table 1, there is still a need for methods that can satisfy multiple goals or provide explicit support to calibrate their balance.

The analysis we proposed offers interesting insights but is limited by the fact that we performed a literature-based review insisting on qualitative aspects. In Section 4 we take one step further by introducing quantitative methods to compare CDD approaches.

## 4 Experimental Analysis

As stated in the Process Mining Manifesto [1], it is still difficult to compare different PM tools and techniques. This problem is aggravated by the variety of goals one can consider when assessing PM. Therefore, one of the challenges in PM research is to provide reliable benchmark datasets consisting of representative scenarios. In addition, identifying quantitative metrics is a pre-requisite to implement strategies conciliating conflicting goals and optimizing *online* PM algorithms. In this Section, we contribute to these aims by proposing an experimental analysis of the five CDD tools [19], [21], [28], [32], [62] we reviewed in Section 3.2.3.

The first stage of our experimental analysis consists of identifying the goals to be assessed. We focused on the **G1-G4** (*memory-accuracy*) dimension, the most discussed in the literature. Accuracy is calculated focusing on the ability to detect drifts by the five tools we consider (Section 4.2). To make the comparison fair, we developed an ad-hoc synthetic log not previously tested by the tools (Section 4.1). The ability to limit memory consumption is assessed by executing a scalability analysis of their memory usage (Section 4.3). Observing how memory consumption varies with logs of increasing sizes provides us with a means for comparing tools running under different software frameworks. The results we obtained are finally discussed in Section 4.4.

### 4.1 Incorporating Drifts in a Synthetic Event Stream

Despite the availability of PM event logs from a public repository,[1] the majority of them was not created for event stream scenarios, and none fits the purpose of this study. Maaradji *et al.* [68] proposed 72 synthetic event logs for *online* PM. Although the datasets simulated concept drifts in business event logs, they do not comprehend the vast set of variables of a streaming scenario: (i) there is only one drift type explored (sudden), (ii) only one perspective (trace) and (iii) no noise was inducted. By ignoring other drift types (incremental, gradual, and recurring) and perspectives, such as time, the proposed event logs are limited for testing *online* PM techniques, i.e., they only represent a limited number of the possible scenarios in online environments. Therefore, inspired by [68], we created synthetic event logs following similar guidelines towards the exploration of additional drift configurations.

Our synthetic event logs incorporate the four drift types identified in the literature [44], articulated according to control-flow and time perspectives. The event logs are publicly available for further adoptions [69]. A business process for assessing loan applications [70] was used as the base business process. Other variants were generated by perturbing the base process with change patterns. Fig. 4 shows this initial process, using Latin letters to label the 15 activities that compose it.

In [68] the authors used twelve simple change patterns from [71] to emulate different deviations of the original base model. Table 2 show the change patterns, which consist of

1. https://data.4tu.nl/repository/collection:event_logs_synthetic

TABLE 2
Simple Control-Flow Change Patterns [68]

| Code | Simple change pattern | Category |
|---|---|---|
| cb | Make fragment skippable/non-skippable | O |
| cd | Synchronize two fragments | R |
| cf | Make two fragments conditional/sequential | R |
| cp | Duplicate fragment | I |
| lp | Make fragment loopable/non-loopable | O |
| pl | Make two fragments parallel/sequential | R |
| pm | Move fragment into/out of parallel branch | I |
| re | Add/remove fragment | I |
| rp | Substitute fragment | I |
| sw | Swap two fragments | I |

adding, removing, looping, swapping, or parallelizing fragments. Moreover, the changes are organized into three categories: insertion (I), resequentialization (R) and optionalization (O), also shown in Table 2. To create more complex drifts, we randomly combined three simple change patterns from different categories, building a composite change pattern, e.g., "IRO", which consists of the combination of insertion, resequentialization, and optionalization simple change patterns. Thus, the proposed change patterns were applied with a broader set of constraints and combinations to extend the degree of variability addressed in the benchmark. The main goal is to provide a wide range of event streams where CDD can be exhaustively represented. BPMN modeller[2] and BIMP[3] were used as supporting tools to model the process and simulate the event stream log, respectively.

All event streams share a few common characteristics: (i) the arrival rate of cases is fixed to 20 minutes, i.e., after every 20 minutes an event from a new case arrives in the stream; (ii) the time distribution between events of the same case follows a normal distribution. For baseline behavior, the mean time was set to 30 minutes, and the standard variation to 3 minutes. While for drifted behavior the mean and standard variation were 5 and 0.5 minutes, respectively; (iii) for time drifts, the model used in a single event stream is the same, i.e., the drift happens only in the time perspective; this way, we avoid introducing other factors; (iv) all drifts were created with 100, 500 or 1,000 cases; (v) noise was introduced in the event stream for all the trace drifts. The noise consisted of removing either the first or the last half of the trace. Then, different percentages were applied (5, 10, 15, and 20 percent) in relation to the total stream size. Note that standard cases were swapped for anomalous ones, this way preserving the event stream size. The drift types we injected are implemented in the following way:

- *Sudden drift*. The first half of the stream is composed of the baseline model, and the second half is composed of the drifted model. The same idea applies for trace and time drifts (for time drifts, the change is only in the time distribution and not the actual model).
- *Recurring drift*. For streams sizes of 100 traces, cases follow the division 33–33–34. The initial and the last groups come from the baseline, and the inner one is the drifted behavior, i.e., the baseline behavior starts the stream, fades after 33 traces, and reappears for the last 34 traces; the same applies for time drifts. For 500 and 1,000 traces, the division is 167–167–166 and 330–330–340, respectively.
- *Gradual drift*. One concept slowly takes place over another. This way, 20 percent of the stream was dedicated to the transition between concepts where one concept fades while the other increase it probability to be observed.
- *Incremental drift*. For the trace perspective, an intermediate model between the baseline and the drift model is required. Only complex change patterns were used because it was possible to create intermediate models from them, whereas, for simple change patterns, the same is not possible since the simple change is already the final form of drift. This way, 20 percent of the stream log was dedicated to the intermediate behavior, so the division was 40–20–40 (baseline–intermediate model–incremental drift). The same applies for the other sizes following the proportion. For the time perspective, all change patterns were used since the time drifts disregard the trace model. The transition state (20 percent of the stream log) was subdivided into four parts where standard time distribution decreases 5 minutes between them, following the incremental change of time.

When combining all drift types and perspectives, a total of 942 event streams were generated following the widely used MXML format [72]. The file names follow the pattern: [A]_[B]_[C]_[D]_[E]. The letters used to compose the event stream names refer to the following values: four drift types: $A \in$ {gradual, incremental, recurring, sudden}; two perspectives: $B \in$ {time, trace}; five noise percentage variations: $C \in$ {0, 5, 10, 15, 20}; three different number of cases: $D \in$ {100, 500, 1000}; 16 patterns: $E \in$ {baseline, cb, cd, cf, cp, IOR, IRO, lp, OIR, pl, pm, re, RIO, ROI, rp, sw}.

### 4.2 Evaluating Concept Drift Detection

As previously mentioned, our experiments used the available software for *drift detection* in PM, which includes: Bose *et al.* [21], Ostovar *et al.* [28], Tavares *et al.* [19], Yeshchenko *et al.* [32] and Zheng *et al.* [62]. Evaluating CDD methods for PM is a complex task as there are no established metrics to assess performance. However, as proposing metrics is out of the scope of this paper, we adopted two traditional regression metrics: Mean Squared Error (MSE) and Root Mean Squared Logarithmic Error (RMSLE), expressed in Equations (1) and (2). For both equations, assume that $n$ is the number of predictions, $Y$ is the predicted value, and $\hat{Y}$ is the real value. Thus, in our setup, $n$ is the number of event streams (942), $\hat{Y}_i$ is 1 (as each event stream contains one concept drift) and $Y_i$ is the predicted number of drifts for an event stream $L_i$.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{1}$$

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(Y_i + 1) - \log(\hat{Y}_i + 1))^2}. \tag{2}$$

2. https://demo.bpmn.io
3. http://bimp.cs.ut.ee

TABLE 3
MSE and RMSLE Scores for Different Approaches Using the 942 Synthetic Event Streams Proposed

| Approach | MSE ($\sigma$) | RMSLE ($\sigma$) |
|---|---|---|
| Bose *et al.* [21] | 1.34 (7.48) | 0.68 (0.16) |
| Ostovar *et al.* [28] | **0.69 (0.52)** | 0.51 (0.31) |
| Tavares *et al.* [19] | 0.95 (3.63) | **0.4 (0.33)** |
| Yeshchenko *et al.* [32] | 12.01 (16.05) | 0.94 (0.33) |
| Zheng *et al.* [62] | 6.09 (7.98) | 0.74 (0.28) |

*The standard deviation ($\sigma$) is shown in parentheses. The best performances are highlighted.*

MSE (Equation (1)) measures the average of the squares of the errors of an estimator, i.e., the distance between predicted and real values. Thus, MSE quantifies the quality of an estimator by evaluating both the variance and the bias of the predictor. RMSLE (Equation (2)) considers the logarithm of predicted and real values and is this way more robust to outliers, as the penalization for out of the curve predictions is lower. More specifically, RMSLE penalized an underestimation more than an overestimation. For both metrics, the closer to 0 a score is, the better the algorithm is performing.

Table 3 presents MSE and RMSLE scores of each tool regarding concept drift detection. Yeshchenko *et al.* [32], Zheng *et al.* [62] and Bose *et al.* [21] were the least performing methods in both metrics. All three approaches have in common the offline assessment of features. As for Bose *et al.* [21], one of the first CDD methods, the techniques applied were still preliminary. Moreover, the experiments only used standard hyperparameters, which may have impacted the performance. However, the non-adaptive behavior of the approaches comes to light because the techniques were not able to adapt itself to concept drifts. Furthermore, for Bose *et al.* [21], its placement in MSE is closer to the best-performing algorithms, while in RMSLE, it is closer to Zheng *et al.* [62]. This shows a tendency of underestimation on drift detection, as RMSLE punishes more heavily underestimations. Yeshchenko *et al.* [32] clearly had the worst performance, mainly in MSE. An important aspect of online processing is to deal with incomplete traces, which is not addressed in this method. The high standard deviation shows that the method tends to predict a massive number of drifts.

Ostovar *et al.* [28] and Tavares *et al.* [19] present an interesting relation between their performances. From the MSE perspective, Ostovar *et al.* [28] is better, but from RMSLE Tavares *et al.* [19] is better. This means that the Tavares *et al.* [19] method is more sensible as it usually mispredicts more than Ostovar *et al.* [28], according to MSE. However, Tavares *et al.* [19] mispredictions tend to overestimate the number of drifts while Ostovar *et al.* [28] tend to underestimate it, according to RMSLE. This behavior is explained by how both methods detect drifts. Ostovar *et al.* [28] is grounded in the application of a statistical test over two non-overlapping windows. Thus, the trace distribution within the two windows has to be different enough to trigger a drift from a statistical analysis. On the other hand, Tavares *et al.* [19] uses an online clustering technique (DenStream) to support the detection of new common behavior, interpreted as a drift, thus being more sensitive to change detection.

TABLE 4
MSE and RMSLE Scores for Different Approaches Per Perspective Using the 942 Synthetic Event Streams Proposed

| Perspective | Approach | MSE ($\sigma$) | RMSLE ($\sigma$) |
|---|---|---|---|
| trace | Bose *et al.* [21] | 0.98 (0.39) | 0.68 (0.14) |
| | Ostovar *et al.* [28] | **0.65 (0.49)** | 0.49 (0.31) |
| | Tavares *et al.* [19] | 0.88 (3.54) | **0.37 (0.31)** |
| | Yeshchenko *et al.* [32] | 12.53 (16.47) | 0.95 (0.33) |
| | Zheng *et al.* [62] | 6.98 (8.61) | 0.78 (0.29) |
| time | Bose *et al.* [21] | 2.76 (16.47) | 0.7 (0.21) |
| | Ostovar *et al.* [28] | **0.85 (0.56)** | 0.6 (0.28) |
| | Tavares *et al.* [19] | 1.2 (3.98) | **0.49 (0.36)** |
| | Yeshchenko *et al.* [32] | 10 (14.08) | 0.89 (0.31) |
| | Zheng *et al.* [62] | 2.6 (2.84) | 0.56 (0.19) |

*The standard deviation ($\sigma$) is shown in parentheses. The best performances are highlighted.*

It emerges that in evaluating CDD methods for *online* PM, it is crucial to determine which is more negative between underestimation and overestimation. Another important note is that no metric captures the behavior of an online PM method completely, as different metrics evaluate different aspects. Hence, a necessity for dedicated metrics for online PM is exposed by the results.

Furthermore, we analyzed drift detection according to different characteristics of the event streams. Table 4 shows the results given two perspectives: trace and time. Generally, the algorithms follow similar performances in both metrics. We can see that Yeshchenko *et al.* [32] and Zheng *et al.* [62] were the only methods with better performances when detecting time-related drifts than trace-related drifts. Though the approaches do not explicitly handle time, the process behavior is also shaped by the events' time distribution, which affects the CDD. Contrarily, the other three approaches had better performance at detecting trace-related drifts. We expected Tavares *et al.* [19] to outperform the other methods in time-related drifts as this method extracts time features from cases. RMSLE confirms this assumption while MSE does not, which also reveals how the metrics might affect interpretation. Moreover, though Ostovar *et al.* [28] had the best MSE time-related drift detection, it was only the third-best in RMSLE. For trace-related drifts, Ostovar *et al.* [28] was better in MSE ranking while Tavares *et al.* [19] was better in RMSLE.

Further, we investigated the capability of the studied approaches in detecting different drift types (Table 5). Though most techniques only claim to identify the sudden drift type, they were able to detect other types satisfactorily. A possible explanation is that even with small changes over time, at one point, the behavior will become entirely different from the reference. In any case, Bose *et al.* [21], Zheng *et al.* [62] and Ostovar *et al.* [28] were better at detecting sudden drifts than the other types. Yeshchenko *et al.* [32] and Tavares *et al.* [19] were better at detecting gradual drifts. Yeshchenko *et al.* [32] profited from the Declare constraints, correctly modeling the small changes of behavior. Tavares *et al.* [19] benefited from the constant adapting characteristic, which is implemented by the online clustering phase. As new events arrive, they slowly change the feature space, hence, gradual drifts become easier to detect. Another interesting observation is that

TABLE 5
MSE and RMSLE Scores for Different Approaches Per Drift Type Using the 942 Synthetic Event Streams Proposed

| Drift type | Approach | MSE ($\sigma$) | RMSLE ($\sigma$) |
|---|---|---|---|
| gradual | Bose *et al.* [21] | 1.07 (1.07) | 0.68 (0.13) |
| | Ostovar *et al.* [28] | 0.62 (0.64) | 0.51 (0.34) |
| | Tavares *et al.* [19] | **0.36 (0.9)** | **0.36 (0.3)** |
| | Yeshchenko *et al.* [32] | 11.73 (16.06) | 0.92 (0.33) |
| | Zheng *et al.* [62] | 6.71 (7.8) | 0.77 (0.29) |
| incremental | Bose *et al.* [21] | 1.63 (4.08) | 0.7 (0.16) |
| | Ostovar *et al.* [28] | **0.64 (0.48)** | 0.51 (0.31) |
| | Tavares *et al.* [19] | 2.47 (8.41) | **0.49 (0.4)** |
| | Yeshchenko *et al.* [32] | 12.57 (16.44) | 0.95 (0.33) |
| | Zheng *et al.* [62] | 5.71 (7.58) | 0.72 (0.28) |
| recurring | Bose *et al.* [21] | 1.88 (13.55) | 0.7 (0.13) |
| | Ostovar *et al.* [28] | **1.0 (0.0)** | 0.55 (0.14) |
| | Tavares *et al.* [19] | 1.19 (2.99) | **0.42 (0.33)** |
| | Yeshchenko *et al.* [32] | 11.94 (15.39) | 0.94 (0.32) |
| | Zheng *et al.* [62] | 7.11 (8.3) | 0.78 (0.3) |
| sudden | Bose *et al.* [21] | 0.93 (0.33) | 0.66 (0.19) |
| | Ostovar *et al.* [28] | **0.46 (0.5)** | 0.47 (0.35) |
| | Tavares *et al.* [19] | 0.6 (1.53) | **0.37 (0.31)** |
| | Yeshchenko *et al.* [32] | 12.11 (16.49) | 0.94 (0.32) |
| | Zheng *et al.* [62] | 4.62 (7.78) | 0.66 (0.24) |

*The standard deviation ($\sigma$) is shown in parentheses. The best performances are highlighted.*

TABLE 6
MSE and RMSLE Scores for Different Approaches per Noise Percentage Type Using the 942 Synthetic Event Streams Proposed

| Noise | Approach | MSE ($\sigma$) | RMSLE ($\sigma$) |
|---|---|---|---|
| 0% | Bose *et al.* [21] | 1.94 (12.37) | 0.68 (0.21) |
| | Ostovar *et al.* [28] | **0.77 (0.56)** | 0.56 (0.3) |
| | Tavares *et al.* [19] | 1.06 (3.7) | **0.44 (0.35)** |
| | Yeshchenko *et al.* [32] | 10.68 (15) | 0.9 (0.32) |
| | Zheng *et al.* [62] | 3.59 (3.39) | 0.63 (0.22) |
| 5% | Bose *et al.* [21] | 0.97 (0.16) | 0.68 (0.12) |
| | Ostovar *et al.* [28] | **0.64 (0.48)** | 0.49 (0.3) |
| | Tavares *et al.* [19] | 0.94 (5.3) | **0.36 (0.31)** |
| | Yeshchenko *et al.* [32] | 13.41 (17.28) | 0.96 (0.34) |
| | Zheng *et al.* [62] | 13.29 (15.28) | 0.94 (0.38) |
| 10% | Bose *et al.* [21] | 1.03 (0.67) | 0.69 (0.1) |
| | Ostovar *et al.* [28] | **0.65 (0.48)** | 0.49 (0.3) |
| | Tavares *et al.* [19] | 0.91 (3.09) | **0.37 (0.32)** |
| | Yeshchenko *et al.* [32] | 12.67 (16.5) | 0.95 (0.33) |
| | Zheng *et al.* [62] | 6.81 (6.23) | 0.78 (0.28) |
| 15% | Bose *et al.* [21] | 0.99 (0.31) | 0.68 (0.12) |
| | Ostovar *et al.* [28] | **0.63 (0.48)** | 0.48 (0.31) |
| | Tavares *et al.* [19] | 0.73 (1.89) | **0.36 (0.3)** |
| | Yeshchenko *et al.* [32] | 12.83 (16.64) | 0.95 (0.33) |
| | Zheng *et al.* [62] | 5.13 (4.4) | 0.72 (0.24) |
| 20% | Bose *et al.* [21] | 0.99 (0.31) | 0.68 (0.13) |
| | Ostovar *et al.* [28] | **0.63 (0.48)** | 0.49 (0.31) |
| | Tavares *et al.* [19] | 0.95 (3.23) | **0.38 (0.32)** |
| | Yeshchenko *et al.* [32] | 12.17 (15.8) | 0.94 (0.32) |
| | Zheng *et al.* [62] | 4.83 (3.67) | 0.71 (0.23) |

*The standard deviation ($\sigma$) is shown in parentheses. The best performance is highlighted.*

Ostovar *et al.* [28] shows a high decay in performance when detecting recurring drifts. Such a phenomenon is probably due to the tool detecting two sudden drifts instead of a recurring behavior. Thus, it was penalized by the scoring metrics. The same can be stated for Tavares *et al.* [19] in incremental drifts, which presents a considerably lower performance, mainly in MSE. The incremental drift is composed of several small-scale changes, which probably were detected as several drifts by the approach instead of a single one.

Most techniques were stable when tested with noisy streams, as shown in Table 6. According to MSE, Bose *et al.* [21] had a worse performance when the stream contains no noise. In other configurations, its performance is very stable. Yeshchenko *et al.* [32] and Zheng *et al.* [62] did not perform well for streams with 5 percent of anomalous cases, which might be due to configuration settings. Note that both approaches assess all traces at once, so their anomaly detection methods are impractical in online scenarios. Differently, Ostovar *et al.* [28] and Tavares *et al.* [19] worst performances are in noiseless streams. This might be due to the approaches identifying less frequent behavior as outliers, thus triggering less change points when no noise is applied. However, as the noise percentage increases, both approaches' performances increase. Moreover, Tavares *et al.* [19] readily identify anomalous or incomplete traces, which positively affect the accuracy in event streams with noise, according to RMSLE.

Regarding stream size, the approaches vary their behaviors widely, according to Table 7 (note that stream sizes are expressed in number of traces). This happens because window size parameters heavily influence change point detection. Table 7 shows that Tavares *et al.* [19] performed better as the stream size increases. Due to Tavares *et al.* [19] constantly adapting approach, smaller streams are more difficult to handle as the number of traces is not enough to characterize a drift. Contrarily, Yeshchenko *et al.* [32] and Zheng *et al.* [62] best performances are for smaller streams.

TABLE 7
MSE and RMSLE Scores for Different Approaches per Stream Size Type Using the 942 Synthetic Event Streams Proposed

| Size | Approach | MSE ($\sigma$) | RMSLE ($\sigma$) |
|---|---|---|---|
| 100 | Bose *et al.* [21] | **1 (0.2)** | 0.69 (0.07) |
| | Ostovar *et al.* [28] | **1 (0)** | 0.69 (0) |
| | Tavares *et al.* [19] | 1.42 (5.07) | 0.55 (0.35) |
| | Yeshchenko *et al.* [32] | **1 (0)** | 0.69 (0) |
| | Zheng *et al.* [62] | **1 (0)** | **0.41 (0)** |
| 500 | Bose *et al.* [21] | 1.21 (2.36) | 0.68 (0.15) |
| | Ostovar *et al.* [28] | **0.42 (0.49)** | 0.35 (0.27) |
| | Tavares *et al.* [19] | 0.75 (2.54) | **0.32 (0.28)** |
| | Yeshchenko *et al.* [32] | 3.48 (1.15) | 0.65 (0.12) |
| | Zheng *et al.* [62] | 6.58 (3.17) | 0.81 (0.18) |
| 1000 | Bose *et al.* [21] | 1.81 (12.72) | 0.68 (0.21) |
| | Ostovar *et al.* [28] | **0.65 (0.62)** | 0.44 (0.29) |
| | Tavares *et al.* [19] | 0.68 (2.66) | **0.26 (0.25)** |
| | Yeshchenko *et al.* [32] | 31.55 (13.98) | 1.31 (0.2) |
| | Zheng *et al.* [62] | 10.69 (11.57) | 0.9 (0.3) |

*The standard deviation ($\sigma$) is shown in parentheses. The best performances are highlighted.*

Ostovar *et al.* [28] dealt better with stream composed of 500 traces, which might be due to configuration settings. Interestingly, Ostovar *et al.* [28], Yeshchenko *et al.* [32] and Zheng *et al.* [62] have a standard deviation of 0 for both metrics when the stream size is 100. We noticed that Ostovar *et al.* [28] and Yeshchenko *et al.* [32] detected no drifts in all streams with 100 traces, while Zheng *et al.* [62] always detected two drifts for the same size. This explains the low standard deviation and also why Zheng *et al.* [62] has a better RMSLE, as this metric heavily punishes underestimations.

Though evaluating *online* PM methods is still a challenge for future research, our experiments supported the identification of some patterns relating each approach to its performances. The MSE and RMSLE metrics enabled to uncover the propensity to overestimation or underestimation in algorithms, but other perspectives could be investigated by adopting different metrics. In a general view, Bose *et al.* [21] presented stable results when submitted to streams with different characteristics. Such behavior is positively affected by the offline assessment of traces. Zheng *et al.* [62] was more affected by different perspectives, meaning that it performs better for specific scenarios. The same phenomenon was observed in Yeshchenko *et al.* [32], though its overall performance is weaker when compared to the other methods. Ostovar *et al.* [28] had the best overall MSE and generally was not very affected by different stream configurations. The same applies to Tavares *et al.* [19], which overall had the best RMSLE scores. Regarding drift types, most approaches state that they can detect only sudden drifts. However, our experiments demonstrated that detecting other drifts is feasible, meaning that different drift types have commonalities within them. The change-point detection of recurring drift is confirmed as the most challenging.

### 4.3 Scalability Analysis on Memory Consumption

The experiments went further investigating memory consumption using a quantitative method. Accurately profiling memory is a difficult task, as the evaluated tools are available in different formats and languages. Tavares *et al.* [19], Yeshchenko *et al.* [32] and Zheng *et al.* [62] are available in Python code, this way, their memory consumption is measured by profiling Python methods. Ostovar *et al.* [28] is available as a standalone tool written in Java. To capture its memory consumption, we assessed the process identification generated by the execution of the tool. Finally, Bose *et al.* [21] is available as a plug-in in the ProM framework.[4] We profiled memory as the difference between the memory consumption when the plug-in is executed with the memory consumption of the framework in standby.

The absolute values that can be measured are biased by several factors and cannot be used for comparison. Thus, the solution we proposed is focused around a scalability analysis that offers us the field for the comparative evaluation of the recorded results. The goal of this experiment is to evaluate how each algorithm scales with event streams of different sizes. Five event streams with a different number of cases were used, making it possible to observe the evolving trend in memory consumption each different approach has. We performed 30 runs of all algorithms for each stream. Memory consumption was measured in megabytes (MB), while time was measured in seconds. The absolute values we recorded measure both memory and time consumption (Table 8). Memory and time consumption increases are presented using a logarithmic view in Figs. 5 and 6, respectively.

TABLE 8
Memory and Time Consumption of the Evaluated Algorithms Given Several Stream Sizes

| # Cases | Approach | Memory in MB ($\sigma$) | Time in sec. ($\sigma$) |
|---|---|---|---|
| 2,500 | Bose *et al.* [21] | 439.78 (58.1) | 9.66 (0.23) |
| | Ostovar *et al.* [28] | 494.25 (22.69) | 11.12 (0.19) |
| | Tavares *et al.* [19] | **73.68 (0.22)** | 7.11 (0.13) |
| | Yeshchenko *et al.* [32] | 209.61 (0.31) | 27.9 (0.76) |
| | Zheng *et al.* [62] | 174.35 (0.13) | **0.52 (0.004)** |
| 12,500 | Bose *et al.* [21] | 706.81 (88.89) | 52.98 (4.86) |
| | Ostovar *et al.* [28] | 1087.61 (27.77) | 29.42 (0.84) |
| | Tavares *et al.* [19] | **107.25 (0.23)** | 32.13 (0.85) |
| | Yeshchenko *et al.* [32] | 391.98 (0.01) | 321.04 (2.49) |
| | Zheng *et al.* [62] | 667.11 (0.15) | **2.67 (0.03)** |
| 25,000 | Bose *et al.* [21] | 1400.67 (147.27) | 103.16 (14.83) |
| | Ostovar *et al.* [28] | 1560.71 (42.25) | 54.13 (1.58) |
| | Tavares *et al.* [19] | **151.29 (1.03)** | 62.49 (0.93) |
| | Yeshchenko *et al.* [32] | 874.87 (1.86) | 3698.62 (24.36) |
| | Zheng *et al.* [62] | 1282.1 (0.1) | **5.43 (0.06)** |
| 37,500 | Bose *et al.* [21] | 1946.71 (158.87) | 127.6 (2.29) |
| | Ostovar *et al.* [28] | 1743.43 (61.28) | 95.13 (5.62) |
| | Tavares *et al.* [19] | **193.42 (0.26)** | 93.29 (2.01) |
| | Yeshchenko *et al.* [32] | 1432.66 (9.27) | 42491.12 (98.07) |
| | Zheng *et al.* [62] | 1890.49 (0.17) | **8.38 (0.09)** |
| 50,000 | Bose *et al.* [21] | 2330.23 (178.84) | 165.53 (9.44) |
| | Ostovar *et al.* [28] | 1963.08 (19.75) | 132.44 (3.24) |
| | Tavares *et al.* [19] | **242.7 (3.51)** | 122.4 (2.9) |
| | Yeshchenko *et al.* [32] | 1832.87 (16.66) | 487899.6 (463.11) |
| | Zheng *et al.* [62] | 2497.23 (0.14) | **11.18 (0.14)** |

*The standard deviation ($\sigma$) is shown in parentheses. The best performances are highlighted.*

Table 8 reports the mean and the standard variation of the absolute values recorded in our experiments. As the results show, there is a clear pattern where Zheng *et al.* [62] approach was the best performing time-wise while Tavares *et al.* [19] approach had the best performance memory-wise.

According to the time perspective, Zheng *et al.* [62] outperforms the other methods because (i) it applies an offline analysis, accessing the complete stream at once, and (ii) performs fewer steps in order to detect drifts. This way, it has an advantage against more robust and sophisticated methods. Regarding memory, both Zheng *et al.* [62] and Bose *et al.* [21] methods consume more memory since they are offline, and thus, they load all the events into the memory instantly. Yeshchenko *et al.* [32] does not suffer as much since it creates sub-logs, diminishing memory consumption. It is also possible to see that in smaller streams, Zheng *et al.* [62] completes the analysis using less memory than Ostovar *et al.* [28]. However, as the stream size grows, Zheng *et al.* [62] suffers from scalability issues because it loads all events at once. Tavares *et al.* [19] performed better memory-wise. This is a direct result of the method being stream grounded and consuming events only once as the stream arrives. Tavares *et al.* [19] outperforms Ostovar *et al.* [28] because the latter uses a window-based approach and passes several times over the same data, leveraging memory consumption.

4. http://www.promtools.org/

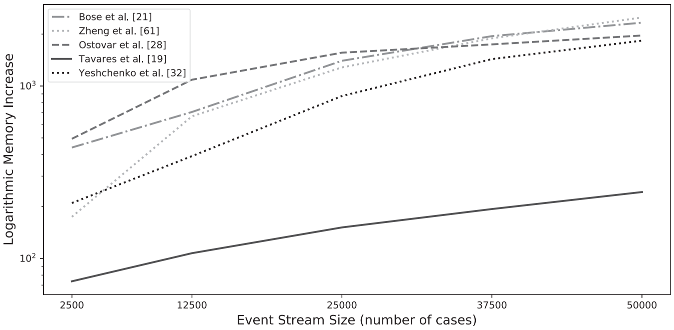


Fig. 5. Logarithmic memory consumption increase for different stream sizes.

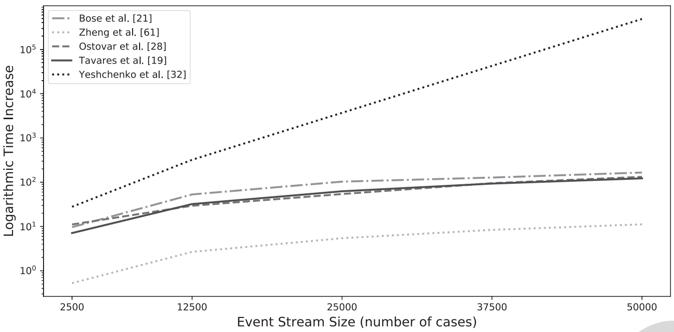


Fig. 6. Logarithmic time consumption increase for different stream sizes.

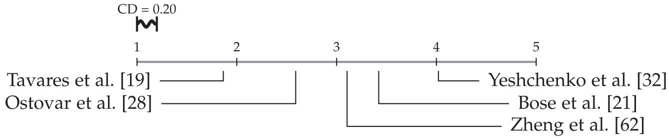


Fig. 7. Analysis of the RMSLE scores of the different methods according to the Friedman and Nemenyi test. Tavares *et al.* [19] was statistically superior to the others.

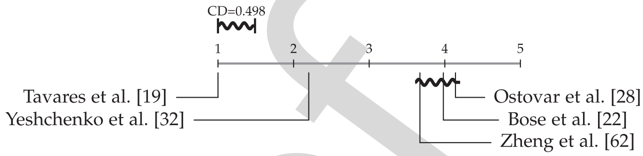


Fig. 8. Analysis of the memory consumption among the different methods according to the Friedman and Nemenyi test. Tavares *et al.* [19] consumes less memory. Yeshchenko *et al.* [32] comes next while the other approaches do not have a statistical difference.

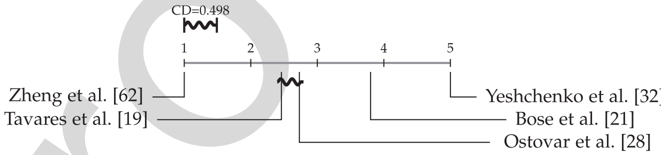


Fig. 9. Analysis of time consumption among the different methods according to the Friedman and Nemenyi test. Zheng *et al.* [62] was statistically superior to the others.

Figs. 5 and 6 show how the methods scale when dealing with larger streams. Tavares *et al.* [19] is the approach that better scales as event stream size increases, a direct result of ingesting stream events without storing them in memory. This behavior is interesting for online settings where events are expected to arrive at high rates. A similar trend was expected from Ostovar *et al.* [28] since it is also an online method, however, its behavior is similar to offline methods, which load all the events to memory at once. For time scaling, Zheng *et al.* [62] exhibited the best performance. Bose *et al.* [21], Ostovar *et al.* [28] and Tavares *et al.* [19] demonstrated a very similar behavior in time scalability. Finally, Yeshchenko *et al.* [32] showed the worst scaling performance time-wise as the method applies several processing steps, and as the data size increases, the processing time tends to increase exponentially.

### 4.4 Discussion

Although there was no hyperparameter tuning, our experiments aimed at understanding if the current solutions meet *online* PM goals. Moreover, the synthetic event logs cover a complex set of scenarios, exploring the approaches from different points of view: drift type, perspective, noise percentage, and event stream size. Furthermore, it is important to notice that Bose *et al.* [21], Yeshchenko *et al.* [32] and Zheng *et al.* [62] do not meet a key requirement (**×R0**) since the methods pre-process the event stream to create an event log.

Regarding *accuracy* the experiments provided a clear ranking of the examined methods (**G4**). This is achieved by assessing the statistical significance of the differences in their scores. For that, Friedman's statistical test and the Nemenyi post-hoc analysis were used [73]. We decide to compare the approaches using *RMSLE*, that punishes underestimation, as we think than in *online* PM reactiveness is crucial and potential changes cannot be disregarded by a monitoring algorithm. Fig. 7 shows the results of this test. If the difference between any two instances is higher than the critical difference (CD), then it can be concluded that their performance is statistically different. According to Fig. 7, there is a statistical difference between all methods, meaning that the presented ranking is maintained for all event streams tested. Tavares *et al.* [19] outperforms the other approaches statistically, corroborating with previous score results. Then, it is followed by Ostovar *et al.* [28], showing that online methods tend to perform better as they take into account event streams characteristics. Following, though Zheng *et al.* [62] and Bose *et al.* [21] performed closely in some scenarios, it can be statistically stated that Zheng *et al.* [62] has a better overall performance. Furthermore, Yeshchenko *et al.* [32] ranks as the least significant approach, which is also supported by Table 3.

We also applied the Friedman statistical test to complement our analysis of memory and time consumption (**G1**). The results are presented in Figs. 8 and 9. Tavares *et al.* [19] always perform better than the other approaches. This way, it is positioned as the best algorithm in this comparison. The next approach is Yeshchenko *et al.* [32], which diminishes memory consumption by applying sub-logs. The other methods do not present a statistical difference between them, meaning that their memory consumption is similar in these experiments. The statistical test corroborates with the analysis of Table 8 as Tavares *et al.* [19] has the best memory performance in all configurations, followed by Yeshchenko *et al.* [32] in most cases, while Bose *et al.* [21], Ostovar *et al.* [28] and Zheng *et al.* [62] change positions in different configurations.

In a general note, our experiments confirm that the online methods scale better. Offline methods tend to have scalability problems due to memory limitations. Regarding time-processing, though, the offline methods usually perform better due to having access to the complete log. However, in real streaming scenarios, their applicability is impractical.

## 5 Conclusion

This paper highlights that the current research on *online* PM lacks a shared comprehension of the requirements framing this field. Different works target different requirements, the explicit assertion of goals addressed by specific solutions is not always available, parameters and techniques for handling the trade-off between conflicting goals are rarely proposed. Developing strategies for conciliating conflicting goals, possibly at run-time, is essential to design optimal and adaptive *online* PM algorithms.

This paper contains three main contributions to progressing the field. We identified a set of goals motivating the adoption in organizations of *online* PM and underlined their conflicting relationships. As emerged in the discussion, CDD is an important pre-requirement of many stream processing approaches. We then proposed a benchmark dataset dedicated to *online* CDD, composed of a total of 942 event streams. The event streams explore different characteristics of an online scenario, such as drift types based both on trace and time perspectives, cases of varied size and noise percentage, including incomplete cases. We developed experiments to quantitatively measure *accuracy* and *memory consumption* highlighting initial insights for creating strategies to trade-off conflicting goals. The window of analysis significantly impacts all the conflicting goals we identified, therefore *adaptive and parametric methods*, connected with PM *appropriateness metrics*, are required for effectively handling CDD. The impact of drift types did not emerge as a critical issue for CDD, with the notable exception of *recurring drifts* that will require the investigation of ad-hoc techniques. Contrarily, stream size significantly affects both memory consumption and accuracy, with memory-wise methods that typically have worse accuracy for streams of small size, due to the incremental learning procedures they implement. Moreover, it emerged that within the same goal multiple propensities can be considered and *well suited metrics* are required to assess them. For example, algorithms focusing on incremental analysis tend to overestimate drifts, while algorithms exploiting statistical tests tend to underestimate them.

Our future work will focus on the definition of quantitative metrics for assessing the entire set of goals and requirements we identified and to develop more exhaustive benchmarks.

## References


[1] W. van der Aalst *et al.*, “Process mining manifesto,” in *Proc. Int. Conf. Bus. Process Manage. Workshops*, 2012, pp. 169–194.

[2] M. Zur Muehlen and R. Shapiro, “Business process analytics,” in *Handbook on Business Process Management 2*. Berlin, Germany: Springer, 2015, pp. 243–263.

[3] P. Coughlan and D. Coghlan, “Action research for operations management,” *Int. J. Oper. Prod. Manage.*, vol. 22, no. 2, pp. 220–240, 2002.

[4] A. Burattin, “Streaming process discovery and conformance checking,” in *Encyclopedia of Big Data Technologies*. Berlin, Germany: Springer, 2018.

[5] S. J. van Zelst, B. F. van Dongen, and W. M. van der Aalst, “Event stream-based process discovery using abstract representations,” *Knowl. Inf. Syst.*, vol. 54, no. 2, pp. 407–435, 2018.

[6] L. Rutkowski, M. Jaworski, and P. Duda, *Basic Concepts of Data Stream Mining*. Cham, Switzerland: Springer, 2020, pp. 13–33.

[7] A. Burattin, A. Sperduti, and W. M. van der Aalst, “Control-flow discovery from event streams,” in *Proc. IEEE Congress Evol. Comput.*, 2014, pp. 2420–2427.

[8] M. Hassani, S. Siccha, F. Richter, and T. Seidl, “Efficient process discovery from event streams using sequential pattern mining,” in *Proc. IEEE Symp. Series Comput. Intell.*, 2015, pp. 1366–1373.

[9] M. Hassani, S. J. van Zelst, and W. M. P. van der Aalst, “On the application of sequential pattern mining primitives to process discovery: Overview, outlook and opportunity identification,” *Wiley Interdisciplinary Rev. Data Mining Knowl. Discov.*, vol. 9, no. 6, 2019, Art. no. e1315. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1315

[10] V. Leno, A. Armas-Cervantes, M. Dumas, M. La Rosa, and F. M. Maggi, “Discovering process maps from event streams,” in *Proc. Int. Conf. Softw. Syst. Process*, 2018, pp. 86–95. [Online]. Available: http://doi.acm.org/10.1145/3202710.3203154

[11] F. M. Maggi, A. Burattin, M. Cimitile, and A. Sperduti, “Online process discovery to detect concept drifts in LTL-based declarative process models,” in *Proc. OTM Confederated Int. Conf. “On the Move Meaningful Internet Syst.”*, 2013, pp. 94–111.

[12] F. Stertz and S. Rinderle-Ma, “Detecting and identifying data drifts in process event streams based on process histories,” in *Proc. Int. Conf. Inf. Syst. Eng. Responsible Inf. Syst.*, 2019, pp. 240–252.

[13] A. Burattin and J. Carmona, “A framework for online conformance checking,” in *Proc. Int. Conf. Bus. Process Manage.*, 2017, pp. 165–177.

[14] A. Burattin, S. J. van Zelst, A. Armas-Cervantes, B. F. van Dongen, and J. Carmona, “Online conformance checking using behavioural patterns,” in *Proc. Int. Conf. Bus. Process Manage.*, 2018, pp. 250–267.

[15] P. Koenig, J. Mangler, and S. Rinderle-Ma , “Compliance monitoring on process event streams from multiple sources,” in *Proc. 1st Int. Conf. Process Mining*, 2019, pp. 113–120. [Online]. Available: http://eprints.cs.univie.ac.at/6066/

[16] G. M. Tavares, V. G. T. da Costa, V. E. Martins, P. Ceravolo, and S. Barbon Jr.,, “Anomaly detection in business process based on data stream mining,” in *Proc. XIV Brazilian Symp. Inf. Syst.*, 2018, pp. 16:1–16:8. [Online]. Available: http://doi.acm.org/10.1145/3229345.3229362

[17] S. J. van Zelst, A. Bolt, M. Hassani, B. F. van Dongen, and W. M. P. van der Aalst, “Online conformance checking: Relating event streams to process models using prefix-alignments,” *Int. J. Data Sci. Analytics*, vol. 8, pp. 269–284, 2019. [Online]. Available: https://doi.org/10.1007/s41060-017-0078-6

[18] G. Tello, G. Gianini, R. Mizouni, and E. Damiani, “Machine learning-based framework for log-lifting in business process mining applications,” in *Proc. Int. Conf. Bus. Process Manage.*, 2019, pp. 232–249.

[19] G. M. Tavares, S. Barbon Junior, P. Ceravolo, and E. Damiani, “Overlapping analytic stages in online process mining,” in *Proc. IEEE Int. Conf. Service Comput.*, 2019, pp. 167–175.

[20] 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 Proc. Web Conf.*, 2018, pp. 319–326. [Online]. Available: https://doi.org/10.1145/3184558.3186343

[21] R. P. J. C. Bose, W. M. P. van der Aalst, I. liobait, and M. Pechenizkiy, “Dealing with concept drifts in process mining,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 1, pp. 154–171, Jan. 2014.

[22] R. P. J. C. Bose, W. M. P. van der Aalst, I. Žliobaitė, and M. Pechenizkiy, “Handling concept drift in process mining,” in *Proc. Int. Conf. Advanced Inf. Syst. Eng.*, 2011, pp. 391–405.

[23] B. Hompes, J. C. Buijs, W. M. van der Aalst, P. Dixit, and J. Buurman, “Detecting changes in process behavior using comparative case clustering,” in *Proc. Int. Symp. Data-Driven Process Discov. Anal.*, 2015, pp. 54–75.

[24] J. Martjushev, R. P. J. C. Bose, and W. M. P. van der Aalst, “Change point detection and dealing with gradual and multi-order dynamics in process mining,” in *Proc. Int. Conf. Perspectives Bus. Informat. Res.*, 2015, pp. 161–178.

[25] 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, Oct. 2017.

[26] T. Li, T. He, Z. Wang, Y. Zhang, and D. Chu, "Unraveling process evolution by handling concept drifts in process mining," in *Proc. IEEE Int. Conf. Services Comput.*, 2017, pp. 442–449.

[27] M. Maisenbacher and M. Weidlich, "Handling concept drift in predictive process monitoring," in *Proc. IEEE Int. Conf. Services Comput.*, 2017, pp. 1–8.

[28] A. Ostovar, A. Maaradji, M. La Rosa, A. H. M. ter Hofstede, and B. F. V. van Dongen, "Detecting drift from event streams of unpredictable business processes," in *Proc. Int. Conf. Conceptual Model.*, 2016, pp. 330–346.

[29] F. Stertz and S. Rinderle-Ma , "Process histories-detecting and representing concept drifts based on event streams," in *Proc. OTM Confederated Int. Conf. "On Move Meaningful Internet Syst."*, 2018, pp. 318–335.

[30] N. Liu, J. Huang, and L. Cui, "A framework for online process concept drift detection from event streams," in *Proc. IEEE Int. Conf. Services Comput.*, 2018, pp. 105–112.

[31] F. Prathama, B. N. Yahya, D. D. Harjono, and E. Mahendrawathi, "Trace clustering exploration for detecting sudden drift: A case study in logistic process," *Procedia Comput. Sci.*, vol. 161, pp. 1122–1130, 2019.

[32] A. Yeshchenko, C. Di Ciccio, J. Mendling, and A. Polyvyanyy, "Comprehensive process drift detection with visual analytics," in *Proc. Int. Conf. Conceptual Model.*, 2019, pp. 119–135.

[33] F. Richter and T. Seidl, "Looking into the tesseract: Time-drifts in event streams using series of evolving rolling averages of completion times," *Inf. Syst.*, vol. 84, pp. 265–282, 2019.

[34] I. Firouzian, M. Zahedi, and H. Hassanpour, "Investigation of the effect of concept drift on data-aware remaining time prediction of business processes," *Int. J. Nonlinear Anal. Appl.*, vol. 10, no. 2, pp. 153–166, 2019.

[35] A. Ostovar, S. J. Leemans, and M. L. Rosa, "Robust drift characterization from event streams of business processes," *ACM Trans. Knowl. Discov. Data*, vol. 14, no. 3, pp. 1–57, 2020.

[36] S. J. van Zelst, M. Fani Sani, A. Ostovar, R. Conforti, and M. La Rosa , "Filtering spurious events from event streams of business processes," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.*, 2018, pp. 35–52.

[37] S. J. van Zelst, M. F. Sani, A. Ostovar, R. Conforti, and M. La Rosa , "Detection and removal of infrequent behavior from event streams of business processes," *Inf. Syst.*, vol. 90, 2019, Art. no. 101451.

[38] S. J. van Zelst, F. Mannhardt, M. de Leoni, and A. Koschmider, "Event abstraction in process mining: Literature review and taxonomy," *Granular Comput.*, 2020.

[39] J. D. Weerdt, M. D. Backer, J. Vanthienen, and B. Baesens, "A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs," *Inf. Syst.*, vol. 37, no. 7, pp. 654–676, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437912000464

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

[41] A. Augusto, A. Armas-Cervantes, R. Conforti, M. Dumas, M. La Rosa, and D. Reissner, "Abstract-and-compare: A family of scalable precision measures for automated process discovery," in *Proc. Int. Conf. Bus. Process Manage.*, 2018, pp. 158–175.

[42] J. Gama, P. P. Rodrigues, E. Spinosa, and A. Carvalho, "Knowledge discovery from data streams," in *Web Intelligence and Security - Advances in Data and Text Mining Techniques for Detecting and Preventing Terrorist Activities on the Web*. Amsterdam, Netherlands: IOS Press, 2010, pp. 125–138.

[43] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woniak, "Ensemble learning for data stream analysis: A survey," *Inf. Fusion*, vol. 37, pp. 132–156, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1566253516302329

[44] J. A. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 44:1–44:37, Mar. 2014. [Online]. Available: http://doi.acm.org/10.1145/2523813

[45] J. Carmona and R. Gavaldà, "Online techniques for dealing with concept drift in process mining," in *Proc. Int. Symp. Intell. Data Anal.*, 2012, pp. 90–102.

[46] M. Dumas and L. García-Bañuelos, "Process mining reloaded: Event structures as a unified representation of process models and event logs," in *Proc. Int. Conf. Appl. Theory Petri Nets Concurrency*, 2015, pp. 33–48.

[47] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: A review," *ACM SIGMOD Rec.*, vol. 34, no. 2, pp. 18–26, 2005.

[48] J. Gama and P. P. Rodrigues, "Data stream processing," in *Learning from Data Streams*. Berlin, Germany: Springer, 2007, pp. 25–39.

[49] D. Redlich, T. Molka, W. Gilani, G. S. Blair, and A. Rashid, "Scalable dynamic business process discovery with the constructs competition miner," in *Proc. 4th Int. Symp. Data-driven Process Discov. Anal.*, 2014, pp. 91–107.

[50] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Scalable process discovery and conformance checking," *Softw. Syst. Model.*, vol. 17, no. 2, pp. 599–631, May 2018. [Online]. Available: https://doi.org/10.1007/s10270–016-0545-x

[51] V. G. T. da Costa *et al.*, "Strict very fast decision tree: A memory conservative algorithm for data stream mining," *Pattern Recognit. Lett.*, vol. 116, pp. 22–28, 2018.

[52] D. Brzezinski and J. Stefanowski, "Reacting to different types of concept drift: The accuracy updated ensemble algorithm," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 1, pp. 81–94, Jan. 2014.

[53] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Trans. Neural Netw.*, vol. 22, no. 10, pp. 1517–1531, Oct. 2011.

[54] P. Weber, P. Tino, and B. Bordbar, "Process mining in non-stationary environments," in *Proc. Eur. Symp. Artif. Neural Netw. Comput. Intell. Mach. Learn.*, 2012. [Online]. Available: https://www.i6doc.com/en/book/?GCOI=28001100967420

[55] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proc. 28th Int. Conf. Very Large Data Bases*, 2002, pp. 346–357. [Online]. Available: http://dl.acm.org/citation.cfm?id=1287369.1287400

[56] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst, "Conformance checking using cost-based fitness analysis," in *Proc. IEEE 15th Int. Enterprise Distrib. Object Comput. Conf.*, 2011, pp. 55–64.

[57] F. Taymouri and J. Carmona, "A recursive paradigm for aligning observed behavior of large structured process models," in *Proc. Int. Conf. Bus. Process Manage.*, 2016, pp. 197–214.

[58] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, Mar. 1985. [Online]. Available: http://doi.acm.org/10.1145/3147.3165

[59] G. Cormode, V. Shkapenyuk, D. Srivastava, and B. Xu, "Forward decay: A practical time decay model for streaming systems," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2009, pp. 138–149.

[60] F. M. Maggi, M. Montali, and W. M. P. van der Aalst, "An operational decision support framework for monitoring business constraints," in *Proc. Int. Conf. Fundam. Approaches Softw. Eng.*, 2012, pp. 146–162.

[61] W. Li, Y. Fan, W. Liu, M. Xin, H. Wang, and Q. Jin, "A self-adaptive process mining algorithm based on information entropy to deal with uncertain data," *IEEE Access*, vol. 7, pp. 131 681–131 691, 2019.

[62] C. Zheng, L. Wen, and J. Wang, "Detecting process concept drifts from event logs," in *Proc. On Move Meaningful Internet Syst.*, 2017, pp. 524–542.

[63] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discov. Data Mining*, 1996, pp. 226–231. [Online]. Available: http://dl.acm.org/citation.cfm?id=3001460.3001507

[64] A. Solti, L. Vana, and J. Mendling, "Time series petri net models," in *Proc. Int. Symp. Data-Driven Process Discov. Anal.*, 2015, pp. 124–141.

[65] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg, "Declarative workflows: Balancing between flexibility and support," *Comput. Sci. - Res. Develop.*, vol. 23, no. 2, pp. 99–113, 2009. [Online]. Available: https://doi.org/10.1007/s00450–009-0057-9

[66] A. Alves De Medeiros, B. Dongen van, W. Aalst van der, and A. Weijters, "Process mining : Extending the alpha-algorithm to mine short loops,", 2004.

[67] F. Cao, M. Estert, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *Proc. SIAM Int. Conf. Data Mining*, 2006, pp. 328–339.

[68] A. Maaradji, M. Dumas, M. L. Rosa, and A. Ostovar, “Fast and accurate business process drift detection,” in *Proc. 13th Int. Conf. Bus. Process Manage.*, 2015, pp. 406–422. [Online]. Available: https://eprints.qut.edu.au/83013/

[69] G. M. Tavares, S. Barbon, and P. Ceravolo, “Synthetic event streams,” 2019. [Online]. Available: http://dx.doi.org/10.21227/2kxd-m509

[70] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*. Berlin, Germany: Springer, 2013.

[71] B. Weber, M. Reichert, and S. Rinderle-Ma , “Change patterns and change support features enhancing flexibility in process-aware information systems,” *Data Knowl. Eng.*, vol. 66, no. 3, pp. 438–466, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169023X0800058X

[72] B. F. van Dongen and W. M. P. van der Aalst, “A meta model for process mining data,” in *Proc. Open Interop Workshop Enterprise Modelling Ontologies Interoperability*, 2005, pp. 309–320.

[73] J. Demšar, “Statistical comparisons of classifiers over multiple data sets,” *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Dec. 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1248547.1248548

**Paolo Ceravolo** is currently an associate professor with the Dipartimento di Informatica, Università degli Studi di Milano, Italy. His research interests include data representation and integration, business process monitoring, empirical software engineering. On these topics, he has published several scientific papers. As a data scientist, he was involved in several international research projects and innovative startups. For more information please visit: http://www.di.unimi.it/ceravolo

**Gabriel Marques Tavares** received the graduate degree from the Londrina State University (UEL), Brazil. He is currently working toward the PhD degree at the Università degli Studi di Milano, Italy. In 2014 he participated in an exchange program at the University of Michigan, Ann Arbor, Michigan. His research interests include machine learning for online process mining with particular attention to Process Discovery and Concept Drift Detection. Currently, his exploration has expanded for anomaly detection and conformance checking.

**Sylvio Barbon Junior** is currently an assistant professor with the Computer Science Department, Londrina State University (UEL), Brazil. His research interests are focused on pattern recognition and their applications, with several international dissemination achieved on topics from image processing, text mining, stream mining, and process mining. For more information please visit: http://www.barbon.com.br

**Ernesto Damiani** is currently a full professor with the Università degli Studi di Milano, Italy, where he leads the SESAR research lab, and the leader of the Big Data Initiative at the EBTIC/Khalifa University in Abu Dhabi, UAE. He is the Principal Investigator of several H2020 projects. He was a recipient of the Chester-Sall Award from the IEEE IES Society (2007). He was named ACM Distinguished Scientist (2008) and received the Stephen S. Yau Services Computing Award (2016).