A Survey on Concept Drift in Process Mining

DENISE MARIA VECINO SATO and SHEILA CRISTIANA DE FREITAS, Graduate Program in Informatics (PPGIa), Pontifícia Universidade Católica do Paraná, Brazil and Instituto Federal do Paraná, Brazil
JEAN PAUL BARDDAL and EDSON EMILIO SCALABRIN, Graduate Program in Informatics (PPGIa), Pontifícia Universidade Católica do Paraná, Brazil

Concept drift in process mining (PM) is a challenge as classical methods assume processes are in a steady-state, i.e., events share the same process version. We conducted a systematic literature review on the intersection of these areas, and thus, we review concept drift in process mining and bring forward a taxonomy of existing techniques for drift detection and online process mining for evolving environments. Existing works depict that (i) PM still primarily focuses on offline analysis, and (ii) the assessment of concept drift techniques in processes is cumbersome due to the lack of common evaluation protocol, datasets, and metrics.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computing methodologies** → **Artificial intelligence**.

Additional Key Words and Phrases: concept drift, process mining, drift detection, change point detection, adaptive process discovery

1 INTRODUCTION: BACKGROUND AND MOTIVATION

Process Mining (PM) is a growing research area that provides techniques to understand and improve processes in different application domains [29]. PM's goal is to extract non-trivial process-related information from event data (observed behavior) recorded by the information systems available. The PM area is drawing more attention because of the increasing availability of data events (more data is being generated and recorded) and the need to develop and improve business processes in fast-changing environments. In [71], van der Aalst defines three types of PM: discovery, conformance, and enhancement. Discovery aims at producing a process model that generalizes the observed behavior from the event data without any *a priori* information. The discovered process model may describe only the control-flow of events or other aspects, e.g., organizational or time. In conformance, the goal is to compare the event data with a process model (discovered or designed) to reveal discrepancies. Enhancement focuses on improving or extending the current process model

Authors' addresses: Denise Maria Vecino Sato, denise.sato@ppgia.pucpr.br, denise.sato@ifpr.edu.br; Sheila Cristiana de Freitas, freitas.sheila@ppgia.pucpr.br, sheila.freitas@ifpr.edu.br, Graduate Program in Informatics (PPGIa), Pontifícia Universidade Católica do Paraná, R. Imac. Conceição, 1155, Curitiba, Paraná, Brazil, 80215-901, Instituto Federal do Paraná, R. João Negrão, 1285, Curitiba, Paraná, Brazil, 80230-150; Jean Paul Barddal, jean.barddal@ppgia.pucpr.br; Edson Emilio Scalabrin, scalabrin@ppgia.pucpr.br, Graduate Program in Informatics (PPGIa), Pontifícia Universidade Católica do Paraná, R. Imac. Conceição, 1155, Curitiba, Paraná, Brazil, 80215-901.

based on information obtained from event data. The key elements for any PM technique are the event data and the process model.

The event data are usually recorded by information systems and can be accessed in event logs or event streams. An event log contains a historical record of occurred events for a specific process, providing information for the PM techniques in an offline manner. Event streams allow the PM techniques to operate online, accessing the events as they occurred. When comparing event streams to event logs, the main differences are that the event stream is potentially infinite, and the cases inside it can be incomplete [76]. Regardless of the format, the event data contain information about the execution of a single business process, e.g., buying items online. The process model in Figure 1 shows the sequence of activities executed in an online store using the Petri Net notation [71].
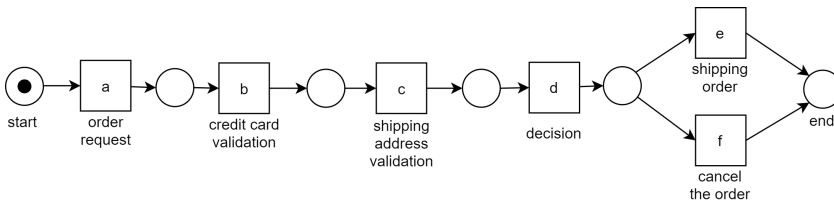

Fig. 1. Petri net model for the e-commerce process.

Each event refers to a single process instance (case), is linked to some activity of the process (e.g., order request), and must be ordered within a case. Figure 2 shows an example of event data recorded for the e-commerce process (Figure 1). Each case has an identifier and represents one execution of the process, recording what is called a trace in the event data, i.e., the sequence of events for the specific case. In the described cases, the trace $< a, b, c, d, e >$ is recorded for cases 1 and 2; but case 3 generates a different trace $< a, b, c, d, f >$. The event data in Figure 2 contains the minimum information needed for PM techniques (case identifier, activity, and timestamp), but other case or event attributes can be recorded and included in the process analysis. Because several cases can record the same trace, a simple representation of the event log contains the traces and its frequency, denoted as a superscript: $L = [< a, b, c, d, e >^2, < a, b, c, d, f >^1, ...]$.

Traces

<a,b,c,d,e>
<a,b,c,d,e>
<a,b,c,d,f>
...

Case Id	Event Id	Activity	Timestamp	...
1	1	(a) Order request	03/01/2020 10:00	
	2	(b) Credit card validation	03/01/2020 10:10	
	3	(c) Shipping address validation	03/01/2020 10:15	
	4	(d) Decision	03/02/2020 15:00	
	5	(e) Shipping order	03/03/2020 09:10	
2	6	(a) Order request	03/01/2020 15:10	
	7	(b) Credit card validation	03/01/2020 16:00	
	8	(c) Shipping address validation	03/01/2020 17:10	
	9	(d) Decision	03/01/2020 17:45	
	10	(e) Shipping order	03/02/2020 09:00	
3	11	(a) Order request	03/04/2020 18:10	
	12	(b) Credit card validation	03/04/2020 18:20	
	13	(c) Shipping address validation	03/05/2020 10:00	
	14	(d) Decision	03/05/2020 12:00	
	15	(f) Cancel order	03/05/2020 12:10	
...	...	...	...	...

Fig. 2. Example of event data.

Most existing PM methods are designed to work with event logs and consider the processes in a steady-state, i.e., all traces are related to the same version of the process. However, real-life

processes are likely to change over time in response to market dynamics, new regulations, or even improving or repairing the current process. This scenario imposes a new challenge: dealing with these changes in the process analysis. A change inside a process can be planned or unexpected, and the situation where the process is changing while being analyzed is named concept drift [10].

Concept drift is a relevant problem in different domains, e.g., business process analysis. Companies are always trying to adapt and evolve their business process to handle different situations, e.g., changes in regulations and seasonal demands, so PM techniques should consider the concept drift challenge to allow process analysis in evolving business. The PM manifesto [72] also highlights dealing with concept drift in PM as one of the core challenges for the area. However, there is no structured review of the topic describing the available published methods. This survey covers different aspects of dealing with concept drift in PM, providing an overview of the current techniques and the challenges as a contribution to the area.

The paper is organized as follows. In Section 2, we introduce the concept drift in the PM context, presenting its main challenges. Sections 3 and 4 overview current methods that can be applied to evolving processes. Section 5 discusses the main achievements of the existing approaches and highlights open challenges. Finally, Section 6 concludes the survey.

2 CONCEPT DRIFT IN PROCESS MINING

Concept drift is a well-known problem in data mining, and it refers to an online supervised learning scenario when the relation between the input data and the target variable changes over time [23, 42]. In this situation, the learning algorithms need to adapt themselves on the fly to react to concept drifts appropriately [28]. In PM, we also have a relationship between the input data, i.e., the event data, and the target variable, i.e., the process model. Nevertheless, data mining techniques, including techniques to deal with concept drift, handle structures as the target variable, as categorical or continuous values in the form of a vector as input. PM techniques deal with process models, which contain more complex structures, like concurrency, choices, and loops. Therefore, it is not possible to use the same methods developed for data mining in PM, and thus, it motivates the development of new strategies and techniques to handle concept drift in PM. To understand the proposed approaches for dealing with concept drift in PM, we conducted a Systematic Literature Review (SLR) to sustain the discoveries of this survey.

2.1 Research method

The SLR's goal is to identify the current approaches for dealing with concept drift in PM. We only included peer-reviewed documents and defined the following research questions (RQ):

- RQ1: Which approaches authors proposed to deal with concept drift detection in processes?
- RQ2: Which approaches provide tools for concept drift detection in processes?
- RQ3: How did the authors validate the proposed approaches?

The first author executed the search protocol by applying the search strings in the databases of four digital libraries, as described in Table 1. We have also included the workshops from the International Conference on PM Series (ICPM) because of their relevance to the PM area.

After obtaining the first set of papers from the digital libraries, the next step was to read and classify them based on the inclusion (IC) and exclusion criteria (EC). The ICs define the relevant studies, while the ECs guarantee the quality of the selected ones based on the pre-defined RQs [35]:

- IC1: Peer-reviewed documents written in English describing approaches for dealing with concept drift in process models using only event data as input - 88 papers.
- IC2: Peer-reviewed documents written in English comparing approaches or tools for dealing with concept drift in process models - 2 papers.

Table 1. Searches in Digital Libraries and proceedings from ICPM Workshops.

Origin	Search String	Results
ACM Digital Library	[[All: "concept drift"] OR [All: "process drift"]] AND [All: "process mining"]	20
IEEE Xplore	(("All Metadata":"concept drift" OR "process drift") AND "All Metadata":"process mining") AND [All: "process mining"]	13
Scopus	(TITLE-ABS-KEY ("concept drift" OR "process drift") AND TITLE-ABS-KEY ("process mining")) AND (LIMIT-TO (LANGUAGE , "English")) AND [All: "process mining"]	56
Springer Link	("concept drift" OR "process drift") AND "process mining" - LIMITED TO ENGLISH	140
ICPM Workshops	"concept drift"	2
Total		231

- EC1: Duplicated documents - 35 papers.
- EC2: Papers published before 2015 with less than five citations - 5 papers.
- EC3: Approaches dealing with concept drift in a business process that do not consider changes in a complete perspective of the model, e.g., control-flow or time. For instance, a predictive model for a specific time difference in the process, e.g., delivery time as the target adapted when a concept drift occurred - 6 papers.
- EC4: Approaches for dealing with concept drift in processes but reporting experiments without complete detailing, e.g., without describing the dataset configuration - 2 papers.

Finally, we applied backward snowballing [80] while following the ICs and ECs, including three papers summing up a total of 45 papers. After reading the selected papers, we organized an overview of the concept drift in PM, thus providing its main concepts. Sections 3 and 4 describe the identified approaches for dealing with concept drifts.

2.2 Concept drift definition

Concept drift research in PM focused on two directions: (i) detecting drifts and providing information for its analysis, and (ii) offering online PM techniques for dealing with evolving environments. We have also identified two papers [20, 54] comparing approaches for concept drift detection in process models, which are included in the first branch.

The concept drift in PM, also named business process drift in [45], is the situation when the process changes while being analyzed [71]. Therefore, we cannot assume that the event log or stream contains traces of a unique version of the process. Process models may include different perspectives of the process: control-flow, data, timing, or others, and the changes can also occur in any of these perspectives. Several notations exist to model business processes, and we will refer to them as process models. The concept drift in process models includes different aspects reported in the related works, summarized in Figure 3. The following topics illustrate and explain the aspects depicted above.

2.3 Perspectives of change

A process model typically contains information about different perspectives; therefore, the changes can occur in one or more of them. Authors in [9, 10] highlighted three essential perspectives in business processes: control-flow, data, and resource (also named as organizational perspective by [71]). Another vital view in PM is the time perspective, which provides performance indicators as
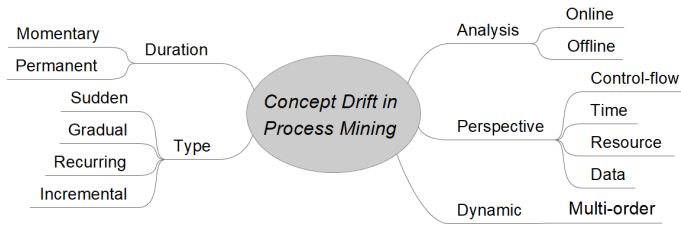
Fig. 3. Different aspects of defining and characterizing concept drift in process mining.

lead, waiting, service, and synchronization times. Despite this perspective being widely explored for process analysis, we found fewer papers exploring it for concept drifts [4, 12, 53, 62, 63, 69].

2.3.1 Control-flow. Changes in this perspective represent behavioral or structural changes in the process model. Process models can be imperative, e.g., Petri nets or BPMN, or declarative, e.g., Declare models [73]. Authors in [77] suggest a set of 13 structural change patterns for imperative business process models, organized into adding/deleting fragments, moving/replacing fragments, adding/removing levels (sub-processes), adapting control dependencies, and changing transition conditions. In a declarative process model, a structural change can be adding or deleting a constraint. Figure 4 shows two structural changes in the e-commerce process: two activities for delivery are included within a choice construction (regular or express), and the activity *shipping address validation* can now be executed in parallel with *credit card validation*.

(a) Process model before the drift.

(b) Process model after the drift.
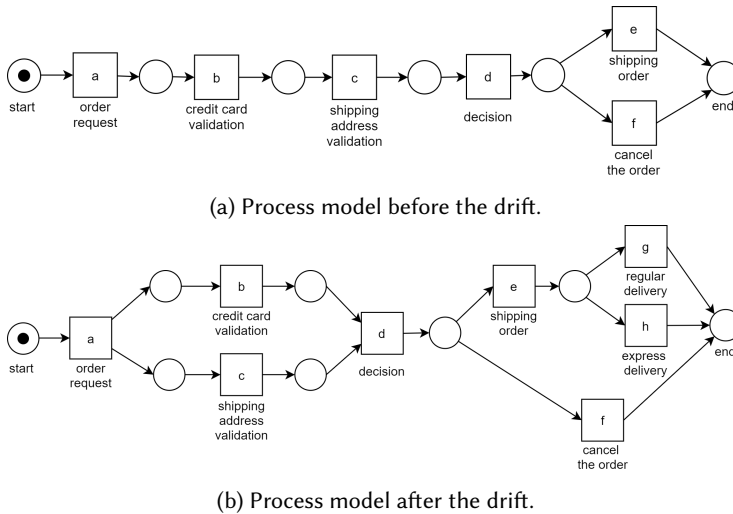
Fig. 4. Concept drift in the control-flow perspective.

The process behavior can also change without any structural modification on the model. In the e-commerce process example, customers can be classified as *silver* or *gold*, depending on their purchase history. *Gold* customers may start having express deliveries without extra charges. The process structure remains intact, but the routing of cases changes as all orders from *gold* customers are routed to the *express delivery* activity.

2.3.2   Time. This perspective concerns the timing and frequency of events in the process. Events recorded with timestamps make it possible to identify bottlenecks or measure service levels [71]. A change in this perspective deals with significant changes in process performance over time [4, 12, 53, 69]. An example can be an activity that is manually executed, e.g., credit card validation, and after a specific date is changed to be automatically performed by the system. The automatic activity can reduce its time of execution, changing the time perspective of the process.

2.3.3   Resource. A change in the resource perspective regards the influence of the resources on the execution of activities, including changes in the organization structure, roles, and resource availability. An activity may not be active until a specific resource, e.g., a machine on a production line, is available. A situation like this can change the execution paths of the process. In another process, some activities can be redirected to be executed by a different department. In the e-commerce process example, we can change the department responsible for the *decision* activity, for instance, changing the resource perspective.

2.3.4   Data. This perspective concerns the data produced or consumed by the process during the execution of its activities. A change in this perspective represents a change in the data related to the case or the event. In the e-commerce process example, it may no longer be required to attach the credit card image to execute the *credit card validation* activity. The proposals depicted in [32, 59, 68] explicitly deal with this perspective.

2.4   Duration, type, and dynamic of the change

Changes can be classified as momentary or permanent, depending on the period the change is active. Momentary changes are short-lived and affect only a few cases, whether permanent changes are persistent and stay for a while [64]. In PM, a momentary change can be understood as an outlier (or blip) that represents an unusual behavior. Typically, PM techniques filter out the outliers. Thus, the approaches to deal with concept drift usually focus only on permanent drifts. Authors in [9, 10] identified four distinct types of drifts: sudden, gradual, recurring, and incremental.

2.4.1   Type of Drift. Figure 5 shows: (a) a sudden drift, (b) a gradual drift, (c) a recurring drift, and (d) an incremental drift. The x-axis represents the time component, while the y-axis indicates distinct process models. A line inside the shaded rectangles represents a case.
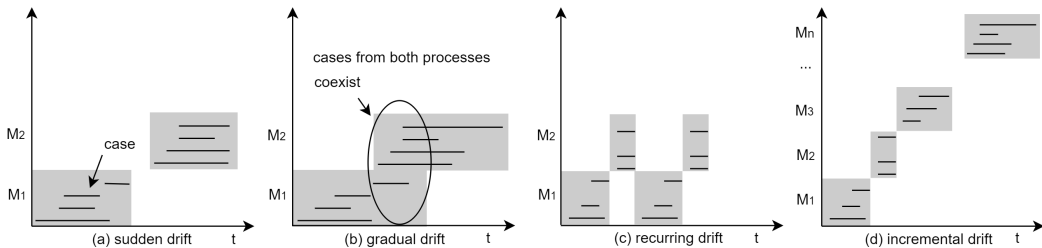
Fig. 5. Four different types of drift. Adapted from [10].

Sudden drift. When a sudden drift occurs, a new version of the process model starts to handle all the ongoing cases. It represents a complete substitution of the current process M_1 with a new one M_2, as shown in Figure 5 (a). This type of drift can occur in emergencies or even when new regulations must be followed. In a sudden drift, incomplete cases should be redirected to the new process, especially in real situations; this is also named intra-trace drift by [57]. In this situation,

a case can contain events related to different versions of the process. Several approaches handle sudden drifts, but the synthetic logs used for validation do not always contain intra-trace drifts.

Gradual drift. A gradual drift occurs when the current process M_1 is replaced by a new one M_2, but instances of both processes coexist for a while. For instance, if only the new customers of the e-commerce process should provide a photo of the identification document. As indicated in Figure 5 (b), after some time, only instances of M_2 will exist. Figure 6 illustrates different gradual drifts showing the probability of a process model emanating instances in the y-axis and time in the x-axis. We can artificially model gradual drifts in different ways by using distinct functions to describe how things grow or decay as time passes. Figure 6 (a-b) shows a gradual drift characterized by a linear change between M_1 and M_2, i.e., the cases from both processes continuously decrease and increase. The slope defines the degree of decrease/increase (Figure 6 a–b), and after t_2, all cases emanate only from M_2. Another common approach is to follow an exponential rate of increase/decrease of cases from two processes, such as depicted in Figure 6 (c–d). In this example, between t_1 and t_2, which define the beginning and the end of the drift region, M_1 emanates cases with $P[M_1] = e^{-\lambda t}$ probability and M_2 follows $P[M_2] = (1 - P[M_1])$ [51].
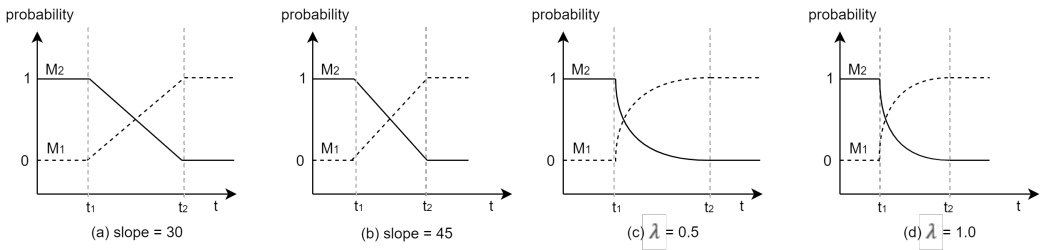

Fig. 6. (a) and (b) shows variants of gradual linear drift, with slope defining the rate of change. (c) and (d) shows variants of exponential gradual drift, characterized by the function $e^{-\lambda t}$ for M_1. Adapted from [51].

Recurring drift. Recurring drifts represent situations where a process M_1 is replaced by the process M_2, but after some time, M_1 reappears, as in Figure 5 (c). If a model, or a set of models, reappears after a while, we have a recurrent drift. It is also noteworthy to mention that models' replacements in recurring drifts can be either sudden or gradual. Furthermore, a process that changes based on seasonal influences is an example of a recurring drift.

Incremental drift. Incremental drifts represent the situation where a model M_1 is replaced to the model M_n by incremental minor changes, as illustrated in Figure 5 (d). As in the recurring drift, each change inside an incremental drift can represent a sudden or gradual drift.

Summing up, the sudden and gradual drifts can be considered basic patterns of change. Recurring and incremental can combine gradual, sudden, or even both types of drifts. In Figure 5, they are shown as discrete sudden changes.

2.4.2 *Dynamic of change.* In [51] authors described multi-order dynamic, i.e., a situation where process changes happen at different time granularity levels. For instance, let us assume a problem where a company changes its process after a four week span. Nonetheless, during such a period, the company has two variants of the process both before and after the drift. Approaches for dealing with concept drift in PM must consider multi-order dynamics, which indicates a different time

scale for the changes. In the example, we can identify micro-level changes every week and macro-level changes after four weeks. This situation is complicated when the approaches use fixed-time windows. Adaptive time windows can be more suitable for this scenario. However, in both cases, the window size hyperparameter affects drift identification.

2.5 Type of analysis

PM techniques can be executed either online or offline. In an offline analysis, data are collected in event logs, representing historical information from a time period and afterward analyzed. Most PM techniques are designed for offline analysis, and concept drift is also explored in this scenario. Possible applications for offline concept drift analysis are: (i) splitting the event log into smaller parts for a better understanding of the process, thus avoiding more complex and imprecise models; (ii) indicating unknown and probably unexpected changes in the processes, improving the process analysis; and (iii) using the results of the analysis for redesigning the processes. In an offline setting, the time for drift detection is not a vital issue, and thus, the approaches can consider data after the drift on the analysis, and the event logs can be filtered to maintain only complete traces.

An online analysis (also known as operational support [71]) provides a way to influence and to react to the ongoing cases by accessing the events as they occurred, usually as event streams, which are an infinite sequence of events generated over time [76]. An online setting usually requires some assumptions inherited from the data mining domain: data should have a fixed and small number of attributes; algorithms should be able to process unlimited data without exceeding memory restrictions; algorithms should consider a finite amount of available memory in a reasonable time; there is a small upper bound on time allowed to process an event, e.g., usually algorithms work with one pass of the data; and stream "concepts" may be stationary or evolving [16]. We need to address the concept drift problem in online mode when the presence of changes or the occurrence of drifts needs to be uncovered in real-time or in near real-time. The online analysis is appropriate if the organization is interested in reacting to a change when it is happening, using real-time alarms. For concept drift detection in PM, this means identifying the process changes as soon as possible but with confidence that the change is significant. Therefore, we can highlight two main constraints for online drift detection: accuracy and time. The online methods should handle both constraints to find a good trade-off between them.

The approaches to online detect concept drift usually define a period where events are collected to determine a reference model. After defining the reference model, the drift mechanism starts to process new events. Concept drift detection approaches should not store all events from the stream, so it is essential to define a forgetting mechanism. The methods can forget events by adopting a windowing strategy (considering the recent events on the analysis) or applying additional methods, e.g., the aging factor. Some authors proposed concept drift online detection using a stream of traces [41, 45]. However, we only considered online strategies when event streams are the input, as a stream of traces requires waiting for traces to be complete before its inclusion into the stream.

2.6 Dealing with concept drifts in processes

We identified two branches in the 45 identified papers: (i) concept drift detection; and (ii) online PM dealing with evolving environments (Table 2). Sections 3 and 4 detail the approaches.

3 CONCEPT DRIFT DETECTION

The approaches for concept drift detection usually addressed one or more challenges described in [9, 10]. We differentiate drift detection from change point (CP) detection because some approaches only detect the drift without reporting the CP [18, 31, 36, 37]. We also separate "Change localization and characterization", defined in [9, 10], as such approaches usually focus on localization.

Table 2. Classification of papers dealing with concept drift in PM.

Approach	Papers	Number of papers
Concept drift detection	[1, 4, 9, 10, 12, 18, 20, 31–33, 36, 37, 40, 41, 44–46, 49, 51, 53–57, 59, 61–63, 66–69, 82–86, 88]	38
Online PM dealing with evolving environments	[6, 13–15, 47, 60, 76]	7

(1) **Drift detection.** Detects that a process has changed without providing exact information about the time period or the trace/event the change occurred.

(2) **Change point detection.** Detects that a process has changed and identifies the time period or the exact point (event or trace) where the drift occurred. These approaches usually report the case or event identifier as the CP, but we have also identified one approach [82–85], named VDD, which reports the day as the CP.

(3) **Change localization.** Identifies the region(s) of change in the process model. The method that deals with change localization should identify the exact point inside the model where the drift occurs, e.g., between activities A and B, without needing a process model as input.

(4) **Change characterization.** Characterizes the nature of the change, defining the perspective of change and the type of drift, e.g., sudden or gradual. As several approaches deal with a specific perspective and type, e.g., sudden drifts in the control-flow perspective, this challenge is unusually reported.

(5) **Unravel process evolution or change process discovery.** Reveals the complete change process based on the identification, localization, and characterization of a change. Process analysts need tools to explore and relate all the discoveries, resulting in discovering the change process describing the drift dynamics.

Table 3 provides an overview of the drift detection approaches in PM. If the link of the public dataset is not available anymore, we do not classify it as publicly available. When the authors validate the approaches using synthetic datasets, we also reported if an objective metric is calculated. We identified two metrics in the papers: *F-score* and *detection delay*. The *F-score* is the most common metric and reports the geometric mean of the *precision* and *recall*, which rely on true positives (TP), false positives (FP), and false negatives (FN). Yet, the definition of a (*TP*), (*FP*), and (*FN*) is unclear in 14 papers. In Table 3 only the papers with the *F-score* clearly defined are marked with *CD*. For instance, if the method returns the trace index as the CP, a *TP* can consider a range of indexes around the actual drift, and this should be specified. In [40, 51], the authors clearly define the evaluation mechanism (Figure 7). The difference from other papers is that the authors apply a lag period *l* surrounding a detected or an actual drift. In [88], the same idea is applied with the name Error Tolerance (*ET*). Techniques able to detect drifts with high precision and recall are preferred over others. The *detection, average or mean delay* indicates the average number of traces/events processed by the method between the actual drift and the moment the drift is alerted [45]. It points out how early the approach is able to detect an actual change [66]. For the sake of homogeneity, hereafter, we use the *detection delay* term.
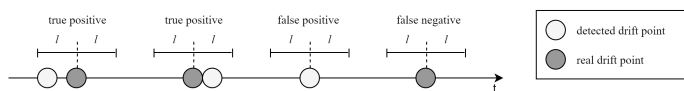

Fig. 7. Definition of evaluation metrics. Adapted from [51].

The authors in [44] report the percentage of correctly classified traces based on the known process version. The approach is based on trace clustering, and the idea is to verify if the clustering strategy correctly classifies the traces. It is not exactly a generic metric, but it works on trace clustering approaches. In [69], the authors compare the total number of detected drifts with the actual number of drifts. We do not consider this an accurate metric because it can evaluate detection accuracy with a bias; however, we reported it in Table 3. The authors in [33] report an accuracy value (from 0 to 1) without defining the metric, so we did not include it in Table 3.

We only considered online approaches those that use event streams as input. The authors in [18] propose a method that can be applied online. However, the validation applied used a stream of traces. Each trace is converted into several Parikh vectors (complete traces, one by one). The Parikh vectors should be created based on an event stream for online use. The method proposed in [41] is classified as offline because the event stream defined is, in fact, a stream of traces.

We grouped the papers by approach, and, in this case, we identified the characteristics addressed in some of the papers using citation. The two papers comparing tools for drift detection are not included and are described in Section 3.8. The papers are organized by the year of publication and title (when there is more than one paper, the ordering considers the first one), in ascending order.

Table 3 highlights that synthetic datasets containing artificially injected drifts are scarce. We only found three publicly available datasets that we further detail in Appendix A.

We organized the approaches to detect concept drift in processes based on the strategy for drift detection: statistical hypothesis testing, trace clustering, visual analysis, CP detection, change detection, trend detection, and other approaches. We classified the 36 papers in Table 4, and the most common approach is statistical hypothesis testing.

The two papers [20, 54] that compare drift detection approaches are excluded from this classification. It is important to highlight that some of the proposed methods combine other strategies before or after applying the drift detection, e.g., clustering.

The following sections describe each approach in detail. Different aspects from the distinct addressed challenge(s) addressed by each approach are also described. Section 3.8 describes the two papers that compare drift detection approaches.

3.1 Statistical hypothesis testing

Several papers use statistical-based approaches to detect concept drifts in processes. In this section, we group these approaches by the challenge addressed and the type of drift detected.

3.1.1 Detecting sudden drifts. The majority of the papers bring forward techniques that apply statistical hypothesis testing to identify significant changes in the event data, process model, or both over time, thus indicating a potential concept drift. These approaches assume that there should be a statistically significant difference in traces, process models, or both; before and after the CPs [9, 10]. The main idea is to apply a hypothesis test to confirm if there is statistical evidence indicating the two samples are equal or not. Most of the approaches using statistical hypothesis testing are tailored to deal with the first two challenges of process drift detection: drift detection and CP detection.

Figure 8 shows an overview of the papers' steps to explain the approach to detect and pinpoint a process drift (challenges 1 and 2 from Section 3) based on statistical hypothesis testing. The grey rectangles are the main steps for any standard statistical hypothesis testing that is based on event data. The options for each step were described based on the papers identified in this survey.

Table 3. Drift detection papers in PM grouped by approach. SD means synthetic datasets, RD means real-world datasets, PA means publicly available, NA means not available, and CD means clearly defined.

Paper(s)	Challenge(s)	Perspective(s)	Type(s)	Analysis	Software	Dataset	SD evaluation
[9, 10, 51]	1,2,3	Control-flow	Sudden, Gradual[10, 51]	Offline	ProM (ConceptDrift)	SD, RD (PA)	F-score CD[51]
[44]	3	Control-flow	Gradual	Offline	Tool NA	SD	% of correctly classified traces
[1]	1,2,3	Control-flow	Sudden	Offline	Tool NA	SD	-
[18]	1	Control-flow	Sudden	Online	Tool NA	SD	-
[49]	1,2	Control-flow	Sudden	Offline	Tool NA	SD (PA)	-
[45, 46, 55–57]	1,2,3	Control-flow	Sudden, Gradual[46]	Offline[45, 46], Online[55–57]	Apromore (ProDrift)	SD (PA), RD (PA)	F-score, detection delay
[32]	1,2	Control-flow, Data	Sudden	Offline	ProM (TraceClustering)	SD, RD (PA)	-
[66]	1,2,3	Control-flow	Sudden	Offline	Experimental ProM plug-in[1]	SD (PA)	F-score, detection delay
[88]	1,2	Control-flow	Sudden	Offline	TPCDD[2]	SD	F-score CD
[62, 63]	1,2,3	Time	Sudden, Incremental, Recurring	Online	ProM (Tesseract)	SD, RD (PA)	Detection delay
[4, 53, 69]	1,2	Control-flow and time combined	Not defined, Sudden [69]	Online	CDESF[3]	SD (PA), RD (PA)	N. of detected drifts [69]
[41]	1,2	Control-flow	Sudden, Recurring	Offline	Tool NA	RD (PA)	F-score

[1]Source code available at https://github.com/alexsee/processdriftdetector.
[2]Source code available at https://github.com/THUBPM/process-drift-detection. Event logs are not made available.
[3]Source code available at https://github.com/gbrltv/CDESF [4]. Updated version available in https://github.com/gbrltv/cdesf2 [53, 69].

Table 3 – continued from previous page

Paper(s)	Challenge(s)	Perspective(s)	Type(s)	Analysis	Software	Dataset	SD evaluation
[67, 68]	1,2	Control-flow [67], data[68]	Sudden, Gradual, Incremental, Recurring	Online	Tool NA	SD	-
[36, 37]	1	Control-flow	Not defined	Offline	Manual method	RD	-
[59]	1,2	Control-flow and data combined	Not defined	Offline	EDBN[4]	RD (PA)	-
[31]	1	Control-flow	Sudden	Online	StrProMCDD NA	SD, RD (PA)	F-score, detection delay
[82–85]	1,2,3,4 [84, 85]	Control-flow	Sudden, Gradual, Incremental, Recurring	Offline	VDD[5]	SD (PA), RD (PA)	F-score [83, 84]
[86]	1,2	Control-flow	Recurring	Offline	DOA[6]	SD, RD (adapted)	F-score
[61]	1,2	Control-flow	Not defined	Online	OTOSO[7]	RD (PA)	-
[12]	1,2	Control-flow and time combined	Sudden	Offline	ProM	SD	-
[40]	1,2	Control-flow	Sudden	Offline	LCDD[8]	SD (PA), RD (PA)	F-score CD
[33]	1,2,3	Control-flow	Sudden	Offline	Tool NA	SD, RD (PA)	-

---

[4]Source code available at https://github.com/StephenPauwels/edbn.
[5]Source code available at https://github.com/yesanton/Process-Drift-Visualization-With-Declare. Web client available at https://yesanton.github.io/Process-Drift-Visualization-With-Declare/client/build/.
[6]Source code available at https://github.com/zellnerlu/DOA.
[7]Source code available at https://github.com/Skarvir/OTOSO.
[8]Source code available at https://github.com/lll-lin/THUBPM.

Table 4. Classification of papers dealing with concept drift detection in PM based on the detection approach.

Approach for Drift Detection	Papers	Number of papers
Statistical hypothesis testing	[9, 10, 45, 46, 49, 51, 55–57, 59, 66]	11
Trace clustering	[1, 4, 44, 53, 62, 69, 86]	7
CP detection	[82–85]	4
Visual analysis	[12, 32, 36, 37]	4
Change detection	[18, 31, 33]	3
Trend detection	[62, 63]	2
Other approaches	[40, 41, 67, 68, 88]	5

Figure 8 overviews the possible combinations that a method can choose to identify process drifts. The following topics describe each of the main steps explaining how the authors apply the proposed approach. Finally, Table 5 details the choice for each step per paper.

Define what to compare. The statistical-based approaches for detecting drifts in processes need to define what information the statistical test should compare. This information can be obtained from event data, discovered process models, or a combination of both. Most of the approaches [9, 10, 45, 46, 49, 51, 55–57, 59] use a representation of the event data (complete traces or sets of events) to obtain relevant information about them, which is named *features* in some works. In this paper, we use the term *feature* to represent any abstraction calculated from event data or process model. Only one approach uses the actual process model as input for the statistical test. In [66], the authors obtain graph metrics from discovered process models (discovered applying the Heuristics Miner algorithm [79]): (i) number of nodes/edges; (ii) graph density; (iii) in and out-degree of each node; and (iv) occurrence of node/edges. The method summarizes the computed metrics in two vectors (reference and detection) with the size specified by the number of nodes in the graph. For drift detection, the approach uses only the occurrence of edges. The other graph metrics are used to determine the structure of the change.

In the approaches that use a representation of the event data, we identified several distinct *features* that somehow capture some characteristics from the traces or the events. Authors in [9, 10] define the concepts of *local* and *global* features. The former is calculated per trace, while the latter is calculated over a log or part of it. Both papers describe the global features relation type count (*RC*) and relation entropy (*RE*); and the local window count (*WC*) and *J measure* features. One drawback of the proposed method is that the user has to manually pick the feature to be used, implying that one has some *a priori* knowledge about the possible characteristics of the drifts to be detected. Another issue is that the defined features generate a potentially large set of high-dimensional vectors, affecting the scalability of the method for complex real-life logs or even for real-time drift detection [45]. The approach defined in [51] used the *J measure* feature former defined by [9, 10], so it shares the same drawbacks aforementioned. Authors in [49] define one global feature named event class correlation (*ecc*), which indicates if two event classes are linked. A higher *ecc* indicates that the two events commonly happen closely in the traces from the event log. On the other hand, authors in [45, 46] proposed a run-based representation of the traces and the notion of run-equivalence, derived from the field of concurrency theory [75]. The set of runs and frequencies are calculated for a given sub-log, indicating this is a global feature. A run can be understood as a representation of a set of traces considering concurrency. In [55–57], authors inherit the $\alpha+$ relations defined by the $\alpha+$ algorithm [22] to generate a contingency matrix containing the frequencies of each identified relation, obtained from sub-logs derived from a set
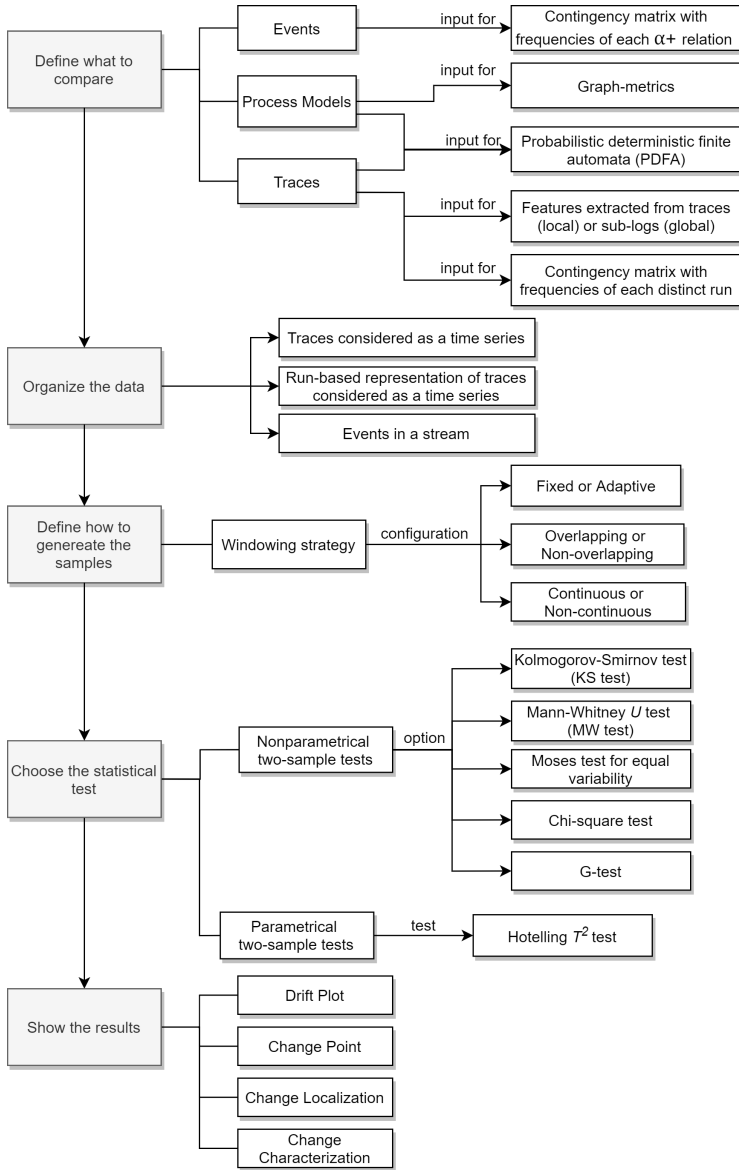
Fig. 8. Steps of statistical hypothesis testing approaches to detect concept drift in processes.

of events. The $\alpha+$ relations matrix can be considered a global feature because it is calculated over a sub-log containing a set of events, which may contain uncompleted traces. The scalability of this feature is validated using one real-life log containing 42 activities and 1,121 traces. Authors in [59] calculates a score for all the cases (after an initial period of training) based on an Extended Dynamic Bayesian Networks (*EDBN*) proposed in the same study. The case score is the mean score calculated overall events within the case and can be classified as a local feature.

Organize the data. The approaches that use the traces or discovered process models to identify the drifts must organize the input data to generate the samples for applying the statistical test. In the case of comparing complete traces, the methods consider the traces as a time series based on the timestamp of the first event. Next, the features obtained from the traces represent a dataset ordered by the trace arrival time [9, 10, 49, 51, 59]. The method applied in [45, 46] also organizes the traces as time series by their arrival time. Next, it converts the traces into runs, i.e., a representation that groups together traces with concurrent activities. As the run represents the trace, we consider it a feature obtained from the traces. The process models from [66] are also discovered based on the traces as time series. In contrast, the approach for drift detection applied in [55–57] uses an event stream, where the events are emitted according to their timestamp without the need for any particular organization. The features are calculated on a set of events ordered by their arrival time.

Organizing the event log as a time series may give rise to a tricky situation when dealing with drifts. A case may start with a version of the process, and during its execution, another version can take place (intra-trace drifts). If the traces have a long duration, they can be placed next to each other based on their arrival time, but they can be related to different versions of the process. This situation can confuse the method when identifying the exact point in time where a change begins.

Define how to generate the samples. Next, the methods should define how to generate the event samples using a windowing strategy over the dataset of features. The windows can be *fixed* or *adaptive*, *overlapping* or *non-overlapping*, *continuous* or *non-continuous*. The statistical hypothesis test is then applied in two adjacent windows to detect if and when the features significantly change (using a two-sample test), which is the drift detection itself. Usually, the two windows used by the test are named reference and detection windows. The definition of the window size, when set by the user, directly affects the accuracy of the results. The choice of the window size is critical in any drift detection approach because a small window size may lead to false positives, and a large one may lead to false negatives, thus rendering the identification of the drift location challenging [45]. This can be explained by the stability-plasticity dilemma, where the main idea is that learning a concept requires plasticity for the integration of new knowledge, but also stability in order to prevent the forgetting of previous knowledge [52]. Even the adaptive window approach proposed in [51] requires the user to establish a minimum and maximum window size, resulting in false positives or negatives, depending on the values selected. The adaptive window proposed in [45, 46] dynamically changes the window size based on the variation observed in the event log, inspired by the ADWIN method proposed in [7]. The method needs an initial window size (defined by the user), and despite the authors describe that this parameter can be empirically set, this is not well explored in the paper. When the authors compared the fixed-size window versus the adaptive-size window, they set the initial value for the adaptive approach with the same value of the fixed-window size. Still, there is an open question on how one should define this initial value. In [57], the authors use the same adaptive window approach without specifying the initial window size. In Table 5 we detail the windowing approach for each paper, describing if the window size is a parameter or not. Unfortunately, the authors in [59] do not explicitly describe how the samples are generated.

Choose the statistical test. The statistical hypothesis testing method's choice depends on the feature (scalar or vector) and the data distribution. Since the event data distribution is not *a priori* known, the authors usually choose non-parametric tests. The only exception is the *Hotelling T^2 test* used in [9, 10]. The statistical test provides the *p-value*, which is the evidence against the null hypothesis. The null hypothesis indicates that the two samples are equal; i.e., there is no drift between them: the smaller the *p-value*, the more substantial the evidence that the null hypothesis should be rejected. The rejection of the null hypothesis indicates a drift. Because the *p-value* may be under the threshold (indicating a drift) in case of noise, some approaches include a filter to

discard abrupt stochastic oscillations in the *p-value*. Authors in [45, 46, 57] named this filter as the "oscillation filter", which allows the method to detect a drift only if a given number of successive statistical tests have a p-value less the typical threshold. The main idea is that a persistent *p-value* under the threshold is much more reliable than a sporadic value happening abruptly.

Show the results. The most common way to report the results is to plot the *p-value* calculated by the statistical method (often named Drift Plot), with the trace/event index in the x-axis. The Drift Plot may also highlight the CP in the x-axis. The paper [66] only reports the CPs, and the paper [59] plots the case scores instead of the *p-value*, highlighting the drift points.

Table 5 reports the details for each of the steps defined in Figure 8 for the papers using a statistical hypothesis testing approach to detect concept drift in processes.

Both approaches [55, 56] applied the same drift detection mechanism proposed in [57]. The papers are organized by their year of publication and title, using ascending order.

3.1.2 Detecting gradual drifts. Detect gradual drifts are more challenging than sudden drifts because there is no specific point when the cases start emanating from a different version of the process. We identified three statistical-based approaches addressing gradual drifts [10, 46, 51]. The method implemented in the Concept Drift plugin [10] tailored for ProM [74] was validated using synthetic logs containing linear gradual changes. The experiments have shown the potential to identify the gradual drifts in the Drift Plot. This approach was extended in [51], using a noncontinuous sliding window to generate the samples for the statistical hypothesis testing. Therefore, there is a gap between each pair of subsequent windows, allowing for the statistical test to easily identify the drift. The main idea is that the gap will cover when both versions of the process co-exist. The maximum size of the gap has to be manually initialized by the user. This approach assumes the gap period between the samples will cover the time period where both versions of the process coexist. If this assumption is true, a statistical hypothesis test on these two samples should yield a significantly lower *p-value*, thus facilitating the detection of a change. Yet, defining a proper value for the gap parameter is not easy because, depending on how this parameter is set, some drifts may be missed.

Another drawback of this approach is that the detection of sudden and gradual drifts is not integrated, i.e., the user has to choose the type of drift to be detected. When applying the gradual drift detection in event logs containing only sudden drifts, the detected drifts are reported as gradual ones [46]. Thus, it is not possible to use this method to identify the type of the detected drift.

Another approach that deals with gradual drifts is [46], implemented in the Apromore Tool [38]. Apromore integrates the detection of sudden and gradual drifts without the need for the user to specify *a priori* the kind of drift he wants to detect. The method relies on the idea that a gradual drift (between M_1 and M_2) is an interval between two sudden drifts. The first drift indicates the point in time M_2 starts to emanate instances, and the second drift is when there are no more instances from M_1. The approach implements a post-processing stage after the detection of the sudden drifts to identify the gradual drifts. For each consecutive pair of sudden drifts detected, the approach applies another statistical test to check whether the distribution of runs inside the interval is a mixture of runs before the first drift and after the second drift. The authors assume that the distribution in the interval is a linear combination of the distributions before and after the interval and apply the Chi-square goodness-of-fit statistical test to confirm it. This test checks if a set of observations is consistent with a hypothesized distribution. The method uses the histogram of runs as a proxy for the statistical test instead of the full statistical distribution of runs over a given time interval, which is unknown. The validation uses 18 artificial logs that include the gradual drift using a linear probability function with a slope of 0.2 percent.

Table 5. Statistical hypothesis testing papers. CP means change point, DP means Drift Plot, WS means window size, and FT means feature.

Paper	Data	Organization	Samples (windowing strategy)	Statistical Test	Results
[9]	Traces	FTs from traces as time series (local or global)	Non-overlapping, continuous, fixed-size (parameter)	Kolmogorov-Smirnov, Mann-Whitney U (univariate data), Hotelling T^2 (multivariate data).	DP (trace)
[10]	Traces	FTs from traces as time series (local or global)	Non-overlapping, continuous, fixed-size (parameter)	Kolmogorov-Smirnov, Mann-Whitney U (univariate data), Hotelling T^2 (multivariate data)	Interactive DP (trace)
[49]	Traces	FTs from traces as time series (global)	Non-overlapping, continuous, fixed-size (parameter)	Mann-Whitney U, Moses for equal variability	DP (trace)
[51]	Traces	FTs from traces as time series (local or global)	Non-overlapping, continuous and non-continuous, adaptive-size (minimum and a maximum WS as parameter)	Kolmogorov-Smirnov	DP (trace) with CP
[45]	Traces	Runs as time series (global)	Non-overlapping, continuous, fixed-size (parameter) and adaptive-size (initial WS as parameter)	Chi-square	DP (trace) with CP
[57]	Events	α+ relations (events within a window build a sub-log)	Non-overlapping, continuous, adaptive-size (initial WS as parameter)	G-test	DP (trace, event) with CP
[56]	Events	α+ relations (events within a window build a sub-log)	Non-overlapping, continuous, adaptive-size (initial WS as parameter)	G-test	DP (trace, event) with CP
[66]	Process Models	Graph-metrics as time series	Non-overlapping, continuous, adaptive-size (estimated)	G-test	CPs
[46]	Traces	Runs as time series (global)	Non-overlapping, continuous, fixed-size (parameter) and adaptive-size (initial WS as parameter)	Chi-square, goodness-of-fit	Chi-square, DP (trace) with CP
[59]	Traces	Case scores based on the EDBN	Not defined	Kolmogorov-Smirnov	Case scores plot with CP
[55]	Events	α+ relations (events within a window build a sub-log)	Non-overlapping, continuous, adaptive-size (initial WS as parameter)	G-test	DP (trace, event) with CP

The performance is determined by the *F-score* and *detection delay*. A *TP* is computed if its detected interval includes the central point of the interval of the actual gradual drift. One drawback of this validation scheme is that the distribution of traces during the period when both process models coexisting can be other than linear, which is not validated. However, the method is also applied in a real-life log, the same log used by [51]. The results of the real-life log are validated by the business analyst, confirming the detected drift. In [46], the authors reported that the business analyst did not recognize the gradual drift reported in [51], meaning that the detected drift is a FP.

3.1.3 Change localization. The methods for dealing with concept drift proposed in [9, 10] provide an approach for localizing the drifts. The user can interact with the ProM software [74], using the Concept Drift plugin, and select two activities to identify whether there is a drift between them.

Authors in [56] proposed a fully automated method for characterizing drifts in a process. In the pre-processing step, the approach obtains two sets of data points containing the $\alpha+$ relations (defined by $\alpha+$ algorithm [22]) from two windows, one before and another after the drift using the drift detection approach from [57]. The size of the data points' set is based on the characterization delay (n), which is automatically defined to 500, based on performed and reported experiments. In the first stage, the approach performs a statistical test to measure the association between the detected drift and the distribution of the $\alpha+$ relations before and after it. K-sample permutation test (*KSPT*), as suggested in [27], measures the association between each $\alpha+$ relation (attributes) and the label indicating pre or post-drift (target), to discard irrelevant relations. Next, the method ranks the remaining $\alpha+$ relations by using the relative frequency change (*RFC*), calculated for each relation. There is also a filter based on the percentage of total relative frequency change (*TRFC*), which considers only a part of the ordered relations, defined as 95% based on the reported experiments. The second stage of the method matches the considered relations with their RFC with a set of pre-defined change templates and reports the best matches to the user in natural language. The approach was validated using 25 synthetic logs and using a real-life log (BPIC 2011). One limitation is that the method does not characterize simultaneous changes if they have overlapping activities. The characterization delay (n) also limits the inter-drifts distance that the method can detect. The authors preferred to provide a fully automated method, not allowing the user to change the parameters, which can also be considered a limitation. Another drawback is that the set of pre-defined templates describes changes in a low level of abstraction, e.g., adding an activity, which results in reporting a lot of low-level changes. The main consequence of the identified limitations is that the drift characterization method hardly works in a real-world environment [68].

In [55], the authors propose a new method for characterizing the drifts, also based on the drift detection mechanism presented in [57], to overcome the limitations to handle overlapping and nested changes. First, the approach discovers two process trees from an event stream (pre and post-drift) applying the Inductive Miner – partial traces (IMpt) proposed by the authors. Next, it discovers the sequence of edit operations with minimal cost to convert the pre-drift tree (P) into the post-drift tree (P') by applying a process tree transformation technique. This technique first determines the valid mappings to convert P into P', then applies the A* algorithm (described by the authors for the specific application) to find the valid mapping with minimal cost. The authors also implement an option to apply a greedy algorithm to choose the valid mapping, which is faster but provides a non-optimal approach. The implementation is available in the Apromore ProDrift plugin [55], and it was extensively validated using 365 artificial logs and two real-life logs.

3.2 Trace clustering

Approaches to detect concept drift in processes based on trace clustering analyze different cluster compositions over time. The main idea is to consider the time information obtained from traces (or

events) in the clustering stage to follow the different compositions of clusters over time. New or missing clusters indicate a change in behavior, i.e., concept drift. Usually, some traces (or events) are processed as a training step (also named "grace period" or "burnout window") to determine the default behavior before monitoring different clusters compositions. Sometimes this training step is not explicitly defined; in [61], for instance, the authors recommend, as a rule of thumb, neglecting insights from the first k cases, where k indicates the size of the hash table used to store the events.

Authors in [4, 53, 69] proposed the Concept-Drift in Event Stream Framework (CDESF), which monitors the cluster composition over time using time windows to define checkpoints (CPs). When a CP has reached, the information about activities and time intervals from the traces are updated. The method applies a forgetting mechanism to define the traces that should be considered in the update. If the cluster composition changes at the checkpoint, the method triggers a drift. For instance, if the method clusters a case previously identified as anomalous within instances of standard behavior. CDSEF combines frequency of activities and time intervals from traces as features for the clustering strategy, detecting drift in the control-flow and time perspectives combined. Authors in [1] proposed a new clustering strategy applied to the activity distance feature to detect drifts. The strategy checks each trace over time, thus identifying a potential drift when a new cluster is created (cluster cut). The activity distance is calculated between a pair of activities, so a cluster cut between two activities indicates a local drift. The authors proposed a graph showing the combined cluster cuts to provide information about more general drifts to the business analyst. In [44], the authors include the starting time of each process instance as an additional feature used by the clustering approaches. In [86], the authors apply the Local Outlier Factor (LOF) to measure non-conforming traces' distance. Next, traces are aggregated based on a sliding windowing strategy and the calculated LOF. Finally, authors in [61] propose a monitoring tool based on OPTICS to plot density-based trace clusters in process' event streams over time. It identifies temporal deviation clusters in a time-dependent graph. We identified the relevant aspects of the approaches for detecting drifts in processes based on trace clustering: defining the features, the clustering strategy, including the time component in the clustering, and the output. Table 6 describes each aspect for the identified papers. The papers are organized according to their publication year and title, in ascending order.

Features. Trace clustering techniques define a set of features obtained per trace, then group traces with similar features together, calculating the similarity between them using a metric, e.g., Euclidian distance. Thus, the approaches should define how to translate the trace into a feature vector. Authors in [44] used the Maximal Repeat Feature Set (MR) [8], obtained by counting the maximal repeats in the entire log (concatenating all the traces with a different delimiter) followed by a grouping task of traces with a similar sequence of activities incurring in drift detection on the control-flow perspective. Authors in [1] defined a feature named *activity distance* that indicates the distance between a pair of activities within a trace. This feature is calculated over the whole log as a pre-processing step and allows drift detection on the control-flow perspective. A limitation is that it cannot be used in processes containing loops. The authors in [4] defined three features that do represent the trace: edit weighted distance (*EWD*), time-weighted distance (*TWD*), and *Time* (timestamp of the last event of the trace). Both *EWD* and *TWD* are calculated using trace and time histograms. These histograms represent the current behavior in terms of frequency of activities (trace histogram) and time differences (time histogram). Both features indicate the difference between the new trace and histogram information regarding the frequency of activities or time difference. In [53, 69], the authors updated CDESF to use graph-distance trace (*GDtrace*) and graph-distance time (*GDtime*) because histograms do not consider the order relation between activities. CDESF calculates *GDtrace* and *GDtime* comparing the new trace (obtained from the arrived event) and the process graph (*PMG*) normalized. In [61], the method uses z-scoring for the

Table 6. Trace clustering considering the time component.

Paper	Features	Clustering strategy	Time	Output
[44]	Maximal Repeat Feature Set (*MR*)	Agglomerative Hierarchical Clustering (AHC) [2]	Timestamp of the 1st event within the trace as a feature.	Clusters
[1]	Activity distance	Cluster cuts algorithm (defined by the authors)	After training, each trace is processed by the clustering algorithm	Combined cluster cuts graph
[4]	Edit and time weighted distances (*EWD*, *TWD*) and time (last event's timestamp)	DBSCAN [25]	Time interval triggers the clustering, and time composes features for clustering	Flow analysis graph and snapshots of clusters
[53, 69]	Graph-distance trace and time (*GDtrace*, *GDtime*)	DenStream [17]	Time interval triggers the clustering, and time is a feature	Drift plot and snapshots of clusters
[61]	Z-scoring of temporal deviation signature (*TDS*)	OPTICS [3]	Events stored in a hash table using the Cuckoo-Hashing	OTOSO plot (time-dependent graph)
[86]	Local outlier factor (*LOF*) [11], calculated for non-conforming traces	Aggregate traces with *LOF* scores below (*T*) in one micro-cluster, when the n^o of traces exceeds *K*	The *LOF* computation is performed in the latest traces by using a fixed-size sliding window	Clusters plotted in a Gantt Chart

temporal deviation signature (*TDS*), calculated based on the mean and variance of all time intervals from each relationship obtained from the cases. The distance between the two traces is calculated using the Euclidean distance. Finally, the authors in [86] calculate the (*LOF*) [11], which describes its "outlierness" concerning the surrounding neighborhood. A difference from other approaches is that the LOF is only calculated for non-conforming traces.

Clustering strategy. Clustering algorithms group sets of data based on their similarity, maximizing intra-cluster similarity, and minimizing inter-cluster similarity [2]. They can use different techniques: partitioning, hierarchical, density-based, grid-based, and model-based. Authors in [44] apply the Agglomerative Hierarchical Clustering (AHC) [2] with the minimum variance criterion, using the Euclidian distance between feature vectors. In [1], the authors define a new algorithm to get the cluster cuts based on the variations of the activity distance (defined as an interval), a feature also defined by the authors. The metric to calculate the similarity is defined by rules based on the four possible interval changes: interval border outrun, smaller interval, no boundary changes, and observation changes. In [4], the authors use DBSCAN [25], an algorithm that starts with an arbitrary instance and expands its cluster according to density-based metrics. The algorithm expands regions until all instances are contained in a cluster, or they are considered outliers (an outlier has its density monitored). The main idea is that an increase in the number of instances within the radius of an outlier over time indicates a concept drift. The updated CDESF [53, 69] applies DenStream [17] because it is a density-based clustering method suitable for online scenarios. In [61], the proposed monitoring tool applies OPTICS [3] to derive the density-based clusters. Authors [86] aggregate traces with similar *LOF* values together without using a previously defined clustering method. They check the traces with a *LOF* score below a threshold *T*, which indicates the affiliation of a trace to a micro-cluster. When the number of traces exceeds a user-given number *K*, these traces are aggregated into a micro-cluster.

Time. The trace clustering approaches must consider the timestamp of the trace in the cluster definition to detect drifts. In [44], the authors included the timestamp of the first event within the trace as an additional feature on the definition of clusters. In the clustering strategy defined by [1], each trace is processed as a time series (based on its timestamp), and after each trace, a new cluster may be declared. Authors in [4] provided a framework where the amount of anomalous traces can be obtained over time. The time component is firstly considered by retrieving events inside a time horizon (TH) defined by a hyperparameter. By the end of the TH, the framework updates its memory component and triggers the trace and time histograms update. Each case is represented by a triplet [*EWD* – edit-weighted distance, *TWD* – time-weighted distance, *Time* – global time], used in the clustering strategy. *EWD* is calculated using the trace histogram and *TWD* the time histogram. Global time concerns the last event of a given case. The first CDESF version [4] also includes time by the feature vector [*EWD*, *TWD*, *Time*]. In the CDESF updated version, the end of the TH triggers the update of the process graph, indicating the current behavior. The tuple [*GDtrace* – trace distance, *GDtime* – trace time distance] represents each case, and it is used in the clustering strategy. Both *GDtrace* and *GDtime* are calculated from graph distance metrics applied between the new processed trace and the current process graph. Then, the feature vector [*GDtrace*, *GDtime*], also considers time for clustering. The Cuckoo-Hashing is applied in the hash table containing cases as a helpful discarding technique for considering the time component in [61]. The hash table represents a finite set of recent cases, but some older behavior is potentially maintained because the swap operations partially regard the table. The authors in [86] include the time component by reading the traces from a stream using a fixed-size sliding window. Gradually, the latest incoming non-conforming traces have the *LOF* calculated.

Output. The output of this type of approach can be the clusters [44], a plot indicating the CP [1], or the set of cases representing the concept drift [4]. In [44], the clusters are presented, each one indicating a set of traces sharing the same behavior over time. A plot indicating the trace or event where the clustering strategy decides to insert a new cluster is provided in [1]. And the CDESF framework [4] plots all the cases considering three monitored dimensions (*EWD*, *TWD*, and *Time*), highlighting the anomalous cases. The clusters that had an increase in the number of samples within the radius of an anomalous one can be identified by the user interaction, indicating a concept drift. In [53, 69] CDESF outputs a drift plot and a snapshot of the cluster formation. The OTOSO plot is the output provided in [61], which is a time-based graph that allows the detection of different structural changes in an event stream by analyzing the clusters and their connections over time. Authors in [86] show the micro-clusters using a Gantt chart. The drifts can be visualized when new micro-clusters appeared in the plot, and the x-axis reports the CP.

CDESF [4, 53, 69] is an approach based on trace clustering that detects concept drifts in the control-flow and time perspectives online. The following aspects support online analysis: the input is an event stream, it handles incomplete traces, it defines a forgetting mechanism, and it is update version uses DenStream clustering algorithm [17], which is suitable for online scenarios [69]. OTOSO [61] is also a method for online concept drifts detection in the control-flow perspective. Authors apply a Cuckoo-Hashing forgetting mechanism and handle incomplete traces. They apply OPTICS [3] to derive the clusters and use a time-based graph to show the different structural changes. On the other hand, the approach proposed in [1] is offline. It relies on the user-given window size choice, as a low window size leads to false positives, and larger windows lead to false negatives (undetected drifts). Besides, the method is not designed to deal with loops [45]. The *Dynamic Outlier Aggregation* [86] is also classified as offline because it assumes that the streaming events are already gathered to traces in a stream of traces. The validation of CDESF using synthetic datasets does not apply an accuracy metric, e.g., F-score. In [20], the authors evaluate CDESF

applying MSE and RMSLE. These metrics are not suitable for evaluating concept drift detection accuracy in our understanding because they only indicate if the number of drifts detected is close to the number of actual drifts, while their actual positions are disregarded.

3.3 Change point detection

CP detection methods identify the points in which multivariate time series, showing changes in their values [70]. Authors in [82–85] apply a CP detection method named Pruned Exact Linear Time (PELT) [34], which is indicated for datasets with limited size when the number of CPs is not *a priori* known. The Visual Drift Detection (VDD) applies PELT in a pre-defined measure calculated over Declare constraints derived from the log. Firstly, the method mines the log deriving the complete Declare constraints alphabet (*#cns* is the total of constraints). Next, it splits the log using a fixed-size sliding window and calculates the confidence (or other measure) of each Declare constraint for each window, generating several time series $T_i = (\text{Conf}_{i,1}, \dots, \text{Conf}_{i,\#\text{win}})$. T_i contains the confidence value for the constraint i over time (*#win* is the total of windows). Other metrics can be used, as the support metric, defined in [83]. The method derives a multivariate time series $D = \{T_1, T_2, \dots, T_{\#\text{cns}}\}$ representing the full spectrum of constraints' confidence.

For drift and CP detection, the method applies the PELT algorithm in D, identifying the CPs where a general change in the constraints' behavior occurred. Another option is combining a clustering strategy (hierarchical clustering) for splitting D into groups of constraints with similar confidence trends, then apply PELT. The resulting clusters indicate similar behavior and allow VDD to identify local behavior changes within the clusters. The method reports the detected CPs in one Drift Map (describing the behavior in all clusters) and several Drift Charts (showing the behavior for each cluster). The Drift Chart allows the user to localize the constraints related to change and visually characterize the local drifts [82, 83] into sudden, gradual, recurring, or incremental (challenges 3 and 4). The user can also inspect if the detected CP has an outlier behavior. In [84, 85], the tool reported different measures to complement the visual analysis of Drift Charts for identifying the type of the reported drifts. VDD shows the PELT results for sudden drifts, stationarity analysis (Augmented Dickey-Fuller test) for identifying gradual and incremental drifts, and autocorrelation plots for recurring drifts. VDD also allows the visualization of the changes identified in the drifts within the process model. The tool shows the process map (graph) for the complete event log enriched with the different constraints identified for each cluster [84, 85], providing a mixed visualization.

VDD requires three parameters: window size (to split the event log), window step (value for shifting the sub-log window), and cut threshold (for the clustering strategy). One drawback of this method is that the detected CPs are sensitive to the parameter configuration because the window size determines the granularity of analysis. The method suggests values for the window size and window step in [83, 84], but this suggestion is based on a good visualization of the plots, not on tuning the accuracy of the tool. In [83, 84], the authors conclude that the window size does not introduce a significant difference in the results; however, this conclusion is not supported by the performed experiments.

The evaluation of VDD using synthetic datasets performed in [83, 84] is unclear. The authors compare VDD with Apromore [57], but the experiments do not include the complete dataset (datasets are also described using different names), and it is not clear how the authors calculate the F-score. The evaluation using real-world datasets [83, 84] reported that VDD described all types of process drifts comprehensively. Using the BPIC 2011 dataset, the experiment detected all the drifts reported in [46]. All the experiments using real-world datasets in [83, 84] provided an overview of the tool and its user interface, which is complemented in [84] by a user study performed with PM experts reporting that the "visualizations are easy to interact with and useful".

3.4 Visual analysis

Some authors propose to compare traces (or process models) using a windowing strategy and plotting the differences for visual analysis. Comparing the current time window with the previous one is performed by comparing some specific aspects from the traces without using any statistical test. These works' output is usually a plot with the aspect calculated for a set of traces over time. As a result, the business analyst must analyze the plot to identify the drifts. We have considered studies able to detect drifts in a complete perspective of the process, e.g., control-flow or time. Works comparing a specific attribute, e.g., the time interval between two activities, were not considered.

Authors in [32] propose to split the event log based on non-overlapping time windows (by events or minutes) and calculate a similarity matrix for each sub-log. One advantage of this approach is that the matrix can consider any attribute of the event log (from case or event), and an age-decay factor corrects the similarity values. The age factor is applied to reduce the similarity of events apart in time. The differences between adjacent matrices are plotted over time, and the spikes indicate potential drifts. After identifying the drifts, the authors use a clustering strategy to explain them based on previously chosen attributes. This approach is not listed in the clustering approaches because clustering is not used to detect the drift but to analyze the effect of the changing behavior.

The method proposed in [12] calculates the Earth Mover's Distance (*EMD*) based on the trace descriptors obtained for sub-logs using a sliding fixed-size window. The approach plots the *EMD* value in a heatmap showing the values for different window sizes. The user can also inspect the *EMD* value in a plot for a specific window size. Based on the *EMD* values, the user can identify potential drifts. A differential of this approach is that *EMD* can be calculated using the control-flow perspective considering frequencies, time perspective (service and sojourn), and a holistic combination of control-flow with time. This drift detection approach allows different perspectives on the process models because of *EMD*'s flexibility regarding the choice of the representation and the distance measure. The approach was experimentally evaluated using one synthetic log, showing that different control-flow types and time-dependent control-flow drifts can successfully be detected. However, further evaluation is needed for applying the approach in real-world scenarios.

In [37], the authors proposed a manual structured method for concept drift analysis based on the PM2 method [24]. The method compares processes in three levels: process model, trace, and activity levels. The comparison between process models generates a plot showing replay fitness, precision, and generalization metrics over time. Using this plot, users can identify potential drifts (without the exact location). All the metrics are calculated using a process model discovered by the interactive Data-Aware Heuristics Miner (iDHM). In [36], the authors extended the work applying four miners for performance comparison: Integer Linear Programming (ILP), the interactive Data-Aware Heuristics Miner (iDHM), the Inductive Miner (IM), and the Inductive Miner Infrequent (IMf). The trace and activity analysis can provide information about time and data perspectives but not considering the whole perspective. The authors applied the method to the route to the diagnosis of patients with endometrial cancer over fifteen years. The outputs graphical data visualizations supported discussions about process evolution and changes with domain experts.

3.5 Change detection

Authors in [18] report the use of a change detection algorithm, ADWIN [7], to identify a drift in event data. The method translates the events into an abstract representation by converting the initial traces from a stream into Parikh vectors [58]. From the Parikh vector, it is possible to derive the polyhedron ($P_{\hat{\sigma}}$). Next, the polyhedra $P_{\hat{\sigma}1}, P_{\hat{\sigma}2}, \ldots, P_{\hat{\sigma}k}$ can be derived from the k traces. Finally, the approach learn the convex-hull of the points represented by the polyhedra $P_{\hat{\sigma}1}, P_{\hat{\sigma}2}, ..., P_{\hat{\sigma}k}$, which is a representation of the log (the learned concept). The detection of the drift is then estimated by

the ADWIN method, which maintains a window W of variable size of instances, that is compared at a certain point in time with a previous window. The algorithm automatically grows its window when no change is apparent and shrinks it when data changes. The mass of the polyhedra identifies drifts by updating the W with 1s (ones) or 0s (zeroes). If the polyhedra (initial concept learned) include the new point observed in the detection stage, its internal window W is updated with 1, otherwise with 0. When the mass of the polyhedra changes significantly, ADWIN algorithms flags a change with no need to evaluate other parameters. ADWIN also provides rigorous guarantees of its performance, bounding the rates of both false positives and false negatives [58].

The main drawback of this approach is that it only detects the first drift without identifying the CP. Furthermore, its validation was performed on a small dataset with no considerably set of change patterns. However, this is the first online method to detect concept drifts in processes. This method is also complex and time-consuming since each trace is transformed into multiple prefixes, which are then randomly selected to derive the polyhedron [41]. Another consideration should be highlighted as, despite being an online method by definition, the validation performed on the paper considers a stream of traces as each trace is converted into several Parikh vectors instead of events. Therefore, to verify the method in an online setting, the Parikh vectors should be created based on an event stream.

In [31], the authors propose StrProMCDD, which applies the ADWIN method [7] to detect drift in an event stream. StrProMCDD collects a batch of events in a pruning period (fixed size defined by the user), computes the frequency list for these events, and includes the new frequency list in a temporally ordered list used by ADWIN. A frequency list is a structure containing each pair of activities observed with the respective directly follows relation's frequency. A directly follows relation indicates that an event follows another event within a trace [71]. Because ADWIN builds its observation window based on real numbers, the authors proposed different distance measures derived from the frequency lists: relation frequency distance, dependency and edge distances, activity frequency distance, routing distance, and relative importance distance.

An advantage of StrProMCDD is that it reads events in a stream in a single pass and inherits the time and memory efficiency from the ADWIN method [7]. However, the method reads the events in batches, and no validation about the impact of different sizes of these batches is reported. Also, the authors report the F-score metric in [31] without clearly defining what is considered as a drift in the method. The plots indicate the size of the adaptive window, and a sharp drop indicates a drift, but it is not clearly defined what threshold triggers a drift. The method also solely addresses the drift detection challenge without reporting the CPs.

Another approach based on a change detection method is proposed in [33]. The authors adapted a pattern-based change detection (PBCD) algorithm, named KARMATree, to detect and characterize drifts on event logs. The events are encoded as a dynamic network to be consistent with the data representation requirements of the PBCD, which represents a sequence of graphs obtained from the traces in the log (treated as time series). Each graph snapshot (G) represents a trace (T), having an edge for each activity name. One advantage of this representation is that more perspectives of the process model can be included. Yet, the authors only validate the approach using control-flow changes. The authors in [33] validate the accuracy of the KARMATree approach compared with Apromore ProDrift fixed/adaptive using Runs [45], and ProM adaptive [51] using synthetic logs. The comparison considers F-score and running times with a well-defined experimental protocol (clearly reporting that parameter *minMC* was tuned for the proposed approach). However, neither the synthetic dataset nor the source code is made publicly available.

3.6 Trend detection

Authors in [62, 63] applied a trend detection method called Tesseract adapted from the text mining area for temporal drift detection in event streams. Tesseract contains three main parts: (i) monitoring the event stream for collecting the activities' completion times; (ii) calculating a significance score based on the calculated times and an indicator function; and (iii) visualizing the results. The significance score indicates how far a new observation (time interval between activities) is from the mean value, and it is an adaption from the SigniTrend [65] approach (trend detection method from text mining area). However, because of the non-stationary stream environment, the proposed score has exponentially decaying means and variances, and then it is smoothed again (second decay factor) to be a stable indicator. Both decay factors are defined by the user and affect the drift detection. Choosing much smoothing allows the method to rely on significant drift alerts but increases the detection delay and makes it hard to detect short-term anomalies.

Tesseract is implemented as a ProM [74] plugin and considers the stream requirements: proposes an adapted Cuckoo hash-table [26] as a data structure with fast performance for look-up, updating, and deleting; controls memory consumption; deal with a stream of events; minimizes the detection delay. A limitation of Tesseract is that the approach only detects drifts in the time perspective. The reported experiments validate the approach using synthetic and real datasets, indicating that Tesseract is robust to noise and can detect sudden, incremental, and recurring drifts (only by visual inspection). The authors do not evaluate the accuracy of the method using a metric like the F-score. They evaluate the drift in the synthetic dataset by calculating the detection delay, which is the number of events between the start of the drift and the event that triggers the detection.

The Tesseract is not classified as a visual analysis approach because it adapts a trend detection method to automatically triggers the drifts when the significance score is of a minimum number of events shift the value out of the in-control limits, addressing both drift and CP detection. The output of the method is a Control chart with Tesseract values and a Gantt-Chart that plots the Tesseract values exceeding the thresholds, indicating the temporal drifts. The user has to select a pair of activities to visualize both plots, which also provides the change localization of the drift.

3.7 Other approaches

Some authors propose different approaches to deal with concept drift in processes that do not fit into any of the previous categories. These approaches are detailed below.

3.7.1 Tsinghua Process Concept Drift Detection (TPCDD). TPCDD [88], is a three-stage approach based on two process similarity algorithm, TAR [87] and BP [78], and a clustering strategy. TPCDD handles both drift and CP detection. First, TPCDD creates a relation matrix containing all relations on the traces in the lines and one column for each trace of the event log. This is a limitation of the method to handle complex and large logs. In the second stage, each line of the matrix is inspected to identify intervals containing a stable same frequency level, classified in always, never, and sometimes intervals. The intervals have a minimum size defined by a parameter named minimum relation invariance distance (*MRID*). Each cut between intervals is defined as a candidate CP. In the third stage, TPCDD applies DBSCAN [25] in the set of candidates CPs to identify the points to be reported as change. DBSCAN parameters: the maximum radius of a neighborhood (*eps*), and the minimum number of points required to form a dense region (*minPts*), are also TPCDD parameters.

The validation of the approach uses the process models from [45] but with different configurations. The authors generated 32 mixed logs with different models and sizes that were not made publicly available. The accuracy of the method is highly affected by the *MRID* and the DBSCAN main parameters (*eps*, and *minPts*), and the authors did not provide insights on how to define them. Another drawback is that the reported CP indicates the center of the cluster, which can be an

invalid trace index. The authors described a TP when the CP detected is inside a neighborhood of the precise drift timestamp regarding the F-score metric. Yet, by analyzing the plots with the results, the neighborhood is defined by a number of traces defined by the Error Tolerance (*ET*) parameter. Even though the paper describes the validation using the F-score compared to the baseline in [45], there is a lack of information that does not support the validation process. The authors define two types of relations: direct succession relation (*DSR*) and weak order relation (*WOR*), and TPCDD uses both; however, the experimental protocol did not define which one is applied. The hyperparameters used during the comparison against the baseline (*MRID*, *eps* or *minPts*) were left undefined. We consider this approach relevant as it uses a simplification of the process model (graph) to extract the relations, avoiding the bias of the discovery algorithm. Also, the clustering strategy in the candidate CPs differentiates this approach from the others identified in this survey. However, the variation trend analysis has a lack of statistical foundation.

3.7.2 Local Complete-based Drift Detection (LCDD). The approach proposed in [40] applies the local completeness (LC) property of event logs defined in [81] for drift detection. The idea is that it is possible to assert (with a defined confidence level *K*) that a log satisfies LC in a limited length. Based on this assumption, LCDD defines the minimal length (*ML*), i.e., the minimal number of traces for a log *L* to assure LC with *K*. The detection approach starts by obtaining the *direct succession* (*DS*) relations in a complete window (*CW*), which can be defined based on the calculated *ML* values. Then, it reads traces of the next detection window (*DW*), obtains the *DSs*, and identifies two features: new *DSs* and disappeared *DSs*. A new *DS* or a disappeared one after *CW* in the *DW* indicates a drift. The CW and DW have the same effect as applying sliding windows. LCDD defines the trace with the new *DS* as a CP. If a pre-existent *DS* disappears in the *DW*, LCDD considers the initial trace of *DW* as a candidate CP. LCDD combines the *stable period* (*sp*) to define the exact CP. The value of *sp* indicates the number of traces to be considered a stable period (configured by parameter). After reading *sp* traces, if there are no more disappeared *DSs*, the start of *sp* period is declared as a CP. LCDD also implements an adaptive window based on increasing the complete window size by *minWin* (parameter) in each round.

The authors in [40] performed several experiments to evaluate the accuracy of LCDD using the *ML* equation to support the definition of the *CW* parameter. The accuracy of the fixed windows obtained using synthetic logs is considerably higher than other approaches: Runs and Alpha options from Apromore ProDrift [45, 57], and TPCDD [88]. However, all the compared approaches apply sliding window strategies, but the window size parameter was set to the default value. In LCDD, the *CW* parameter was defined based on the *ML* value. Unfortunately, the papers do not report experiments applying the value calculated for LCDD (200) as the parameter for the compared approaches. There is an interesting statistical foundation in the definition of *ML*, which is a differential of LCDD when comparing it with other fixed sliding window approaches. However, the synthetic datasets used in the validation contain a fixed interval between drifts, which may not represent the real-world behavior. The adaptive approach experiment shows that the accuracy of LCDD is sensitive to the *minWin* parameter. Yet, tuning this parameter can result in higher accuracy when comparing to the default parameters applied to the compared approaches. Authors in [40] also report a real-world experiment comparing LCDD performance against the same methods applied to the synthetic datasets. Because LCDD proved to be not robust to noise, the authors apply the method defined in [21] to obtain similar results to the compared methods. In summary, the accuracy of LCDD is sensitive to the *CW* parameter (but the *ML* equation addressed this drawback), LCDD is not robust to noise, and the accuracy of the proposed adaptive window is sensitive to the parameter *minWin*.

3.7.3 Framework for "online" process concept drift detection. In [41], the authors proposed an approach based on the footprint matrixes containing the α relations between each pair of activities. First, the method generates a process model from initial traces (a parameter defines the number of traces). Then, the approach checks each new trace by extracting its footprint matrix. Next, the framework compares the matrix of the new trace with the matrix of the current model to identify differences, classified in adding existent or nonexistent activities, removing activities, and altering adjacent relationships between activities. The method applies a metric named process model precision to identify the need to remove process model activities. A difference between the matrixes indicates a drift, and the method returns the activities and the difference to localize and characterize the drift, indicating if the drift is sudden or recurring. There is a filter to consider a trace when it appears more times than a threshold set by the user to avoid noise traces.

Despite being reported as an online method, the definition of an event stream represents a stream of traces. Consequently, the method is not suitable for an online setting. The paper reports that the method is able to identify incremental and gradual drifts, but there are no details about the types of drifts handled by the algorithms or in the validation protocol. Besides, the localization and characterization returned by the algorithms are not validated by the reported experiments. The validation process is not clearly defined; the F-score definition is provided, yet, it is limited for assessment of single-drift scenarios, and the detection error is not explained in the paper. Despite the shortcomings, the approach shows a distinct method to detect drifts based on the footprint matrix and the precision of the process model.

3.7.4 Detecting concept drifts based on process histories from event streams. The approach proposed in [67] is based on the discovery of process models providing what the authors named a process history. After reading a new event, the method calculates the fitness of the updated trace (from the case of the new event) for the current model in the process history; a small fitness value (under a pre-defined threshold) triggers the discovery of a new process model, which represents a concept drift. The approach uses an adaptation of the Inductive Miner [39] for streams for discovering the process models, based on the Stream-based Process Discovery (S-BAR) architecture defined in [76]. The proposed algorithm handles memory restriction by delimiting the $trace_{map}$ structure, which is responsible for storing the k last traces updated with an event. For conformance checking, the algorithm applies the fitness based on alignments, using the cost for move on log only. One drawback of using the proposed fitness calculation is that if the algorithm discovers a generic model, the fitness will stay with high values, not triggering the insertion of a new process model in the history (no drift will be detected). This is acknowledged by the authors, which propose a periodical model discovery using all traces in the $trace_{map}$, to detect a stricter model, but this could lead to big mixed process models. Another drawback of this approach is that the validation indicates that there is no initial setup of the process history. When the first event is received from the stream, a new model is mined. As a result, the method identifies incremental concept drifts during the initial events processing, potentially generating false alarms. The method does not report the CPs, just the drift indication and its type (sudden, gradual, incremental, or recurring).

The validation of the method shows that the four types of drift have been detected. Nonetheless, the validation also indicates that the approach misses some concept drifts injected in the synthetic dataset (for the online environment). The dataset used contains a small process with simple control-flow changes, and authors do not evaluate the processing time for mining a new model or calculating the fitness after each new event is received. Despite the identified drawbacks, authors in [67] present an entirely distinct approach to detect drifts based on discovery and conformance for online environments. This work is extended in [68] to identify concept drifts from the data perspective, validated using a real-world dataset.

3.8   Comparing tools for drift detection

We identified two papers addressing drift detection without proposing a new technique. They compare existing drift detection tools and report the results. It is essential to highlight that we did not find any paper comparing online PM approaches considering evolving environments.

In the first paper [54], the authors compare Apromore ProDrift and the Concept Drift plugin from ProM considering a set of characteristics: detection method, windowing, feature selection, sensitivity, noise handling, type of drift, type of input (log or stream), and user interface. The qualitative analysis of each aspect helps describe both tools. However, there is no accuracy evaluation for the methods based on objective metrics, e.g., F-score. Both methods were applied to a real-world dataset.

Authors in [20] discuss the properties needed for online PM techniques by defining a set of evaluation goals and evaluate the fulfillment of the defined goals for identified techniques from the state-of-the-art. The paper identified the following goals for online PM: (i) minimize memory consumption, (ii) minimize response latency, (iii) minimize the number of runs, and (iv) optimize accuracy. A set of online process discovery and online conformance checking techniques have been evaluated considering the defined goals. It is essential to highlight that the papers selected in [20] do not follow the same criteria defined in our SLR, which is that the online PM approach should report a validation scenario containing drifts. An exciting conclusion of the study proposed in [20] is that concept drift detection is a central issue for online PM. However, concept drift techniques are usually not integrated with online PM tasks. This conclusion corroborates with the low number of papers identified on the branch named *online PM dealing with evolving environments*. Because of the importance of concept drift detection for the online PM, the authors compared different concept drift detection tools, limited to open-source software or tools that provide the source code. The paper reports the fulfillment of the goals defined for online PM for each tool and reports a performance evaluation, applying two metrics (from regression methods): mean squared error (*MSE*) and root mean squared logarithmic error (*RMSLE*). We argue that both metrics are not adequate to measure the accuracy of the drift detection because they consider the presence of the drift but not the CP. So, the conclusions mainly concern the drift detection challenge. The paper publicizes a new dataset including the four drift types identified in the literature (sudden, gradual, incremental, and recurring), articulated according to control-flow and time perspectives [19].

4   ONLINE PM DEALING WITH EVOLVING ENVIRONMENTS

PM techniques are usually classified into discovery, conformance checking, and enhancement [71]. All these tasks can be executed online or offline, and in both situations, concept drift can introduce a challenge. In our SLR, we identified several approaches for detecting concept drifts that can be combined with offline techniques for discovery, conformance checking, or enhancement. When the analyst identifies the drift point, it is possible to split the event log and apply traditional offline techniques on the resulted sub-logs. However, in online analysis, drift detection may not be enough. For instance, in a process discovery scenario, even if some online drift detection technique reports a drift in an event stream, there still some open issues, i.e., is it possible to mine the model from scratch considering response time limitations? How many events should be used to mine or update the new model? Some papers dealing with concept drift in PM focus on proposing online PM techniques in an evolving environment because of questions like these. We have only found seven papers classified into this branch, and we justified this number of studies (fewer than concept drift detection) because PM research still mainly focuses on offline analysis. Yet, the identified papers only describe process discovery techniques that handle concept drift in online analysis. In the approaches to adapt discovery algorithms after a drift, the challenge may not explicitly detect the drift but enhance the algorithms with adaptive or incremental abilities. We name this challenge as

streaming process discovery dealing with evolving environments. We have considered solely the works that validate the streaming discovery approach with a scenario involving concept drifts.

4.1 Streaming process discovery dealing with evolving environments

In [15], the authors propose different versions of the Heuristics Miner (HM) [79] to deal with event streams, adapting the algorithm using a moving window. Three approaches are designed for evolving environments: HM with Sliding Window (SW), HM with Lossy Counting (LC), and HM with Lossy Counting with Budget (LCB). HW with SW is the simplest approach: first, the process collects events for a given time window, derives an event log, and then applies the classical version of HM. Besides the simplicity, this approach can apply any existing PM algorithm. However, in this approach, only the more recent events are considered, with equal importance. The model update is not constantly triggered, and the algorithm must handle each event at least two times: to store it in the event log and for deriving a model update (not desired for online analysis). The HM with LC applies an adaptation of the standard LC approach [48], where the batch version of the directly-follows measure is calculated over an event stream. The algorithm interacts with three data structures: one to count the frequencies of the activities (DA), one to count the frequencies of the direct succession relations (DR), and another one to keep track of distinct cases running at the same time (DC). Because of the concept of "buckets" and the cleanup procedure (based on the maximum approximation error allowed) from LC, the algorithm forgets the old behavior. The LC strategy does not limit memory usage, which can be a problem in streaming environments. The LCB [50] addresses this issue by adapting the approximation error according to the stream and the available budget, i.e., the maximum memory that the algorithm can use. The authors compared the three algorithms using a synthetic event log simulating two concept drifts. The HM with SW reports almost always the worst model-to-model similarity. HM with LC and HM with LCB are basically equivalent, showing the capability to detect the drifts and the corrected model. Experimental results show the effectiveness of control-flow discovery algorithms for streams on a real dataset.

In [47], the authors also propose a framework for the online discovery of declarative process models from streaming event data. Two algorithms for frequency counting are implemented in ProM [74]: SW and LC. The authors validate both algorithms using a synthetic log containing sudden drifts. The authors designed three algorithms, each tailored to identify one kind of change template in the Declare constraints: response and not response constraints, precedence and not precedence constraints, and responded existence and not responded existence constraints. The approach is extended in [14], by including the LCB algorithm for frequency counting. The discovery algorithms are also adapted to apply two different notions of constraint support: event-based and trace-based, and they are now able to discover the entire Declare language. The output of the updated discovery algorithms is also enhanced by providing an updated picture of the process model at runtime (LTL-based business constraints) and information about the most significant concept drifts identified during the process execution. The authors performed different experiments with a larger set of synthetic logs and with a real-life event log (public available). The complete implementation is available in the StreamDeclareDiscovery plugin in the ProM framework [74]. Another study [13] complements the approaches mentioned above by focusing on the dynamic visualization of the Declare models over time. The concept drifts can be visually inspected using the trend line plot containing the total of Declare rules discovered over time.

Authors in [76] proposed a generic architecture, named stream-based abstract representation (*S-BAR*), which allows the adaptation of conventional discovery algorithms for online analysis. Any discovery algorithm that maps events into an abstract representation to derive a process model can be adapted by two steps: defining a function for mapping an event log to the abstract representation needed for the algorithm and maintaining a data structure tracking the abstraction's information.

S-BAR architecture defines some options for the data structure, which should be combined with a forgetting mechanism because of the stream requirements. Several instantiations of the S-BAR are implemented in plugins in ProM [74]: α-Miner, Inductive Miner, Heuristics Miner, Transition System Miner (state-based regions), and ILP (language-based regions). The Inductive Miner-based instantiation of S-BAR (based on LC) is validated using an event stream containing one gradual drift with two change patterns. The validation showed that the algorithm is able to adapt the model to the new concept, which is indicated by the replay fitness calculated over time. The experiment also showed that a data structure with more capacity implies the drift being reflected longer.

In [6], the authors adapted the Fuzzy Miner algorithm [30] to the streaming scenario. The implementation of the streaming Fuzzy Miner is integrated into the real-time business process monitoring (RTBPM) software [5], following the steps: (i) receive and store event into the database; (ii) classify the events; (iii) check if the cases are completed; and (iv) calculate log-based metrics. Specifically, one step of the original Fuzzy Miner is adapted: measuring log-based metrics. The adaptation consisted of organizing calculations to show the changes in a process model in near real-time mode (using as minimal computational resources as possible) to the user. For dealing with drifts, the authors include new metrics: drift unary significance metric, drift binary significance metric, and drift binary correlation metric. The included metrics were calculated between the log-based and derivate metrics (from the original Fuzzy Miner), so the logic for the other metrics remained almost the same. The validation applies the new method to a real-world dataset, comparing the process model generated from the original Fuzzy Miner with the one obtained with the new algorithm. The original Fuzzy Miner derives a process model with an infinite loop, which is not present in the model derived from the streaming Fuzzy Miner. Nonetheless, neither the source code or executable files were made available by the authors [5].

Another streaming process discovery algorithm is proposed in [60]. The authors adapted the Constructs Competition Miner (CCM) algorithm to the streaming environment, considering changes in the behavior. The original CCM applies a divide-and-conquer strategy to mine a block-structured process model from the footprint matrix describing directly-follow relations. In CCM, distinct calculated global relations between activities compete with each other for the most suitable solution. The discovery process also handles noise and not-supported behavior. The proposed Dynamic CCM (DCCM) [60]) splits up the original CCM into two parts: (i) footprint update and (ii) the footprint interpretation. For every event, the footprint update module updates the dynamic footprints. DCMM can then compute the dynamic footprints at any point in time be compiled to a business process by the footprint interpretation, scheduled based on a configured number of completed traces. Older behavior is handled by an aging approach applied by the footprint update module. It is created an individual trace footprint (TFP) for each trace and adds it multiplied by the trace influence factor $\texttt{tif} \in R$ to the current dynamic overall footprint (DFP) multiplied by $1 - \texttt{tif}$. With this mechanism the influence of a trace completed 60 traces ago became almost irrelevant for $\texttt{tif} = 0.1$. New activities are added to the overall footprint as the trace terminates. Activities that do not appear anymore are removed from the dynamic footprint based on a removal threshold $\texttt{tr}$. The authors [60] validate DCMM in a scenario with a sudden drift in the control-flow perspective. They analyze the change to the business process and report the behavior of the DCCM, the change detection (comparing the trace with actual drift and the detected one), and the change transition period. The reported conclusions indicate that the trace influence factor is a pre-specified value, but it depends on how many traces one must consider representing all the behavior of the process model, which renders the definition of an optimum or even fair value challenging. We did not find any paper comparing the online PM approaches focusing on concept drift scenarios.

5 ACHIEVEMENTS AND CHALLENGES

Drift detection is a hot research topic, and the authors address different challenges: drift detection, CP detection, change localization, change characterization, and revealing the changing process. Of the 45 selected papers, 38 of them (approximately 84%) deal with process drift detection, and the most addressed challenge is CP detection (29 papers). Seven papers propose adaptive process discovery techniques for dealing with evolving environments, thus depicting that the PM area is still focused on offline analysis. The most addressed problem is CP detection (Figure 9 b) in the control-flow perspective (Figure 9 a). Papers addressing more than one perspective are counted for each perspective; the same applies to the challenges plot (Figure 9 b). Regarding the drift perspective, most approaches deal with drifts in the control-flow perspective. The data perspective is explored in [32, 59, 68], and the time perspective is explored in [4, 12, 53, 62, 63, 69].

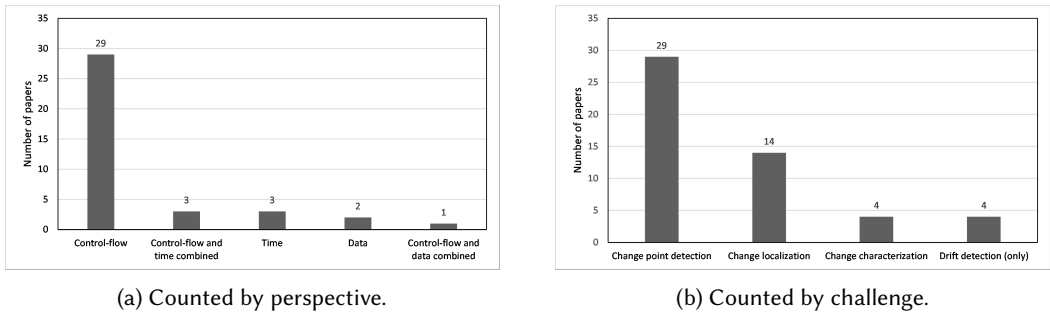(a) Counted by perspective.

(b) Counted by challenge.

Fig. 9. Drift detection papers.

Despite being the most addressed problem, it is difficult to compare the distinct approaches for drift detection in PM. The validation of the detected CP is not always performed using an objective and well-defined metric. Even when the papers report a metric, there are some open questions about how they are computed or regarding the validation protocol. For instance, from the 29 papers that propose a new technique for CP detection, only 14 papers [31, 40, 41, 45, 46, 51, 55−57, 66, 83−86] use the F-score to validate their results. Yet, the definition of what a TP, TN, FP, or FN stands for is overlooked or different from the definition provided in other papers. We understand that the F-score metric properly describes the accuracy of the detected CPs if the authors define how the true/false positives are counted. We suggest that the TPs can only score CPs detected after the real CP inside a neighborhood pre-defined. The idea of an Error Tolerance (ET), described in [88], similar to the objective evaluation using a lag period proposed in [40, 51], can be adapted to define this neighborhood. The main difference between the current proposals is that we argue that the lag period should be considered only after the real drift position and not before it, as drift detection is a reactive process rather than a preemptive one.

The two leading software that provide a tool for drift detection, Apromore [38], and ProM [74], do not report validation metrics in the user interface, e.g., F-score. This hardens the comparison between different methods. We identified only two papers [20, 54] that compare tools for drift detection in PM. The authors in [54] do not report a metric for comparing the accuracy of the detected drifts and perform the experiments using a real-life dataset, which is not the best choice when comparing approaches because the real drift points are not known *a priori*. In [20], the drift detection accuracy is compared using two metrics applied to regression models (*MSE* and *RMSLE*). In our understanding, by calculating these metrics, the authors could verify if the approaches detected the correct number of drifts, but the detected drift can be very distant from the actual

one. Therefore, we were unable to find any tool available to compare different approaches for drift detection. Providing an easy and common way of comparing different approaches is a challenge in the area. Another challenge is the set of synthetic event logs publicly available. We only identified three datasets (Appendix A) with concept drifts in event logs, and their configuration does not reproduce different drift intervals or a mixed combination of different types of drift expected in real life. The approaches can evaluate their accuracy based on real-world datasets when experts are available for validation to overcome this issue. However, we argue that the tools for generating synthetic datasets should be improved considering changes in processes for future research.

In several approaches for drift detection, the user must set different parameters, and these parameters affect the accuracy of the results. Another challenge is to provide an intuitive and interactive interface, where one can easily change the parameters to tune the results. Some approaches intend to be fully automated, using pre-defined values for the parameters [45, 46, 57]. However, for different datasets, it should be interesting to provide a way to tune the hyperparameters' values.

Most papers propose statistical tests over some features extracted from event logs or streams considering the adopted approaches. They apply the statistical tests using fixed or adaptive windows. However, the authors usually propose the adaptive approaches without rigorous guarantees of performance as bounds on the rates of false positives and false negatives. Only two approaches [18, 31] for adaptive windows use the ADWIN method [7], which provides such guarantees. We understand that statistical-based approaches can also enhance adaptive windowing strategies. Also, the application of change detector algorithms (PELT) is reported only in one approach [82–85].

The validation of online and offline approaches should follow different experimental protocols. The online setting requires an event stream and has other limitations, e.g., memory consumption and processing time. The experimental protocol of some online approaches only considers the processing time or fails to validate the complete online scenario of analysis. We argue that an event stream is a proper input for online methods because of the restrictions of the online setting, and the experimental protocol should explicitly detail how the method handles: memory, time, and accuracy restrictions. For dealing with concept drifts, the online setting imposes new challenges: the time for drift detection is a critical factor, the number of events needed after the drift occurred is essential, the number of events stored for the analysis must be considered, and the approaches should consider incomplete traces. Besides that, some approaches for online concept drift detection in PM deal with a stream of traces, waiting until the trace is complete and thus, violating the restriction to deal with incomplete traces. We classified these approaches for offline analysis.

6 CONCLUSION

Despite being a relatively new research topic in the PM community, we found different approaches to deal with concept drift in processes. We categorize the approaches into two main branches: explicitly detect the drift and adapt process mining techniques to deal with event streams in an evolving environment. The primary efforts concern the detection of the concept drift, usually handling the drift and the CP detection in the control-flow perspective. Seven studies have been found on stream process discovery dealing with evolving environments. We believe this reflects that the PM area is still focused on offline analysis, especially when the need is for understanding the real processes. The IC defined for the SLR only considers papers with online process mining approaches that evaluate the method using a scenario containing drifts. We identify few synthetic datasets for this type of validation, and real-world datasets are not suitable because we do not know *a priori* if there is a concept drift. We summarize the following challenges for concept drift in PM:

- Enhance the experimental protocol. The use of a detailed metric to evaluate the approaches, either for online and offline analysis, is required. Our suggestion is the use of the F-score, yet,

following a precise and clear definition on how to count the true/false positives/negatives, considering an error tolerance, and applying the metric in artificial datasets.

- Enhance the online experiment protocol, considering the restrictions of the online environment: detection delay, memory usage, the processing time for each event, and input as an event stream, for instance.
- Include validation in datasets without drifts. The methods are highly sensitive to parametrization, which may lead to false positives. The validation of different thresholds for avoiding the false positives for the proposed parameter can improve the evaluation and robustness of the proposed methods.
- Develop tools for easily comparing the results of different approaches within distinct datasets. Such tools should provide an easy interface to change the parameters and evaluate results. Also, these tools may provide synthetic datasets.

We are in the fourth industrial revolution, Industry 4.0, where one of the five significant features is automation and adaptation. Business process management (BPM) is one of the industrial information integration methods needed, fulfilling the continuous improvement of business processes [43]. In the context of Industry 4.0, the business process is even more automated and flexible, increasing the need for tools for monitoring and improving the business process in evolving environments. PM provides powerful tools for analyzing, monitoring, and improving the processes; however, the main state-of-the-art techniques do not focus on online and evolving environments. Dealing with concept drift in PM can improve the benefits of this application in the Industry 4.0 context, thus allowing the application of PM techniques in evolving environments.

ACKNOWLEDGMENTS

A APPENDIX - SYNTHETIC DATASETS DESCRIPTION

(1) **Business Process Drift – [45]** - 75 event logs
    Link: https://data.4tu.nl/articles/dataset/Business_Process_Drift/12712436
    Type of drift: sudden. **Perspective:** control-flow (12 simple and 6 complex patterns).
    Size and interval between drifts (traces): 2500 (250); 5000 (500); 7500 (750); 10000 (1000).
    Number of drifts: 9 drifts in each log

(2) **Logs Characterization – [55]** - 375 event logs
    Link: https://drive.google.com/file/d/1xYuai8-HBrCZLSAuZMGPv7IJzOjbEsaY/view
    Type of drift: sudden. **Perspective:** control-flow (65 logs with single changes, 30 logs with composite changes, and 30 logs with nested changes)
    Size and interval between drifts (traces): 3000 (1000)
    Number of drifts: 2 drift in each log
    Noise: two variants with 2.5% and 5% noise (inserting random events into the traces)
    Complementary information: Highly variable logs with trace variability around 80%.

(3) **Synthetic Event Streams - [20]** - 942 event streams
    Link: https://ieee-dataport.org/open-access/synthetic-event-streams
    Type of drift: sudden, gradual, incremental, recurring. **Perspective:** control-flow (16 change patterns), time. **Size (traces):** 100, 500, and 1000

Number of drifts. Sudden: 1 drift injected in the middle of the stream. **Recurring:** 2 drifts – for streams with 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); for 500 and 1000 traces, the division is 167–167–166 and 330–330–340. **Gradual:** 1 drift (20% of the stream represents the transition between concepts). **Incremental:** 2 drifts (an intermediate model between the baseline and the drifted model is required; 20% of the stream contains the intermediate behavior, so the division was 40–20–40, baseline–intermediate model–incremental drift) **Time drifts.** Baseline behavior: the meantime is 30 min, standard variation is 3 min. Drifted behavior: the mean and standard variations were 5 and 0.5 min. **Incremental drift:** the transition state (20% of the stream) was split into 4 parts where standard time distribution decreases 5 min between them, following the incremental change of time.

Noise: 4 variants with 5%, 10%, 15%, and 20% (removing the first or the last half of the trace)

Complementary information: The arrival rate of events fixed to 20 min, the time distribution between events of the same case follows a normal distribution. For time drift, the change affects only the time perspective.

REFERENCES

[1] Rafael Accorsi, Thomas Stocker, Rafael Accorsi, Thomas Stocker, Discovering Workflow, Time-based Trace Clustering, Rafael Accorsi, and Thomas Stocker. 2012. Discovering Workflow Changes with Time-Based Trace Clustering. In *Data-Driven Process Discovery and Analysis*. Springer, Berlin, Heidelberg, 154–168.

[2] Charu C. Aggarwal and Chandan K. Reddy (Eds.). 2014. *DATA CLUSTERING Algorithms and Apllications*. CRC Press, Boca Raton, USA.

[3] Mihael Ankerst, Markus M. Breunig, Hans Peter Kriegel, and Jörg Sander. 1999. OPTICS: Ordering Points to Identify the Clustering Structure. *SIGMOD Record (ACM Special Interest Group on Management of Data)* 28, 2 (jun 1999), 49–60.

[4] Sylvio Barbon Junior, Gabriel Marques Tavares, Victor G. Turrisi da Costa, Paolo Ceravolo, and Ernesto Damiani. 2018. A Framework for Human-in-the-loop Monitoring of Concept-drift Detection in Event Log Stream. In *Companion Proceedings of the The Web Conf. 2018*, Vol. 2. ACM, Lyon, France, 319–326.

[5] A Batyuk, V V Oityshyn, and V Verhun. 2018. Software Architecture Design of the Real- Time Processes Monitoring Platform. In *2018 IEEE 2nd Int. Conf. on Data Stream Mining Processing (DSMP)*. IEEE, Lviv, Ukraine, 98–101.

[6] Anatoliy Batyuk and Volodymyr Voityshyn. 2020. Streaming process discovery method for semi-structured business processes. In *Int. Conf. on Data Stream Mining and Processing, DSMP 2020*. IEEE, Lviv, Ukraine, 444–448.

[7] Albert Bifet and Ricard Gavaldà. 2007. Learning from time-changing data with adaptive windowing. In *Int. Conf. on Data Mining*. Springer, Minneapolis, USA, 443–448.

[8] R. P. Jagadeesh Chandra Bose and Wil M.P. P van der Aalst. 2010. Trace clustering based on conserved patterns: Towards achieving better process models. *Lecture Notes in Business Information Processing* 43 LNBIP (2010), 170–181.

[9] R. P. Jagadeesh Chandra Bose, Wil M. P. van der Aalst, Indrė Žliobaitė, and Mykola Pechenizkiy. 2011. Handling Concept Drift in Process Mining. In *Advanced Information Systems Engineering*, Haralambos Mouratidis and Colette Rolland (Eds.). Springer, Berlin, Heidelberg, 391–405.

[10] R. P. Jagadeesh Chandra Bose, Wil M. P. van der Aalst, Indre Zliobaite, and Mykola Pechenizkiy. 2014. Dealing With Concept Drifts in Process Mining. *IEEE Transactions on Neural Networks and Learning Systems* 25, 1 (jan 2014), 154–171.

[11] Markus M. Breuniq, Hans Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: Identifying density-based local outliers. *SIGMOD Record (ACM Special Interest Group on Management of Data)* 29, 2 (jun 2000), 93–104.

[12] Tobias Brockhoff, Merih Seran Uysal, and Wil M.P. van der Aalst. 2020. Time-aware Concept Drift Detection Using the Earth Mover's Distance. In *2nd Int. Conf. on Process Mining (ICPM)*. IEEE, Padua, Italy, 33–40.

[13] Andrea Burattin, Marta Cimitile, and Fabrizio Maria Maggi. 2015. Lights, camera, action! Business process movies for online process discovery. In *Business Process Management Workshops (BPM 2014)*, Fabiana Fournier, , and Jan Mendling (Eds.). Springer, Eindhoven, The Netherlands, 408–-419.

[14] A Burattin, M Cimitile, F M Maggi, and A Sperduti. 2015. Online Discovery of Declarative Process Models from Event Streams. *IEEE Transactions on Services Computing* 8, 6 (2015), 833–846.

[15] A Burattin, A Sperduti, and W M P van der Aalst. 2014. Control-flow discovery from event streams. In *2014 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, Beijing, China, 2420–2427.

[16] Giorgio C. Buttazzo. 2011. *Hard Real-Time Computing Systems* (3rd ed.). Real-Time Systems Series, Vol. 24. Springer US, New York, USA. 1689–1699 pages. arXiv:arXiv:1011.1669v3

[17] Feng Cao, Martin Estert, Weining Qian, and Aoying Zhou. 2006. Density-Based Clustering over an Evolving Data Stream with Noise. In *Int. Conf. on Data Mining*. SIAM, Philadelphia, PA, 328–339.

[18] Josep Carmona and Ricard Gavaldà. 2012. Online Techniques for Dealing with Concept Drift in Process Mining. In *Advances in Intelligent Data Analysis XI*. Springer-Verlag, Berlin, Heidelberg, 90–102.

[19] Gabriel Marques Tavares; Sylvio Barbon Jr.; Paolo Ceravolo. 2019. Synthetic Event Streams.

[20] Paolo Ceravolo, Gabriel Marques Tavares, Sylvio Barbon Junior, and Ernesto Damiani. 2020. Evaluation Goals for Online Process Mining: a Concept Drift Perspective. *IEEE Transactions on Services Computing* (2020). Early Access.

[21] R Conforti, M L Rosa, and A H M t. Hofstede. 2017. Filtering Out Infrequent Behavior from Business Process Event Logs. *IEEE Transactions on Knowledge and Data Engineering* 29, 2 (2017), 300–314.

[22] A. K.A. De Medeiros, B. F. Van Dongen, W. M.P. Van Der Aalst, and A. J.M.M. Weijters. 2004. Process mining for ubiquitous mobile systems: An overview and a concrete algorithm. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3272 (2004), 151–165.

[23] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. 2015. Learning in Nonstationary Environments: A Survey. *IEEE Computational Intelligence Magazine* 10, 4 (nov 2015), 12–25.

[24] Maikel Eck, Xixi Lu, Sander Leemans, and Wil Aalst. 2015. PM2: A Process Mining Project Methodology. In *Int. Conf. on Advanced Information Systems Engineering*. Springer, Stockholm, Sweden, 297–313.

[25] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conf. on Knowledge Discovery and Data Mining*. AAAI Press, Portland, USA, 226–231.

[26] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. 2014. Cuckoo Filter: Practically Better Than Bloom. In *10th ACM Int. on Conf. on Emerging Networking Experiments and Technologies (CoNEXT '14)*. Association for Computing Machinery, New York, USA, 75–88.

[27] Eibe Frank and Ian H. Witten. 1998. Using a permutation test for attribute selection in decision trees. In *15th International Conf. on Machine Learning*. Morgan Kaufmann Publishers, San Francisco, USA, 152–160.

[28] João Gama, Indre Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A Survey on Concept Drift Adaptation. *ACM Comput. Surv.* 46, 4 (2014), 44:1-–44:37.

[29] Cleiton dos Santos Garcia, Alex Meincheim, Elio Ribeiro Faria Junior, Marcelo Rosano Dallagassa, Denise Maria Vecino Sato, Deborah Ribeiro Carvalho, Eduardo Alves Portela Santos, and Edson Emilio Scalabrin. 2019. Process mining techniques and applications – A systematic mapping study. *Expert Systems with Applications* 133 (nov 2019), 260–295.

[30] Christian W. Günther and Wil M.P. Van Der Aalst. 2007. Fuzzy mining - Adaptive process simplification based on multi-perspective metrics. In *Int. Conf. on Business Process Management BPM 2007: Business Process Management*, Vol. 4714 LNCS. Springer, Berlin, Heidelberg, Brisbane, Australia, 328–343.

[31] Marwan Hassani. 2019. Concept drift detection of event streams using an adaptive window. In *European Council for Modelling and Simulation, ECMS*, Vol. 33. European Council for Modelling and Simulation, Napoli, Italy, 230–239.

[32] Bart Hompes, Wil M P van der Aalst, and Prabhakar Dixit. 2017. Detecting Changes in Process Behavior Using Comparative Case Clustering. In *Data-Driven Process Discovery and Analysis*, Vol. 244. Springer, Vienna, Austria, 0–22.

[33] Angelo Impedovo, Paolo Mignone, Corrado Loglisci, and Michelangelo Ceci. 2020. Simultaneous Process Drift Detection and Characterization with Pattern-Based Change Detectors. In *Int. Conf. on Discovery Science (DS2020)*, Vol. 12323 LNAI. Springer, Thessaloniki, Greece, 451–467.

[34] R. Killick, P. Fearnhead, and I. A. Eckley. 2012. Optimal detection of changepoints with a linear computational cost. *J. Amer. Statist. Assoc.* 107, 500 (jan 2012), 1590–1598. arXiv:1101.1438

[35] B. Kitchenham and S. Charters. 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Technical Report 3. Keele University and Durham University Joint Report.

[36] Angelina Prima Kurniati, Ciarán McInerney, Kieran Zucker, Geoff Hall, David Hogg, and Owen Johnson. 2020. Using a multi-level process comparison for process change analysis in cancer pathways. *Int. Journal of Environmental Research and Public Health* 17, 19 (2020), 1–16.

[37] Johnson O. Kurniati A.P., McInerney C., Zucker K., Hall G., Hogg D. 2019. A Multi-level Approach for Identifying Process Change in Cancer Pathways. In *Int. Conf. on Business Process Management BPM 2019: Business Process Management Workshops*. Springer, Vienna, Austria, 595–607.

[38] Marcello La Rosa, Hajo A. Reijers, Wil M.P. Van Der Aalst, Remco M. Dijkman, Jan Mendling, Marlon Dumas, and Luciano García-Bañuelos. 2011. APROMORE: An advanced process model repository. *Expert Systems with Applications* 38, 6 (jun 2011), 7029–7040.

[39] Sander J.J. Leemans, Dirk Fahland, and Wil M.P. Van Der Aalst. 2013. Discovering block-structured process models from event logs - A constructive approach. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 7927 LNCS. Springer, Berlin, Heidelberg, 311–329.

[40] Leilei Lin, Lijie Wen, Li Lin, Jisheng Pei, and Hedong Yang. 2020. LCDD: Detecting Business Process Drifts Based on Local Completeness. *IEEE Transactions on Services Computing* 14, 8 (2020), 1–1.

[41] N Liu, J Huang, and L Cui. 2018. A Framework for Online Process Concept Drift Detection from Event Streams. In *2018 IEEE Int. Conf. on Services Computing (SCC)*. IEEE, San Francisco, USA, 105–112.

[42] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2019. Learning under Concept Drift: A Review. *IEEE Transactions on Knowledge and Data Engineering* 31, 12 (dec 2019), 2346–2363. arXiv:2004.05785

[43] Yang Lu. 2017. Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration* 6 (jun 2017), 1–10.

[44] Daniela Luengo and Marcos Sepúlveda. 2012. Applying Clustering in Process Mining to Find Different Versions of a Business Process that Changes over Time. In *Business Process Management Workshops*, Florian Daniel, Kamel Barkaoui, and Schahram Dustdar (Eds.). Springer, Berlin, Heidelberg, 153–158.

[45] Abderrahmane Maaradji, Marlon Dumas, Marcello La Rosa, and Alireza Ostovar. 2015. Fast and Accurate Business Process Drift Detection. In *Int. Conf. on Business Process Management BPM 2016: Business Process Management*. Springer, Cham, Innsbruck, Austria, 406–422.

[46] A Maaradji, M Dumas, M L Rosa, and A Ostovar. 2017. Detecting Sudden and Gradual Drifts in Business Processes from Execution Traces. *IEEE Transactions on Knowledge and Data Engineering* 29, 10 (2017), 2140–2154.

[47] Fabrizio Maria Maggi, Andrea Burattin, Marta Cimitile, and Alessandro Sperduti. 2013. Online process discovery to detect concept drifts in LTL-based declarative process models. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8185 LNCS, September (2013), 94–111.

[48] Gurmeet Singh Manku and Rajeev Motwani. 2002. Approximate Frequency Counts over Data Streams. In *VLDB '02: 28th Int. Conf. on Very Large Databases*, Philip A Bernstein, Yannis E Ioannidis, Raghu Ramakrishnan, and Dimitris Papadias (Eds.). Morgan Kaufmann, San Francisco, 346–357.

[49] M. V. Manoj Kumar, Likewin Thomas, and B. Annappa. 2015. Capturing the sudden concept drift in process mining. In *CEUR Workshop Proceedings*. CEUR-WS.org, Brussels, Belgium, 132–143.

[50] Giovanni Da San Martino, Nicolò Navarin, and Alessandro Sperduti. 2013. A Lossy Counting Based Approach for Learning on Streams of Graphs on a Budget. In *Twenty-Third Int. Joint Conf. on Artificial Intelligence (IJCAI '13)*. AAAI Press, Beijing, China, 1294–1301.

[51] J Martjushev, R. P. Jagadeesh Chandra Bose, and Wil M P Van Der Aalst. 2015. Change Point Detection and Dealing with Gradual and Multi-Order Dynamics in Process Mining. In *Int. Conf. on Business Informatics Research*. Springer, Tartu, Estonia, 1–15.

[52] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. 2013. The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology* 4 (aug 2013), 504.

[53] Davide Mora, Paolo Ceravolo, Ernesto Damiani, and Gabriel Marques Tavares. 2020. The CDESF toolkit: An introduction. In *ICPM Doctoral Consortium and Tool Demonstration Track 2020*, Vol. 2703. CEUR-WS.org, Padua, Italy, 47–50.

[54] Nicolas Jashchenko Omori, Gabriel Marques Tavares, Paolo Ceravolo, and Sylvio Barbon. 2019. Comparing Concept Drift Detection with Process Mining Tools. In *SBSI'19: Proceedings of the XV Brazilian Symposium on Information Systems*. ACM, Aracaju, Brazil, 1–8.

[55] Alireza Ostovar, Sander J. J. Leemans, and Marcello La Rosa. 2020. Robust Drift Characterization from Event Streams of Business Processes. *ACM Trans. Knowl. Discov. Data* 14, 3, Article 30 (March 2020), 57 pages.

[56] Alireza Ostovar, Abderrahmane Maaradji, Marcello La Rosa, and Arthur Ter. 2017. Characterizing Drift from Event Streams of Business Processes. In *Lecture Notes in Computer Science*, Vol. 10253. Springer, Cham, 210–228.

[57] Alireza Ostovar, Abderrahmane Maaradji, Marcello La Rosa, Arthur H. M. ter Hofstede, and Boudewijn F. V. van Dongen. 2016. Detecting Drift from Event Streams of Unpredictable Business Processes. In *Conceptual Modeling ER 2016 (Lecture Notes in Computer Science)*. Springer, Cham, 330–346.

[58] Rohit J Parikh. 1966. On Context-Free Languages. *Journal of the Association for Computing Machinery* 13, 4 (1966), 570–581.

[59] Stephen Pauwels and Toon Calders. 2019. An anomaly detection technique for business processes based on extended dynamic Bayesian networks. *ACM Symposium on Applied Computing* Part F1477, April (2019), 494–501.

[60] David Redlich, Thomas Molka, Wasif Gilani, Gordon Blair, and Awais Rashid. 2014. Scalable dynamic business process discovery with the constructs competition miner. In *Int. Symposium on Data-Driven Process Discovery and Analysis (SIMPDA 2014)*, Vol. 1293. CEUR-WS.org, Milan, Italy, 91–107.

[61] Florian Richter, Andrea Maldonado, Ludwig Zellner, and Thomas Seidl. 2020. OTOSO: Online Trace Ordering for Structural Overviews. In *Process Mining Workshops - 1st Int. Workshop on Streaming Analytics for Process Mining (SA4PM'20)*. Springer, Padua, Italy, 189–192.

[62] Florian Richter and Thomas Seidl. 2017. TESSERACT: Time-Drifts in Event Streams Using Series of Evolving Rolling Averages of Completion Times. In *Int. Conf. on Business Process Management BPM 2017: Business Process Management*. Springer, Barcelona, Spain, 289–305.

[63] Florian Richter and Thomas Seidl. 2019. Looking into the TESSERACT: Time-drifts in event streams using series of evolving rolling averages of completion times. *Information Systems* 84 (2019), 265–282.

[64] Helen Schonenberg, Ronny Mans, Nick Russell, Nataliya Mulyar, and Wil van der Aalst. 2008. Process Flexibility: A Survey of Contemporary Approaches. In *Advances in Enterprise Engineering I*, Jan L. G. Dietz, Antonia Albani, and Joseph Barjis (Eds.). Springer, Berlin, Heidelberg, 16–30.

[65] Erich Schubert, Michael Weiler, and Hans-Peter Kriegel. 2014. SigniTrend: Scalable Detection of Emerging Topics in Textual Streams by Hashed Significance Thresholds. In *20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD '14)*. Association for Computing Machinery, New York, USA, 871–880.

[66] Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. 2017. Detecting Concept Drift in Processes Using Graph Metrics on Process Graphs. In *Proceedings of the 9th Conf. on Subject-Oriented Business Process Management* (Darmstadt, Germany) *(S-BPM ONE '17)*. Association for Computing Machinery, New York, USA, Article 6, 10 pages.

[67] Florian Stertz and Stefanie Rinderle-Ma. 2018. Process Histories - Detecting and Representing Concept Drifts Based on Event Streams. In *On the Move to Meaningful Internet Systems - OTM (Lecture Notes in Computer Science, Vol. 11229)*, Panetto H. et al. (Ed.). Springer, Valletta, Malta, 318–335.

[68] Florian Stertz and Stefanie Rinderle-Ma. 2019. Detecting and Identifying Data Drifts in Process Event Streams Based on Process Histories. In *Information Systems Engineering in Responsible Information Systems. CAiSE 2019.*, Vol. 350. Springer, Rome, Italy, 133–144.

[69] Gabriel Marques Tavares, Paolo Ceravolo, Victor G.Turrisi Da Costa, Ernesto Damiani, and Sylvio Barbon Jr. 2019. Overlapping analytic stages in online process mining. In *Int. Conf. on Services Computing*. IEEE, Milan, Italy, 167–175.

[70] Charles Truong, Laurent Oudre, and Nicolas Vayatis. 2020. Selective review of offline change point detection methods. *Signal Processing* 167 (2020), 107299.

[71] Wil Van der Aalst. 2016. *Process mining: Data science in action.* Springer, Berlin, Germany. 1–467 pages.

[72] Wil M.P. Van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos Buijs, et al. 2011. Process Mining Manifesto. In *Int. Conf. on Business Process Management BPM 2011: Business Process Management Workshops*, Vol. 99. Springer, Clermont-Ferrand, France, 169–194.

[73] W. M.P. Van Der Aalst, M. Pesic, and H. Schonenberg. 2009. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development* 23, 2 (2009), 99–113.

[74] W. M.P. van der Aalst, B. F. van Dongen, C. Günther, A. Rozinat, H. M.W. Verbeek, and A. J.M.M. Weijters. 2009. Prom: The process mining toolkit. In *BPM 2009 Demonstration Track*, Vol. 489. CEUR-WS.org, Ulm, Germany, 1–4.

[75] R J van Van and U Goltz. 1989. Equivalence notions for concurrent systems and refinement of actions. In *14th Symposium on Mathematical Foundations of Computer Science*, Vol. 379. Springer-Verlag, Berlin, Heidelberg, 237–248.

[76] Sebastiaan J. van Zelst, Boudewijn F. van Dongen, and Wil M.P. van der Aalst. 2018. Event stream-based process discovery using abstract representations. *Knowledge and Information Systems* 54, 2 (2018), 407–435. arXiv:1704.08101

[77] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. 2008. Change patterns and change support features - Enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering* 66, 3 (2008), 438–466.

[78] Matthias Weidlich, Jan Mendling, and Mathias Weske. 2011. Efficient consistency measurement based on behavioral profiles of process models. *IEEE Transactions on Software Engineering* 37, 3 (2011), 410–429.

[79] A J M M Weijters, W M P Van Der Aalst, and A K Alves De Medeiros. 2006. *Process Mining with the HeuristicsMiner Algorithm.* Technische Universiteit Eindhoven, Eindhoven, The Netherlands.

[80] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *18th Int. Conf. on Evaluation and Assessment in Software Engineering*. ACM Press, New York, USA, 1–10.

[81] H Yang, L Wen, and J Wang. 2012. An Approach to Evaluate the Local Completeness of an Event Log. In *2012 IEEE 12th Int. Conf. on Data Mining*. IEEE, Brussels, Belgium, 1164–1169.

[82] Anton Yeshchenko, Claudio Di Ciccio, Jan Mendling, and Artem Polyvyanyy. 2019. Comprehensive Process Drift Analysis with the Visual Drift Detection Tool. In *CEUR Workshop Proceedings*. CEUR-WS, Salvador, Brazil, 108–112.

[83] Anton Yeshchenko, Claudio Di Ciccio, Jan Mendling, and Artem Polyvyanyy. 2019. Comprehensive Process Drift Detection with Visual Analytics. In *Conceptual Modeling ER 2019*. Springer, Salvador, Brazil, 119–135. arXiv:1907.06386

[84] Anton Yeshchenko, Claudio Di Ciccio, Jan Mendling, and Artem Polyvyanyy. 2021. Visual Drift Detection for Sequence Data Analysis of Business Processes. *IEEE Transactions on Visualization and Computer Graphics* (2021), 1–1.

[85] Anton Yeshchenko, Jan Mendling, Claudio Di Ciccio, and Artem Polyvyanyy. 2020. VDD: A Visual Drift Detection System for Process Mining. In *Doctoral Consortium and Tool Demo Track (ICPM)*. CEUR-WS.org, Padua, Italy, 31–34.

[86] Ludwig Zellner, Florian Richter, Janina Sontheim, Andrea Maldonado, and Thomas Seidl. 2020. Concept Drift Detection on Streaming Data with Dynamic Outlier Aggregation. In *Process Mining Workshops - 1st Int. Workshop on Streaming Analytics for Process Mining (SA4PM'20)*, Springer (Ed.). Springer, Padua, Italy, 189–192.

[87] Haiping Zha, Jianmin Wang, Lijie Wen, Chaokun Wang, and Jiaguang Sun. 2010. A workflow net similarity measure based on transition adjacency relations. *Computers in Industry* 61, 5 (jun 2010), 463–471.

[88] Canbin Zheng, Lijie Wen B, Jianmin Wang, Lijie Wen, and Jianmin Wang. 2017. Detecting Process Concept Drifts from Event Logs. In *On the Move to Meaningful Internet Systems - OTM*, Panetto H. et al. (Ed.). Springer, Cham, 524–542.