PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA (PPGIa)

DENISE MARIA VECINO SATO

CONCEPT DRIFT DETECTION IN PROCESS MODELS

CURITIBA

2022

DENISE MARIA VECINO SATO

CONCEPT DRIFT DETECTION IN PROCESS MODELS

This thesis was presented to the Graduate Program in Informatics (PPGIa) at the Pontifícia Universidade Católica do Paraná, as a partial requirement to obtain the title of Doctor of Informatics.

Advisor: Prof. Dr. Edson Emilio Scalabrin

Co-advisor: Prof. Dr. Jean Paul Barddal

CURITIBA

2022

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central
Luci Eduarda Wielganczuk – CRB 9/1118

Sato, Denise Maria Vecino
S253c Concept drift detection in process models / Denise Maria Vecino Sato ;
2022 advisor: Edson Emilio Scalabrin ; co-advisor: Jean Paul Barddal. – 2022.
166 f. : il. ; 30 cm

Tese (doutorado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2022
Bibliografia: f. 157-166

1. Mineração de dados (Computação). 2. Mineração de processos.
3. Aprendizado do computador. 4. Algoritmos de computador. I. Scalabrin, Edson Emilio. II. Barddal, Jean Paul III. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. IV. Título.

CDD 22. ed. – 006.312

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Informática

61-2022

DECLARAÇÃO

Declaro para os devidos fins que **DENISE MARIA VECINO SATO**, defendeu a tese de Doutorado intitulada "**CONCEPT DRIFT DETECTION IN PROCESS MODELS",** na área de concentração Ciência da Computação, no dia 08 de julho de 2022, no qual foi aprovada.

Declaro ainda que foram feitas todas as alterações solicitadas pela Banca Examinadora, cumprindo todas as normas de formatação definidas pelo Programa.

Por ser verdade, firmo a presente declaração.

Curitiba, 13 de setembro de 2022.

Prof. Dr. Emerson Cabrera Paraiso
Coordenador do Programa de Pós-Graduação em Informática
Pontifícia Universidade Católica do Paraná

Rua Imaculada Conceição 1155, Prado Velho, CEP 80215-901 Curitiba Paraná Brasil
Fone: (41) 3271-1669 e-mail: secretaria@ppgia.pucpr.br

Koji, Lara e Enzo, dedico essa tese a vocês, que sempre me apoiam a buscar o melhor de mim.

AGRADECIMENTOS

Agradeço inicialmente a minha família, especialmente ao meu marido Koji e meus filhos Lara e Enzo, eles são minha força para sempre buscar meus objetivos.

Agradeço especialmente ao meu orientador, prof. Edson Scalabrin, que sempre foi um incentivador do meu trabalho, me direcionando com ideias, mas sempre permitindo autonomia ao percorrer meu caminho de pesquisa. Os ensinamentos e conselhos foram fundamentais durante todo o processo.

Agradeço também ao meu coorientador, prof. Jean Barddal, que sempre esteve disposto a me auxiliar na discussão de abordagens e demais desafios. A todos os professores do PPGIa minha gratidão pela contribuição na minha formação acadêmica durante o desenvolvimento do curso de doutorado.

Agradeço ao prof. Eduardo Portela e seus orientados Letícia e Alexandre, que sempre me auxiliaram em diferentes parcerias. Agradeço especialmente ao Edson Ruschel, que me permitiu aplicar parte da minha pesquisa em um cenário real. Também agradeço aos colegas do PPGTS, que me permitiram conhecer um pouco mais sobre os desafios da área da saúde. Agradeço especialmente minha amiga Rafaela, que sempre acompanhou minha pesquisa e me auxiliou muito para melhorar a interface da ferramenta desenvolvida. Também aos meus amigos e colegas de laboratório Juliano, Luiz, Jair, e Eduardo que sempre contribuíram para desenvolvimento do meu projeto. Especialmente a minha amiga Sheila, que me acompanhou e sempre esteve ao meu lado em todos os momentos.

Agradeço ao IFPR, que me permitiu desenvolver minha pesquisa de forma integral, algo que foi essencial para o resultado desse trabalho.

Á Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pela concessão de bolsa de estudos durante a realização do doutorado. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

RESUMO

A mineração de processos provê uma visão real dos processos de negócio a partir de eventos dos sistemas de informação. Normalmente, os métodos assumem que os processos não mudam, porém, isso é pouco provável em processos reais que mudam para atender demandas de mercado, novas regulamentações, ou para melhorias e correções. Essas mudanças são chamadas de *concept* ou *process drifts* e a sua detecção é relevante por diferentes razões: melhorar a análise de processos fornecendo um modelo mais aderente a realidade; melhorar a ação de gestores por atuar a partir das mudanças; melhorar a acurácia de mecanismos de predição e recomendação. Diferentes propostas foram apresentadas para detecção de *process drifts*; entretanto, elas são muito sensíveis a escolha dos parâmetros e muitas vezes a análise de acurácia não utiliza métricas objetivas, ou as métricas não estão claras. Essa pesquisa buscou superar esses desafios a partir da apresentação de uma nova ferramenta para detecção de *process drifts*, o *Interactive Process Drift Detection (IPDD) framework*, que define uma arquitetura genérica para implementação de métodos de detecção de *drifts*. Essa arquitetura permitiu a implementação de três abordagens: (i) *Fixed IPDD for control-flow drifts*; (ii) *Adaptive IPDD for time and data drifts*; e (iii) *Adaptive IPDD for control-flow drifts*, que buscaram reduzir a quantidade de parâmetros – aplicando o ADWIN change detector – e facilitar a verificação dos *drifts* pelo usuário. O *Fixed IPDD for control-flow drifts* foi validado em uma base sintética. Detectou 17 dos 18 padrões de mudança da base e permitiu a localização dos *drifts* no modelo de processo e a visualização de suas versões ao longo do tempo. O *Adaptive IPDD for time and data drifts* foi aplicado em um estudo de caso no contexto industrial para *condition-based maintenance* e validado em bases sintéticas e uma base real. O *Adaptive IPDD for control-flow drifts* foi validado em bases sintéticas com duas abordagens (*trace by trace* e *windowing*) e comparado com o Apromore ProDrift e Visual Drift Detection (VDD). O *Adaptive IPDD for control-flow drifts windowing* é estatisticamente equivalente ao Apromore para detecção de drifts, mas supera o Apromore considerando o *mean delay*. Já o *Adaptive IPDD for control-flow drifts trace by trace* é estatisticamente equivalente ao Apromore no *dataset 2* e supera o *mean delay* do Apromore AWIN no *dataset 1* e o *mean delay* do Apromore FWIN no *dataset 2*. VDD apresentou os resultados mais baixos de *F-score* e *mean delay* para ambos os *datasets*. O *Adaptive IPDD for control-flow* foi aplicado em um log real e detectou três *drifts*, localizando os mesmos e apresentando as versões do processo ao longo do tempo. A aplicação do *IPDD* em processos reais é promissora devido a validação realizada e a interface de usuário que permite uma facilidade para visualização dos drifts detectados pelo especialista de negócio.

Palavras-chave: process drifts, concept drift, modelo de processo, ADWIN.

ABSTRACT

Process mining provides an objective view of business processes from events collected from the information systems. Usually, the methods assume that processes do not change; however, this is not probable in real-life processes that change to meet market demands, new regulations, or improvements and corrections. These changes are called concept or process drifts. Their detection is relevant for different reasons: to improve process analysis by providing a model more adherent to reality; to improve the action of managers by acting based on the detected changes; improve the accuracy of prediction and recommendation mechanisms. Different proposals were presented for detecting process drifts; however, they are susceptible to the parameter configuration, and often the accuracy analysis does not use objective metrics, or the metrics are not clear. This research aims to overcome these challenges by presenting a new tool for process drift detection, the *Interactive Process Drift Detection (IPDD) framework*, which defines a generic architecture for implementing drift detection methods. Three implementations based on this architecture were presented: (i) *Fixed IPDD for control-flow drifts*; (ii) *Adaptive IPDD for time and data drifts*; and (iii) *Adaptive IPDD for control-flow drifts*, aiming to reduce the number of parameters – by applying the ADWIN change detector – and facilitate the verification of drifts by the user. *Fixed IPDD for control-flow drifts* was validated on a synthetic dataset. It detected 17 of the 18 change patterns, allowing the drift localization in the process model and the visualization of their versions over time. The *Adaptive IPDD for time and data drifts* was applied in a case study in the industrial context for condition-based maintenance and validated on synthetic datasets and a real-life event log. The *Adaptive IPDD for control-flow drifts* was validated on synthetic datasets with two approaches (*trace by trace* and *windowing*) and compared with Apromore ProDrift and Visual Drift Detection (VDD). *Adaptive IPDD for control-flow drifts windowing* is statistically equivalent to Apromore for drift detection but outperforms Apromore in mean delay. The *Adaptive IPDD for control-flow drifts trace by trace* is statistically equivalent to Apromore in dataset 2 and surpasses the mean delay of Apromore AWIN in dataset 1 and the mean delay of Apromore FWIN in dataset 2. VDD presented the lowest F -score and mean delay results for both datasets. The *Adaptive IPDD for control-flow* was applied to a real-life log and detected three drifts, localizing them and presenting the process versions over time. The application of the *IPDD* in real processes is promising because of the validation performed and the focus on providing a simple user interface for the business analyst to verify the detected drifts and the evolution of the process over time.

Keywords: process drifts, concept drift, process models, ADWIN.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ADWIN	Adaptive Windowing
AHC	Agglomerative Hierarchical Clustering
AWIN	Apromore ProDrift Adaptive Approach
BPIC	Business Process Intelligence Challenge
CCM	Constructs Competition Miner
CD	Clear Defined
CDESF	Concept Drift in Event Stream Framework
CP	Change Point
CW	Complete Window
DA	Frequency of Activities
DC	Distinct Cases Running at the Same Time
DCCM	Dynamic CCM
DFG	Directly-follows Graph
DFP	Dynamic Overall Footprint
DOA	Dynamic Outlier Aggregation
DP	Drift Plot
DS	Direct Succession
DSR	Direct Succession Relation
DW	Detection Window
EC	Exclusion Criteria
EDBN	Extended Dynamic Bayesian Networks
EMD	Earth Mover’s Distance
EPS	Maximum Radius of a Neighborhood
ET	Error Tolerance
EWD	Edit Weighted Distance
FN	False Negative
FP	False Positive
FT	Feature
FWIN	Apromore ProDrift Fixed Approach
GDTime	Graph-distance Time
GDTrace	Graph-distance Trace
HM	Heuristics Miner

IC	Inclusion Criteria
ICPM	International Conference on Process Mining Series
ID	Identifier
iDHM	Interactive Data-Aware Heuristics Miner
ILP	Language-based Regions Miner
IM	Inductive Miner
IMf	Inductive Miner Infrequent
IMpt	Inductive Miner – partial traces
IPDD	Interactive Process Drift Detection
KSPT	K-sample permutation test
LC	Lossy Counting
LCB	Lossy Counting with Budget
LCO	Local Completeness
LCDD	Local Complete-based Drift Detection
LOF	Local Outlier Factor
MINPTS	Minimum Number of Points Required to Form a Dense Region
MINWIN	Minimal Window Size
ML	Minimal Length
MR	Maximal Repeat Feature Set
MRID	Minimum Relation Invariance Distance
MSE	Mean Squared Error
NA	Not Available
OTOSO	Online Trace Ordering for Structural Overviews
PA	Public Available
PBCD	Pattern-based Change Detection
PELT	Pruned Exact Linear Time
PM	Process Mining
PMG	Process Graph Normalized
PSM	Performance Spectrum Miner
RC	Relation Type Count
RD	Real-world Dataset
RE	Relation Entropy
RFC	Relative Frequency Change
RMSE	Root Mean Squared Error

RQ	Research Question
S-BAR	Stream-based Process Discovery
SD	Synthetic Dataset
SLR	Systematic Literature Review
SP	Stable Period
SW	Sliding Window
TDS	Temporal Deviation Signature
TFP	Individual Trace Footprint
TH	Time Horizon
TN	True Negative
TP	True Positive
TPCDD	Tsinghua Process Concept Drift Detection
TRFC	Total Relative Frequency Change
TWD	Time Weighted Distance
VDD	Visual Drift Detection System
WC	Window Count
WOR	Weak Order Relation
WS	Window Size

SUMMARY

1 INTRODUCTION

Process mining is a growing research area that links the process analysis and modeling techniques with data mining. The goal is to obtain valuable information about processes from data collected from the information systems' event logs. Business analysts and companies can then improve their business processes based on what is happening instead of trying to understand reality based on assumptions or interviews. Process mining provides powerful tools and methods for understanding, analyzing, and improving business processes to achieve better performance, customer experience, or any other aspect that may raise an advantage in the competitive business world.

Process mining includes three main branches: process discovery, conformance checking, and enhancement (van der Aalst, 2016). Process discovery automatically derives a process model from an event log without any *a priori* information. The techniques compare the event logs and a process model (discovered or designed) to detect deviations in conformance checking. In process enhancement, the discovered process model can be extended by adding additional perspectives obtained from event logs, e.g., performance, and data. The enhancement also includes repairing or improving a business process based on the discovery, conformance, and extension analysis. Process mining also includes social network/organizational mining, automated construction of simulation models, case prediction (Choueiri et al., 2020), and history-based recommendations (van der Aalst et al., 2011).

Nowadays, we can find different process mining tools available. Some of them are designed for commercial use, e.g., Disco[1] (Günther & Rozinat, 2012), Apromore[2] (La Rosa et al., 2011), Upflux[3], Celonis[4], and, in addition, provide academic licenses for academics to support their research. Some tools are designed for research purposes, e.g., ProM Framework[5] (van der Aalst et al., 2009) and bupaR[6] (Janssenswillen et al., 2019). A more recent tool, the PM4Py[7] (Berti et al., 2019), started as an open-source framework including several state-of-the-art process mining algorithms for researchers, mainly because of their programmatic user interface. However, the same research group recently released the

[1] Disco: https://fluxicon.com/disco/

[2] Apromore: https://apromore.org/platform/editions/

[3] Upflux: https://upflux.net/pt/

[4] Celonis: https://www.celonis.com/process-mining/what-is-process-mining/

[5] ProM Framework: https://www.promtools.org/

[6] bupaR: https://www.bupar.net/index.html

[7] PM4Py: https://pm4py.fit.fraunhofer.de/

PMTK, a web-based process mining tool built on the open-source PM4Py project, providing a user-friendly interface for commercial and academic purposes.

The starting point for any process mining technique is the event data, accessed as an event log or an event stream. Event logs contain historical information about processes, represented by sequentially recorded events related to a process instance (case). A case *id* identifies each process instance, and each event refers to an activity, which is a well-defined step in the process (van der Aalst et al., 2011). An event stream is a potentially infinite sequence of events emitted as they occur and are executed in the context of a business process (van Zelst et al., 2018). Any event in the stream must be related to a case and an activity, as in an event log. Event streams allow process mining online analysis, but include additional challenges such as dealing with incomplete traces and memory consumption. Regardless of the format (event log or event stream), the event data contains information about the execution of a single business process, such as buying items online.

The process model in Figure 1.1 shows the sequence of activities executed from the order request to the shipping or canceling order activities from an online store using the Petri Net notation. The possible events in the process are (a) order request, (b) credit card validation, (c) shipping address validation, (d) decision, (e) shipping order, and (f) cancel the order.

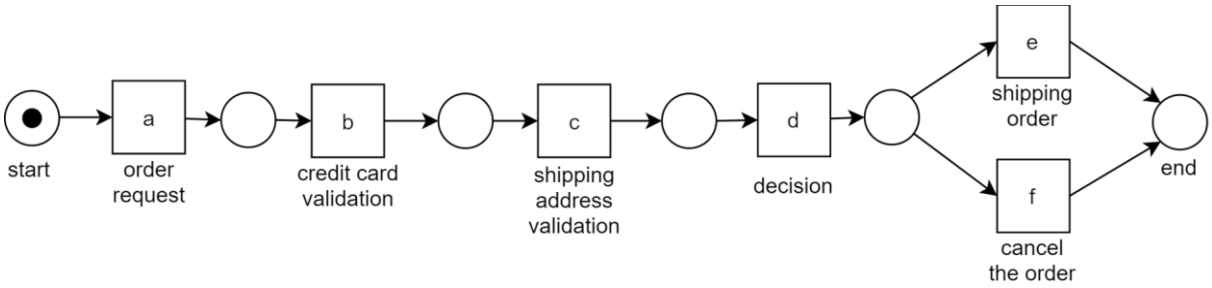


Figure 1.1. E-commerce process model using Petri Net notation.

Figure 1.2 shows an example of an event log recorded for the e-commerce process described in Figure 1.1. Each case has an identifier and represents the execution of one instance of the process, storing what is called a trace in the event data. A trace indicates the sequence of events performed by a specific case, i.e., a path in the process. In the described cases, the trace $< a, b, c, d, e >$ is stored for cases 1 and 2; but case 3 generates a different trace $< a, b, c, d, f >$. The event log in Figure 1.2 contains the minimum information needed for process mining techniques (case identifier, activity, and timestamp), but other case or event attributes can be included in the process analysis if available. Since several cases can record the same trace (cases 1 and 2), a simple representation of the event log can describe

each trace and its frequency (hereafter denoted as a superscript coupled with each trace): $L = [< a, b, c, d, e >^2, < a, b, c, d, f >^1, \dots]$.

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	
...	...	...	...	

Figure 1.2. Event log recorded for the e-commerce process.

The process model in Figure 1.1 shows only the control-flow perspective, i.e., the possible sequence of activities. However, additional perspectives can be included when extending process models through enhancement. For example, one can discover the average sojourn time for the “Shipping order” activity by applying a replay technique on the discovered process model and assigning performance information to the activities and paths. Therefore, we can identify different perspectives when applying process mining techniques, e.g., time and resource.

Most existing process mining methods and techniques are designed to work with event logs and consider the processes in a “steady-state”. In other words, they assume the process does not change during the recording of the event log, i.e., all traces are related to the same version of the process. This assumption is naive as real-life processes are likely to change over time in response to market dynamics, new regulations, or improving/repairing the current process. For instance, most process discovery algorithms have difficulties when an event log contains a drift: causal relations between events may appear, disappear, and vice-versa (Carmona & Gavaldà, 2012). 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 defined as a concept drift (Bose et al., 2014).

Mechanisms for detecting concept drifts in process models are essential for different reasons, such as:

- Improving the process analysis by deriving process models more adherent to reality;
- Minimizing the complexity of process discovered models by avoiding a mixed model;
- Improving managers' actions by explicitly identifying changes in the processes;
- Improving the accuracy of prediction or recommendation mechanisms, which usually degrade performance over time because of concept drifts;
- Implementing triggers in case of changes in the process.

The process mining manifesto (van der Aalst et al., 2011) also indicated that dealing with concept drift in process mining is one of the area's challenges, confirming the topic's importance.

We identified different approaches to handle the concept drift situation in processes, especially for detecting concept drifts in the control-flow perspective of the process models. Section 2 of this thesis describes the concept drift problem in process mining in further detail and provides a structured review of the current approaches for dealing with it. Based on this review, we identified that the current approaches have some issues that still need to be addressed in real-world business analyses. The following sections provide a brief overview of the main topic and the open challenges for applying the current approaches.

1.1 CONCEPT DRIFT IN PROCESSES

Concept drift in processes refers to the process changes that may happen when the process is being analyzed (Bose et al., 2014). Few processes are in a "steady-state," and understanding, detecting, and analyzing concept drift in processes is essential for business process management. The changes can affect the different perspectives of the process, e.g., control-flow, time, and resources. For instance, one activity may be removed or added, affecting the control-flow perspective; in another change, one can automate a previously manually performed activity, reducing the sojourn time of the process, which affects the time perspective.

The changes in the processes can affect the current instances suddenly or gradually. Furthermore, versions of the processes can reappear, or the new version can be

implemented using small changes over time. Therefore, the concept drift in processes can be classified into sudden, gradual, recurrent, and incremental, as described in Figure 1.3.

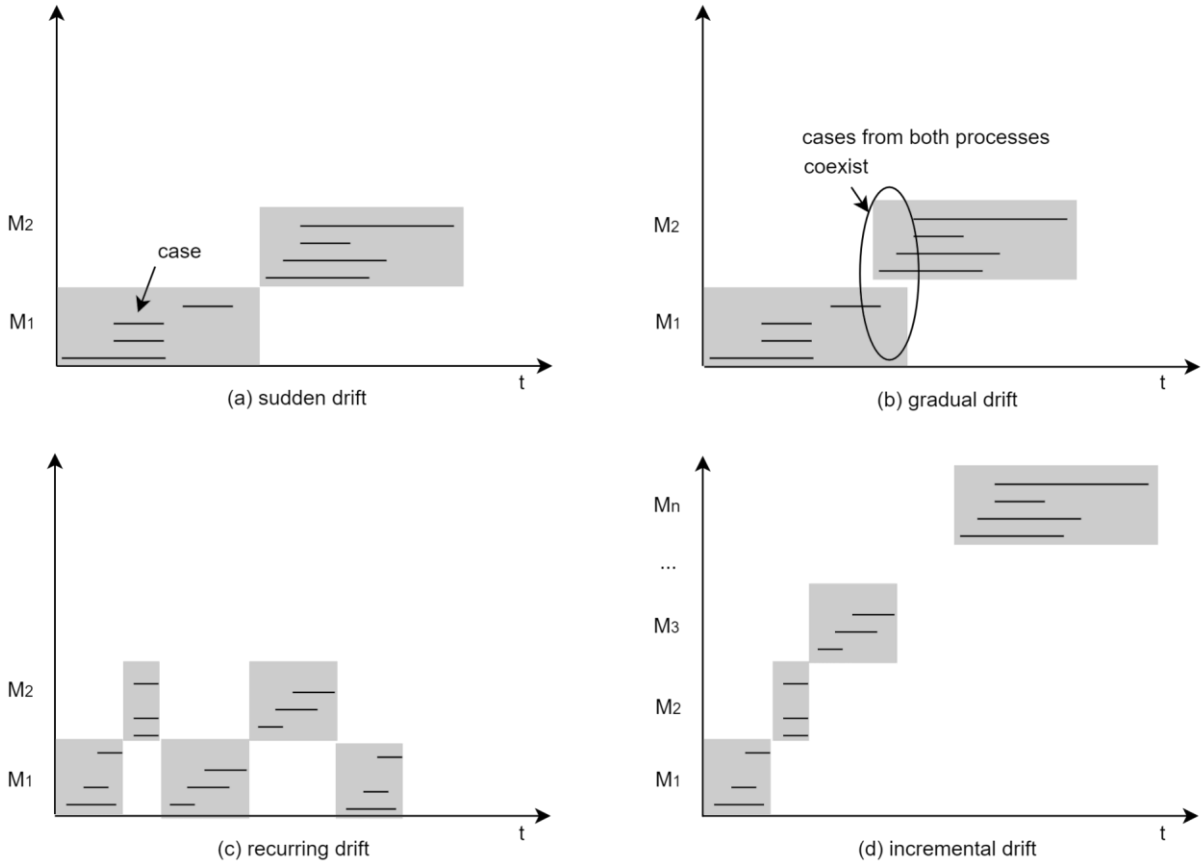


Figure 1.3. Four types of concept drift in processes.

The analysis of concept drift in processes may occur offline or online. In an offline analysis, the input is an event log representing the historical information about the process over a period containing complete traces. The offline analysis of concept drift improves the process analysis by highlighting unknown and probably unexpected changes. As a complementary analysis, we can apply the online concept drift detection when the changes need to be uncovered in real-time or near real-time, e.g., if the organization is interested in reacting to a change when it is happening, using real-time alarms. For this purpose, the online analysis requires an event stream, representing an infinite sequence of events generated over time (van Zelst et al., 2018).

The concept drift detection in process models includes five challenges: drift detection, change point detection, change localization, change characterization, and unraveling process evolution or change process discovery (Sato et al., 2022). Drift detection means reporting that a drift occurred and change point detection complements this information by reporting the point in the time where it occurred. The change localization identifies the region in the process model where the change occurred, and the characterization reports the nature of the

change by informing its type and perspective. Finally, the unravel the process evolution challenge allows the business analyst to visualize the complete change process based on the identification, localization, and characterization of a change.

1.2 APPROACHES FOR DETECTING CONCEPT DRIFT IN PROCESSES

Section 2 describes the identified approaches for dealing with concept drift in processes in more detail, as published in (Sato et al., 2022). However, we included this section to position the research aim against what was already found in the literature. Here we only report a brief overview of the main aspects of the identified approaches. Most papers for handling concept drift in process mining aim to detect the drifts (38 papers from 45) in the control-flow perspective (29 papers from 38) of the process offline. Most detection approaches apply statistical hypothesis tests over features calculated from the event log (Bose et al., 2014, 2011; A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Manoj Kumar et al., 2015; Martjushev et al., 2015; Ostovar et al., 2020, 2017, 2016; Pauwels & Calders, 2019; Seeliger et al., 2017), and some apply change point detection or change detector algorithms (Carmona & Gavaldà, 2012; Hassani, 2019; Impedovo et al., 2020; Yeshchenko et al., 2021; Yeshchenko, Ciccio, et al., 2019; Yeshchenko, Di Ciccio, et al., 2019; Yeshchenko, Mendling, et al., 2020). We also identified trace clustering approaches considering the time component (Rafael Accorsi et al., 2012; Barbon Junior et al., 2018; Luengo & Sepúlveda, 2012; Mora et al., 2020; Richter & Seidl, 2017; Tavares et al., 2019; Zellner et al., 2020). Some approaches provided visual analysis: performance spectrum (Denisov et al., 2018), plots providing similarity matrices over time (Hompes et al., 2017), or heat maps plotting the Earth Mover's Distance (*EMD*) value to identify potential drifts (Brockhoff et al., 2020). Some other approaches are clustering over intervals obtained from a relation matrix (Zheng et al., 2017), comparing footprint matrices over time (N. Liu et al., 2018), and discovery and conformance checking analysis of process histories for online environments (Stertz & Rinderle-Ma, 2018, 2019).

Some of the approaches are part of the process mining tools: ProM – Concept Drift plugin (Bose et al., 2014, 2011; Martjushev et al., 2015), Trace Clustering plugin (Hompes et al., 2017), PSM plugin (Denisov et al., 2018); and Apromore – ProDrift plugin (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Ostovar et al., 2020, 2016, 2017). Other approaches have the source code available on GitHub[8]: Process Drift Detector plugin for

[8] Source code repository: https://github.com/

ProM[9] (Seeliger et al., 2017), Tsinghua Process Concept Drift Detection – TPCDD[10] (Zheng et al., 2017), Concept Drift in Event Stream Framework – CDESF[11] (Barbon Junior et al., 2018; Mora et al., 2020; Tavares et al., 2019), Visual Drift Detection System – VDD[12] (Yeshchenko et al., 2021; Yeshchenko, Ciccio, et al., 2019; Yeshchenko, Di Ciccio, et al., 2019; Yeshchenko, Mendling, et al., 2020), Dynamic Outlier Aggregation[13] (Zellner et al., 2020), and OTOSO[14] (Richter et al., 2020). However, eight approaches do not provide the source code nor the experimental tool developed for download.

Analyzing the available approaches, we still find some challenges.

(i) Accuracy is sensitive to the parameter configuration

The methods are susceptible to parameter configuration, such as window size for statistical approaches or even specific parameters for the clustering approaches. Even the adaptive approaches show this dependency.

(ii) Lack of objective evaluation metrics reports

The available tools do not report the evaluation metrics (e.g., F-score, mean delay) in the user interface. The evaluation metrics are also not reported in all proposed approaches, and when they are reported, it is sometimes unclear how the authors calculate them. As the current methods are sensitive to the parameter configuration, comparing distinct approaches and performing parameter tuning is challenging. We did not identify the possibility of changing the parameters and obtaining the drift detection accuracy results based on an objective metric in the available tools.

(iii) Absence of a simple visualization of the evolution of the process

Most approaches only deal with drift and change point detection, localization, and characterization. We identified only one approach, the VDD (Yeshchenko et al., 2021; Yeshchenko, Ciccio, et al., 2019; Yeshchenko, Di Ciccio, et al., 2019; Yeshchenko, Mendling, et al., 2020), which addresses all the challenges. However, the visualization of the changing process mixes a directly-follows graph with constraints derived from the Declare

---

[9] Source code available at https://github.com/alexsee/processdriftdetector

[10] Source code available at https://github.com/THUBPM/process-drift-detection

[11] Source code available at https://github.com/gbrltv/CDESF and https://github.com/gbrltv/CDESF2

[12] Source code available at https://github.com/yesanton/Process-Drift-Visualization-With-Declare

[13] Source code available at https://github.com/zellnerlu/DOA

[14] Source code available at https://github.com/Skarvir/OTOSO

models (van der Aalst et al., 2009) applied in the approach, which does not provide a simple visualization of the changes in the process.

Applying the available tools in real-world situations is an arduous and, sometimes, not a viable task because of the identified problems.

1.3 RESEARCH AIM

This research aims to develop a tool for concept drift detection in process models that address the previously identified issues. In other words, the developed tool aims to minimize the impact of the parameter configuration on the detection accuracy, providing an objective evaluation metric report for supporting the user to evaluate the results quickly and supplying the user with a simple visualization of the process evolution.

1.4 RESEARCH OBJECTIVES AND CONTRIBUTIONS

The following objectives support the aim of this research:

1. Identify and understand the tools and approaches for concept drift detection in process models.
2. Propose and validate a sudden concept drift detection tool that allows the user to change the parameters and quickly evaluate the results based on an objective, showing the process' evolution over time.
3. Extend the tool, including an adaptive approach for the control-flow perspective, aiming to reduce the impact of user-defined parameters on detection accuracy.

The first contribution of this research is the structured revision of the topic of concept drift in process mining (Section 2), describing the main concepts and the existing approaches. Based on the performed review, it was possible to identify the current challenges to applying concept drift detection analysis in real-world scenarios.

Another contribution is the proposed tool for concept drift detection (Sections 3 and 4), which detects drift in process models with a simplified parameter configuration, outputting the result of the detection mechanism using the process models (process evolution replay) and showing evaluation metrics. Also, the tool performs drift detection in the control-flow based on an adaptive windowing mechanism (Section 5), aiming to reduce the dependency of the parameter configuration.

We extended the concept drift detection tool to handle drifts in the time and data perspectives based on an adaptive approach. Although this is not included in the research objectives, the extension provided information about time drifts for maintenance decision-making in the condition-based maintenance policy. Tests on synthetic databases and a case study with a real-life dataset were conducted to validate the method and prove its

effectiveness by numerical evaluation. The main contribution is applying the developed tool to a real problem in the industry, showing promising results in a practical problem with a structured process. Section 6 contains the paper addressing the case study applied in a maintenance process.

1.5 RESEARCH METHODOLOGY

We applied the design science research process (Peffers et al., 2014) in this research. Table 1.1 shows the description of each step of the process.

Table 1.1. Design Science Research Process Description

Process step	Description
Identify problem & motivate	Identify the pros and cons of the current tools for concept drift detection in process models.
Define Objectives of a Solution	Develop a tool for concept drift detection in process models that handles some of the issues of the current approaches.
Design & Development	Tool for concept drift detection in process models.
Demonstration	Evaluate the accuracy of the developed tool using a synthetic dataset.
Evaluation	Compare the accuracy of the developed tool with other available tools.
Communication	Publish the papers (one per research objective) and the thesis. Publicize the source code of the developed tool.

1.6 DOCUMENT STRUCTURE

The following sections of this document are written as a research paper, each addressing one research objective. At the beginning of each section, the problem of concept drift in process models and the related works may become repetitive. After reading Section 2, which contains the literature review about the topic and details of the current approaches, the reader may skip the introduction and related work topics from the following sections.

Section 1 introduces the problem addressed by this research and presents the research goals and methodology. Section 2 describes a systematic literature review of the concept drift in process mining, explaining the main concepts and the current approaches for dealing with concept drift in processes. Sections 3 and 4 present the Interactive Process Drift Detection (IPDD) framework with the first implementation and validation. Section 5 extends the *IPDD* to detect control-flow drifts using an adaptive approach and compares it to other available tools. Section 6 describes the application of the IPDD to support condition-based maintenance policies by detecting time and data drifts. Finally, Section 7 concludes this thesis by highlighting the main results and future work. Figure 1.4 shows the structure of this document graphically.

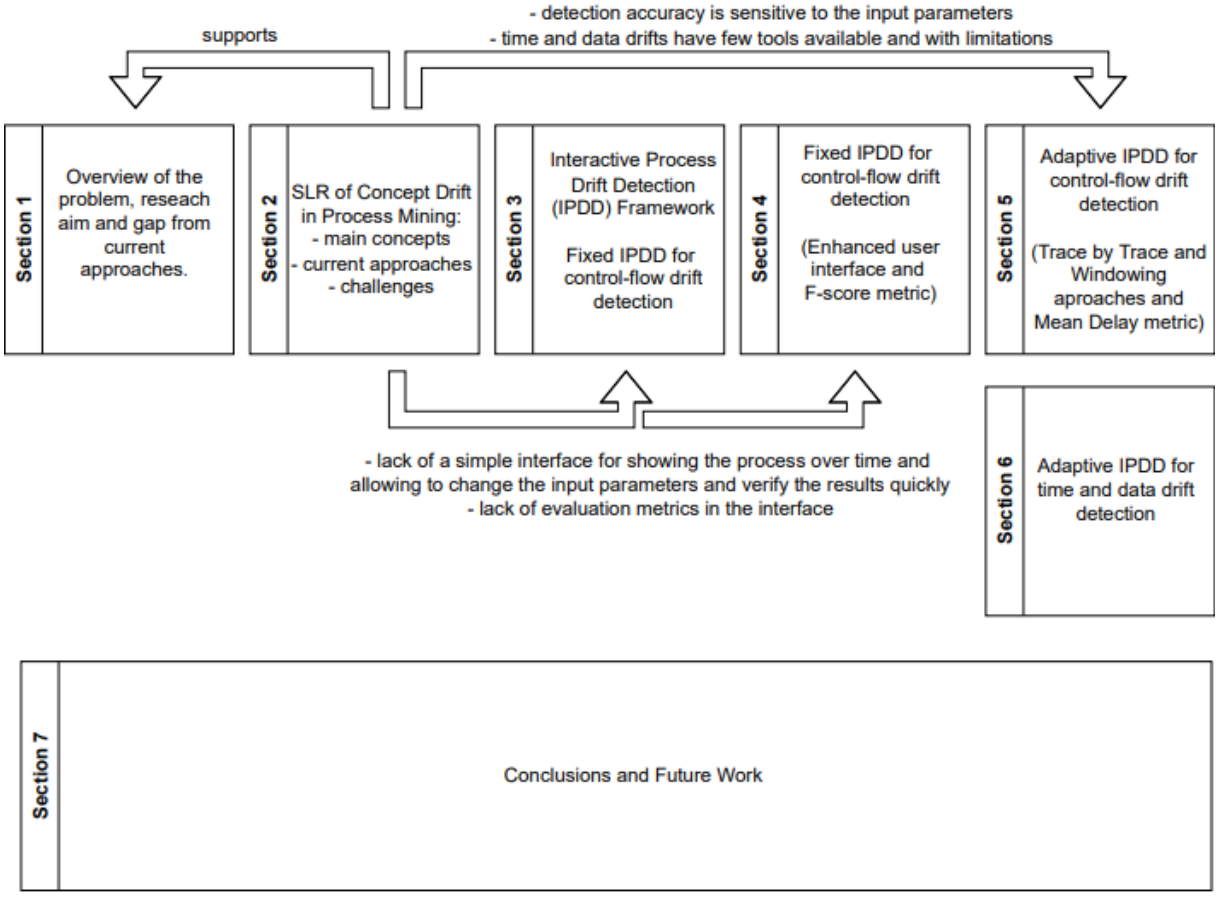


Figure 1.4. Thesis structure.

2 PAPER #1: A SURVEY ON CONCEPT DRIFT IN PROCESS MINING

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

2.1 INTRODUCTION: BACKGROUND AND MOTIVATION

Process Mining (PM) is a growing research area that provides techniques to understand and improve processes in different application domains (Garcia et al., 2019). 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 (van der Aalst, 2016), 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 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 (van Zelst et al., 2018). 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 2.1 shows the sequence of activities executed in an online store using the Petri Net notation (van der Aalst, 2016).

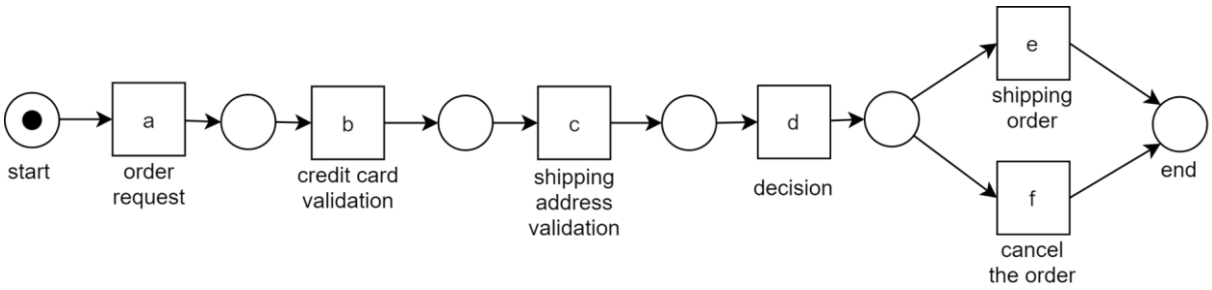


Figure 2.1. Petri net model for the e-commerce process.

The process contains the following activities: order request (a), credit card validation (b), shipping address validation (c), decision (d), shipping order (e), and cancel the order (f). Activities a to d are sequentially executed, then there is a choice between e and f.

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 2.2 shows an example of event data recorded for the e-commerce process (Figure 2.1). 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 2.2 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: $L = [< a, b, c, d, e >^2, < a, b, c, d, f >^1, \dots]$.

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	
...	...	...	...	

Figure 2.2. 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 (Bose et al., 2014).

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 (van der Aalst et al., 2011) 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 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 2.2, we introduce the concept drift in the PM context, presenting its main challenges. Sections 2.3 and 2.4 overview current methods that can be applied to evolving processes. Section 2.5 discusses the main achievements of the existing approaches and highlights open challenges. Finally, Section 2.6 concludes the survey.

2.2 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 (Ditzler et al., 2015; J. Lu et al., 2019). In this situation, the learning algorithms need to adapt themselves on the fly to react to concept drifts appropriately (Gama et al., 2014). 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.

2.2.1 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 2.1. We have also included the workshops from the International Conference on PM Series (ICPM) because of their relevance to the PM area.

Table 2.1. 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”))	56
Springer Link	(“concept drift” OR “process drift”) AND “process mining” - LIMITED TO ENGLISH	140
ICPM Workshops	“concept drift”	2
Total		**231**

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 (Kitchenham & Charters, 2007):

- 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.
- 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 (Wohlin, 2014) 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 2.3 and 2.4 describe the identified approaches for dealing with concept drifts.

2.2.2 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 (P. Ceravolo et al., 2020; Omori et al.,

2019) 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 (Abderrahmane Maaradji et al., 2015), is the situation when the process changes while being analyzed (van der Aalst, 2016). 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 2.3. The following topics illustrate and explain the aspects depicted above.

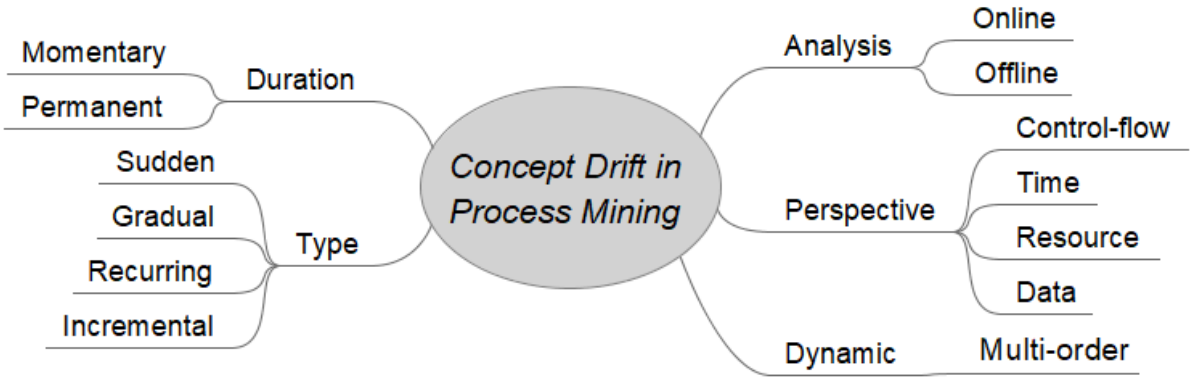


Figure 2.3. Different aspects of defining and characterizing concept drift in process mining.

2.2.3 PERSPECTIVES OF CHANGE

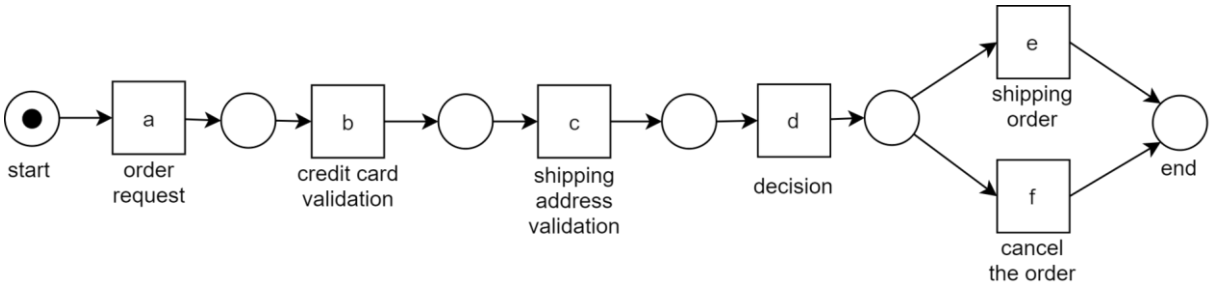
A process model typically contains information about different perspectives; therefore, the changes can occur in one or more of them. Authors in (Bose et al., 2014, 2011) highlighted three essential perspectives in business processes: control-flow, data, and resource – also named as organizational perspective by (van der Aalst, 2016). Another vital view in PM is the time perspective, which provides performance indicators as lead, waiting, service, and synchronization times. Despite this perspective being widely explored for process analysis, we found fewer papers exploring it for concept drifts (Barbon Junior et al., 2018; Brockhoff et al., 2020; Mora et al., 2020; Richter & Seidl, 2019, 2017; Tavares et al., 2019).

2.2.3.1 Control-flow

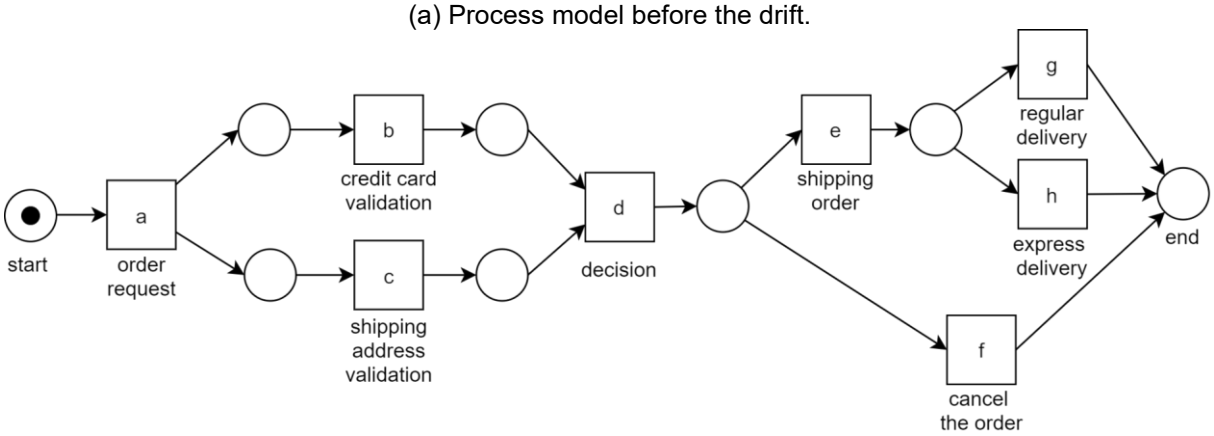
Changes in this perspective represent behavioral or structural changes in the process model. Process models can be imperative, e.g., Petri nets or BPMN, or declarative, e.g., Declare models (van der Aalst et al., 2009). Authors in (B. Weber et al., 2008) suggest a set of 13 structural change patterns 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 2.4 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.

Figure 2.4. 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.

2.2.3.2 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 (van der Aalst, 2016). A change in this perspective deals with significant changes in process performance over time (Barbon Junior et al., 2018; Brockhoff et al., 2020; Mora et al., 2020; Tavares et al., 2019). 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.

2.2.3.3 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.

2.2.3.4 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 (Hompes et al., 2017; Pauwels & Calders, 2019; Stertz & Rinderle-Ma, 2019) explicitly deal with this perspective.

2.2.4 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 (Schonenberg et al., 2008). 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 (Bose et al., 2014, 2011) identified four distinct types of drifts: sudden, gradual, recurring, and incremental.

2.2.4.1 Type of Drift

Figure 2.5 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.

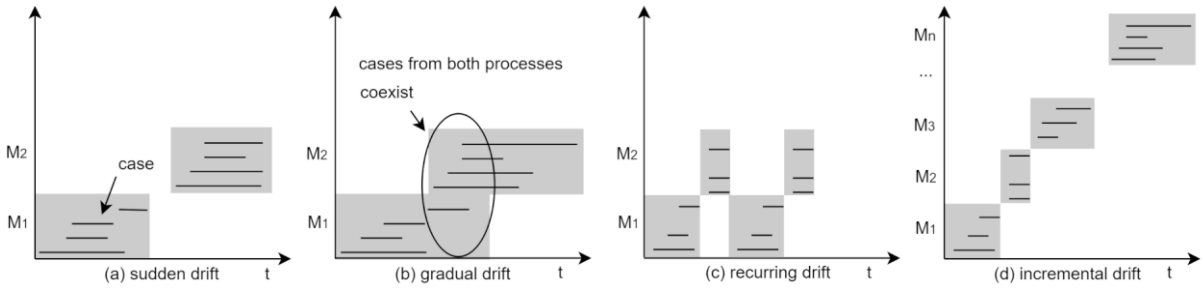


Figure 2.5. Four different types of drift. Adapted from (Bose et al., 2014).

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 2.5 (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 (Ostovar et al., 2016). 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 2.5 (b), after some time, only instances of M_2 will exist. Figure 2.6 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 2.6 (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 2.6 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 2.6 (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 $P[M_2] = (1 - P[M_1])$ (Martjushev et al., 2015).

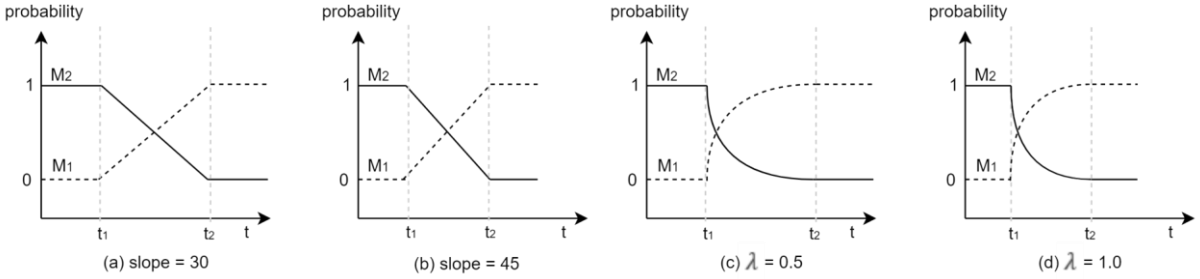


Figure 2.6. (a) and (b) shows variants of gradual linear drift, with slope defining the rate of change. (c) and (d) shows variants of exponential gradual drift, characterized by the function $e^{-\lambda t}$ for M_1. Adapted from (Martjushev et al., 2015).

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 2.5 (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 2.5 (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 2.5, they are shown as discrete sudden changes.

2.2.4.2 Dynamic of change

In (Martjushev et al., 2015) 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.

2.2.5 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 (van der Aalst, 2016)) 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 (van Zelst et al., 2018). 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 (Buttazzo, 2011). 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 (N. Liu et al., 2018; Abderrahmane Maaradji et al., 2015). 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.

2.2.6 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 2.2). Sections 2.3 and 2.4 detail the approaches.

Table 2.2. Classification of papers dealing with concept drift in PM.

Approach	Papers	Number of papers
Concept drift detection	(Rafael Accorsi et al., 2012; Barbon Junior et al., 2018; Bose et al., 2014, 2011; Brockhoff et al., 2020; Carmona & Gavaldà, 2012; P. Ceravolo et al., 2020; Hassani, 2019; Hompes et al., 2017; Impedovo et al., 2020; Kurniati et al., 2020; Kurniati A.P., McInerney C., Zucker K., Hall G., Hogg D., 2019; L. Lin et al., 2020; N. Liu et al., 2018; Luengo & Sepúlveda, 2012; A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Manoj Kumar et al., 2015; Martjushev et al., 2015; Mora et al., 2020; Omori et al., 2019; Ostovar et al., 2020, 2017, 2016; Pauwels & Calders, 2019; Richter et al., 2020; Richter & Seidl, 2019, 2017; Seeliger et al., 2017; Stertz & Rinderle-Ma, 2019, 2018; Tavares et al., 2019; Yeshchenko et al., 2021; Yeshchenko, Ciccio, et al., 2019; Yeshchenko, Di Ciccio, et al., 2019; Yeshchenko, Mendling, et al., 2020; Zellner et al., 2020; Zheng et al., 2017)	38
Online PM dealing with evolving environments	(Anatoliy Batyuk & Voityshyn, 2020; A Burattin et al., 2015, 2014; Andrea Burattin et al., 2015; Maggi et al., 2013; Redlich et al., 2014; van Zelst et al., 2018)	7

2.3 CONCEPT DRIFT DETECTION

The approaches for concept drift detection usually addressed one or more challenges described in (Bose et al., 2014, 2011). We differentiate drift detection from change point (CP) detection because some approaches only detect the drift without reporting the CP (Carmona & Gavaldà, 2012; Hassani, 2019; Kurniati et al., 2020; Kurniati A.P., McInerney C., Zucker K., Hall G., Hogg D., 2019). We also separate “Change localization and characterization”, defined in (Bose et al., 2014, 2011), as such approaches usually focus on localization.

(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 (Yeshchenko et al., 2021; Yeshchenko, Ciccio, et al., 2019; Yeshchenko, Di Ciccio, et al., 2019) , 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 2.3 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 2.3 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 (L. Lin et al., 2020; Martjushev et al., 2015), the authors clearly define the evaluation mechanism (Figure 2.7). The difference from other papers is that the authors apply a lag period *l* surrounding a detected or an actual drift. In (Zheng et al., 2017), 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 (Abderrahmane Maaradji et al., 2015). It points out how early the approach is able to detect an actual change (Seeliger et al., 2017). For the sake of homogeneity, hereafter, we use the *detection delay* term.

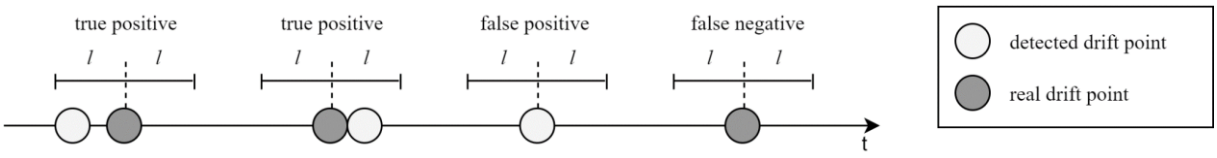


Figure 2.7. Definition of evaluation metrics. Adapted from (Martjushev et al., 2015).

The authors in (Luengo & Sepúlveda, 2012) 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 (Tavares et al., 2019), 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 2.3. The authors in (Impedovo et al., 2020) report an accuracy value (from 0 to 1) without defining the metric, so we did not include it in Table 2.3.

We only considered online approaches those that use event streams as input. The authors in (Carmona & Gavaldà, 2012) 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 (N. Liu et al., 2018) 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 2.3.8. 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 2.3 highlights that synthetic datasets containing artificially injected drifts are scarce. We only found three publicly available datasets that we further detail in Appendix 2.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 2.4, and the most common approach is statistical hypothesis testing.

The two papers (P. Ceravolo et al., 2020; Omori et al., 2019) 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 2.3.8 describes the two papers that compare drift detection approaches.

Table 2.3. 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
(Bose et al., 2014, 2011; Martjushev et al., 2015)	1,2,3	Control-flow	Sudden, Gradual (Bose et al., 2014; Martjushev et al., 2015)	Offline	ProM (ConceptDrift)	SD, RD (PA)	F-score CD (Martjushev et al., 2015)
(Luengo & Sepúlveda, 2012)	3	Control-flow	Gradual	Offline	Tool NA	SD	% of correctly classified traces
(Rafael Accorsi et al., 2012)	1,2,3	Control-flow	Sudden	Offline	Tool NA	SD	-
(Carmona & Gavaldà, 2012)	1	Control-flow	Sudden	Online	Tool NA	SD	-
(Manoj Kumar et al., 2015)	1,2	Control-flow	Sudden	Offline	Tool NA	SD (PA)	-
(A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Ostovar et al., 2020, 2016, 2017)	1,2,3	Control-flow	Sudden, Gradual(A Maaradji et al., 2017)	Offline(A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015), Online (Ostovar et al., 2020, 2016, 2017)	Apromore (ProDrift)	SD (PA), RD (PA)	F-score, detection delay
(Hompes et al., 2017)	1,2	Control-flow, Data	Sudden	Offline	ProM (TraceClustering)	SD, RD (PA)	-
(Seeliger et al., 2017)	1,2,3	Control-flow	Sudden	Offline	Experimental ProM plug-in[15]	SD (PA)	F-score, detection delay
(Zheng et al.,	1,2	Control-flow	Sudden	Offline	TPCDD[16]	SD	F-score CD

[15] Source code available at https://github.com/alexsee/processdriftdetector.

Paper(s)	Challenge(s)	Perspective(s)	Type(s)	Analysis	Software	Dataset	SD evaluation
2017)							
(Richter & Seidl, 2019, 2017)	1,2,3	Time	Sudden, Incremental, Recurring	Online	ProM (Tesseract)	SD, RD (PA)	Detection delay
(Barbon Junior et al., 2018; Mora et al., 2020; Tavares et al., 2019)	1,2	Control-flow and time combined	Not defined, Sudden (Tavares et al., 2019)	Online	CDESF[17]	SD (PA), RD (PA)	N. of detected drifts
(N. Liu et al., 2018)	1,2	Control-flow	Sudden, Recurring	Offline	Tool NA	RD (PA)	F-score
(Stertz & Rinderle-Ma, 2018, 2019)	1,2	Control-flow (Stertz & Rinderle-Ma, 2018), data (Stertz & Rinderle-Ma, 2019)	Sudden, Gradual, Incremental, Recurring	Online	Tool NA	SD	-
(Kurniati et al., 2020; Kurniati A.P., McInerney C., Zucker K., Hall G., Hogg D., 2019)	1	Control-flow	Not defined	Offline	Manual method	RD	-
(Pauwels & Calders, 2019)	1,2	Control-flow and data combined	Not defined	Offline	EDBN[18]	RD (PA)	-

[16] Source code available at https://github.com/THUBPM/process-drift-detection. Event logs are not made available.

[17] Source code available at https://github.com/gbrltv/CDESF (Barbon Junior et al., 2018). Updated version available in https://github.com/gbrltv/cdesf2 (Mora et al., 2020; Tavares et al., 2019).

[18] Source code available at https://github.com/StephenPauwels/edbn.

Paper(s)	Challenge(s)	Perspective(s)	Type(s)	Analysis	Software	Dataset	SD evaluation
(Hassani, 2019)	1	Control-flow	Sudden	Online	StrProMCDD NA	SD, RD (PA)	F-score, detection delay
(Yeshchenko et al., 2021; Yeshchenko, Ciccio, et al., 2019; Yeshchenko, Di Ciccio, et al., 2019; Yeshchenko, Mendling, et al., 2020)	1,2,3,4 (Yeshchenko et al., 2021; Yeshchenko, Mendling, et al., 2020)	Control-flow	Sudden, Gradual, Incremental, Recurring	Offline	VDD[19]	SD (PA), RD (PA)	F-score (Yeshchenko et al., 2021; Yeshchenko, Di Ciccio, et al., 2019)
(Zellner et al., 2020)	1,2	Control-flow	Recurring	Offline	DOA[20]	SD, RD (adapted)	F-score
(Richter et al., 2020)	1,2	Control-flow	Not defined	Online	OTOSO[21]	RD (PA)	-
(Brockhoff et al., 2020)	1,2	Control-flow and time combined	Sudden	Offline	ProM	SD	-
(L. Lin et al., 2020)	1,2	Control-flow	Sudden	Offline	LCDD[22]	SD (PA), RD (PA)	F-score CD
(Impedovo et al., 2020)	1,2,3	Control-flow	Sudden	Offline	Tool NA	SD, RD (PA)	-

[19] 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/.

[20] Source code available at https://github.com/zellnerlu/DOA.

[21] Source code available at https://github.com/Skarvir/OTOSO.

[22] Source code available at https://github.com/lll-lin/THUBPM.

Table 2.4. 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	(Bose et al., 2014, 2011; A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Manoj Kumar et al., 2015; Martjushev et al., 2015; Ostovar et al., 2020, 2017, 2016; Pauwels & Calders, 2019; Seeliger et al., 2017)	11
Trace clustering	(Rafael Accorsi et al., 2012; Barbon Junior et al., 2018; Luengo & Sepúlveda, 2012; Mora et al., 2020; Richter & Seidl, 2017; Tavares et al., 2019; Zellner et al., 2020)	7
CP detection	(Yeshchenko et al., 2021; Yeshchenko, Ciccio, et al., 2019; Yeshchenko, Di Ciccio, et al., 2019; Yeshchenko, Mendling, et al., 2020)	4
Visual analysis	(Brockhoff et al., 2020; Hompes et al., 2017; Kurniati et al., 2020; Kurniati A.P., McInerney C., Zucker K., Hall G., Hogg D., 2019)	4
Change detection	(Carmona & Gavaldà, 2012; Hassani, 2019; Impedovo et al., 2020)	3
Trend detection	(Richter & Seidl, 2019, 2017)	2
Other approaches	(L. Lin et al., 2020; N. Liu et al., 2018; Stertz & Rinderle-Ma, 2018, 2019; Zheng et al., 2017)	5

2.3.1 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.

2.3.1.1 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 (Bose et al., 2014, 2011). 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 2.8 shows an overview of the papers' steps to explain the approach to detect and pinpoint a process drift (challenges 1 and 2 from Section 2.3) 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.

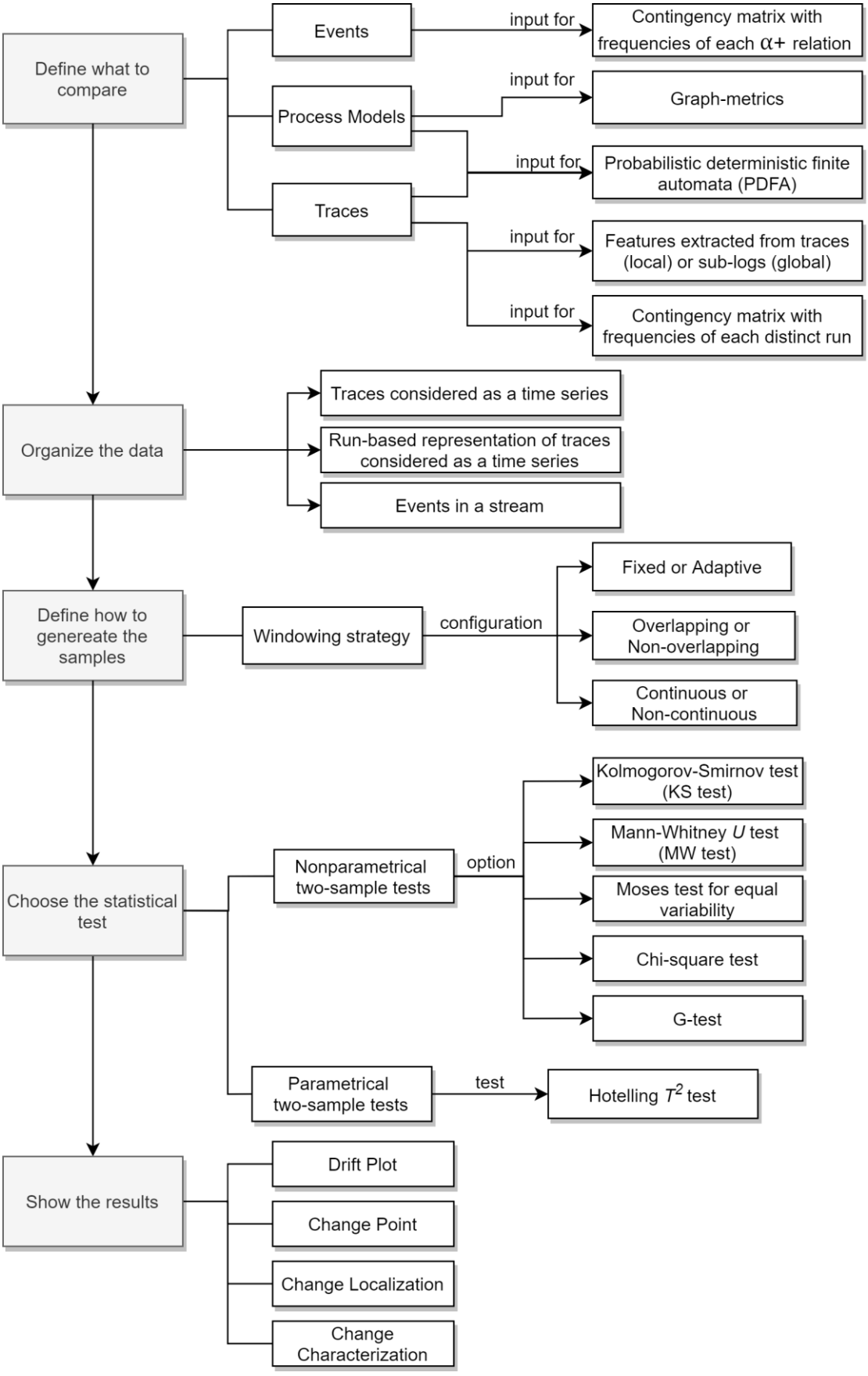


Figure 2.8. Steps of statistical hypothesis testing approaches to detect concept drift in processes.

Figure 2.8 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 2.5 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 (Bose et al., 2014, 2011; A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Manoj Kumar et al., 2015; Martjushev et al., 2015; Ostovar et al., 2020, 2016, 2017; Pauwels & Calders, 2019) 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 (Seeliger et al., 2017), the authors obtain graph metrics from discovered process models (discovered applying the Heuristics Miner algorithm (Weijters et al., 2006): (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 (Bose et al., 2014, 2011) 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 (Abderrahmane Maaradji et al., 2015). The approach defined in (Martjushev et al., 2015) used the *J measure* feature former defined by (Bose et al., 2014, 2011), so it shares the same drawbacks aforementioned. Authors in (Manoj Kumar et al., 2015) 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 (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015) proposed a run-based representation of the traces and the notion of

run-equivalence, derived from the field of concurrency theory (van Glabbeek & Goltz, 1989). 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 (Ostovar et al., 2020, 2016, 2017), authors inherit the α+ relations defined by the α+ algorithm (De Medeiros et al., 2004) to generate a contingency matrix containing the frequencies of each identified relation, obtained from sub-logs derived from a set of events. The α+ 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 (Pauwels & Calders, 2019) 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 (Bose et al., 2014, 2011; Manoj Kumar et al., 2015; Martjushev et al., 2015; Pauwels & Calders, 2019). The method applied in (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015) 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 (Seeliger et al., 2017) are also discovered based on the traces as time series. In contrast, the approach for drift detection applied in (Ostovar et al., 2020, 2016, 2017) 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 (Abderrahmane Maaradji et al., 2015). 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 (Mermillod et al., 2013). Even the adaptive window approach proposed in (Martjushev et al., 2015) 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 (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015) dynamically changes the window size based on the variation observed in the event log, inspired by the ADWIN method proposed in (Bifet & Gavaldà, 2007). 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 (Ostovar et al., 2016), the authors use the same adaptive window approach without specifying the initial window size. In Table 2.5 we detail the windowing approach for each paper, describing if the window size is a parameter or not. Unfortunately, the authors in (Pauwels & Calders, 2019) 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 used in (Bose et al., 2014, 2011). 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 (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Ostovar et al., 2016) 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 (Seeliger et al., 2017) only reports the CPs, and the paper (Pauwels & Calders, 2019) plots the case scores instead of the p-value, highlighting the drift points.

Table 2.5 reports the details for each of the steps defined in Figure 2.8 for the papers using a statistical hypothesis testing approach to detect concept drift in processes.

Both approaches (Ostovar et al., 2020, 2017) applied the same drift detection mechanism proposed in (Ostovar et al., 2016). The papers are organized by their year of publication and title, using ascending order.

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 (Bose et al., 2014; A Maaradji et al., 2017; Martjushev et al., 2015). The method implemented in the Concept Drift plugin (Bose et al., 2014) tailored for ProM (van der Aalst et al., 2009) 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 (Martjushev et al., 2015), 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 (A Maaradji et al., 2017). 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 (A Maaradji et al., 2017), implemented in the Apromore Tool (La Rosa et al., 2011). 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.

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 (Martjushev et al., 2015). The results of the real-life log are validated by the business analyst, confirming the detected drift. In (A Maaradji et al., 2017), the authors reported that the business analyst did not recognize the gradual drift reported in (Martjushev et al., 2015), meaning that the detected drift is a FP.

Table 2.5. 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
(Bose et al., 2011)	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)
(Bose et al., 2014)	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)
(Manoj Kumar et al., 2015)	Traces	FTs from traces as time series (global)	Non-overlapping, continuous, fixed-size (parameter)	Mann-Whitney *U*, Moses for equal variability	DP (trace)
(Abderrahmane Maaradji et al., 2015)	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
(Ostovar et al., 2016)	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
(Ostovar et al., 2017)	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
(Seeliger et al., 2017)	Process Models	Graph-metrics as time series	Non-overlapping, continuous, adaptive-size (estimated)	G-test	CPs
(A Maaradji et al., 2017)	Traces	Runs as time series (global)	Non-overlapping, continuous, fixed-size (parameter) and adaptive-size (initial WS as parameter)	Chi-square, Chi-square goodness-of-fit	DP (trace) with CP
(Pauwels & Calders, 2019)	Traces	Case scores based on the EDBN	Not defined	Kolmogorov-Smirnov	Case scores plot with CP
(Ostovar et al., 2020)	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

2.3.1.2 Change localization

The methods for dealing with concept drift proposed in (Bose et al., 2014, 2011) provide an approach for localizing the drifts. The user can interact with the ProM software (van der Aalst et al., 2009), using the Concept Drift plugin, and select two activities to identify whether there is a drift between them.

Authors in (Ostovar et al., 2017) 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 α+ relations - defined by α+ algorithm (De Medeiros et al., 2004) - from two windows, one before and another after the drift using the drift detection approach from (Ostovar et al., 2016). 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 α+ relations before and after it. K-sample permutation test (*KSPT*), as suggested in (Frank & Witten, 1998), measures the association between each α+ relation (attributes) and the label indicating pre or post-drift (target), to discard irrelevant relations. Next, the method ranks the remaining α+ 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 (Stertz & Rinderle-Ma, 2019).

In (Ostovar et al., 2020), the authors propose a new method for characterizing the drifts, also based on the drift detection mechanism presented in (Ostovar et al., 2016), 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 (Ostovar et al., 2020), and it was extensively validated using 365 artificial logs and two real-life logs.

2.3.2 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 (Richter et al., 2020), 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 (Barbon Junior et al., 2018; Mora et al., 2020; Tavares et al., 2019) 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 (Rafael Accorsi et al., 2012) 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 (Luengo & Sepúlveda, 2012), the authors include the starting time of each process instance as an additional feature used by the clustering approaches. In (Zellner et al., 2020), 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 (Richter et al., 2020) 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 2.6 describes each aspect for the identified papers. The papers are organized according to their publication year and title, in ascending order.

Table 2.6. Trace clustering considering the time component.

Paper	**Features**	**Clustering strategy**	**Time**	**Output**
(Luengo & Sepúlveda, 2012)	Maximal Repeat Feature Set (*MR*)	Agglomerative Hierarchical Clustering (AHC) (Crc, 2014)	Timestamp of the 1st event within the trace as a feature.	Clusters
(Rafael Accorsi et al., 2012)	Activity distance	Cluster cuts algorithm (defined by the authors)	After training, each trace is processed by the clustering algorithm	Combined cluster cuts graph
(Barbon Junior et al., 2018)	Edit and time weighted distances (*EWD*, *TWD*) and time (last event's timestamp)	DBSCAN (Ester et al., 1996)	Time interval triggers the clustering, and time composes features for clustering	Flow analysis graph and snapshots of clusters
(Mora et al., 2020; Tavares et al., 2019)	Graph-distance trace and time (*GDtrace*, *GDtime*)	DenStream (F. Cao et al., 2006)	Time interval triggers the clustering, and time is a feature	Drift plot and snapshots of clusters
(Richter et al., 2020)	Z-scoring of temporal deviation signature (*TDS*)	OPTICS (Ankerst et al., 1999)	Events stored in a hash table using the Cuckoo-Hashing	OTOSO plot (time-dependent graph)
(Zellner et al., 2020)	Local outlier factor (*LOF*) (Breuniq et al., 2000), calculated for non-conforming traces	Aggregate traces with *LOF* scores below (*T*) in one micro-cluster, when the nº 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

2.3.2.1 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 (Luengo & Sepúlveda, 2012) used the Maximal Repeat Feature Set (MR) (Bose & van der Aalst, 2010), 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 (Rafael Accorsi et al., 2012) 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 (Barbon Junior et al., 2018) 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 (Mora et al., 2020; Tavares et al., 2019), 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 (Richter et al., 2020), the method uses z-scoring for the 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 (Zellner et al., 2020) calculate the (*LOF*) (Breuniq et al., 2000), which describes its “outlierness” concerning the surrounding neighborhood. A difference from other approaches is that the LOF is only calculated for non-conforming traces.

2.3.2.2 Clustering strategy

Clustering algorithms group sets of data based on their similarity, maximizing intra-cluster similarity, and minimizing inter-cluster similarity (Crc, 2014). They can use different techniques: partitioning, hierarchical, density-based, grid-based, and model-based. Authors in (Luengo & Sepúlveda, 2012) apply the Agglomerative Hierarchical Clustering (AHC) (Crc, 2014) with the minimum variance criterion, using the Euclidian distance between feature vectors. In (Rafael Accorsi et al., 2012), 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 (Barbon Junior et al., 2018), the authors use DBSCAN (Ester et al., 1996), 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 (Mora et al., 2020; Tavares et al., 2019) applies DenStream (F. Cao et al., 2006) because it is a density-based clustering method suitable for online scenarios. In (Richter et al., 2020), the proposed monitoring tool applies OPTICS (Ankerst et al., 1999) to derive the density-based clusters. Authors (Zellner et al., 2020) 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.

2.3.2.3 Time

The trace clustering approaches must consider the timestamp of the trace in the cluster definition to detect drifts. In (Luengo & Sepúlveda, 2012), 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 (Rafael Accorsi et al., 2012), each trace is processed as a time series (based on its timestamp), and after each trace, a new cluster may be declared. Authors in (Barbon Junior et al., 2018) 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 (Barbon Junior et al., 2018) 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 (Richter et al., 2020). 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 (Zellner et al., 2020) 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.

2.3.2.4 Output

The output of this type of approach can be the clusters (Luengo & Sepúlveda, 2012), a plot indicating the CP (Rafael Accorsi et al., 2012), or the set of cases representing the concept drift (Barbon Junior et al., 2018). In (Luengo & Sepúlveda, 2012), 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 (Rafael Accorsi et al., 2012). And the CDESF framework (Barbon Junior et al., 2018) 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 (Mora et al., 2020; Tavares et al., 2019) CDESF outputs a drift plot and a snapshot of the cluster formation. The OTOSO plot is the output provided in (Richter et al., 2020), 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 (Zellner et al., 2020) 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 (Barbon Junior et al., 2018; Mora et al., 2020; Tavares et al., 2019) 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 (F. Cao et al., 2006), which is suitable for online scenarios (Tavares et al., 2019). OTOSO (Richter et al., 2020) 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 (Ankerst et al., 1999) to derive the clusters and use a time-based graph to show the different structural changes. On the other hand, the approach proposed in (Rafael Accorsi et al., 2012) 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 (Abderrahmane Maaradji et al., 2015). The *Dynamic Outlier Aggregation* (Zellner et al., 2020) 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 (P. Ceravolo et al., 2020), 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.

2.3.3 CHANGE POINT DETECTION

CP detection methods identify the points in which multivariate time series, showing changes in their values (Truong et al., 2020). Authors in (Yeshchenko et al., 2021; Yeshchenko, Ciccio, et al., 2019; Yeshchenko, Di Ciccio, et al., 2019; Yeshchenko, Mendling, et al., 2020) apply a CP detection method named Pruned Exact Linear Time (PELT) (Killick et al., 2012), 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 $T_i = Conf_{i,1}, \dots, Conf_{i,\#win}$. 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 (Yeshchenko, Di Ciccio, et al., 2019). The method derives a multivariate time series $D = \{T_1, T_2, \dots T_{\#cns}\}$ 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 (Yeshchenko, Ciccio, et al., 2019; Yeshchenko, Di Ciccio, et al., 2019) into sudden, gradual, recurring, or incremental (challenges 3 and 4). The user can also inspect if the detected CP has an outlier behavior. In (Yeshchenko et al., 2021; Yeshchenko, Mendling, et al., 2020), 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 (Yeshchenko et al., 2021; Yeshchenko, Mendling, et al., 2020), 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 (Yeshchenko et al., 2021; Yeshchenko, Di Ciccio, et al., 2019), but this suggestion is based on a good visualization of the plots, not on tuning the accuracy of the tool. In (Yeshchenko et al., 2021; Yeshchenko, Di Ciccio, et al., 2019), 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 (Yeshchenko et al., 2021; Yeshchenko, Di Ciccio, et al., 2019) is unclear.The authors compare VDD with Apromore (Ostovar et al., 2016), 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 (Yeshchenko et al., 2021; Yeshchenko, Di Ciccio, et al., 2019) reported that VDD described all types of process drifts comprehensively. Using the BPIC 2011 dataset, the experiment detected all the drifts reported in (A Maaradji et al., 2017). All the experiments using real-world datasets in (Yeshchenko et al., 2021; Yeshchenko, Di Ciccio, et al., 2019) provided an overview of the tool and its user interface, which is complemented in (Yeshchenko et al., 2021) by a user study performed with PM experts reporting that the “visualizations are easy to interact with and useful”.

2.3.4 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 (Hompes et al., 2017) 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 (Brockhoff et al., 2020) 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 (Kurniati A.P., McInerney C., Zucker K., Hall G., Hogg D., 2019), the authors proposed a manual structured method for concept drift analysis based on the PM2 method (van Eck et al., 2015). 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 (Kurniati et al., 2020), the authors extended the work applying four miners for performance comparison: Integer Linear Programming, 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.

2.3.5 CHANGE DETECTION

Authors in (Carmona & Gavaldà, 2012) report the use of a change detection algorithm, ADWIN (Bifet & Gavaldà, 2007), 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 (Parikh, 1966). From the Parikh vector, it is possible to derive the polyhedron ($P_{\widehat{\sigma}}$). Next, the polyhedra $P_{\widehat{\sigma 1}}, P_{\widehat{\sigma 2}}, \dots, P_{\widehat{\sigma k}}$ can be derived from the *k* traces. Finally, the approach learn the convex-hull of the points represented by the polyhedra$P_{\widehat{\sigma 1}}, P_{\widehat{\sigma 2}}, \dots, P_{\widehat{\sigma k}}$, 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 (Bifet & Gavaldà, 2007).

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 (N. Liu et al., 2018). 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 (Hassani, 2019), the authors propose StrProMCDD, which applies the ADWIN method (Bifet & Gavaldà, 2007) 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 (van der Aalst, 2016). 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 (Bifet & Gavaldà, 2007). 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 (Hassani, 2019) 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 (Impedovo et al., 2020). 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 (Impedovo et al., 2020) validate the accuracy of the KARMATree approach compared with Apromore ProDrift fixed/adaptive using Runs (Abderrahmane Maaradji et al., 2015), and ProM adaptive (Martjushev et al., 2015) 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.

2.3.6 TREND DETECTION

Authors in (Richter & Seidl, 2019, 2017) 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 (Schubert et al., 2014) 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 (van der Aalst et al., 2009) plugin and considers the stream requirements: proposes an adapted Cuckoo hash-table (Fan et al., 2014) 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.

2.3.7 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.

2.3.7.1 Tsinghua Process Concept Drift Detection (TPCDD)

TPCDD (Zheng et al., 2017), is a three-stage approach based on two process similarity algorithm, TAR (Zha et al., 2010) and BP (Weidlich et al., 2011), 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 (Ester et al., 1996) 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 (Abderrahmane Maaradji et al., 2015) 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 (Abderrahmane Maaradji et al., 2015), 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.

2.3.7.2 Local Complete-based Drift Detection (LCDD)

The approach proposed in (L. Lin et al., 2020) applies the local completeness (LCO) property of event logs defined in (Yang et al., 2012) for drift detection. The idea is that it is possible to assert (with a defined confidence level *K*) that a log satisfies LCO 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 LCO 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 (L. Lin et al., 2020) 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 (Abderrahmane Maaradji et al., 2015; Ostovar et al., 2016), and TPCDD (Zheng et al., 2017). 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 (L. Lin et al., 2020) 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 (Conforti et al., 2017) 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*.

2.3.7.3 Framework for "online" process concept drift detection

In (N. Liu et al., 2018), 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.

2.3.7.4 Detecting concept drifts based on process histories from event streams

The approach proposed in (Stertz & Rinderle-Ma, 2018) 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 (Leemans et al., 2013) for streams for discovering the process models, based on the Stream-based Process Discovery (S-BAR) architecture defined in (van Zelst et al., 2018). 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 (Stertz & Rinderle-Ma, 2018) present an entirely distinct approach to detect drifts based on discovery and conformance for online environments. This work is extended in (Stertz & Rinderle-Ma, 2019) to identify concept drifts from the data perspective, validated using a real-world dataset.

2.3.8 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 (Omori et al., 2019), 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 (P. Ceravolo et al., 2020) 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 (P. Ceravolo et al., 2020) 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 (P. Ceravolo et al., 2020) 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 (G. M. T. S. B. J. . P. Ceravolo, 2019).

2.4 ONLINE PM DEALING WITH EVOLVING ENVIRONMENTS

PM techniques are usually classified into discovery, conformance checking, and enhancement (van der Aalst, 2016). 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.

2.4.1 STREAMING PROCESS DISCOVERY DEALING WITH EVOLVING ENVIRONMENTS

In (A Burattin et al., 2014), the authors propose different versions of the Heuristics Miner (HM) (Weijters et al., 2006) 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 (Manku & Motwani, 2002), 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 (DRS), 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 (Martino et al., 2013) 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 (Maggi et al., 2013), 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 (van der Aalst et al., 2009): 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 (A Burattin et al., 2015), 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 (van der Aalst et al., 2009). Another study (Andrea Burattin et al., 2015) 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 (van Zelst et al., 2018) 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 (van der Aalst et al., 2009): 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.

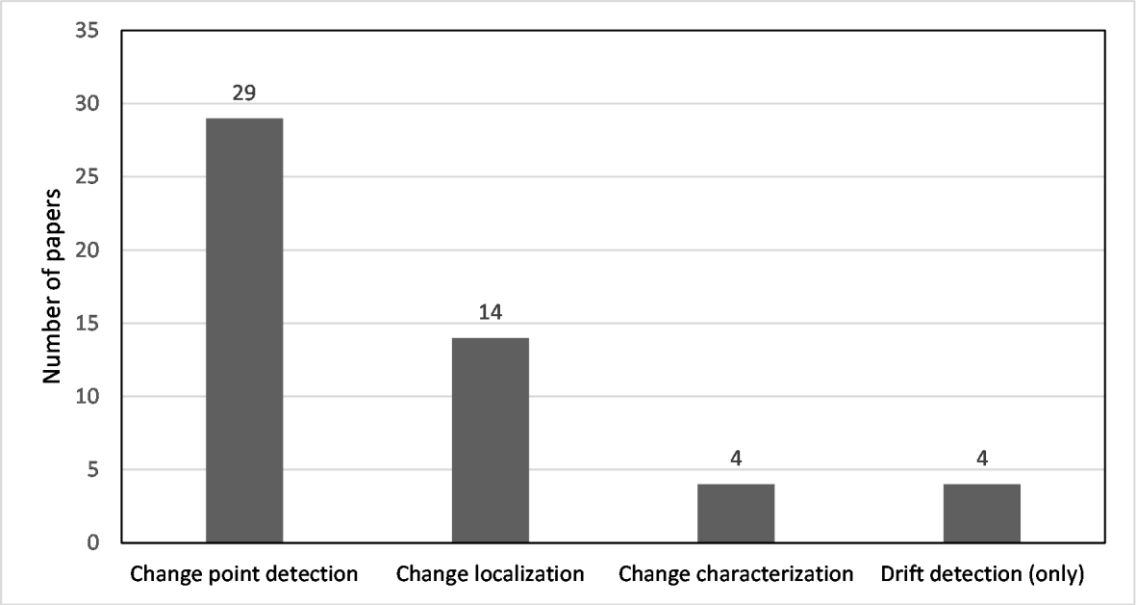
In (Anatoliy Batyuk & Voityshyn, 2020), the authors adapted the Fuzzy Miner algorithm (Günther & van der Aalst, 2007) to the streaming scenario. The implementation of the streaming Fuzzy Miner is integrated into the real-time business process monitoring (RTBPM) software (A Batyuk et al., 2018), following 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 (A Batyuk et al., 2018).

Another streaming process discovery algorithm is proposed in (Redlich et al., 2014). 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

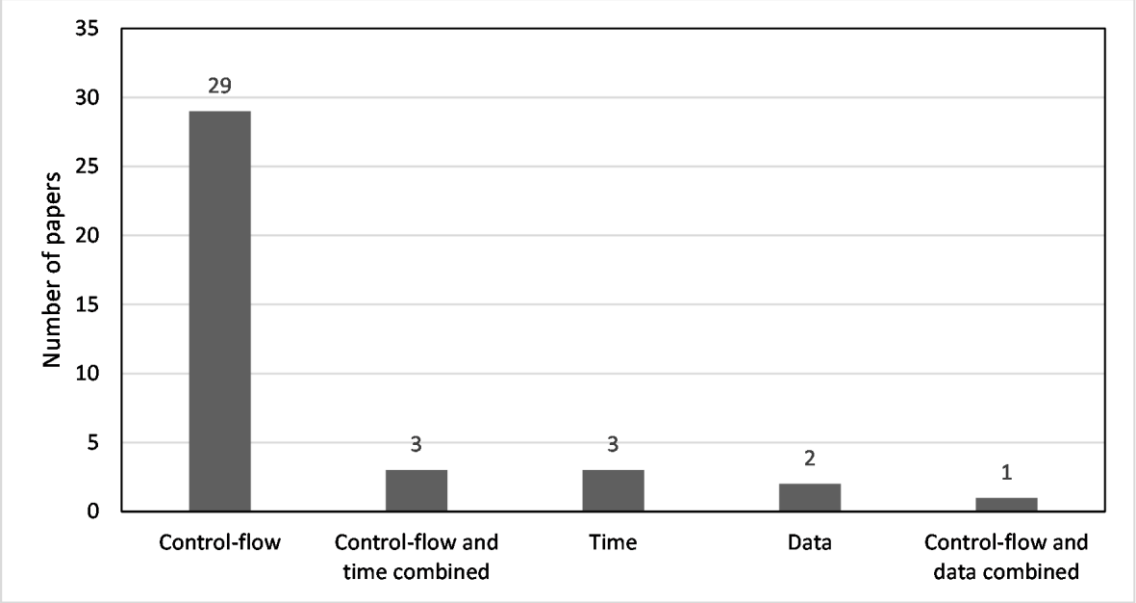
handles noise and not-supported behavior. The proposed Dynamic CCM (DCCM) (Redlich et al., 2014) splits up the original CCM into two parts: (i) footprint update and (ii) the footprint interpretation. For every event, the footprint update module updates the dynamic footprints. DCMM can then compute the 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 $tif \in R$ to the current dynamic overall footprint (DFP) multiplied by $1 - tif$. With this mechanism the influence of a trace completed 60 traces ago became almost irrelevant for $tif = 0.1$. 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 *tr*. The authors (Redlich et al., 2014) 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.

2.5 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 2.9 b) in the control-flow perspective (Figure 2.9 a). Papers addressing more than one perspective are counted for each perspective; the same applies to the challenges plot (Figure 2.9 b). Regarding the drift perspective, most approaches deal with drifts in the control-flow perspective. The data perspective is explored in (Hompes et al., 2017; Pauwels & Calders, 2019; Stertz & Rinderle-Ma, 2019), and the time perspective is explored in (Barbon Junior et al., 2018; Brockhoff et al., 2020; Mora et al., 2020; Richter & Seidl, 2019, 2017; Tavares et al., 2019).



(a) Counted by perspective.



(b) Counted by challenge.

Figure 2.9. 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 (Hassani, 2019; L. Lin et al., 2020; N. Liu et al., 2018; A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Martjushev et al., 2015; Ostovar et al., 2020, 2017,

2016; Seeliger et al., 2017; Yeshchenko et al., 2021; Yeshchenko, Di Ciccio, et al., 2019; Yeshchenko, Mendling, et al., 2020; Zellner et al., 2020) 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 (Zheng et al., 2017), similar to the objective evaluation using a lag period proposed in (L. Lin et al., 2020; Martjushev et al., 2015), 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 (La Rosa et al., 2011), and ProM (van der Aalst et al., 2009), 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 (P. Ceravolo et al., 2020; Omori et al., 2019) that compare tools for drift detection in PM. The authors in (Omori et al., 2019) 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 (P. Ceravolo et al., 2020), 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 2.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 (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Ostovar et al., 2016). 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 (Carmona & Gavaldà, 2012; Hassani, 2019) for adaptive windows use the ADWIN method (Bifet & Gavaldà, 2007), 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 (Yeshchenko et al., 2021; Yeshchenko, Ciccio, et al., 2019; Yeshchenko, Di Ciccio, et al., 2019; Yeshchenko, Mendling, et al., 2020).

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.

2.6 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 (Y. Lu, 2017). 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.

APPENDIX 2.A - SYNTHETIC DATASETS DESCRIPTION

(1) Business Process Drift – (Abderrahmane Maaradji et al., 2015) – 72 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 – (Ostovar et al., 2020) – 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 - (P. Ceravolo et al., 2020) - 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.

2.7 RECENT PAPERS

We identified two more papers dealing with concept drift detection in process mining: (Adams et al., 2021) and (De Sousa et al., 2021). The former presented a Framework for Explainable Concept Drift Detection in Process Mining and can be classified as CP detection because it applies the PELT (Killick et al., 2012) algorithm for the change point detection. The method splits the event log using a tumbling time window (defined in days), extracts some features from each time window, then apply the PELT algorithm for obtaining the change points. One advantage of this method is the definition of the generic framework, which allows the application of different functions to construct a real-valued representation of different perspectives of the process. The authors evaluate the drift detection using a synthetic and a real dataset against Apromore ProDrift (Abderrahmane Maaradji et al., 2015) and VDD (Yeshchenko, Di Ciccio, et al., 2019), even the drift detection is not the final goal of the proposed framework. However, there is no objective metric reported in the evaluation, and the parameters applied in the software are not explicitly defined. The detection accuracy was also not evaluated considering distinct window sizes, probably because the drift detection is one part of the framework which aims to explain a root cause for the concept drift.

In (De Sousa et al., 2021), the authors apply a trace clustering approach to different traces profiles obtained from windows of traces of fixed size (the traces are ordered by their timestamp), and combine drift detection and localization. The authors in (De Sousa et al., 2021) stated that the proposed approach is based on an online setting; however, we classify it as offline analysis because it handles a stream of traces. The trace clustering is applied on two trace profiles: trace – based on the occurrence of activities (Actv-Occ) or the frequency of activities (Actv-Frq), and control-flow – based on the occurrence of transitions between activities (Trns-Occ). The clustering applies centroid-based clustering algorithms, using the Euclidean distance between all centroids in a cluster. The concept drift detection verifies whether a significant variation occurs in the current value of a feature when comparing the clusters. The authors proposed the following cluster evolution features: intra-cluster distance, inter-cluster distance, silhouette coefficient, Davies-Bouldin index (DBi), MSE, and the distance between centroid positions average. The approach is evaluated using the synthetic dataset from (Abderrahmane Maaradji et al., 2015), considering the log sizes of 5,000, 7,500, and 10,000 traces, and compared with the approach (Apromore ProDrift) from the same paper (however it is not specified if the fixed or adaptive setting is selected). The approach does not outperform the Apromore ProDrift method in all cases; however, it was able to localize the drifts in most cases. The detection's accuracy is limited to trace vector representation, clustering algorithms, and parametrization.

Table 2.7. Classification for the recent papers with concept drift detection approaches for PM. 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
(Adams et al., 2021)	1,2	All – depend on the function, SD (time and control-flow), RD (time and resource)	Sudden	Offline	Explainable Concept Drift PM[23]	SD (PA), RD (PA)	-
(De Sousa et al., 2021)	1,2,3	Control-flow	Sudden	Offline	Experimental tool NA	SD (PA)	F-score

2.8 SECTION NOTES

This section is published in (Sato et al., 2022). Topic 2.7 describes recent papers related to the topic published after the SLR publication's date. Changes from the original paper:

- We correctly cite the paper that describes the ADWIN method by the end of the first paragraph of Section 2.3.5
- Appendix 2.A – the first log contains 72 event logs (not 75).

[23] Source code available at: https://github.com/niklasadams/explainable_concept_drift_pm.git

3 PAPER #2: INTERACTIVE PROCESS DRIFT DETECTION FRAMEWORK

This paper presents a novel tool for detecting drifts in process models. The tool targets the challenge of defining the better parameter configuration for detecting drifts by providing an interactive user interface. Using this interface, the user can quickly change the parameters and verify how the process evolved. The process evolution is presented in a timeline of process models, simulating a “replay” of models over time. One instantiation of the framework was implemented using a fixed-size sliding window, discovering process maps using directly-follows graphs (DFGs), and calculating nodes and edges similarities. This instantiation was evaluated using a benchmarking dataset of simple and complex drift patterns. The tool correctly detected 17 from the 18 change patterns, thus confirming its potential when an adequate window size is set. The user interface shows that replaying the process models provides a visual understanding of the changing process. The concept drift is explained by the similarity metrics’ differences, thus allowing drift localization.

Keywords: Process drift, Concept drift, Drift detection, Evolving environment.

3.1 PROCESS MINING IN EVOLVING ENVIRONMENTS

Process Mining (PM) is gathering more enthusiasts in recent years. The growing interest can be explained by the current availability of process data recorded by the informatics systems (big data) and the increasing development of tools to provide easy access to different PM techniques. The primary input of any PM technique is event data, which contains information about business process executions, and can be accessed in the form of event logs (historical information about the process) or event streams (continuous flow of events associated with processing instances). The event data must include at least an identifier of the process instance (case), the event (indicating the occurrence of activity), and the timestamp in which the event occurred.

There are three types of PM: discovery, conformance checking, and enhancement (van der Aalst, 2016). Discovery aims at learning what is happening with the processes by automatically discovering process models from the event logs without any *a priori* knowledge. Conformance checking compares a discovered or designed process model to the event log to pinpoint possible deviations. The goal is to help business analysts to understand problems or even unexpected behavior to support improvements. In the enhancement, an existent process model is extended (including new perspectives, e.g., performance) or improved using information from the event logs. Business analysts can then change the process model to reflect better reality based on the information provided by discovering or conformance. Discovery, conformance checking, and enhancement are

usually applied offline by analyzing an event log, but PM techniques can also perform online analysis.

Regardless of the type of analysis, the business processes are not static. Companies and their analysts are always trying to improve the processes to minimize costs or maximize customer satisfaction. The processes also change when there is a new regulation or in case of unexpected events. Recently, the whole world had to adapt roughly every process to address the COVID-19 pandemic situation. So, it is naïve to assume that models do not change when business processes are placed in evolving environments. Besides, most state-of-the-art PM techniques consider the processes to be in a “steady-state”, i.e., thus assuming that an event log contains information about a single version of the process. This assumption does not consider the existence of process drifts.

A process drift, or concept drift in processes, occurs when the business process changes while being analyzed (van der Aalst et al., 2011). The correct identification of the process drifts is critical when conducting process analysis in evolving environments. By detecting process drifts, the analysts should better understand the actual process model without facing a mixed model from different versions of the process. Process drift detection can improve reporting and diagnosis analysis, predictions, recommendations, and operational support by assuring that a more adherent version of the process model is considered. In online analysis, the process drift detection can help discovery or conformance checking techniques to maintain an up-to-dated process model, without the need to continually rediscover it.

Process drift was one of the challenges cited in the Process Mining Manifesto (van der Aalst et al., 2011). Since 2011, researchers have developed different process drift detection tools, such as ProM ConceptDrift plugin (Bose et al., 2014, 2011; Martjushev et al., 2015), Apromore ProDrift (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Ostovar et al., 2020, 2016, 2017), and other experimental tools (Barbon Junior et al., 2018; Mora et al., 2020; Richter et al., 2020; Seeliger et al., 2017; Yeshchenko, Di Ciccio, et al., 2019; Yeshchenko, Mendling, et al., 2020; Zellner et al., 2020; Zheng et al., 2017). However, specific issues make it difficult to compare the tools affecting the adoption in real scenarios.

We identified the following experimental tools for process drift detection: Process Drift Detector Plugin (PDD) for ProM (Seeliger et al., 2017), Tsinghua Process Concept Drift Detection (TPCDD) (Zheng et al., 2017), Concept-Drift in Event Stream Framework (CDESF) (Barbon Junior et al., 2018; Mora et al., 2020), Visual Drift Detection (VDD) (Yeshchenko, Di Ciccio, et al., 2019; Yeshchenko, Mendling, et al., 2020), Dynamic Outlier Aggregation (DOA) (Zellner et al., 2020), Online Trace Ordering for Structural Overviews (OTOSO) (Richter et

al., 2020). These tools share the drawback of ProM and Apromore, i.e., drift detection accuracy is highly affected by the hyperparameter configuration. Some of them are essentially affected by the window size: TPCDD, CDESF (the window size is defined by the time-horizon parameter, controlling model updates), and VDD. The approaches based on clustering, i.e., TPCDD, CDESF, VDD, DOA, and OTOSO, are also sensitive to the clustering hyperparameters. The PDD applies an adaptive window approach aiming to solve the window size choice, but the adaptation is limited to user-given upper and lower bounds.

This paper proposes the Interactive Process Drift Detection (IPDD) Framework to provide a practical tool for process drift detection that can be easily applied in real scenarios. Section 3.2 describes process drifts and explores the available tools. In Section 3.3, the new framework is described. Section 3.4 presents results obtained with an instantiation of the proposed framework. Finally, Section 3.5 concludes the paper and indicates next steps for IPDD enhancement.

3.2 PROCESS DRIFT

A process drift indicates a point in time where the process changes for a reason. The changes can be planned and documented or unexpected. A change in the process reflects a change in its process model, meaning a new one replaces the existing process model's version. The new process model can affect the ongoing process instances in different ways: sudden or gradual (Bose et al., 2014, 2011).

In a sudden drift, all the ongoing process instances start to emanate from the new process model when the drift occurs. It can occur when a new regulation should be followed or even within an epidemic situation. In a gradual drift, the new process starts emanating process instances, but both versions coexist for some time, indicating a gradual replacement of the model. The authors in (Bose et al., 2014, 2011) also describe two different dynamics for the changes: recurring and incremental. In the recurring drift, the current model is replaced by a new one, but the new model is replaced by the previous one after some time. The incremental drift represents minor incremental changes implemented during some time. The recurring drift can indicate seasonal changes, for instance. An incremental drift can occur in companies to minimize risks for significant changes. In each process model transition, we can have a sudden or a gradual drift.

Process drifts can occur at different time granularities. For instance, we can have a recurring drift that occurs every season, e.g., summer, winter, fall, and spring; and another drift occurring at the last week of the month, e.g., for specific accounting tasks. Both drifts coexist, yet they occur at different time granularities. In (Martjushev et al., 2015), the authors

named this situation as multi-order dynamics, reinforcing that any drift detection mechanism should deal with different time granularities when identifying process drifts.

A process drift can affect one or more perspectives of the model, which are partially overlapping. The most common perspectives described in (van der Aalst, 2016) are:

(1) Control-flow. It represents the process model's behavior based on the structure of activities (sequential, parallel, choice, loops).

(2) Organizational. Resources related to the process activities, which can be people, systems, departments, or others.

(3) Data. It is related to the information relevant to the process associated with the case, e.g., supplier, or to a specific activity, e.g., a machine.

(4) Time. Time and frequency of activities. Tools and methods form handling process drifts should consider the types and perspectives of change and can also address different problems (Bose et al., 2014, 2011):

(1) Change point detection. Identify the point in time (timestamp) that the drift occurred. The change point can be reported by the case index, the event index, or the date/time where the change starts. Change point detection is the most common problem addressed by the available tools.

(2) Change localization. Report the process model region which has changed, e.g., between activities A and B. This problem partially overlaps with “unravel process evolution”, however, ProM and Apromore addressed this challenge without providing the process evolution. This problem is more about local changes and not the global picture of the drift.

(3) Change characterization. Specify the type of change and in which perspective it occurred. This problem is less explored because few methods detect different process drifts or consider different perspectives.

(4) Unravel process evolution. Relate and explore the former discoveries, putting everything together to understand the process’ evolution over time. We did not identify any tool that explored this problem.

We focus on problems 1, 2, and 4. We propose IPDD, a framework for detecting change points and visually presenting its localization and process model evolution. IPDD deals with sudden drifts in the control-flow perspective. Incremental, recurring drifts, or multi-order dynamics are addressed by the user interface, allowing to check the detected drifts with different parameters.

3.2.1 PROCESS DRIFT DETECTION TOOLS

Process drift detection tools can be divided into two categories: academic and experimental tools. We did not find any commercial tool with a process drift detection mechanism. We only report the tools that provide the source code or an executable interface due to space limitations.

The first identified tool for process drift detection is the Concept Drift plugin in ProM[24]. Different approaches have been implemented in this plugin (Bose et al., 2014, 2011; Martjushev et al., 2015) to address change point detection for sudden and gradual drifts and multi-order dynamics. The user can also se.arch for change localization by selecting a pair of activities; in this case, the tool checks if there is a change between the selected activities. The user has to select many options for using the plugin: log configuration (join logs, split the log or not), feature to be compared (global or local), parameters of the feature, window strategy (fixed or adaptive, sliding over traces or time periods) with parameters, type of drift to detect (gradual or sudden), and the statistical test applied along its parameters. The user is required to set many configurations before even know if there is potential drift in the event log. Furthermore, the two types of drifts detected (sudden and gradual) must be separately checked. The plugin is designed for offline analysis, and it is not working if any global feature is selected, thus raising an exception. The results are highly sensitive to the parameters chosen, and there is no user-friendly interface to compare the results from different configurations. It is not possible to obtain the accuracy of the method, e.g., *F-score*, for synthetic logs in the plugin's interface.

Apromore[25] is another academic tool that provides a plugin for concept drift detection reporting change points, change characterization, and localization (ProDrift). ProDrift has different approaches implemented (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Ostovar et al., 2020, 2016, 2017) and can detect drifts from a stream of traces (based on runs, which is an abstraction for the traces) or event streams. The tool can detect sudden and gradual drifts from event streams in an integrated way, i.e., the user does not have to specify the type of drift to be detected. ProDrift also has different types of parameters: approach (runs or events), windowing strategy (fixed or adaptive), and window size. The event-based approach includes a noise filter threshold, drift detection sensitivity,

[24] ProM is an open source framework that provides a big set of tools for the discovery and analysis of process models from event logs: http://www.processmining.org.

[25] Apromore is a collaborative business process analytics platform with distinct editions. The ProDrift is an experimental plugin: https://apromore.org/platform/tools/.

characterization (on/off), characterization method (activity-based or fragment-based), and characterization noise filter threshold. The accuracy of the detection method is highly sensitive to the chosen parameter values. An advantage of Apromore is that it has default values for each parameter. Yet, a drawback is that it is not possible to quickly check the differences in the process model before and after a detected change point. The adaptive window approach uses an initial window size value as input, but the plugin applies an algorithm to initialize it if the user does not specify. However, this initial size definition is not explained in the papers supporting the implementation (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Ostovar et al., 2020, 2016, 2017).

The identified experimental tools for process drift detection are: Process Drift Detector Plugin for ProM (Seeliger et al., 2017), Tsinghua Process Concept Drift Detection (TPCDD) (Zheng et al., 2017), Concept-Drift in Event Stream Framework (CDESF) (Barbon Junior et al., 2018; Mora et al., 2020), Visual Drift Detection (VDD) (Yeshchenko, Bayomie, et al., 2020; Yeshchenko, Di Ciccio, et al., 2019), Dynamic Outlier Aggregation (DOA) (Zellner et al., 2020), Online Trace Ordering for Structural Overviews (OTOSO) (Richter et al., 2020). These tools share the drawback of ProM and Apromore, i.e., drift detection accuracy is highly affected by the hyperparameter configuration. Some of them are essentially affected by the window size: TPCDD, CDESF (the window size is defined by the time-horizon parameter, controlling model updates), and VDD. The approaches based on clustering, i.e., TPCDD, CDESF, VDD, DOA, and OTOSO, are also sensitive to the clustering hyperparameters. The Process Drift Detector Plugin for ProM applies an adaptive window approach aiming to solve the window size choice, but the adaptation is limited to user-given upper and lower bounds.

The remaining issue for the available tools is hyperparameter setup. 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 making it challenging to pinpoint the exact location of the drift (Abderrahmane Maaradji et al., 2015). Hyperparameter tuning is essential for providing a tool for detecting drifts in real-world scenarios. The current tools do not provide accuracy metrics neither reveal process evolution, turning the hyperparameter tuning an arduous task. By default, the tools focus on reporting the change points and leaving the user to understand the process model's change by splitting the log and applying discovery techniques in all the resulted sub-logs. Our proposal (IPDD) fills this gap by providing an interactive process drift detection tool, easy to use, and a reduced number of parameters. The interface allows the user to quickly verify the detection results by visualizing the process models over time.

3.3 INTERACTIVE PROCESS DRIFT DETECTION FRAMEWORK

The proposed framework (IPDD) aims at overcoming the reported issues of the available tools: a not so user-friendly interface, the difficulty in comparing results obtained by different parameter configurations (not allowing hyperparameter tuning), complex configuration, and not reporting the accuracy metrics in a common manner. Figure 3.1 an overview of the proposed framework.

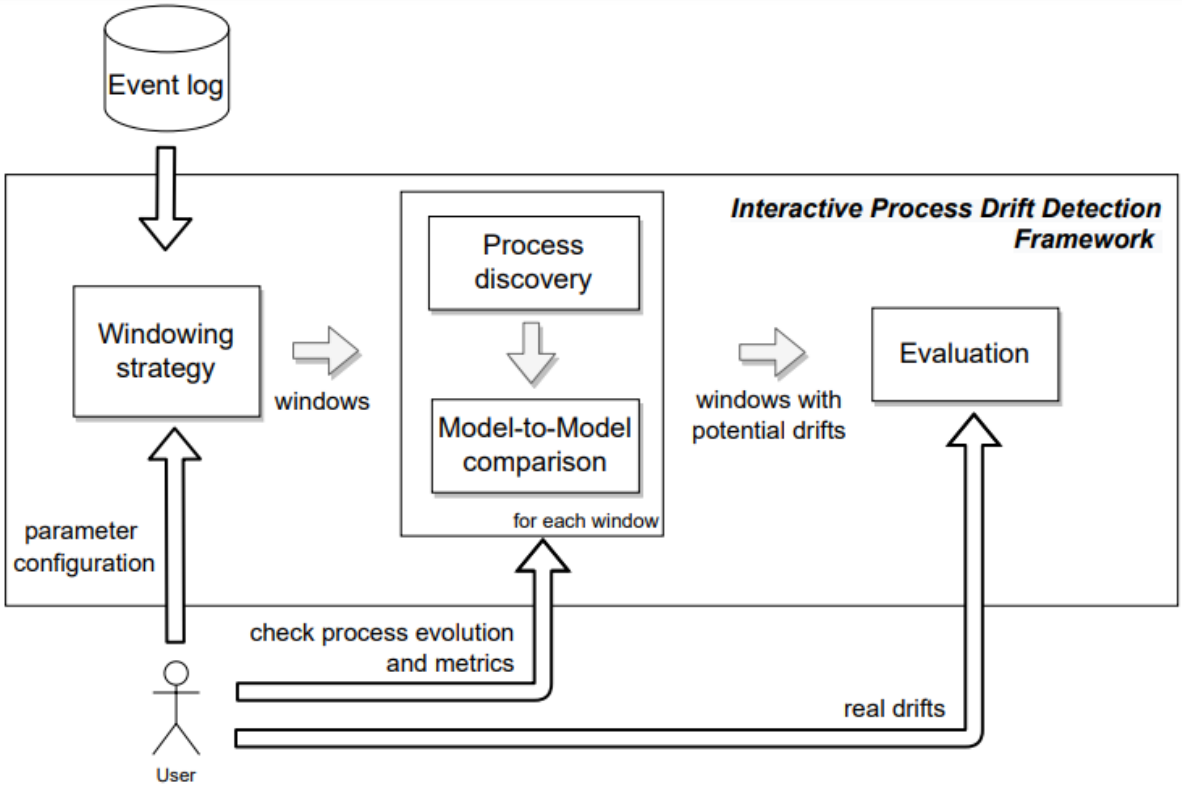


Figure 3.1. Overview of Interactive Process Drift Detection Framework.

The definition of the parameters' values is challenging when starting a process drift analysis because the user has to determine *a priori* what will be a "good" parameter for each situation. We propose to solve this issue by providing an interactive and easy user interface, leaving the user free to test different parameters quickly. The commercial tools for process discovery (e.g., Disco) inspired this format, as they usually present a simplified version of the process model in which the user can navigate by zooming in or out to understand the process. The goal is to allow the user to navigate between different granularities of change and inspect the process evolution. The hyperparameter tuning is allowed by the user interface, which provides the process evolution for each tested parameter configuration. The user can also verify the chosen parameters' accuracy by checking the evaluation module.

3.3.1 WINDOWING STRATEGY

An event log in XES format[26] can be uploaded into the IPDD. Next, by applying a windowing strategy, the events are split into separated windows. The window strategy can consider the event log as time series of traces, i.e., a stream of traces, by ordering the traces by the first event's timestamp. The windows can also be defined based on events, sorted by their timestamp, i.e., time series of events. The size of the window can be fixed or adaptive. Using fixed-size windows, the user must specify the size as the number of traces or event or as a time window, e.g., hours, days. The windows can be overlapping or non-overlapping, continuous, or non-continuous. All of the identified options can be implemented in the windowing strategy step of the framework. The window slots containing a set of traces/events are forward to the next step, named *process discovery*.

3.3.2 PROCESS DISCOVERY

For each window slot provided in the first step, IPDD applies a process discovery algorithm to derive a process model. Several algorithms have been proposed for process discovery, and any discovery technique has its own representational bias, i.e., the process model that can be discovered (van der Aalst, 2016). The resulted process models can be declarative or imperative, and several notations are available (van der Aalst, 2016). Another option is to use a DFG, also named process map, for simplicity and scalability. This option simplifies the mined process models but still shows relevant insights about the process paths with metrics.

The discovery algorithm's choice is related to the resulted process model, its ability to filter noise, its performance, etc. In the *process discovery* step, any implemented discovery algorithm can be called, like a black-box. The derived models are forward to the next step. An advantage of using the process model is that the framework can quickly show the process changes over time to the user.

3.3.3 MODEL-TO-MODEL COMPARISON

In the previous step, IPDD derives a process model for each window slot. Comparing models from adjacent windows allow identifying differences. Any difference can be a potential drift, and the type of change that can be detected is related to the metric chosen for this comparison. There are several metrics for comparing process models (Becker & Laue, 2012), and they are related to the notation of the process model mined. In this step, IPDD calculates the implemented similarity metrics between adjacent models.

---

[26] See www.xes-standard.org for detailed information about the standard.

The metric should be adherent to the process model derived by the chosen discovery algorithm. There is no limitation about the number of metrics that can be included, and the user can select one or more of the available metrics. IPDD implements a timeout mechanism for avoiding freezing the user interface when calculating the metrics. If the timeout is reached, only the finished metric results are shown. The result of this stage is reporting each metric's value. If a metric has a value bounded in the [0,1] interval, with one indicating that the two models are similar, a potential drift can be considered when this metric value is below one. Any metric with a value indicating dissimilarity triggers an update in the user interface, marking the window as a potential drift.

We choose to use a model-to-model comparison instead of trace comparison scheme, e.g., applying statistical or clustering approaches, because we can identify drifts that affect the process model. The chosen process model notation can provide the detection of different drifts. For instance, if we choose to generate a model with the frequencies annotated in the path between activities, IPDD can implement a metric for comparing these frequencies. In this example, IPDD can handle control-flow drifts that do not affect the structure of the process model but affects the routing of cases. It is also possible to localize the drift in the process model by checking the similarity metrics' differences.

3.3.4 EVALUATION

IPDD receives the real drift position as trace/event indexes or date/time values. It can then calculate an accuracy metric to measure if the windows reported as potential drifts include the real drifts. We identified two metrics for measuring the accuracy of concept drift in process models: *F-score* and *mean delay*, reported in (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Martjushev et al., 2015; Ostovar et al., 2020, 2017, 2016; Seeliger et al., 2017; Zheng et al., 2017). IPDD reports a drift by indicating the window that renders a model dissimilar to the one obtained in the previous window, so the *F-score* can be applied to evaluate the accuracy of the detected drifts.

The *F-score* represents the harmonic mean of recall and precision, calculated based on the true positives (TP), false positives (FP), and false negatives (FN). It is critical to define that a TP should consider an interval of indexes because the detection mechanism cannot detect the drift by the time it has occurred. In other words, if the real drift initiates in the *i*-th trace, a TP occurs when the detection method reports a drift in the interval $[i, i + et]$, where *et* indicates an error tolerance and should be configured. The FPs and FNs should also be consistent with the definition of the TP. The *mean delay* represents the distance between the occurrence of the real drift and the drift flagged. This distance relates to the windowing

strategy adopted. For instance, if the instantiation used a window over traces, this distance is the difference between the trace index of the real drift and the detected one.

3.4 RESULTS

We implemented a prototype to validate the IPDD and its user interface[27]. This prototype is an instantiation of the framework that encompasses the following:

(1) Windowing strategy. Non-overlapping and continuous windows of traces (ordered as time series, based on the first activity's timestamp). The windows have a fixed size of traces defined by the user.

(2) Process discovery. We applied the PM4Py[28] framework to discover the DFG, with the frequencies of activities and paths.

(3) Model-to-model comparison. We calculated the node similarity (NS) and edge similarity (ES) scores between two consecutive process maps *P* and *Q*. NS is calculated using Equation (3.1) (Akkiraju & Ivan, 2010), where n_p and n_q are the number of activities in process maps *P* and *Q*, respectively, and n_{cs} indicates the number of common activities between *P* and *Q*. ES is calculated using Equation (3.2), which is similar to NS, however using: e_p is the number of edges *P*, e_q is the number of edges in *Q*, and e_{cs} indicates the number of common edges in both *P* and *Q*.

$$NS = 2 * n_{cs}/(n_p + n_q) \tag{3.1}$$

$$ES = 2 * e_{cs}/(e_p + e_q) \tag{3.2}$$

The prototype calculates both metrics, and if one or both is less than zero, IPDD marks the current window as a drift.

(4) Evaluation. There is no evaluation metric already implemented, but we have the *F-score* metric defined to measure the detected drifts' accuracy. Because of the window strategy choice, a TP represents a window reported as a drift containing a trace inputted as a real drift. An FP should be counted when a window reporting a drift does not contain any of the traces inputted as real drifts. Finally, an FN should be incremented when a window that does not report a drift contains any traces inputted as real drifts.

---

[27] Available at https://github.com/denisesato/InteractiveProcessDriftDetectionFW.

[28] PM4Py is a python open source PM platform: https://pm4py.fit.fraunhofer.de/.

Figure 3.2 shows a snapshot of the prototype. After setting the window size, the prototype mine the models and calculate the similarity metrics between adjacent models (NS and ES). The window with a similarity metric value below one is marked in red, indicating a drift. The user can check the process mined for this window and verify the metric value and the differences. In Figure 3.2 (displayed in the lower-left corner), the edges from activity “Assess eligibility” to “Send acceptance pack” and to “Send home insurance” are included, indicating what has changed in the model. In this example, the activities “Prepare acceptance pack” and “Check if home insurance quote is requested” are optional after the drift; the new edges allow skipping both activities after performing “Assess eligibility”.

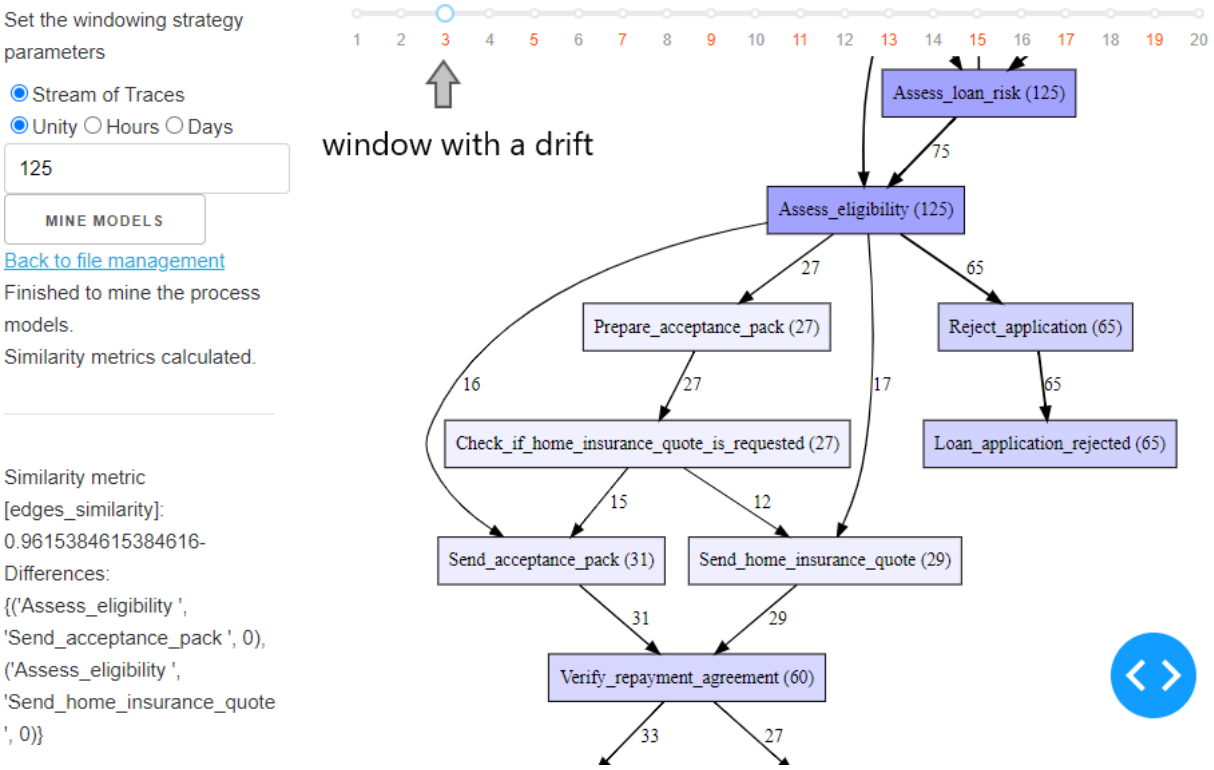


Figure 3.2. Snapshot of the prototype implementation.

To validate the tool’s usage, we apply the drift detection in some of the logs publicized in (Abderrahmane Maaradji et al., 2015). The dataset contains 72 logs with different change patterns and inter-drift distances (distance between each injected drift). The authors use a business process for assessing loan applications containing 15 activities and different control-flow structures as the base model. Next, they injected different types of control-flow changes, simulating sudden recurring drifts (9 drifts in each log). The base model was changed using 12 simple change patterns - described in (P. Weber et al., 2012) and 6 complex patterns (the composition of 3 simple patterns). Each change pattern is injected using 4 inter-drift distances (250, 500, 750, and 1,000). We select all change patterns with the inter-distance of 500 to validate IPDD (Table 3.1). For the change patterns *lp* and *re*, we used the files named 2.5k as these files contain inter-drift distance of 500. Each pattern is

categorized as Insertion (I), Resequentialization (R), and Optionalization (O). For the complex pattern, the authors randomly apply one pattern from each category in a nested way.

Table 3.1. Experiments description.

Change pattern	Category	Drifts detected?	Metric that detects the drifts
cb: make fragment skippable/non-skippable	O	Yes	ES
cd: synchronize two fragments	R	Yes	ES
cf: make two fragments conditional/sequential	R	Yes	ES
cm: move fragment into/out of the conditional branch	I	Yes	ES
cp: duplicate fragment	I	Yes	ES
fr: change branching frequency	O	No	-
lp: make two fragments loopable/not-loopable	O	Yes	ES
pl: make two fragments parallel/sequential	R	Yes	ES
pm: move fragment into/out of parallel branch	I	Yes	ES
re: add/remove fragment	I	Yes	NS and ES
rp: substitute fragment	I	Yes	NS
sw: swap two fragments	I	Yes	ES
IOR	IOR	Yes	NS and ES
IRO	IRO	Yes	NS and ES
OIR	OIR	Yes	NS and ES
ORI	ORI	Yes	NS and ES
RIO	RIO	Yes	ES
ROI	ROI	Yes	ES

To validate the tool's usage, we apply the drift detection in some of the logs publicized in (Abderrahmane Maaradji et al., 2015). The dataset contains 72 logs with different change patterns and inter-drift distances (distance between each injected drift). The authors use a business process for assessing loan applications containing 15 activities and different control-flow structures as the base model. Next, they injected different types of control-flow changes, simulating sudden recurring drifts (9 drifts in each log). The base model was changed using 12 simple change patterns - described in (P. Weber et al., 2012) and 6 complex patterns (the composition of 3 simple patterns). Each change pattern is injected using 4 inter-drift distances (250, 500, 750, and 1,000). We select all change patterns with the inter-distance of 500 to validate IPDD (Table 3.1). For the change patterns *lp* and *re*, we used the files named 2.5k as these files contain inter-drift distance of 500. Each pattern is categorized as Insertion (I), Resequentialization (R), and Optionalization (O). For the complex pattern, the authors randomly apply one pattern from each category in a nested way.

Table 3.1 shows that the implemented instantiation for IPDD correctly detects drifts for 17 out of the 18 patterns using NS and ES. The detection is possible when configuring a window size equal to the inter-drift distance (500). The *fr* pattern is not detectable because there is

no structural difference between the models; the drift only changes the frequencies in one branch of the process. The choice of the window size is still an issue. If the window size is configured with a value higher than the inter-drift distance, the drift will not be detected.

3.5 CONCLUSION

IPDD has been validated by a prototype implementation that demonstrates its use for detecting sudden drifts in the control-flow perspective. The main contribution of the novel approach is the interactive user interface, which provides the user with a tool for quickly checking different values for window sizes. The drift can be visually checked by analyzing the process models from adjacent windows and the metrics' value describing the detected differences. The implemented metrics (NS and ES) can detect 17 (from 18) change patterns in the public dataset. The window size choice is still a challenge, but the interactive user interface provides an easy way of testing different values. We plan to extend IPDD by including new instantiations implementing the defined evaluation metric (*F-score*) and a similarity metric related to the frequencies between activities.

3.6 ADITIONAL INFORMATION ABOUT THE EVALUATION

We evaluated the IPDD implementation using only part (event logs with 5,000 traces) of the complete dataset defined in (Abderrahmane Maaradji et al., 2015). The reason for not evaluating IPDD with other log sizes (2,500; 7,500; and 10,000) is that some event logs do not follow the specification described in the paper concerning log size, the position of the drift, and the adopted change pattern. Appendix 5.A describes the issues of this dataset in more detail.

The fixed tumbling window strategy identifies the drifts by setting the window size equal to the interval between drifts, e.g., 500 for the selected event logs. Other window sizes can also correctly detect the drifts, e.g., 50, 100, 125, 250; values less than 500 and result in rest zero when 500 is divided by the value. This behavior occurs because the windows do not overlap; thus, the chosen window size directly impacts the detection accuracy. However, the current implementation's main contribution is providing an interface allowing the user to test and adjust the window size value interactively, verifying the detected changes in the model and in the similarity metric information.

Another consideration is the change pattern that IPDD was unable to detect: the change branching frequency pattern (fr). This change pattern does not change the structure of the process model, only the routing of cases, i.e., behavioral drift. The current IPDD implementation cannot identify behavioral control-flow drifts because the similarity metrics implemented in the Model-to-Model comparison do not consider the frequency information in

the nodes or edges, which is a limitation of the current implementation. To overcome this limitation, similarity metrics that consider the frequency of nodes and edges may be included.

3.7 SECTION NOTES

This section (from 3.1 to 3.5) is published in (Sato, Barddal, et al., 2021). Changes from the original paper:

- We correctly cite the paper that describes the Process Drift Detector Plugin (PDD) for the ProM in Section 3.1.

Section 3.6 describes the evaluation process in more detail, including additional information about the scenarios and limitations. This section is not included in the original paper due to space limitations.

4 PAPER #3: INTERACTIVE PROCESS DRIFT DETECTION: A FRAMEWORK FOR VISUAL ANALYSIS OF PROCESS DRIFTS (EXTENDED ABSTRACT)

Interactive Process Drift Detection (IPDD) is a framework for visual analysis of process drifts. A process drift indicates a change in the process model occurred at some point in time. IPDD firstly generates process models for subparts of the event log using a sliding window approach. Then, it detects the drifts by evaluating similarity metrics calculated between adjacent process models; a difference in some of the metrics indicates a drift. The current implementation of IPDD generates the process models using the directly-follows graph (DFG) and applies two metrics: nodes and edges similarity. The user interface shows the drifts in the process models over time, allowing the user to visually understand the model changes. Also, the user can easily change the hyperparameters for the analysis and verify the results on the interface. The user interface of IPDD allows the user to evaluate the detected drifts by calculating the F-score metric, which is useful when using artificial datasets. The underlying idea is to ease the choice of a “good” value for the hyperparameter configuration, which is critical for almost any drift detection tool.

Keywords: process drift detection, visual process analysis, process drift, concept drift.

4.1 INTRODUCTION

Process mining aims at creating valuable knowledge about business processes obtained from information systems event data. Usually, process mining techniques assume the processes to be steady-state, i.e., the event data contains information from a unique version of the process. However, this assumption does not reflect the reality of the business processes, which constantly adapt to new regulations, improve performance, or enhance user experience. The situation where a process changes while being analyzed is named concept drift or process drift (van der Aalst et al., 2011).

The change in the process can affect the ongoing instances, sudden or gradually. A sudden drift occurs when all the ongoing instances start to follow the new process model immediately. In a gradual drift, there is a period of time where instances from both versions of the process model coexist. The process drifts can also follow recurrent or incremental patterns. A recurrent drift indicates that a replaced process model can occur again. In an incremental drift, minor changes of the process model are implemented during some time. Sudden, gradual, incremental, and recurring are considered process drift types (Bose et al., 2014). The process drift can also affect one or more perspectives of the process model. However, the most common perspective considered in the available tools is the control flow. Identifying and understanding the process drifts is relevant for business analysts because it improves their knowledge about the processes and enhances the quality of process mining

analysis. Even when analysts perform offline process mining analysis, process drift detection can provide benefits, e.g., avoid complex discovered process models, improve conformance checking, or enhance processes based on their current state.

Different tools for detecting process drifts from event logs have been proposed, but the accuracy of the detection is usually related to the hyperparameter configuration (Sato et al., 2022). The ProDrift plugin in Apromore (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015) and the ConceptDrift plugin in ProM (Bose et al., 2014) can detect different types of drifts (sudden and gradual); however, the focus is the change point and information about it. The user has to complement the drift analysis by executing a more exploratory mining slicing the event log based on the reported change points to understand the evolution of the process. A more recent tool, named VDD (Yeshchenko et al., 2021), detects the four types of drifts and allows the user to explore the drift using the process model. However, the tool is based on constraints mined over Declare models, and it mixes DFGs with the constraints to explain the dynamic of the process over time. None of the identified tools calculate an accuracy metric in the user interface.

Tunning the hyperparameter configuration to enhance the detection accuracy imposes a challenge to the proposed tools because the different approaches are affected by the hyperparameter configuration. IPDD aims to overcome this issue by providing an interactive user interface where the user quickly changes the parameter and visually evaluates the results. The tool provides visual process drift detection analysis by showing the distinct process models over time, in what we can consider a “replay” of the process models. IPDD also provides information about the differences against the previous model for each process model, enhancing the analysis. IPDD’s current implementation detects sudden drifts in the control-flow perspective offline, which is a limitation.

4.2 IPDD MAIN FEATURES

The IPDD framework detects the process drifts by analyzing the event log using a sliding window strategy. First, the user defines the window size based on the number of traces, and IPDD splits the log using tumbling windows. Then, it generates a model for each window and calculates the similarity metrics between adjacent models. The idea is to compare models mined from adjacent time slots using similarity metrics; when they are not similar, IPDD identifies a drift and characterizes the change based on the information provided by the metric.

The IPDD’s current implementation mines the DFGs (process maps) from the traces in the time slots using the PM4Py (Berti et al., 2019). Then, the adjacent derived graphs are compared using the Nodes (NS) and Edges similarity (ES) metrics. NS is calculated using

Equation (4.1), where n_p and n_q are the number of activities in the process maps *P* and *Q* (derived from adjacent windows) respectively, and n_{cs} indicates the number of common activities between *P* and *Q*. ES is calculated using Equation (4.2), similar to NS: e_p is the number of edges in *P*, e_q is the number of edges in *Q*, and e_{cs} indicates the number of common edges in both *P* and *Q*.

$$NS = 2 * n_{cs}/(n_p + n_q) \tag{4.1}$$

$$ES = 2 * e_{cs}/(e_p + e_q) \tag{4.2}$$

IPDD calculates both metrics, and if one or both is less than 0, it marks the window as a drift. The F-score metric uses the True Positives (TP), False Positives (FP), and FN (False Negatives). A TP indicates a window reported as a drift containing a trace inputted as a real drift; an FP is counted when a window reporting a drift does not contain any trace informed as real drifts, and an FN is incremented when a window that does not report a drift contains any traces inputted as actual drifts.

Figure 4.1 shows the tool's main screen, allowing users to easily change parameters and visually check the results. The parameter configuration panel is on top, where users must define the hyperparameter configuration before starting the analysis. After clicking on "Analyze Process Drifts", users can follow the current status in the "Status" area below the parameters panel. When the analysis finishes, IPDD shows the process drift analysis panel. There is a timeline of windows in the upper part of this panel, where users can click to inspect specific windows of the process model. The similarity metrics information (on the left side) is updated for each window selected, providing information about the differences between the current and the previous model. In the example, the ES indicates a drift that is characterized by two edges added. After IPDD finishes the analysis, the user can show the evaluation panel to calculate the F-score metric by clicking "Evaluate results". IPDD framework is described in more detail in (Sato, Barddal, et al., 2021). Its source code is available in a public repository[29], the deployed application is available in a public node[30], and a demo video is available on YouTube[31].

[29] https://github.com/denisesato/InteractiveProcessDriftDetectionFW

[30] https://visual-pro-drift.com.br/

[31] Demonstration video at: https://youtu.be/8feKd6jr8Gs

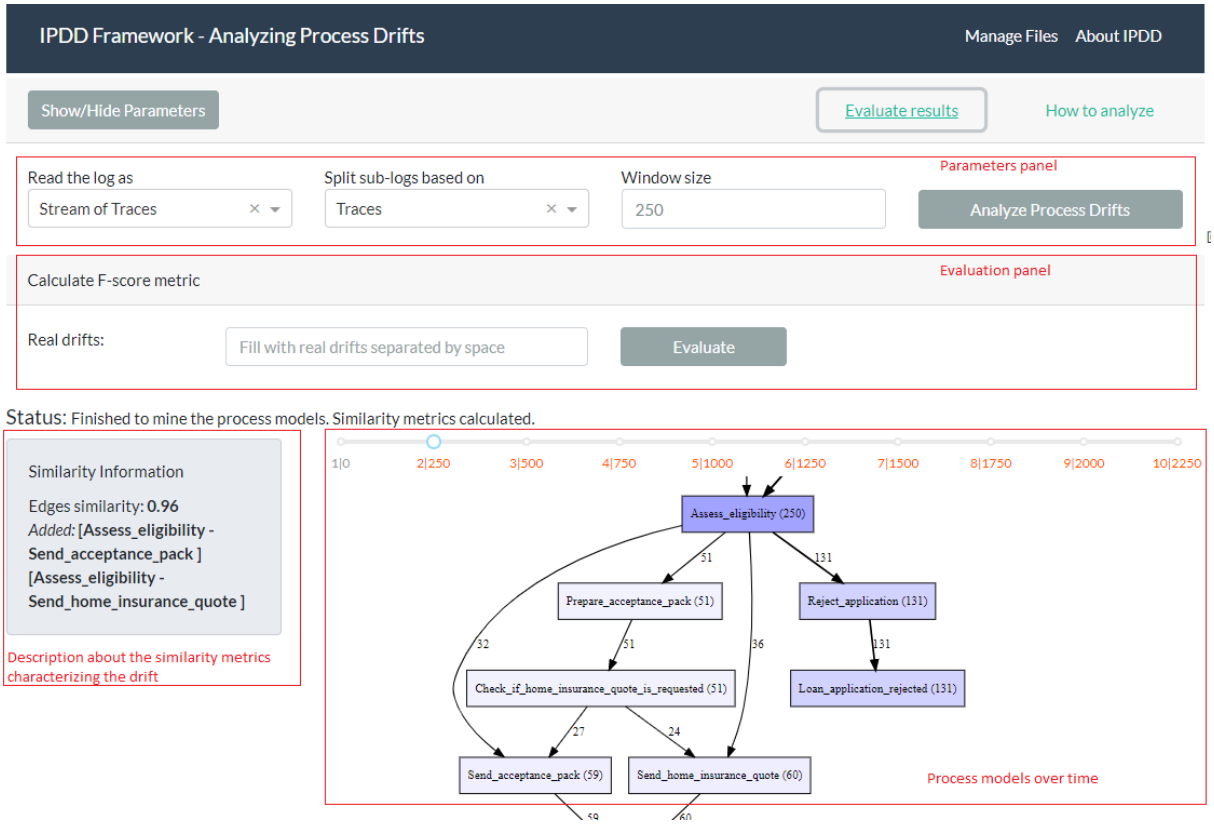


Figure 4.1. Screenshot from the main window.

4.3 CASE STUDIES

Authors have proposed different tools for process drift detection. However, the methods are usually sensitive to the hyperparameter configuration. Moreover, almost all approaches apply windowing strategies – and defining a “good” value for the window size is still a challenge. Also, the adaptive approaches have some drawbacks; other parameters affect the detected drifts (Sato et al., 2022). Our IPDD approach gives users the freedom to check different hyperparameter configurations to overcome this challenge visually.

The tool was presented to our research group in Curitiba (Brazil), including researchers from three post-graduate programs (Informatics, Production and Systems Engineering, and Health Technology). Firstly we have conducted a usability assessment for redesigning the user interface. Currently, we are working on a case study on a manufacturing scenario. The idea is to detect drifts in the temporal perspective of the process (sojourn time). The information about drifts will be used as input for planning the maintenance intervals on the production line.

4.4 USER INTERFACE

The tool described in this Section is an extension of the first implementation of the IPDD Framework described in Section 3. However, after the initial implementation, we evaluated

the usability of the IPPD user interface by presenting the tool to a group of process mining experts (researchers that work with process mining tools) from PUCPR (Brazil). While preparing the tool for the presentation and based on the collected answers, we have enhanced the tool to provide a more straightforward and intuitive user interface. The user interface presented in this extended abstract results from this process.

4.5 SECTION NOTES

This section (from 4.1 to 4.3) is published in (Sato, Fontana, et al., 2021). The current implementation of IPDD extends the options described in this extended abstract and is available at the address: https://visual-pro-drift.com.br/. Section 5 describes the new approaches included in the tool.

5 PAPER #4: SUDDEN PROCESS DRIFT DETECTION IN PROCESS MODELS: AN ADAPTIVE APPROACH

Process mining provides methods for obtaining valuable information about business processes from the actual data collected from the information systems. However, most process mining techniques still assume that the process does not change while being analyzed, which does not reflect the reality in many situations. Detecting and understanding the changes, i.e., process drifts, allows a more reliable process analysis by understanding when and what has changed. An open challenge for process drift detection is that the detection accuracy is very sensitive to the parameter configuration, and there is no standard protocol for comparing the results for different tools. In this paper, we propose the *Adaptive Interactive Process Drift Detection (IPDD)*, which applies the ADWIN change detector to quality metrics of the process model over time. The core idea is to collect a minimum number of traces, discover an initial process model, and continuously evaluate two quality metrics (fitness and precision) using the following traces. If a process drift occurs, the values of the metrics will change. The fitness metric decreases when the new traces raise a behavior not allowed by the current process model. The precision metric measures how much behavior is allowed by the process model and not observed in the event log; thus, if the precision decreases, some behavior allowed by the current model is not observed in the traces anymore. The method inputs the metrics in the ADWIN change detector, which detects a change in the values providing rigorous guarantees of performance as bounds on false positives and false negatives rates. Therefore, a process drift is only detected if the increasing or decreasing of the metric value is significant, thus avoiding false alarms in case of noise. We described and evaluated two approaches, *Adaptive IPDD trace by trace*, and *Adaptive IPDD windowing*. We applied the same experimental protocol in two synthetic datasets and compared it with other tools (F-score and the mean delay metrics). *Adaptive IPDD windowing* is statistically equivalent to Apromore for drift detection but overcomes Apromore considering the mean delay. *Adaptive IPDD trace by trace* is statistically equivalent to Apromore for drift detection only in dataset 2 and overcomes Apromore AWIN's mean delay for dataset 1 and Apromore FWIN's mean delay for dataset 2. The Visual Drift Detection (VDD) System performed the lowest F-score and mean delay values for both datasets. We have also evaluated our approach using a real-world dataset, and the results show that *Adaptive IPDD trace by trace* can detect and localize process drifts and show the process versions over time with a fair value for window size.

Keywords: process drift, process drift detection, ADWIN.

5.1 INTRODUCTION

Process mining aims to obtain valuable knowledge from business processes based on actual data collected from the information systems. The event data must contain at least the activity performed (step in the process), the case identifier (identify the process instance), and the timestamp, usually stored as an event log. Each process execution for a specific case generates a trace in the event log, representing the sequence of activities performed for the case in the analyzed business process. The most common tasks in process mining are discovery, conformance, and enhancement. Using a discovery algorithm, we can obtain a process model from the event log without any a priori information. In conformance checking, we compare the event log with a process model (designed or discovered) to understand deviations in the behavior of the process executions. Moreover, it is possible to extend the discovered model by applying enhancement by including additional perspectives from the event log, e.g., resources and time (van der Aalst, 2016).

Most process mining techniques are designed to deal with event logs and assume that the recorded events are related to the same process version. However, business processes are constantly changing due to new regulations, improving customer experience, and dealing with urgent or other situations. Therefore, detecting these changes in the behavior of the process may raise relevant information for the business analysts, e.g., when the process changed, what are the differences in the distinct versions of the process, and what is the current version of the process. The situation where the process changes while being analyzed is called concept drift or process drift (van der Aalst et al., 2011). In (Bose et al., 2014), the authors define three challenges for dealing with process drifts:

- change point detection – define the point in time where the change occurred;
- change localization and characterization – characterize the change (type and perspective) and localize the change in the model, and;
- change process discovery – discover the change process by putting the previously discovered information into perspective.

Usually, a process describes the control-flow perspective, which indicates the allowed sequence of activities for achieving the goal of the process. However, if available in the event log, more perspectives can be included, such as time, resources, and data. The process drift may occur in all the available perspectives. A process drift in the control-flow perspective might represent a structural change in the process, e.g., adding or removing an activity, changing two activities from a sequential structure to a parallel structure; and also may represent a change in the behavior of the process, i.e., in the routing of cases (Bose et al.,

2014). However, for our approach, we only considered process drifts in the control-flow perspective, the ones that affect the structure of the process.

The process drifts can occur sudden, gradual, recurrent, and incremental, based on the pattern of change between the different versions of the process (Bose et al., 2014), as illustrated by Figure 5.1.

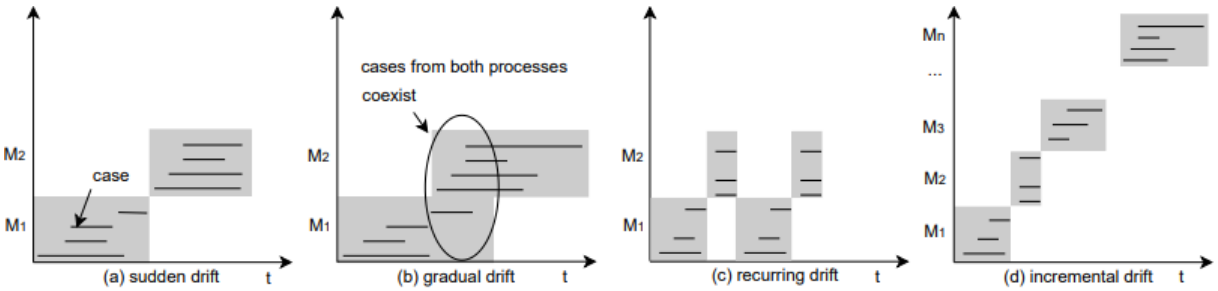


Figure 5.1. The four types of process drifts. Adapted from (Bose et al., 2014).

The process drifts may be analyzed offline to apply the results for understanding, designing, or improving processes. However, an online analysis is more indicated when the changes need to be discovered in near real-time.

In this paper, we presented a new process drift detection tool for offline analysis that addresses the three challenges previously described for sudden drifts in the control-flow perspective of the processes. The tool extends the Interactive Process Drift Detection (IPDD) framework using an adaptive windowing approach named *Adaptive IPDD*.

5.2 RELATED WORK

(Sato et al., 2022) presented a survey describing the current approaches for dealing with process drifts, highlighting that most approaches focus on offline process drift detection, usually detecting the changing points for sudden drifts in the control-flow perspective. Despite being the most addressed topic, sudden drift detection still faces challenges, such as the lack of a standard protocol for comparing results from different tools and the impact of the parameter configuration on the detection accuracy. As described in the introduction, the proposed method deals with offline process drift detection (event log as the input) from the control-flow perspective. Therefore, we only considered the recent tools with the same goal, which provided the source code or an executable version.

The Apromore ProDrift plugin (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Ostovar et al., 2020, 2016, 2017) implements a statistical grounded approach for detecting process drifts in event logs (runs approach) and streams (events approach). For our comparison, we only considered the approach for event logs, named *runs* approach, which is designed for offline analysis because it handles a stream of traces (Abderrahmane

Maaradji et al., 2015). The approach applies statistical hypothesis testing over the distributions of *runs* or *alpha relations* (events approach) observed in adjacent time windows. If a change occurs at a given time point, the distribution of *runs* or *alpha relations* before and after this time point will be statistically different when the time window is sufficiently large for statistical testing. The plugin also provides an adaptive approach for addressing the challenge of defining the window size.

The Apromore ProDrift handles the change point detection (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Ostovar et al., 2016) and provides two methods for change characterization and localization (Ostovar et al., 2020, 2017) that explain the drift using natural language statements (available when applying the events approach). The main challenge in ProDrift is to define a suitable window size for the *runs* approach because even the adaptive window is affected by the initial window size. The method needs an initial window size (defined by the user), and despite the authors stating that this parameter can be empirically set, this is not well explored in the paper. Furthermore, the user interface does not allow visualizing the different version of the process model and do not provide the evaluation metrics (f-score and mean delay). However, this tool handles gradual drifts, which is not included in our tool.

Another tool for process drift detection is the Visual Drift Detection (VDD) System (Yeshchenko et al., 2021; Yeshchenko, Ciccio, et al., 2019; Yeshchenko, Di Ciccio, et al., 2019; Yeshchenko, Mendling, et al., 2020), which provides a user interface for visual analysis of the changes. VDD applies a change point detection algorithm (PELT) in a multivariate time series containing pre-defined measures calculated over Declare constraints (van der Aalst et al., 2009), derived from the event log. Firstly, the method mines the log obtaining the complete Declare constraints alphabet ($\#cns$ is the total of constraints). Then, the event log is subdivided into temporal windows of fixed size, with $\#win$ indicating the total of windows. For each window, VDD calculates the confidence (or another measure) from the traces considering the Declare constraints alphabet, generating a set of time series $T_i = Conf_{i,1}, \dots, Conf_{i,\#win}$, where T_i represents the confidence value for the constraint i over time. Then, the method groups the derived time series into the multivariate time series $D = \{T_1, T_2, \dots T_{\#cns}\}$, where D represents the full spectrum of constraints' confidence. It is also possible to combine a clustering strategy (hierarchical clustering) for splitting D into groups of constraints with similar confidence trends before applying PELT. The resulting clusters indicate similar behavior and allow VDD to identify local behavior changes within the clusters. VDD can detect sudden, gradual, recurring, or incremental drifts and provide plots for visual analysis of the drift characterization and localization. Another advantage is that VDD also handles the change process discovery, using a process map (graph) of the complete event

log annotated with the constraints that vary over time (Yeshchenko et al., 2021; Yeshchenko, Mendling, et al., 2020). However, the provided mixed visualization (process map with constraints) is not that simple, and the user may identify the change point by inspecting other visualizations, e.g., by inspecting each cluster's Drift Chart.

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 change points are sensitive to the parameter configuration because the window size determines the granularity of analysis. The method suggests values for the window size and step (Yeshchenko et al., 2021; Yeshchenko, Di Ciccio, et al., 2019). However, this suggestion is based on a good visualization of the plots, not on tuning the tool's accuracy. Furthermore, the tool does not provide the evaluation metrics, e.g., f-score and mean delay on the user interface.

Because of the described issues in the recent approaches, we propose a new sudden drift detection tool for the control-flow perspective based on evaluating the fitness and precision metrics over time using the ADWIN change detector (Bifet & Gavaldà, 2007).

5.3 SUDDEN PROCESS DRIFT DETECTION

A discovered process model represents the behavior of the process obtained from the event data recorded by the information systems. The quality of the process model, i.e., how well the model describes the behavior of the event log, can be described by four dimensions: fitness, precision, generalization, and simplicity (van der Aalst, 2016). The fitness dimension measures how much of the behavior in the event log is allowed by the model. Fitness is usually the first dimension of analysis because it does not make sense to evaluate other quality dimensions for a model that does not describe the behavior observed in the event log; however, we can evaluate other aspects of a fitting model. Because the event log represents a snapshot of the complete process, the generalization dimension indicates that the model should generalize the example behavior to represent situations not recorded in the log. However, the process model should not allow for behavior completely unrelated to the one observed in the event log, measured by the precision dimension. Finally, the simplicity dimension indicates that the process model should be as simple as possible. The available discovery algorithms pursue a trade-off between these four quality criteria (van der Aalst, 2016).

After collecting a minimal number of traces, it is possible to apply a discovery algorithm to obtain a process model and then evaluate some quality metrics, e.g., fitness and precision. In a steady-state situation, i.e., the process does not change; we can re-evaluate the same quality metrics after collecting more traces, and they should stay stable. However,

in a process drift situation, where the process model changes, the traces represent a different behavior after the change point. Therefore, we believe the values for the same quality metrics evaluated using the previously discovered process model and the new traces (after the drift) will change. The proposed method aims to identify the moment when the process change by applying a change detector algorithm to the fitness and precision metrics.

The proposed sudden concept drift detection method extends the *Interactive Process Drift (IPDD) Framework* (Sato, Barddal, et al., 2021) for detecting process drifts in the control-flow perspective, based on adaptive windows (Figure 5.2). The motivation is to minimize the impact of the parameters for defining the windows by applying an adaptive windowing approach for splitting the event log. The new approach changes the *Windowing strategy* module from the framework by implementing a strategy using the ADWIN change detector algorithm (Bifet & Gavaldà, 2007) in two quality metrics for process models: fitness and precision (Buijs et al., 2014).

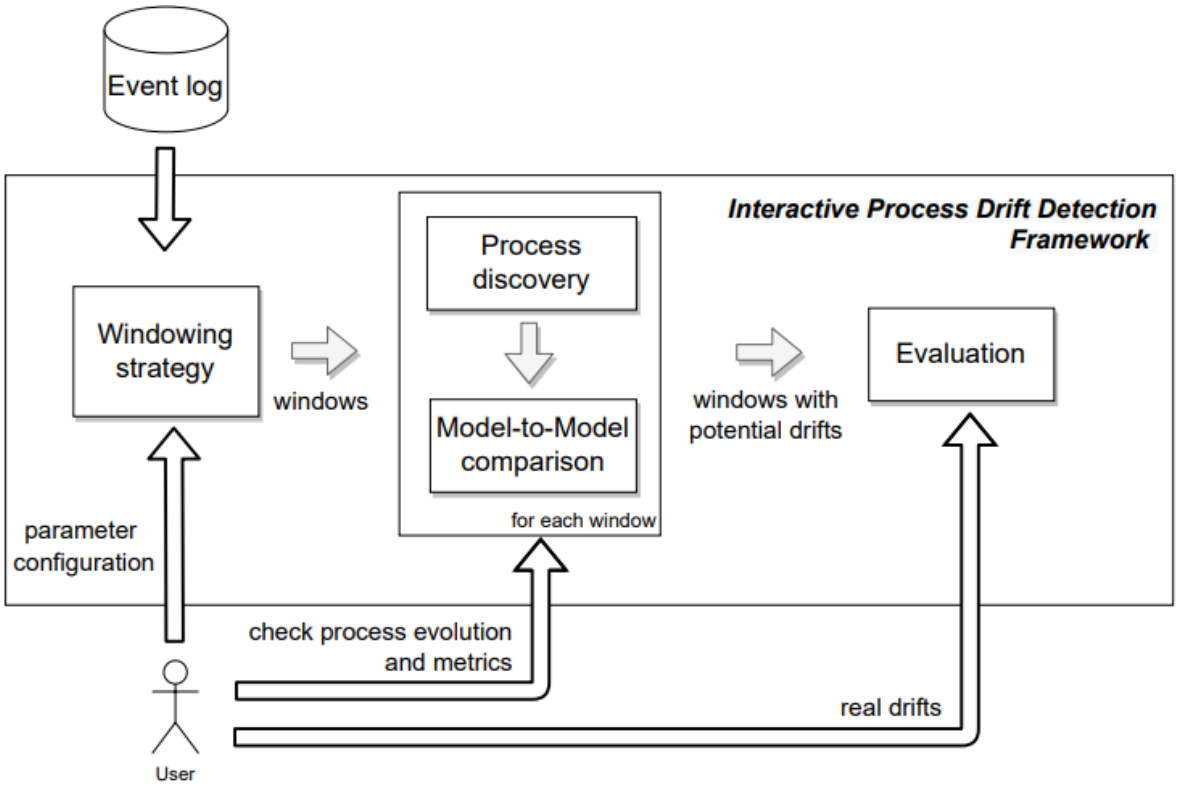


Figure 5.2. Overview of the IPDD Framework.

We assume that the quality metrics stay stable if the process model does not change over time. Based on this assumption, the new windowing strategy contains two stages: the setup and detection phases. In the setup phase, the method first discovers an initial process model (using the initial traces), calculates the fitness and precision for the initial traces, and inputs the ADWIN detector with the calculated values. In the detection phase, the method reads the

following trace and calculates the fitness and precision using the initial model. Next, it updates the ADWIN change detector with the new values and checks for a change. If a drift is detected, the method reports the change point and returns to the setup phase, discovering a new model using the traces after the change point. If no drift is detected, the method continues on the detection phase by reading the following trace in the event log.

We consider two quality metrics in this method: fitness and precision (Buijs et al., 2014). Evaluating the fitness over time allows the detection of new behavior in the log, e.g., adding a new activity or path or removing activities or paths. However, if there is a behavior in the model, e.g., a path that allows skipping some activities or removing a loop after a drift, the fitness metric will stay unaffected because the observed behavior will still fit the model. We included the precision metric for situations like this, where the model included behavior not observed in the log. The precision metric measures the amount of unobserved behavior in the event log present in the process model (Buijs et al., 2014). If a behavior is not observed in the event log after some time, the precision will decrease. Therefore combining precision and fitness, the method can identify changes that include or remove behavior in the process. The following modules from *IPDD* apply the same as described in (Sato, Barddal, et al., 2021; Sato, Fontana, et al., 2021):

- *Process discovery.* DFGs miner discovering process maps from the traces in the windows using the PM4Py (Berti et al., 2019).
- *Model-to-Model comparison.* The adjacent process maps are compared using the Nodes (NS) and Edges similarity (ES) metrics. NS is calculated using Equation (5.1), where n_p and n_q are the numbers of activities in the process maps *P* and *Q* (derived from adjacent windows), respectively, and n_{cs} indicates the number of common activities between *P* and *Q*. ES is calculated using Equation (5.2), similar to NS: e_p is the number of edges in *P*, e_q is the number of edges in *Q*, and e_{cs} indicates the number of common edges in both *P* and *Q*. *IPDD* calculates both metrics, and if one or both is less than 0, it marks the window as a drift.

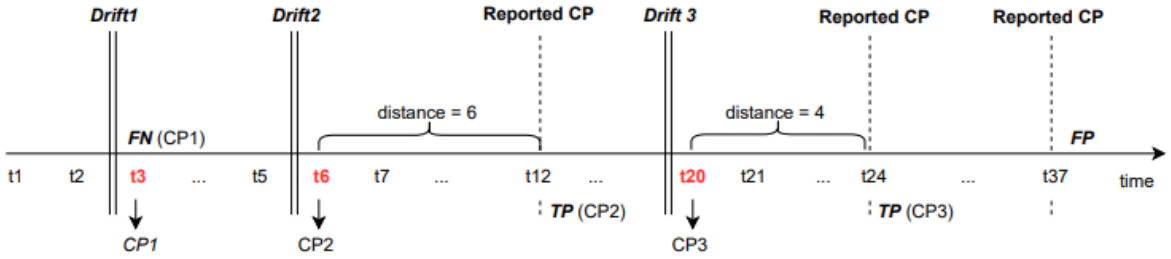
$$NS = 2 * n_{cs}/(n_p + n_q) \tag{5.1}$$

$$ES = 2 * e_{cs}/(e_p + e_q) \tag{5.2}$$

We enhanced the *Evaluation* module from *IPDD*, including the F-score and mean delay metrics based on the detected change points. In the former implementation of *IPDD* (fixed window as the *Windowing strategy*), the F-score metric is calculated based on the windows reported because the tool does not report a trace as the change point.

The F-score is the harmonic mean between precision and recall which relies on true positives (TP), false positives (FP), and false negatives (FN). A TP indicates a reported drift related to a real drift (we only consider the first reported drift after a real one); an FP is a reported drift unrelated to a real drift, and an FN is a real drift not detected. We consider an interval for the TPs $[rd, rd + et]$, where rd is the trace of the informed real drift and et is the number of traces to the next known drift. If there is only one drift in the event log, a TP is counted if the methods report a change point after the real drift point. The F-score reports a value between 0 and 1, measuring the drift detection accuracy. However, the F-score does not measure if the change point reported is "close" to the real change point. The mean delay complements the F-score analysis by measuring the distance between the reported change points and the real change points for the reported drifts considered a TP.

Figure 5.3 shows an example of the two metrics. In the x-axis, we can see the traces ordered by timestamp (*t1, t2, …, tn*). The event log contains three drifts at change points: *t3*, *t6*, and *t20*. The detection method reports three drifts in the example: *t12*, *t24*, and *t37*. The F-score will count a TP only if a process drift is reported after a real drift and before the following know drift. In the example, only *t12* and *t24* are assumed as TP. The distance between the reported drift and the real one is the number of traces between the reported and the real drift.



$$F - score = \frac{TP}{TP + \frac{1}{2}(FP + FN)} = \frac{2}{2 + \frac{1}{2}(1 + 1)} = 0.67$$

Figure 5.3. Example of the evaluation metrics implemented in *IPDD*.

The following topics describe two approaches to include the described adaptive window procedure in the *Windowing strategy* module of the *IPDD*.

5.3.1 ADAPTIVE IPDD: TRACE BY TRACE APPROACH

In the *trace by trace* approach, the detection phase of the windowing strategy reads and evaluates the fitness and precision metrics using the last read trace. The complete process applied to the *trace by trace* approach is described in Figure 5.4. In the setup phase, the approach derives the first process model and inputs the initial values for fitness and precision

into the ADWIN detector. Then in the detection phase, this approach reads the following trace from the event log and calculates the fitness and precision metrics using the new trace and the model discovered in the setup phase. The approach uses the PM4Py framework[32] (Berti et al., 2019) for calculating the fitness and precision metrics, and it is possible to apply one of the available methods. We choose to calculate the fitness using token-replay (Berti & van der Aalst, 2019) and precision using token-replay (Muñoz-Gama & Carmona, 2010) because they are less computation expensive than the alignments based metrics.

The ADWIN change detector (Bifet & Gavaldà, 2007) detects drifts in data sequences that may vary with time. In the *IPDD* approach, we input two data series into the detector: fitness and precision. ADWIN maintains a window of variable size with recently seen items consistent with the hypothesis that "there has been no change in the average output value inside the window according to a confidence bound δ" (Bifet & Gavaldà, 2007). The δ parameter indicates the confidence level, and increasing this parameter will also increase the false positives rate (Bifet & Gavaldà, 2007). We apply the ADWIN change detector because it has upper on false positives and negatives. Also, ADWIN is parameter assumption-free since it automatically detects and adapts to the current rate of change without defining a window size. Despite its theoretical bounds, ADWIN is known for triggering too many false positives (Huang et al., 2015), i.e., it reports changes when they do not occur. We use the implementation of the ADWIN change detector from the scikit-multiflow library[33] (Montiel et al., 2018).

The token-replay fitness implementation available in PM4Py better handles invisible transitions and improved intermediate storage techniques, outperforming the original token-based approaches and the alignment-based approaches for Petri nets with visible and invisible transitions (Berti & van der Aalst, 2019).

The precision metric calculated using token-replay is faster and provides a value that estimates the effort to obtain an accurate model, focusing on the detected discrepancies. The calculation of the precision metric avoids the potential state explosion that might arise when dealing with large and highly concurrent process models by only focusing on the behavior present in the log (Muñoz-Gama & Carmona, 2010).

---

[32] PM4Py is an open source process mining platform: https://pm4py.fit.fraunhofer.de/

[33] Scikit-multiflow is a machine learning package for streaming data in Python: https://scikit-multiflow.github.io/. In the available implementation of the library, ADWIN reports different change points when comparing to MOA (https://moa.cms.waikato.ac.nz/). We changed the source code based on MOA's source code and included the modified library at: https://github.com/denisesato/InteractiveProcessDriftDetectionFW.

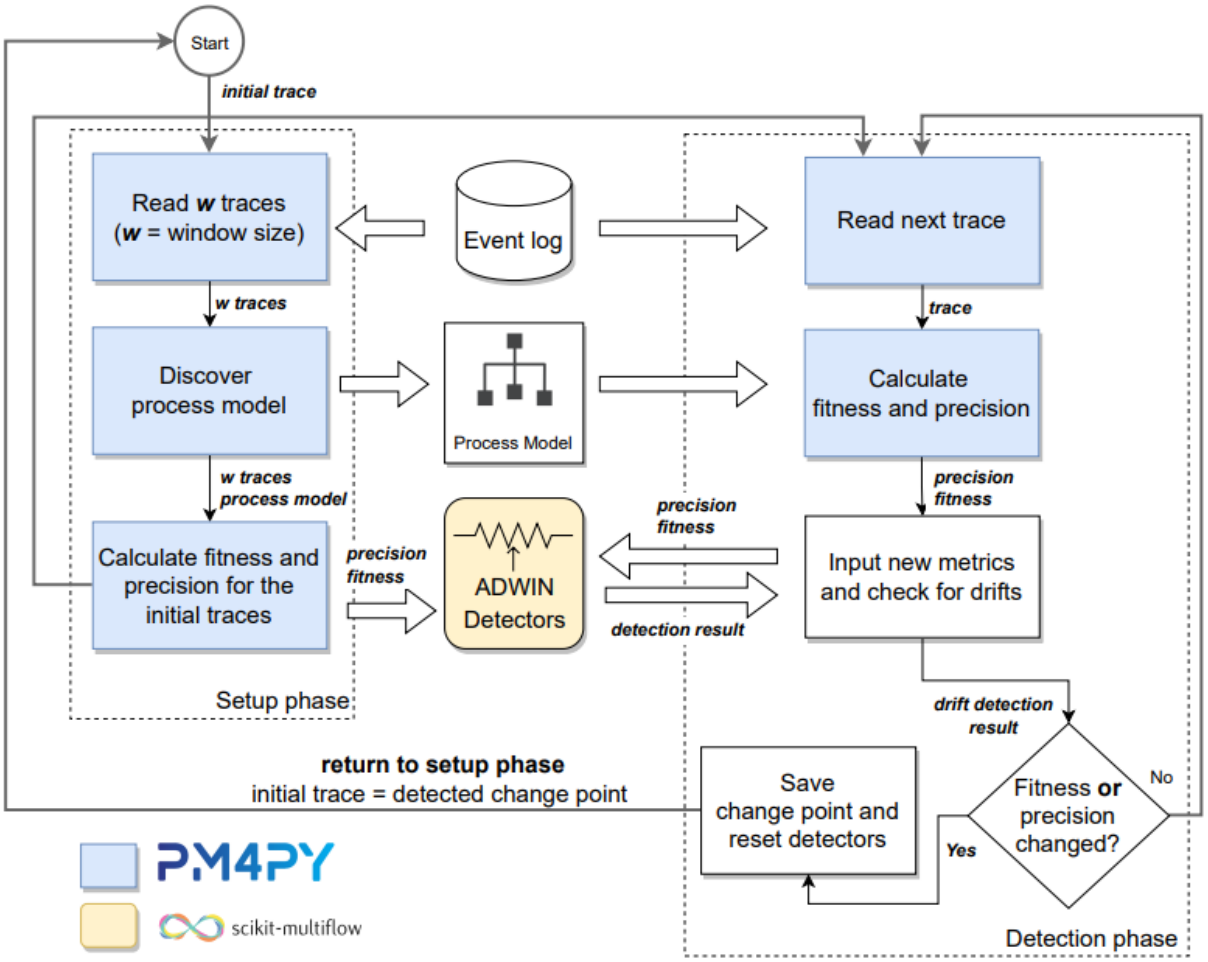


Figure 5.4. Description of the *trace by trace* approach for detecting control-flow drifts.

The *trace by trace* approach receives two inputs: the event log and the number of initial traces, also named window size (*w*). After detecting a drift, the change point and the window used to derive the process model are saved. After reading the complete event log, the adaptive windowing approach reports all the saved windows to the next step of *IPDD*, which performs the *process discovery* and the *model-to-model* comparison. It is essential to highlight that the saved windows do not include all traces between change points. The reason is that the process drift detection is a reactive procedure, so the change point is always identified after the drift occurs. Thus, applying the discovered algorithm on a sub log from the initial trace to the change point will derive a process model containing some traces after the drift.

The user can then verify the evolution of the process over time by visualizing the process models for each window, which addresses the change process discovery challenge. Also, after selecting a window, the user can localize the change by checking the similarity metrics information. Furthermore, the user can evaluate the drifts detected using a synthetic dataset by evaluating the two implemented metrics: F-score and mean delay.

5.3.2 ADAPTIVE IPDD: WINDOWING APPROACH

The *windowing approach* differs from the *trace by trace* approach in the precision metric calculation. Evaluating precision estimates how much behavior is allowed by the model and not observed in the event log (Muñoz-Gama & Carmona, 2010). However, when precision is computed using only one trace (the last one), we are not exactly providing the expected information because one trace does not represent an event log, which motivates using a sliding window for calculating the precision metrics, as defined in Figure 5.5.

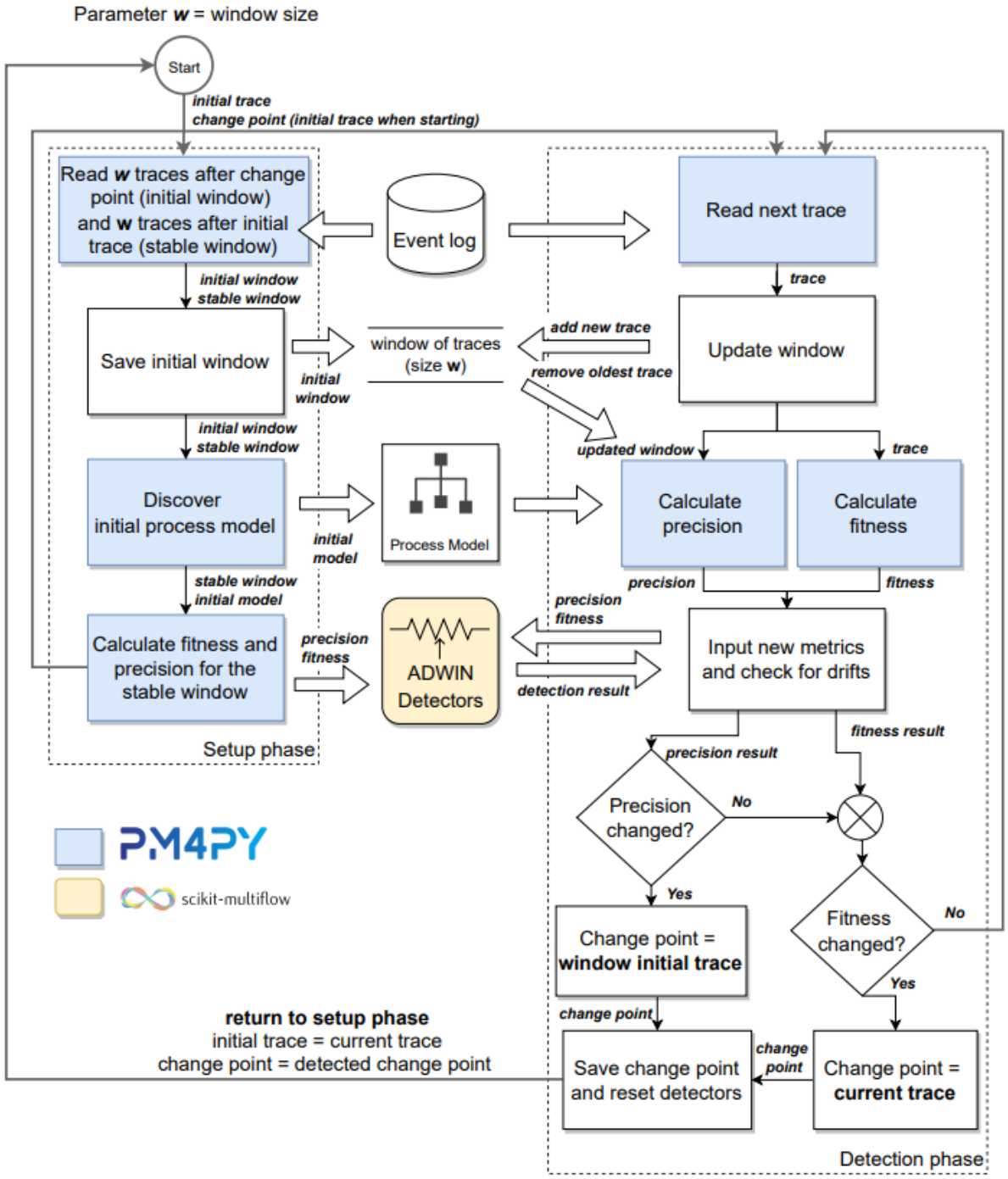


Figure 5.5. Description of the *windowing* approach for detecting control-flow drifts.

In the *windowing* approach, the setup phase saves the initial window. After reading a trace (detection phase), the window is updated by adding the new trace and removing the oldest, maintaining a fixed-size window. The approach calculates the precision metric using the window and the current process model. We apply precision based on footprints because it provides a simple and scalable technique for comparing the behavior between the model and the event log (van der Aalst, 2016).

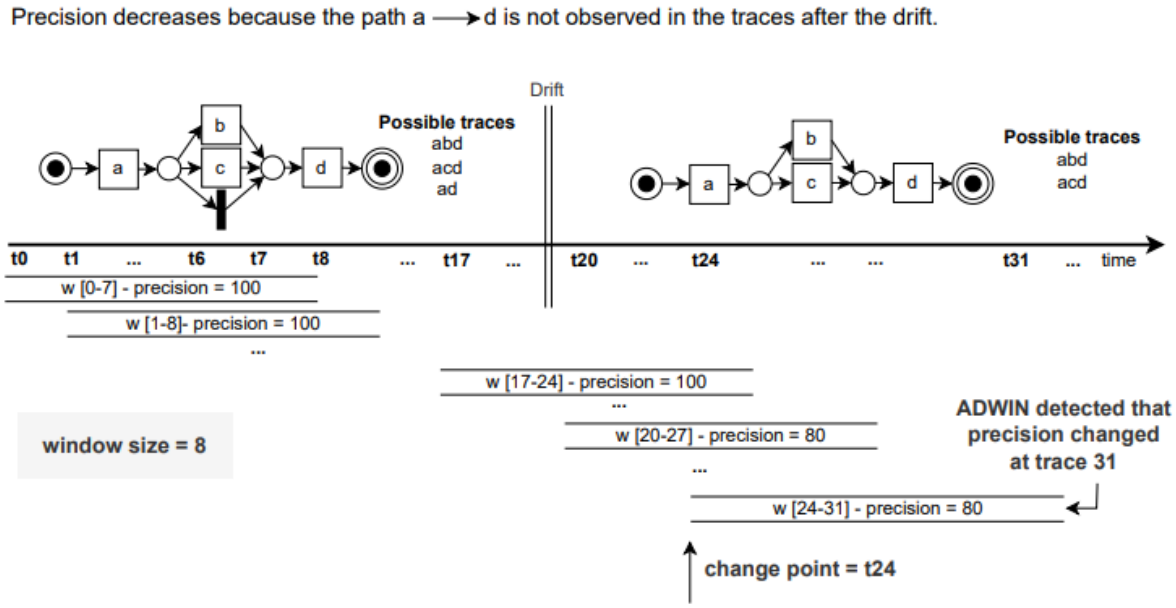


Figure 5.6. Precision metric using a sliding window.

Because the precision detects a “forgotten” behavior, i.e., something in the model that stops being observed in the traces, the change point reported is the initial of the current window. Figure 5.6 shows a situation where the precision metric detects a drift. In this case, the path from “a” to “d”, which allows escaping performing activities “b” or “c”, is removed. In the *windowing* approach, the decrease in the precision metric will start only after the current window contains only traces after the drift change point. Then, the ADWIN change detector needs more inputs of a small precision to report the change and ensure it is not noise. Finally, when the method reads *t31*, ADWIN reports a drift. As we know that some behavior has been forgotten, we report the initial trace of the window as the change point.

5.4 EXPERIMENTAL SETUP FOR SYNTHETIC EVENT LOGS

We choose two synthetic datasets to assess the accuracy of the two proposed approaches. The first dataset, name dataset 1, was initially created for the Apromore ProDrift

Runs approach (Abderrahmane Maaradji et al., 2015), and it is publicly available[34]. However, some of the logs from the original dataset do not follow the specification described in the paper. Appendix 5.A details the event logs in this situation. Also, the event logs are not in the XES format[35]. Because of these issues, we simulated new event logs for the cases where the original ones were not fitting the paper's description and converted all the event logs to the XES format – using the ProM framework (van der Aalst et al., 2009). The new dataset, the Petri nets for the change patterns, and the source code for simulating an event log with sudden and recurrent drifts are publicly available[36]. We do not include the *fr* (change branching frequency) change pattern from the original dataset because it only affects the routing of cases and does not change the structure of the process.

Table 5.1 describes the change patterns applied to dataset 1 and presents some highlighted modifications from the original table (Abderrahmane Maaradji et al., 2015). There are 11 simple patterns and six complex patterns derived from simple patterns. We maintain the same classification proposed in (Abderrahmane Maaradji et al., 2015) for the simple patterns: I – insertion, *R* – resequentialization, and *O* – optionalization. We changed the classification of the *sw* (swap two fragments) pattern to *R* because it better fits the description and was used in the *OIR* pattern. We also changed the name o *RIO* pattern to *RI*, because there is no optionalization included in the altered model defined in the dataset (we considered the BPMN model provided in the original dataset). For each of the 17 change patterns, the dataset included 4 log sizes (2,500; 5,000; 7,500; and 10,000 traces). Each log contains nine drifts injected after 10% of the size (250, 500, 750, and 10,000 traces), changing from the base model to the altered model and vice-versa. Consequently, we simulated nine sudden recurrent drifts. All the event logs start from the base model and change to the altered model, as described in Table 5.1.

Table 5.1. Change patterns of the event logs. Adapted from (Abderrahmane Maaradji et al., 2015).

Code	Change pattern	Category	Description
cb	Make fragment skippable/non-skippable	O	"Prepare acceptance pack" and "Check if home insurance quote is requested" turned skippable.
cd	Synchronize two fragments	R	"Assess loan risk" can only be performed after both "Appraise property" and "Check credit history".
cf	Make two fragments conditional/sequential	R	"Send acceptance pack" and "Send home insurance quote" from conditional to sequential.
cm	Move fragment into/out	I	"Prepare acceptance pack" moved into conditional branch

[34] https://data.4tu.nl/articles/dataset/Business_Process_Drift/12712436

[35] See www.xes-standard.org for detailed information about the standard.

[36] Source code available at: https://github.com/denisesato/SimulateLogsWithDrifts/

	of conditional branch		containing “Send acceptance pack” and “Send home insurance quote”.
cp	Duplicate fragment	I	Fragment “Check credit history” and “Assess loan risk” duplicated after “Verify repayment agreement”.
IOR	I: re O: cb R: cd	IOR	I: “Added activity” included. O: “Reject application” turned skippable. R: “Added activity” e “Reject application" synchronized.
IRO	I: re R: cd O: lp	IRO	I: Fragment containing “Added activity”, “Verify repayment agreement”, “Prepare acceptance pack” included. R: “Added activity” synchronized with “Verify repayment agreement”, “Prepare acceptance pack”. O: Fragment containing “Approve application”, “Verify repayment agreement”, “Prepare acceptance pack”, “Added activity” turned loopable.
lp	Make fragment loopable/non-loopable	O	“Assess eligibility” turned loopable.
OIR	O: lp I: re R: sw	OIR	O: Fragment “Check credit history”, “Assess loan risk”, “Appraise property” turned loopable. I: “Added activity” included. R: “Check credit history” and “Assess loan risk” swapped.
ORI	O: lp R: cf I: re	ORI	O: Fragment “Check if home insurance quote is requested”, “Send home insurance quote”, “Send acceptance pack”, “Verify repayment agreement” turned loopable. R: Change “Send home insurance quote”, “Send acceptance pack” from conditional to sequential. I: “Added activity” included.
pl	Make two fragments parallel/sequential	R	Fragment “Appraise property”, “Check credit history”, “Assess loan risk” from parallel to sequential.
pm	Move fragment into/out of parallel branch	I	“Prepare acceptance pack” moved into a parallel branch with “Send home insurance quote”
re	Add/remove fragment	I	“Assess eligibility” removed.
RI	**R: cf** **I: cp**	**RI**	R: Fragment “Prepare acceptance pack”, “Check if home insurance quote is requested” from sequential to conditional after “Send home insurance quote”. I: Fragment “Prepare acceptance pack”, “Check if home insurance quote is requested” duplicated after “Assess eligibility”
ROI	R: pl O: lp I: rp	ROI	R: Fragment “Check credit history”, “Assess loan risk” and “Appraise property ” from parallel to sequential (“Appraise property” first”). O: Fragment “Appraise property”, Check credit history” turned loopable. I: “Check credit history” replaced by “Replaced activity”.
rp	Substitute fragment	I	“Verify repayment agreement” replaced by “Replaced activity”
sw	**Swap two fragments**	**R**	Swap fragments “Prepare acceptance pack” followed by “Check if home insurance is requested” with “Verify repayment agreement”.

We simulated the second dataset using the same change patterns defined in Table 5.1 and the same proposal of changing from a base to an altered model. However, we simulated different log sizes and intervals between drifts. Dataset 2 contains three log sizes, and the interval between the drifts varies to avoid the bias of a fixed interval between drifts:

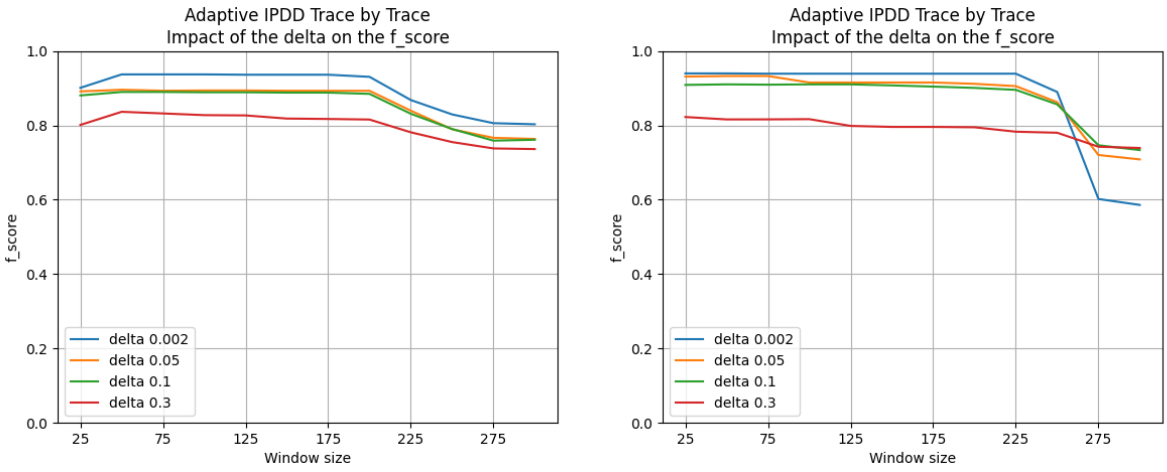
- 3,000 traces: 4 drifts (250, 750, 1500, 2500)
- 4,500 traces: 7 drifts (250, 750, 1500, 2500, 3250, 3750, 4000)
- 8,000 traces: 13 drifts (250, 750, 1500, 2500, 3250, 3750, 4000, 4500, 5250, 6250, 7000, 7500, 7750)

5.5 ADAPTIVE IPDD TRACE BY TRACE ON SYNTHETIC EVENT LOGS

Firstly, we analyzed the *trace by trace approach* using the datasets defined in Section 5.4.

5.5.1 IMPACT OF THE DELTA PARAMETER AND WINDOW SIZE ON ACCURACY

The first analysis evaluates if different configurations of the ADWIN's delta parameter will affect the detection accuracy. We calculated the F-score metric for the two synthetic datasets using a window varying from 25 to 300 traces (with a step of 25). We decided to vary the window size parameter because one of the motivations of the *Adaptive IPDD* approach is to reduce the sensitivity of the accuracy given different hyper-parameter values. We apply four delta values: 0.002, 0.05, 0.1, and 0.3 (0.002 is the default value in the ADWIN implementation; the other values were applied to investigate the rate of false positives in (Bifet & Gavaldà, 2007)).



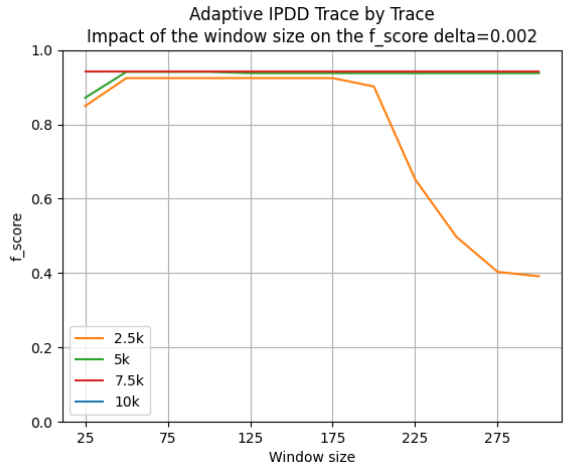
(a) Dataset 1 (b) Dataset 2

Figure 5.7. Impact of the delta parameter on the F-score (mean) – *trace by trace* approach.

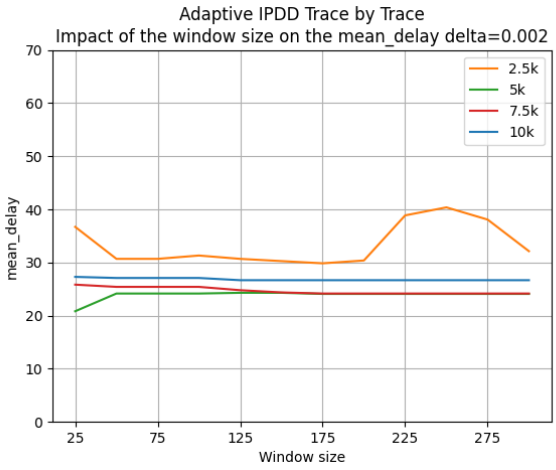
In Figure 5.7, we plot the average F-score considering the 17 change patterns, the four sizes for dataset 1 (2,500; 5,000; 7,500; 10,000 traces), and the three sizes of dataset 2 (3,000; 4,500; 8,000 traces). Visually we can observe that the default value of 0.002 results in better F-score rates. The F-score values decrease when we increase the delta parameter.

Furthermore, the F-score drops after the window size of 200. This behavior is expected because both datasets contain drifts with an interval of 250 traces. Using a window size higher than 200, we are probably considering traces after a drift (in case of an interval

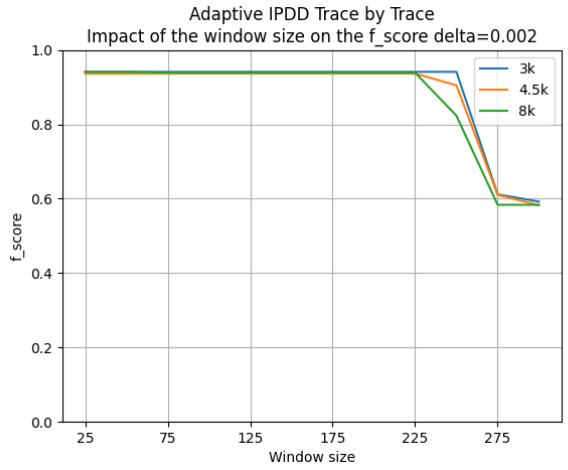
between drifts of 250 traces) for mining the process model. However, the accuracy of the approach is not affected by smaller window sizes.



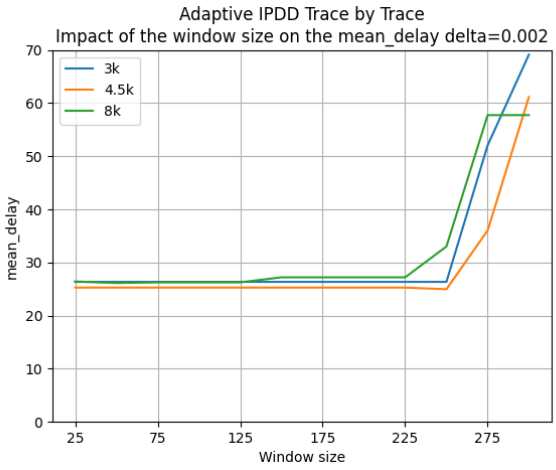
(a) Dataset 1 F-score



(b) Dataset 1 Mean delay



(c) Dataset 2 F-score



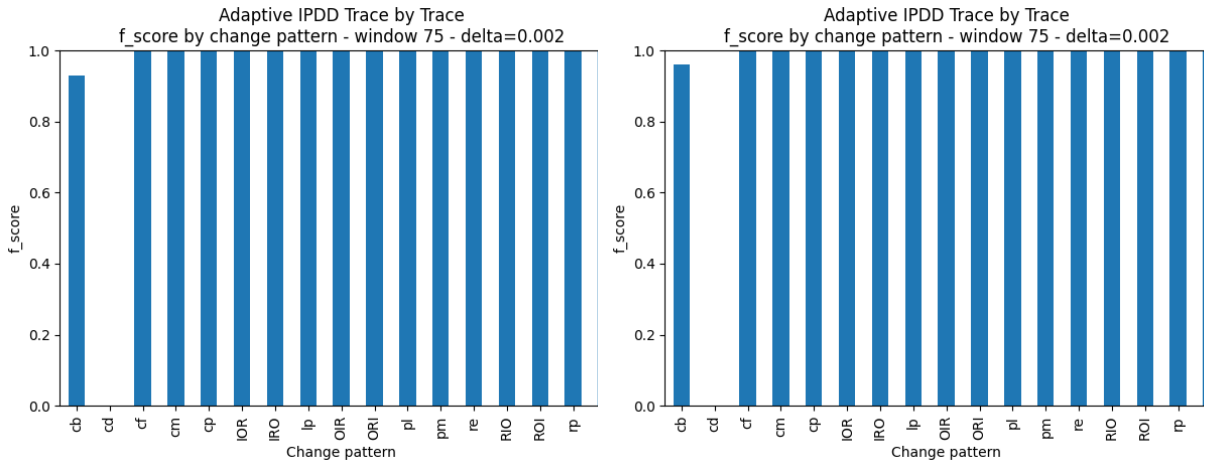
(d) Dataset 2 Mean delay

Figure 5.8. Impact of the window size on the F-score and mean delay – *trace by trace* approach. Event logs are grouped by size.

We can visualize the drop in the F-score for dataset 1 with delta=0.002 when the interval between drifts is 250 (2.5k traces) in Figure 5.8 (a). In the other log sizes, containing drifts with intervals of 500, 750, and 1,000 traces, the F-score values maintain above 0.8 for all the selected window sizes. Figure 5.8 (b) shows the mean delay metric, which complements the F-score by indicating how "close" to the actual change point *IPDD* detects the drifts. We can observe that for the event logs 5k, 7.5k, and 10k, the mean delay is below 30 traces. In log 2.5k, the mean delay increases after the window size of 200, following the decrease of the F-score. In dataset 2, all the logs include an interval between drifts of 250 traces, which explains the drop of the F-score after a window size of 225 (Figure 5.8 c). The mean delays stay stable before the window size of 225, showing that the reported change points are close to the real ones.

5.5.2 DETECTION ACCURACY PER CHANGE PATTERN

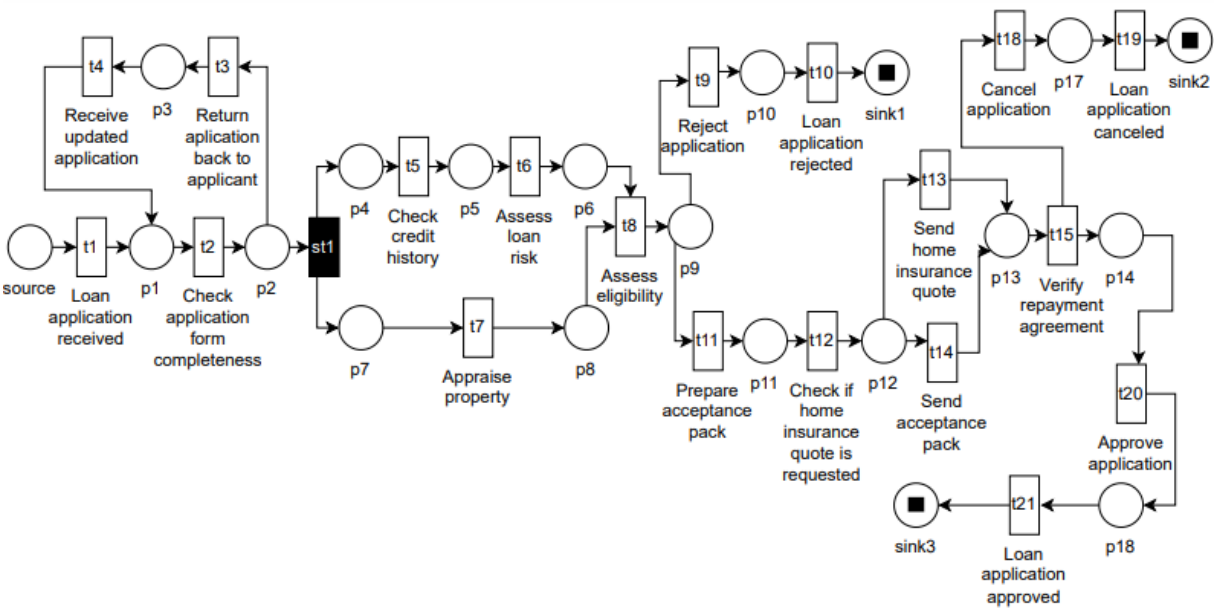
We analyze the accuracy per change pattern to understand better why the F-score does not reach 1.0 even in the “best” parameter configuration. Figure 5.9 plots the average F-score for both datasets, grouping all the event log sizes for window size=75 and delta=0.002.



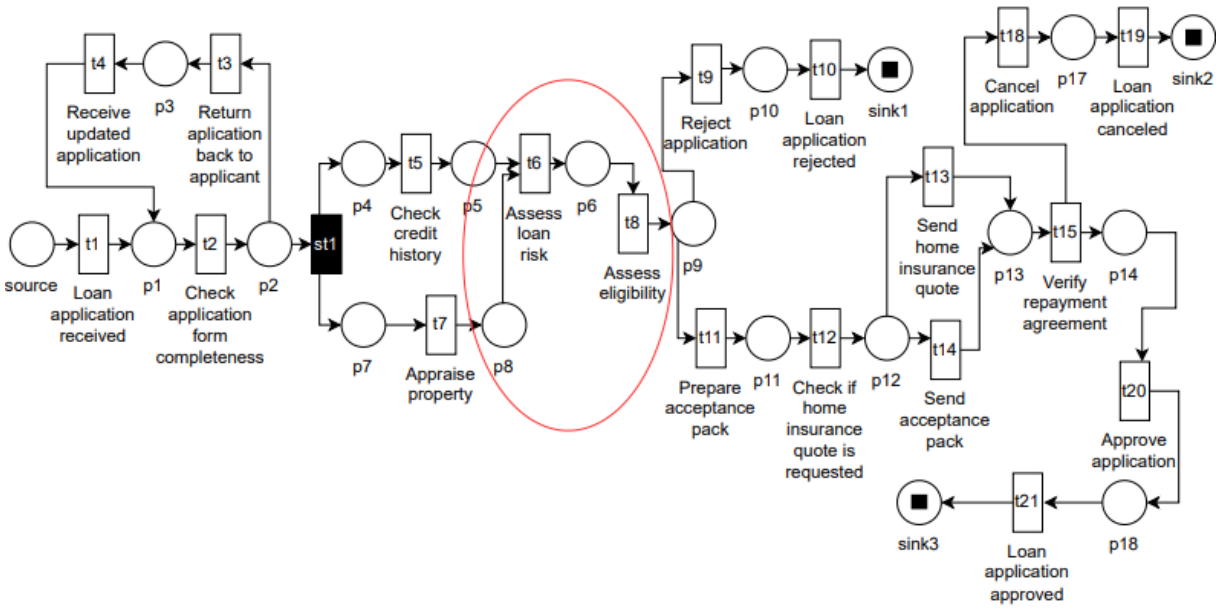
(a) Dataset 1 (b) Dataset 2

Figure 5.9. Accuracy per change pattern window=75 and delta-0.002 – *trace by trace* approach.

We can observe that the approach never detects the pattern *cd*. This changing pattern includes a control dependency in the baseline model by synchronizing the “Assess loan risk” with two activities: “Check credit history” and “Appraise property” (Figure 5.10).



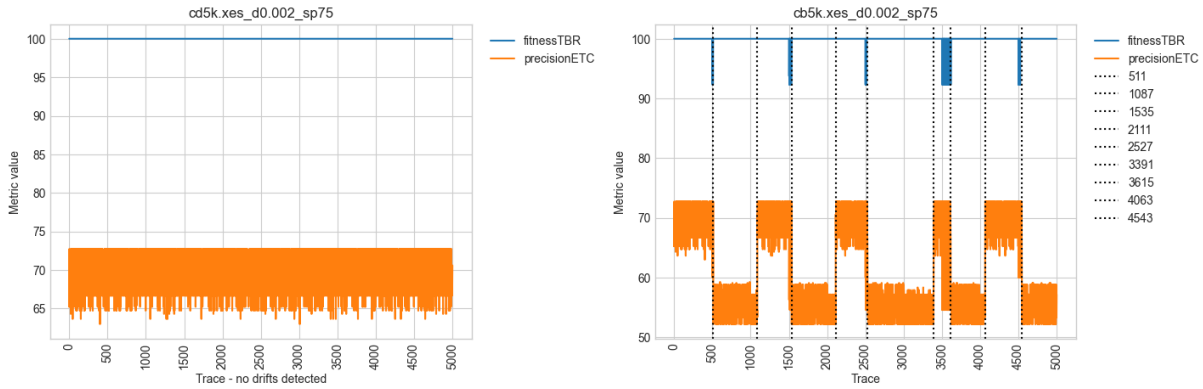
(a) Base model



(b) Altered model (*cd* – Synchronize two fragments)

Figure 5.10. Change pattern *cd* in the model (Petri net).

The fitness metric does not detect the change to the *cd* model because all possible traces in the *cd* process are also allowed by the *base* model, i.e., the *base* model is more generic than *cd*. In this case, we expect the precision metric detects the change. However, because we evaluate precision using only the last read trace in the *trace by trace* approach, the decrease in the precision is not enough to report the change in this situation (Figure 5.11 a). When we compare the precision of the *cb* model (Figure 5.11 b), which creates a path to skip two activities, we can see that the precision decreases by adding the new paths.



(a) *cd* – Synchronize two fragments (b) *cb* – Make fragment skippable/non-skippable

Figure 5.11. Fitness and precision time series window=75 delta=0.002 *trace by trace* approach.

We assume that when evaluating precision using only one trace, the precision stays stable if both models (pre-drift and pos-drift) are similar in the simplicity dimension. The precision metric evaluates how much behavior is allowed by the model and not observed in the log.

When evaluating only one trace, the precision will change if the model changes to a more complex or straightforward model. The change applied in the *cd* model does not reflect in a more complex model. We verify this assumption by calculating the simplicity metric (Blum, 2015) for all change patterns using PM4Py (Table 5.2). We can observe that the pattern *cd* does not change the simplicity metric. For the pattern *re*, the simplicity metric is the same. However, the activity "Assess eligibility" is removed, thus, reflecting a decrease in the precision metric. The fitness metric detects the drifts for the other change patterns (*ROI, rp, sw*).

Table 5.2. Simplicity metric for each change pattern.

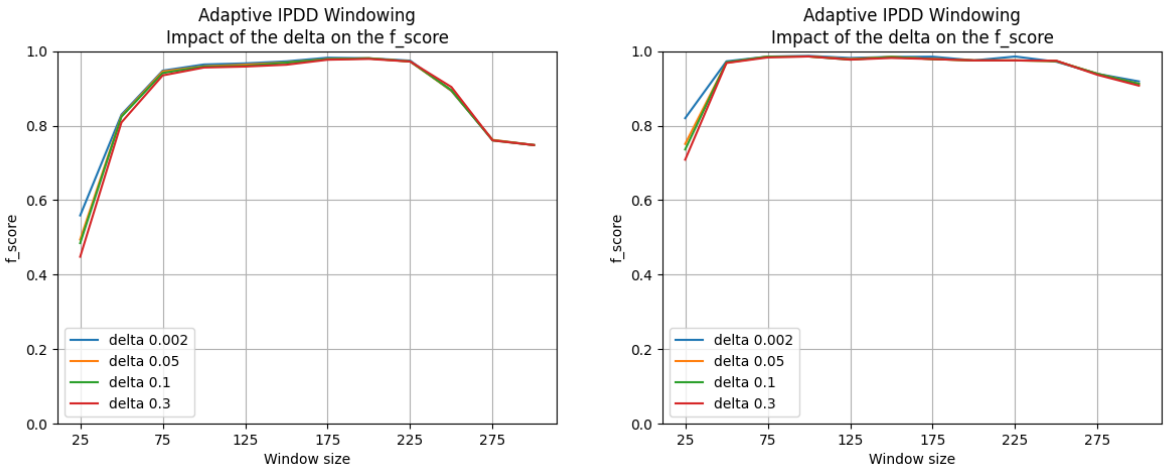
Code	Change pattern	Simplicity
base	Base model	**0.83**
cb	Make fragment skippable/non-skippable	0.80
cd	Synchronize two fragments	**0.83**
cf	Make two fragments conditional/sequential	0.87
cm	Move fragment into/out of conditional branch	0.79
cp	Duplicate fragment	0.92
IOR	I: re, O: cb, R: cd	0.79
IRO	I: re, R: cd, O: lp	0.86
lp	Make fragment loopable/non-loopable	0.81
OIR	O: lp, I: re, R: sw	0.78
ORI	O: lp, R: cf, I: re	0.85
pl	Make two fragments parallel/sequential	0.85
pm	Move fragment into/out of parallel branch	0.81
re	Add/remove fragment	**0.83**
RI	R: cf, I: cp	0.92
ROI	R: pl, O: lp, I: rp	**0.83**
rp	Substitute fragment	**0.83**
sw	Swap two fragments	**0.83**

5.6 ADAPTIVE IPDD WINDOWING ON SYNTHETIC EVENT LOGS

The limitation of the *trace by trace* approach to detect the control dependency pattern (*cd*) and the concept of evaluating the precision of a model related to an event log (and not a trace) motivated the implementation of the *windowing* approach.

5.6.1 IMPACT OF THE DELTA PARAMETER AND WINDOW SIZE ON ACCURACY

We apply the same evaluation performed for the *trace by trace* approach about the impact of the delta parameter and window size on the accuracy of the approach, i.e., F-score metric, window from 25 to 300 traces (step of 25), and delta values 0.002, 0.05, 0.1, and 0.3.

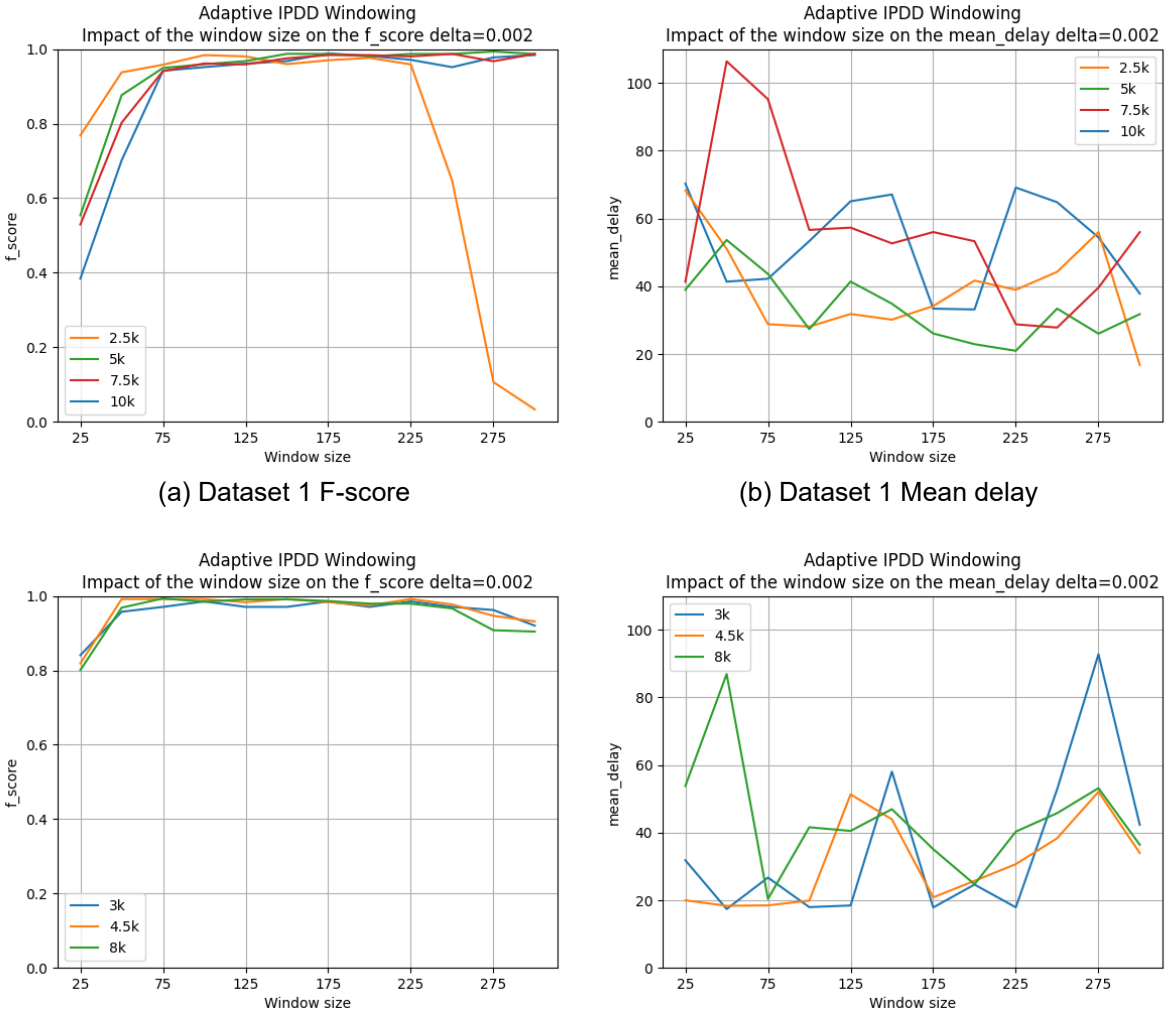


(a) Dataset 1 (b) Dataset 2

Figure 5.12. Impact of the delta parameter on the F-score (mean) *windowing* approach.

The *windowing* approach is more affected by window sizes 25 and 50 (Figure 5.12); however, after the window size of 75 F-score increases (close to 1.0). Again, we can observe a drop in the F-score after window size 200 for dataset 1 (Figure 5.12 a), which is compatible with the behavior of the *trace by trace* approach. However, the decrease in the F-score is not as accentuated as the *trace by trace* approach for dataset 2 (Figure 5.12 b). Another interesting observation is that the delta parameter does not affect the detection accuracy.

We plot the F-score and mean delay for both datasets, showing the results for each log size for delta=0.002 in Figure 5.13. We can observe that the drop in the F-score after the window size of 200 occurs in the 2.5k event logs, which contain a drift after 250 traces. However, when applying a window of traces, the reported change point is not so “close” to the real one compared to the *trace by trace* approach. We can observe this in Figure 5.13 b and Figure 5.13 d.



(a) Dataset 1 F-score

(b) Dataset 1 Mean delay

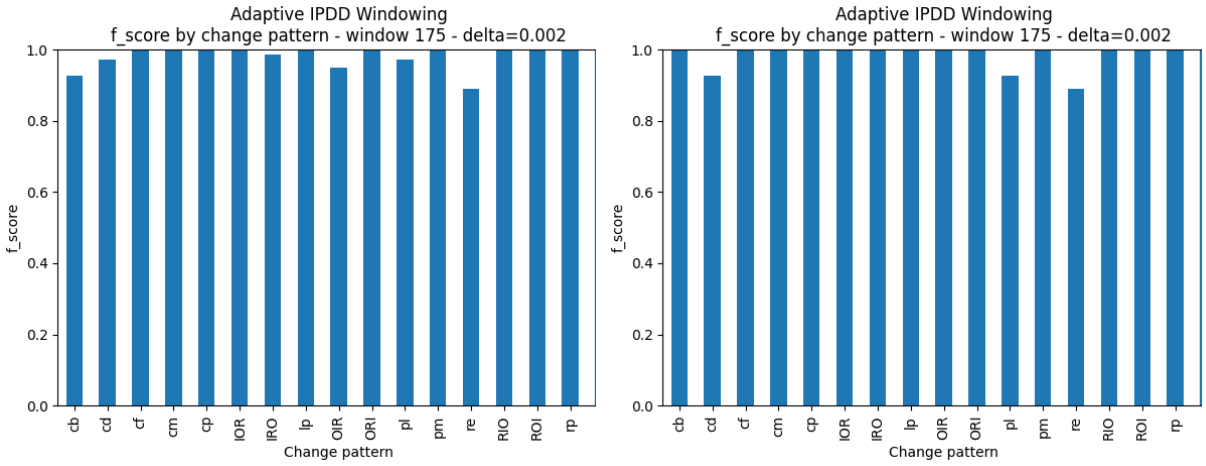
(c) Dataset 2 F-score

(d) Dataset 2 Mean delay

Figure 5.13. Impact of the window size on the F-score and mean delay – *windowing* approach. Event logs are grouped by size.

5.6.2 DETECTION ACCURACY PER CHANGE PATTERN

We analyze the accuracy per change pattern to understand the *windowing* approach's behavior and verify if the *cd* pattern can be accurately detected. We selected window size=175 and delta=0.002 based on the analysis performed in Section 5.6.1. We can observe that the windowing approach can detect the *cd* pattern, and the F-score stays above 0.8 for all change patterns.



(a) Dataset 1 (b) Dataset 2

Figure 5.14. Accuracy per change pattern window=175 and delta-0.002 – *windowing* approach.

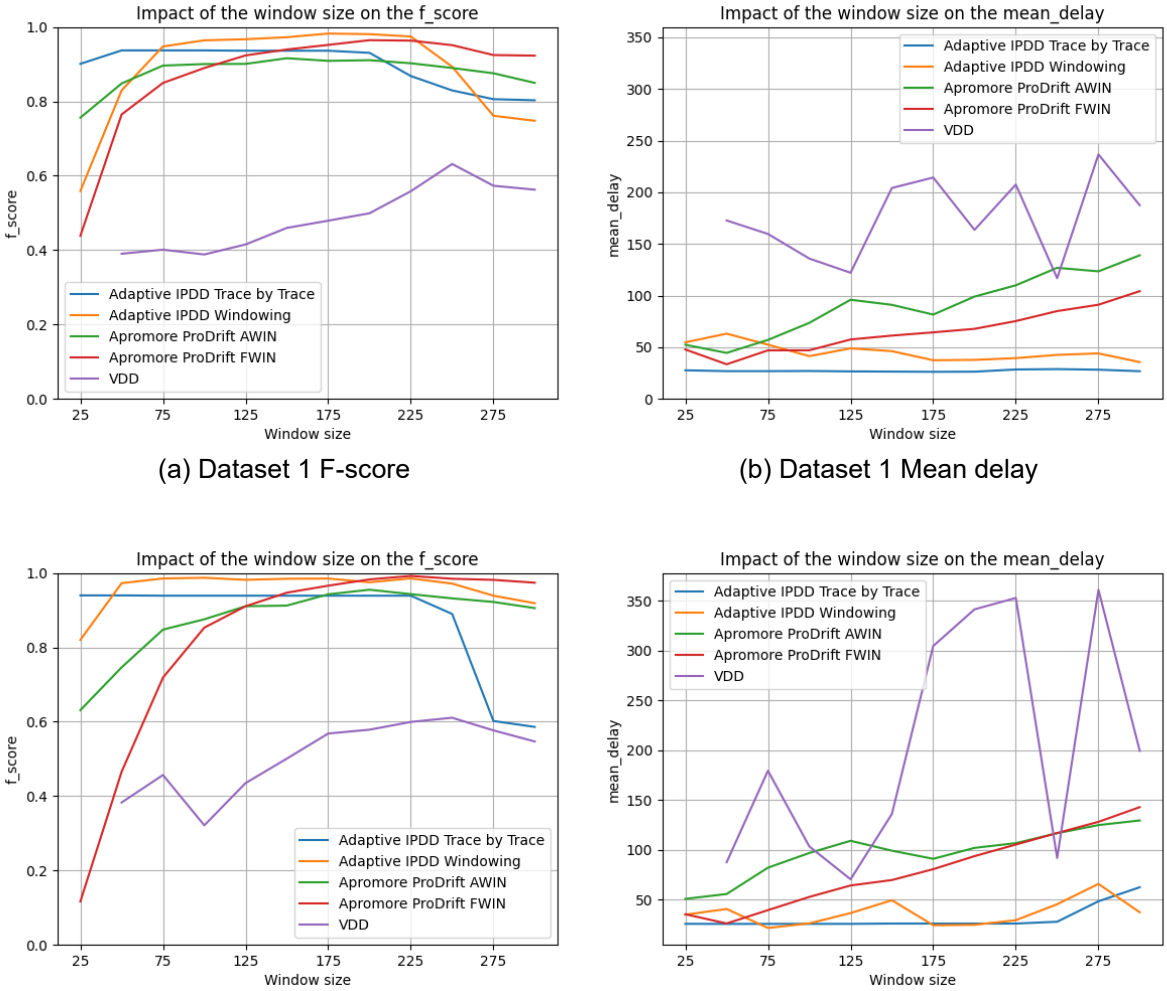
5.7 COMPARING IPDD AGAINST APROMORE PRODRIFT AND VDD

We analyzed datasets 1, and 2 using the same window configuration (25 to 300, step of 25) in the Apromore ProDrift runs approach, applying the fixed (FWIN) and adaptive (AWIN) approaches. The window size is set to the initial window size parameter in the AWIN approach.

We also analyzed the VDD using datasets 1 and 2 and the window configuration (50 to 300, step of 25). We do not include the results using a window size of 25 because VDD takes too much time in the execution (e.g., more than 10 hours for the lo cb10k in an I5-9500 with 8GB RAM). VDD requires another parameter, named window step, defined as $window\ size\ /\ 2$ (integer division). We chose this configuration of the window step because in (Yeshchenko et al., 2021), the authors suggested a $window\ size\ =\ 2\ *\ window\ step$. We performed the experiments using the clustering option (default), where the change points are derived for every cluster. VDD does not inform the trace as the change point in the Drift Map, so we calculated the trace of the drift based on the outputted information from the console (indicating the number of the window with a drift) and the window size parameter.

In Figure 5.15, we plot the average of the F-score and mean delay calculated over the different log sizes and change patterns (mean value) from both synthetic datasets. For *IPDD*, we apply a delta=0.002, which is the default value in the ADWIN implementation. We can observe that VDD performed the lowest F-score rates (Figure 5.15 a and Figure 5.15 c) when calculating the average of all the event logs for both datasets. *IPDD* and Apromore ProDrift achieve higher F-score results, however the visual analysis can not define if the differences are significant. VDD reports the highest values for the mean delay (Figure 5.15 b and Figure 5.15 d), indicating that VDD also reports change points with a considerably

distance from the real ones. Also, we can observe that the *IPDD* approaches performed better than Apromore ProDrift considering the mean delay.



(a) Dataset 1 F-score

(b) Dataset 1 Mean delay

(c) Dataset 2 F-score

(d) Dataset 2 Mean delay

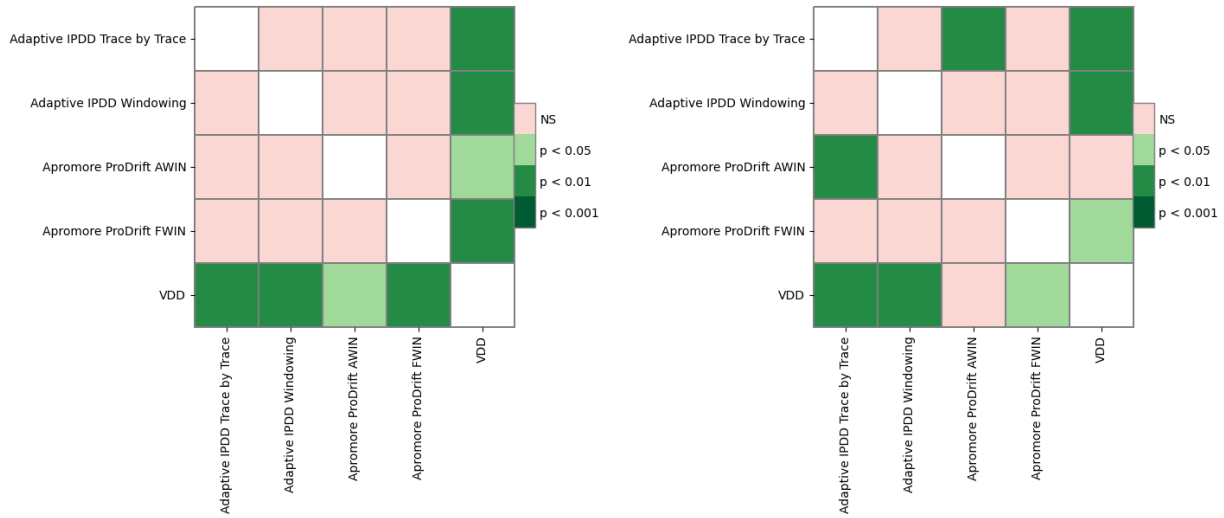
Figure 5.15. Comparing tools considering F-score and mean delay.

To understand the difference between the compared tools, we apply the Friedmann test for each dataset for the F-score and mean delay ($\alpha = 0.05$). We excluded window 25 of the Friedmann analysis because we do not have this configuration for VDD. The Friedmann test rejects the null hypothesis for both datasets and metrics, indicating that the results are significantly different considering F-score and mean delay.

We performed a Nemenyi post hoc test, presenting the results in the significance plots in Figure 5.16, showing different significance levels and NS indicating that the result indicates a non-significant difference. Figure 5.16 (a) shows that VDD F-score results are significantly lower than Apromore and *IPDD* for dataset 1. However, *IPDD* and Apromore performed similar results statistically. The mean delay reported for *Adaptive IPDD trace by*

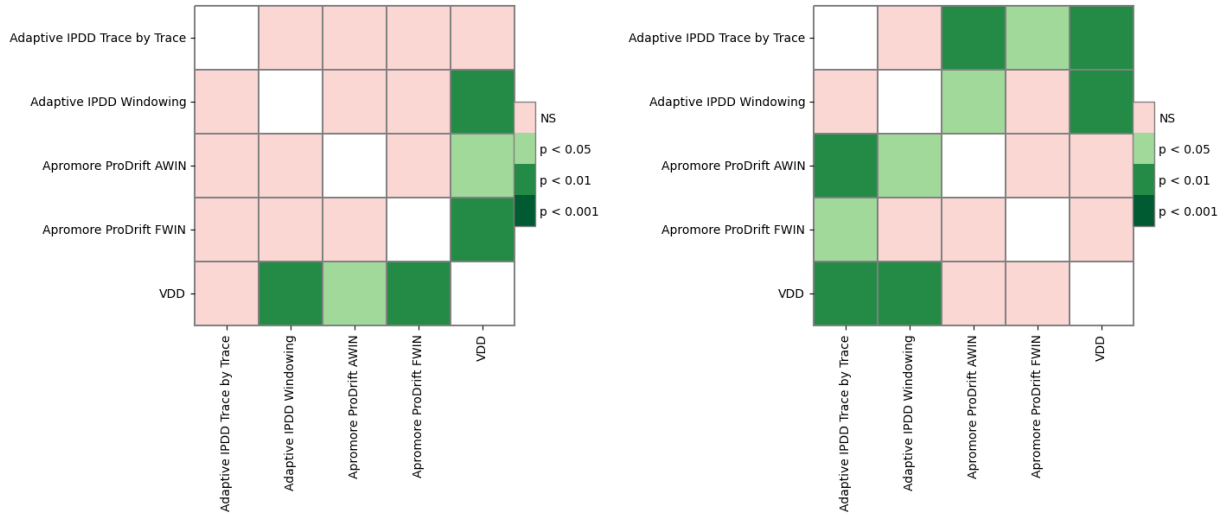
trace is better than the Apromore ProDrift AWIN and VDD. Furthermore, *Adaptive IPDD windowing* also reports better mean delays compared to VDD.

For dataset 2, we have a slightly different result for F-score (Figure 5.16 c). Only *Adaptive IPDD* windowing and Apromore ProDrift overcome VDD results. The reason is the accentuated drop of the F-score after a window size of 250 (Figure 5.15 c). *IPDD* and Apromore ProDrift FWIN presented significantly better results than the mean delay metric for Apromore ProDrift AWIN and VDD. Also, *IPDD trace by trace* is better than Apromore ProDrift AWIN, considering the mean delay.



(a) Dataset 1 – F-score

(b) Dataset 1 – Mean delay



(c) Dataset 2 – F-score

(d) Dataset 2 – Mean delay

Figure 5.16. The significant plot of the Nemenyi post hoc comparing the evaluated tools.

5.8 EVALUATION OF ADAPTIVE IPDD ON REAL-LIFE EVENT LOG

We further evaluated *Adaptive IPDD* on a public real-life event log representing a ticketing management process of an Italian Company[37]. The original CSV file was converted to XES using the Disco software[38] (academic license). The event log contains 4,580 cases and 14 activities from January 2010 to January 214. The same event log was evaluated for drift detection in (Yeshchenko et al., 2021; Yeshchenko, Ciccio, et al., 2019) and drift characterization in (Ostovar et al., 2020).

Firstly we applied the *Adaptive IPDD trace by trace* using a window size of 100, the same configuration applied in (Yeshchenko et al., 2021). *IPDD* reports 3 drifts, at traces 1,695, 2,239, and 3,743. VDD reports three overall drifts in traces 1,000, 1,750, and 3,750. The authors discarded the first drift reported because it reports an outlier behavior (by evaluating the Drift Plot). The following drifts are the same as the first and third change point reported by *IPDD* and are also reported in (Ostovar et al., 2020). However, in the VDD system visualization of the changes in the DFG, we cannot identify which change occurs at which change point. The user has to inspect cluster by cluster based on the erratic measure. It is unclear why clusters 9, 11, and 4 are selected in the table reporting the erratic measure. The autocorrelation results presented other clusters: 9, 12, and 15.

In (Ostovar et al., 2020), the authors apply the event stream approach (Ostovar et al., 2016) with an adaptive window (initial window=1,000 events), reporting the drifts at events 8,757 and 17,307. We read the event log as an event stream to identify the reported event's trace using PM4Py. The events belong to traces 1,716 and 3,754. It is interesting to verify that both reported drifts are also identified by *Adaptive IPDD trace by trace*. Furthermore, as expected by the synthetic dataset analysis, *IPDD* is able to identify the drift early: 21 traces and 11 traces. For the first drift (1,695), *IPDD* reports the same change characterized by Apromore ProDrift, which indicates a new edge in the DFG from "Assign seriousness" to "Resolve ticket". However, our approach also reports similarity metrics between the DFGs (before and after the drift), and identify more changes (Figure 5.17). Furthermore, *IPDD* shows the different versions of the process over time.

---

[37] https://doi.org/10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb

[38] https://fluxicon.com/disco/

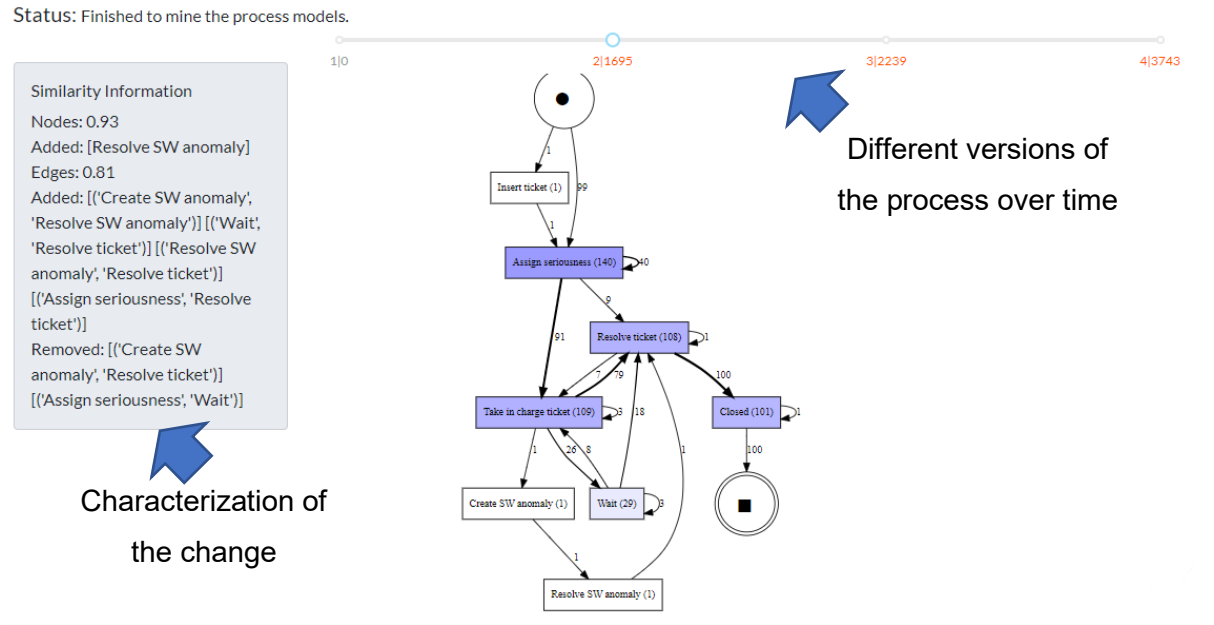


Figure 5.17. IPDD shows the differences for the first drift detected for the real-life event log.

The validation applied in (Ostovar et al., 2020) derives a DFG before and after the change point. As we already stated, as the change point is always reported after the drift, the validation approach derives a pre-drift DFG containing traces after the real drift. In *IPDD*, we derive the DFG using the initial windows reported by the adaptive windowing approach. Using the initial windows, *IPDD* derives different DFGs, which explains why we find out more changes related to the first drift point. *IPDD* detected the second drift reported by (Ostovar et al., 2020). However, we cannot compare the characterizations between both approaches because *IPDD* characterizes the change by comparing the drift reported at trace 2,239, not detected by Apromore ProDrift.

As we do not know the ground truth, we considered the second drift reported by *Adaptive IPDD trace by trace* because the following changes occurred in the process model:

- Nodes similarity: 77%.

  Removed: “Insert ticket”, “Resolve SW anomaly”, “Create SW anomaly”.

- Edges similarity: 76%.

  Added: “Resolve ticket” → “Wait”.

  Removed: “Take in charge ticket” → “Create SW anomaly”, “Take in charge ticket” → “Take in charge ticket”, “Closed” → “Closed”, “Resolve SW anomaly” → “Resolve ticket”, “Insert ticket” → “Assign seriousness”, “Create SW anomaly” → “Resolve SW anomaly”.

We also analyzed the real-life event log with *Adaptive IPDD* windowing using different window sizes (100, 200, 500, 800). *Adaptive IPDD windowing* reports more change points than expected in all scenarios, as informed in Table 5.3.

Table 5.3. The number of change points reported for *Adaptive IPDD windowing* on the real-life event log.

Window size	**Number of reported change points**
100	28
200	17
300	12
500	6
800	5

Analyzing process drift detection tools in real-life event logs raises the challenge of evaluating the results using an objective metric.

5.9 CONCLUSION

We presented two approaches for detecting process drifts in the control-flow perspective: the *Adaptive IPDD trace by trace* and *Adaptive IPDD windowing*. We extensively evaluated both approaches using two publicly available synthetic datasets containing 17 change patterns and different sizes and intervals between drifts. The accuracy results are evaluated using F-score and mean delay metrics, the former measuring the number of correct detections and the second indicating the accuracy of the reported change point in case of correct detection. Both approaches overcome VDD considering F-score and mean delay metric for dataset 1. *IPDD windowing* presented better F-score results than VDD and Apromore for dataset 2. The reason is the accentuated drop in the F-score values using window sizes larger than 200. We conclude that *Adaptive IPDD trace by trace* approach accuracy is affected by larger window sizes, i.e., values larger than the interval between drifts. Another promising result is that *IPDD* overcomes VDD's mean delay in both datasets. Furthermore, *Adaptive IPDD trace by trace* performed better mean delays than Apromore AWIN for dataset1, and better mean delays than Apromore FWIN in dataset 2.

Based on the performed experiments on synthetic datasets, *Adaptive IPDD windowing* is statistically equivalent to Apromore for drift detection but overcomes Apromore considering the mean delay. *Adaptive IPDD trace by trace* is statistically equivalent to Apromore for drift detection only in dataset 2 and overcomes Apromore AWIN's mean delay for dataset 1 and Apromore FWIN's mean delay for dataset 2. VDD performed the lowest F-score and mean delay values for both datasets.

Another important aspect is to reduce the detection sensitivity related to the parameter configuration. *Adaptive IPDD trace by trace* is more robust to small window sizes, decreasing accuracy with window sizes larger than 200 traces. On the other hand, the *windowing* approach shows a lower accuracy with window sizes of 25 and 50, another decrease after 225 (similar to the Apromore ProDrift results). However, the *trace by trace* approach cannot detect change patterns that apply a subtle change in the process, like the *cd* pattern.

We also evaluate the two *Adaptive IPDD trace by trace* in a real-life event log, showing that another contribution of the tool is to localize the drifts and reveal the changing process (addressed only by VDD). We could detect the two drifts reported by Apromore ProDrift and VDD, also characterizing the first drift with more changes than the one reported in ProDrift. *IPDD* also detects another drift, which was explained by the differences between the models. The characterization method of *IPDD* provides two metrics, indicating the similarity between nodes and edges between the different versions of the process. The VDD user interface localizes the changes by the derived cluster instead of indicating the change in an imperative model, e.g., process graph. Revealing the changing process in *IPDD* allows the user to check the process versions over time. However, *IPDD* only detects sudden drifts, while VDD can detect the four types of drifts, and Apromore ProDrift can detect sudden and gradual drifts. *Adaptive IPDD windowing* does not report interesting results using the real-life event log. Thus, *Adaptive IPDD trace by trace* is more suitable for real-life event logs based on the performed experiments. However, it is important to highlight that *Adaptive IPDD trace by trace* approach lows accuracy when applying larger window sizes, which was confirmed by the experiments performed on the synthetic datasets. Another drawback is that the processing time increases if the process model is complex, because of the fitness and precision calculation.

APPENDIX 5.A – DETAILS OF DATASET 1[39]

This appendix describes the event logs from the original dataset, which do not follow the description considering the log's size, interval between drifts, and change pattern (Table 5.4).

[39] Original dataset from (Abderrahmane Maaradji et al., 2015)

Table 5.4. Details of dataset 1.

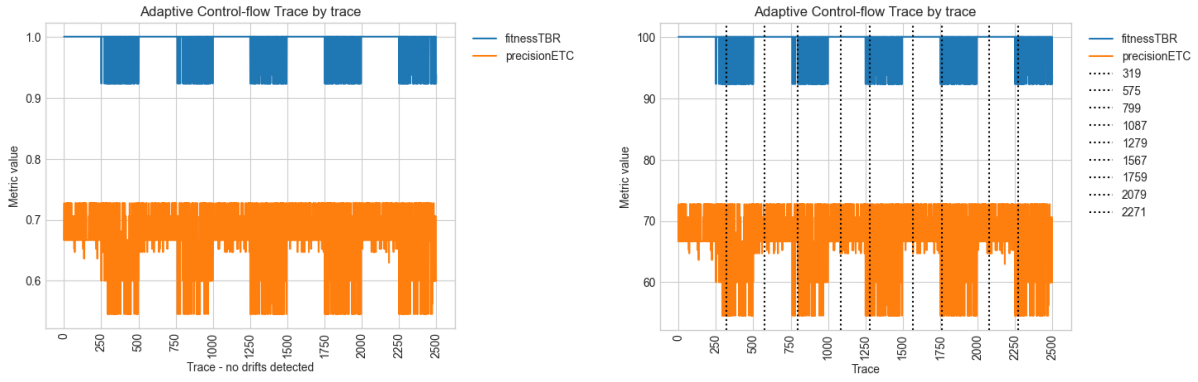
Event log	Problem Description	Solution
cb10k.xes	Contain only one drift after 5000 traces.	New event log created using simulation.
cd2.5k.xes	Contain 9 drifts after 250 traces; however, the altered model is completely different from the base model (also different from the BPMN specification).	New event log created using simulation.
cd7.5k.xes	Contain 9 drifts after 750 traces; however, the altered model is completely different from the base model (also different from the BPMN specification).	New event log created using simulation.
cm2.5k.xes	Contain 9 drifts after 250 traces; however, the altered model is the same from *cb* instead of the defined in the BPM specification.	New event log created using simulation.
cm7.5k.xes	Contain 9 drifts after 750 traces; however, the altered model is the same from *cb* instead of the defined in the BPM specification.	New event log created using simulation.
lp2.5k.xes	Contain 5000 traces with a drift after 500 traces.	Renamed to lp5k.xes.
lp5k.xes	Contain 10000 traces with a drift after 1000 traces.	Renamed to lp10k.xes.
lp7.5k.xes	Contain 15000 traces with drifts not following the specification; drifts at: 1000; 3500; 4000; 6500; 7000; 9500; 10000; 12500; 13000	New event log created using simulation.
lp10k.xes	Contain 15000 traces with drifts not following the specification; drifts at: 1000; 3500; 4000; 6500; 7000; 9500; 10000; 12500; 13000	New event log created using simulation.
re2.5k.xes	Contain 5000 traces with a drift after 500 traces.	Renamed to re5k.xes.
re5k.xes	Contain 10000 traces with a drift after 1000 traces.	Renamed to re10k.xes
re7.5k.xes	Contain 15000 traces with drifts not following the specification drifts at: 1000; 2000; 2500; 3500; 4000; 5000; 5500; 6500; 7000; 8000; 8500; 9500; 10000; 11000; 11500; 12500; 13000	New event log created using simulation.
re10k.xes	Contain 20000 traces with a drift after 2000 traces, and the log contains a loop (the same of the lp logs) instead of a re.	New event log created using simulation.

Also, as reported in Section 5.4, we changed the classification of the *sw* pattern to *R*, and renamed the pattern *RIO* to *RI*.

APPENDIX 5.B – ADWIN DETECTION BEHAVIOR USING A DIFFERENT SCALE

We applied the ADWIN change detector for reporting changes in the two temporal series collected: fitness and precision. Both metrics are calculated using PM4Py, which reports the fitness and precision values in an interval between 0 and 1. However, when evaluating the two implemented methods for drift detection (*trace by trace* and *windowing*), we identified that ADWIN's detection considers the scale of values inputted into the detector. Thus, using an interval between 0 and 1, we cannot detect the desired changes in the behavior of fitness or precision. To illustrate the problem, Figure 5.18 (a) plots the fitness and precision time series calculated based only on the first derived model (using the initial 100 traces) for the

scenario cb2.5k (dataset1). We plot the values using the interval between 0 and 1, as calculated by the PM4Py[40] framework. We can visually observe a considerable change in the fitness and precision behavior; however, the ADWIN change detector (Bifet & Gavaldà, 2007) does not report any changes. When we change the input value using a scale of 100, i.e., the fitness and precision values are multiplied by 100, we get a new outcome from ADWIN, indicated in Figure 5.18 (b).



(a) Original fitness and precision - no drift is detected by ADWIN.

(b) Fitness and precision multiplied by 100 - vertical lines indicate the drifts reported by ADWIN.

Figure 5.18. Temporal series for dataset 1, scenario cb2.5k, without updating the process model.

We can verify that both plots show the same curve using a different scale. The ADWIN change detector can only detect a change when we input the fitness and precision values multiplied by 100. We apply this multiplication factor in both strategies: *trace by trace* and *windowing*. The reported ADWIN behavior was unexpected; thus, we believe further investigation can be performed on the detector.

[40] https://pm4py.fit.fraunhofer.de/

6 PAPER #5: INTERACTIVE PROCESS DRIFT DETECTION FOR CONDITION-BASED MAINTENANCE USING PROCESS MINING TECHNIQUES

The paper addresses problems related to the difficulty of detecting anomalies in manufacturing equipment as soon as they start, and this issue is recurrent in condition-based maintenance (CBM) policies. CBM policies use the P-F curve to estimate the equipment's residual useful life (RUL). The point P of the P-F curve indicates the moment from which a given anomaly that is in progress can be detected by analyzing the monitored parameters. Then, we can consider the actions performed before point P as proactive, close to the exact moment when the anomaly started. Therefore, we present a new approach to concept drift detection in process models called Interactive Process Drift Detection (IPDD) to detect potential failures in advance. In addition to detecting changes in the patterns of the monitored parameters, this tool can also detect variations in the process pattern through process mining techniques, identifying changes in the sequencing of activities and events. Tests on synthetic databases and a case study with a real-life dataset were conducted to validate the method and prove its effectiveness by numerical evaluation.

Keywords: Condition-based maintenance, P-F curve, Interactive process drift detection, Proactive maintenance, Process mining.

6.1 INTRODUCTION

Industries are constantly seeking to improve their manufacturing process by balancing costs, quality, and delivery times (Y. Cao et al., 2012). For these goals to be achieved, the machinery must work in perfect conditions or, at least, in the best possible conditions (Ruschel et al., 2021). Maintaining equipment performance and availability indicators at high levels requires adequate maintenance policies (Ruschel et al., 2020); and among them, the Condition-Based Maintenance (CBM) stands out, which requires constant monitoring of the equipment conditions, using, for this, various information to calculate indicators, such as oil and vibration analysis, temperature, noise and velocity (Bousdekis et al., 2015; Mehta et al., 2015; Tang et al., 2015). One of CBM's premises is that it is possible to act proactively, performing maintenance actions in advance (early actions) when the reading of some data indicates a malfunction of a certain equipment (Bousdekis et al., 2015).

Further, (Bengtsson & Lundström, 2018) point to the need to combine basic concepts and traditional maintenance policies with emerging technological elements of Industry 4.0, e.g., predictive maintenance, cyber-physical systems, connectivity through the internet of things, self-management of assets, and big data. For the authors, considering only the tools conceived with the advancement of technology without looking at "the old", is not enough to obtain maximum effectiveness. Therefore, the combination of these elements is essential.

Furthermore, to correctly read and analyze the data, there is a need for a robust and reliable technological structure to collect and store information and well-defined frameworks and analysis procedures to make decisions coherently and effectively (Ruschel et al., 2017). Therefore, when considering this need and applying the CBM policy, we need to list relevant fundamental tools that can provide the necessary support for the proposition of methods. According to (Bousdekis et al., 2015), one of the most important principles of CBM is the P-F curve, which is used to estimate the remaining useful life (RUL) of the equipment and/or its components. P-F curve shows how and when a malfunction starts and, over time, moves to a stage at which it can be detected as a potential failure (point *P*); however, if the anomaly is not detected or no maintenance action is taken, it will progress at an accelerated rate to a functional failure (point *F*).

In an ideal CBM policy scenario, maintenance should occur before point *P* (at the exact moment the anomaly started), thus ensuring maximum effectiveness of this action and making it possible to keep equipment availability at the best possible level (Prajapati & Ganesan, 2013). Very early maintenance actions, intuitively, may seem beneficial, but the cost tends to be higher if performed long before the failure starts and late actions (Bousdekis et al., 2015). In addition, it is not possible to predict the onset of potential failure (point *P*) with 100% accuracy (Bousdekis et al., 2020), so the effort of most techniques and methods proposed for the CBM policy is to detect the point *P* as soon as possible so that maintenance actions can be performed soon after.

Several ways and methods can assist in analyzing shop floor information for decision-making in maintenance and manufacturing, and one of them is process mining (Choueiri et al., 2020; Choueiri & Portela Santos, 2021). We can analyze the process behavior from an event log and check for changes or anomalies that could indicate the imminence of a failure (Ruschel et al., 2021). In this context, this paper presents a method based on concept drift detection (Dries & Rückert, 2009), with an extension of the Interactive Process Drift Detection (IPDD) Framework (Sato, Barddal, et al., 2021). Our method uses process mining techniques applied to process data (event log), performing analysis and signaling to the manager or maintainer any change or variation in the monitored parameters of the equipment. Further, the IPDD framework also uses process mining techniques to identify changes in the sequencing of activities and events present in the manufacturing process (control-flow perspective) and monitor the operating cycle times and process activities duration (time perspective) also signaling if there are significant changes.

Thus, the framework proposed in this paper contributes to the insertion of these two elements (control-flow and time perspectives) that can also indicate the presence of

anomalies in the equipment (P-F curve), increasing the amount of information available for maintenance decision-making in the CBM policy.

The present paper is structured as follows: Section 6.2, we present more in-depth concepts of CBM policy, P-F curve, concept drift, and process mining, as well as previous proposals from the literature; in Section 6.3, we present the IPDD framework for its application and a synthetic datasets validation; Section 6.4 presents a case study with a real-life dataset from a manufacturing process and an evaluation of the results; Section 6.5 concludes the paper, addressing contributions, limitations, and prospects.

6.2 BACKGROUND

In order to introduce the IPDD as a support tool for the maintenance actions of the CBM policy, initially, we need to understand the evolution and structure of the concept drift detection and of the CBM itself, gathering information from the literature about its concepts, as well as the previously proposed methods.

6.2.1 MAINTENANCE STRATEGIES AND CBM POLICIES

As stated by (Bousdekis et al., 2015), based on studies carried out by (Jardine et al., 2006), there is still no absolute agreement in the literature regarding the types of maintenance, their policies, and strategies. The difficulty of a definitive classification that is accepted by the entire scientific community due to the high variation perceived in how these policies are applied, where there is not a well-defined integration between different techniques and strategies that should be specific to each one of them (Huynh et al., 2015; Zhou et al., 2015). Further, (Ruschel et al., 2017) conducted an extensive literature review on industrial maintenance decision-making, pointing out a certain deficiency in the proposed methods where, in general, there are still difficulties in optimizing maintenance results without negatively and significantly impacting the performance of the production process. In this context, four distinct types of maintenance can be listed based on these works:

- Corrective Maintenance (CM): performed after the occurrence and detection of a functional failure in the equipment.
- Preventive Maintenance (PM): normally, this type of maintenance is time-based, in which inspections are pre-scheduled, considering regular intervals (time windows) calculated through prior analysis of the historical equipment's behavior.
- Predictive Maintenance (PdM): as well as PM, it performs prior analysis of historical data on equipment's behavior, as well as real-time analysis to predict future behavior, enabling early decision-making based on forecasts. Some techniques of this type of maintenance are combined with those present in the CBM mentioned below.

- Condition-Based Maintenance (CBM): maintenance actions are triggered according to the current and future state of the equipment, based on the monitored parameters data, e.g., temperature, vibration, cycle time, and noises.

According to (Jardine et al., 2006), Preventive Maintenance (PM) is an evolution of Corrective Maintenance (CM), widely used in the industry. However, within these same industries, other authors (Riccardo Accorsi et al., 2017; Do et al., 2015; Shin & Jun, 2015; Tang et al., 2015; Tian et al., 2012) also point to significant growth in Predictive Maintenance (PdM) and, mainly, Condition-Based Maintenance (CBM). Further, the correct application of techniques that guide this policy is also necessary (Gulledge et al., 2010; Peng et al., 2010), and the P-F curve concepts are typically used as support tools (Bousdekis et al., 2015, 2020; Prajapati & Ganesan, 2013), considering the possibility of carrying out diagnoses and prognoses about the monitored state of the equipment, thus allowing proactive maintenance actions. However, maintenance proactiveness is not something new addressed in CBM policies (Jardine et al., 2006). (Bousdekis et al., 2015) developed a framework for proactive CBM decision making after conducting an extensive literature review. The authors identified possibilities for proactive online recommendations, considering the use of real-time data extracted from the machinery through sensors. The main contributions were structuring the CBM framework and concepts into two components: information space and decision space, and defining different tools and methods for each one. However, for condition-based maintenance policies, these components need to be well-defined.

According to (Voisin et al., 2010), condition-based maintenance (CBM) has three main steps: diagnosis, prognosis, and decision support. Diagnosis occurs during equipment condition monitoring and failure detection, while the prognosis is performed based on failure mode information and RUL predictions (Jardine et al., 2006; Peng et al., 2010). Figure 6.1 presents in a simplified way the CBM structure presented by (Voisin et al., 2010).

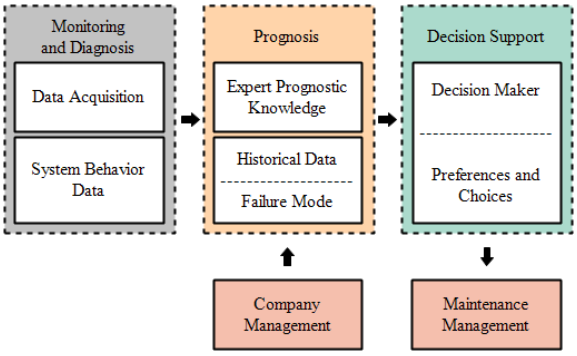


Figure 6.1. Diagnosis, prognosis, and decision support in CBM. Adapted from (Voisin et al., 2010).

CBM structure requires supporting technologies for data acquisition of monitored parameters and adequate tools for their correct implementation (Riccardo Accorsi et al., 2017). According to (Peng et al., 2010), there is a relationship between equipment reliability and maintenance cost, as shown in Figure 6.2. In addition to maintenance costs being related to reliability, there is also a relationship with the equipment availability (Ruschel et al., 2020), as the P-F curve, shown in Figure 6.3, is an essential principle of CBM, used to calculate and estimate the RUL (Bousdekis et al., 2015, 2020; Prajapati & Ganesan, 2013).

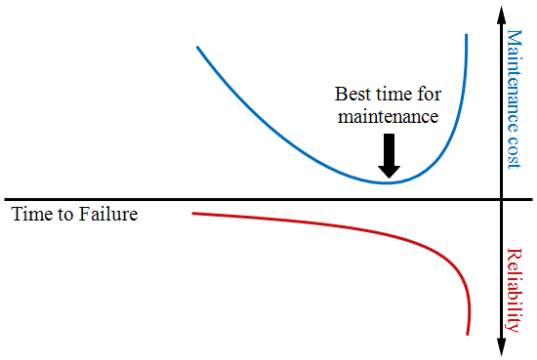


Figure 6.2. Reliability and maintenance cost relationship. Adapted from (Peng et al., 2010).

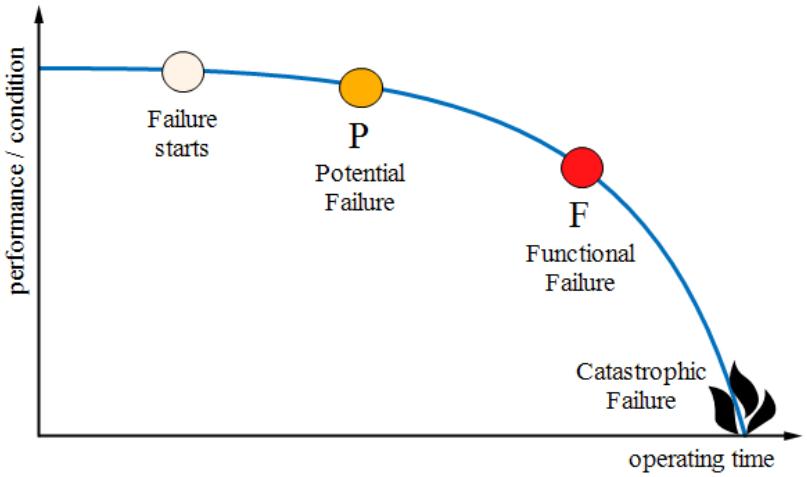


Figure 6.3. The P-F curve.

According to (Bousdekis et al., 2015; Prajapati & Ganesan, 2013), the main elements of the P-F curve are:

- The point where the failure starts — this event is not detected at the exact moment it occurs since it takes time for the monitored parameters to undergo sufficient modifications so that such detection can be performed.

- Point P — is the point where, after a certain time has elapsed after the failures start, the condition monitoring tools are able to detect the anomaly indicated by the monitored parameters variation.
- Point F — when no maintenance action is taken beforehand, point F determines the occurrence of a functional failure, forcing immediate maintenance intervention to avoid a catastrophic failure.

Point P can have different positions along the operating timeline and can be detected soon after the onset of failure or at later periods. This detection moment depends exclusively on the quality of the monitoring and detection methods and tools used, the behavior of the equipment, and the failure mode, presenting milder anomalies (increasing the time to detection) or more critical (allowing a faster detection) (Veldman et al., 2011). Further, (Bousdekis et al., 2015, 2020) consider the functional failure (point F) as the last stage of the P-F curve evolution, not addressing the catastrophic failure (J. Liu et al., 2018) shown in Figure 3. As we are interested in proactive maintenance actions (before point P or immediately after it), such a difference in the P-F curve interpretation is irrelevant since it does not impact the proposal of this paper.

6.2.2 PROCESS MINING AND INDUSTRIAL MAINTENANCE

Process mining is a research area between data mining and business process management to turn event data into insights and actions by combining traditional model-based process analysis and data-centric analysis techniques (van der Aalst, 2016). The critical assets of process mining tasks are the event data and the process models. As shown in Table 6.1, the event data is usually recorded in an event log, containing at least an identifier of the case (i.e., a unique process instance), the activity performed, and the timestamp, but it can contain more information about the events recorded for the business process. The process model indicates the allowed sequence of activities that should be performed for each process instance. The three main types of process mining are discovery, conformance, and enhancement. In discovery, we can derive a process model from the event data containing the historical records of business process events. We can compare the event data and a process model (discovered or designed) to identify discrepancies or deviations in the conformance. Moreover, the process model is extended in the enhancement by adding different perspectives from the event data, e.g., performance or resources (van der Aalst, 2016).

Table 6.1. A sample of an event log.

Case (Instance)	Activity or Event	Timestamp	
		Start	**Complete**
1	A	01/01/2022 08:05 am	01/01/2022 08:07 am
1	B	01/01/2022 08:07 am	01/01/2022 08:10 am
1	C	01/01/2022 08:10 am	01/01/2022 08:13 am
1	D	01/01/2022 08:13 am	01/01/2022 08:15 am
2	A	01/01/2022 08:15 am	01/01/2022 08:27 am
2	B	01/01/2022 08:27 am	01/01/2022 08:36 am
2	D	01/01/2022 08:36 am	01/01/2022 08:49 am
3	A	01/01/2022 08:49 am	01/01/2022 08:51 am

One of the most recurrent applications in several areas is the use of heuristic mining algorithms. We can obtain a causal net (dependency graph) that represents the process model with them. Figure 4 shows an example of a dependency graph, where: rectangles A and B are process activities or events; i_s and i_f represent the number of cases started and finished, respectively; $f_a(A)$ and $f_a(B)$ indicate the number of times activities A and B occurred (absolute frequency); the directed arc $|A >_L B|$ indicates the number of times activity A was followed directly by B in the event log L; and $|B >_L B|$ indicates how many times activity B was followed by itself (Rozinat & van der Aalst, 2008). From this information, it is possible to extract perspectives of process performance, analyzing task time in the activity's statistics and queues and bottlenecks in the directed arcs.

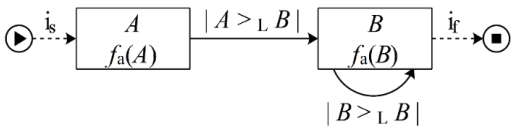


Figure 6.4. A dependency graph example.

Although process mining concepts are broad and complex (van Der Aalst et al., 2010), applications of these techniques are widespread in manufacturing but still little applied in industrial maintenance (Ruschel et al., 2020). One of these applications can be found in (Ruschel et al., 2020). The authors present process mining techniques integrated with prediction and probabilistic models developed in Bayesian networks to determine preventive maintenance intervals. The authors developed two functions: one for maintenance costs; and another for equipment availability. The proposed method calculates the best time for maintenance that provides the lowest cost and the best time for greater availability. The decision maker can choose between one interval or another; or, still, an intermediate interval that allows balancing the two criteria.

Further, (Ruschel et al., 2021) uses methods to create Bayesian networks from event log mining developed by (Kurscheidt Netto et al., 2015; Kurscheidt et al., 2015). The

probabilistic model developed by the authors makes it possible to estimate the completion time of process instances, calculating the probability of activities and failure events to occur. Thus, it is also possible to analyze the process performance.

6.2.3 Concept Drift in Process Mining

Most techniques for process mining still consider that the process model does not change when being analyzed. However, this assumption does not reflect the reality of the business processes, which are usually in constant change for distinct reasons, e.g., adapting to new regulations and improving performance. The concept drift indicates a change in the process when being analyzed (van der Aalst et al., 2011). The process models can represent different business process perspectives, e.g., control-flow and time; concept drift may also occur in all perspectives (Sato et al., 2022). Detecting and understanding the concept drifts in process mining can help better understand the business process dynamics and may help business managers proactively act. The four most common types of concept drift are listed below and, according to (J. Lu et al., 2019):

- Sudden drift: when a change occurs in a short period of time.
- Gradual drift: A new pattern gradually takes the place of an older one over time.
- Recurring drift: an old pattern may reoccur after a certain time.
- Incremental drift: an old pattern changes to a new pattern over time.

Figure 6.5 presents these four scenarios for the process changes according to (Sato et al., 2022). The y-axis indicates different versions of the process, and the x-axis indicates the time. Figure 6.6 presents the same scenarios from a time perspective. The y-axis indicates different durations of the process activities, and the x-axis indicates the time.

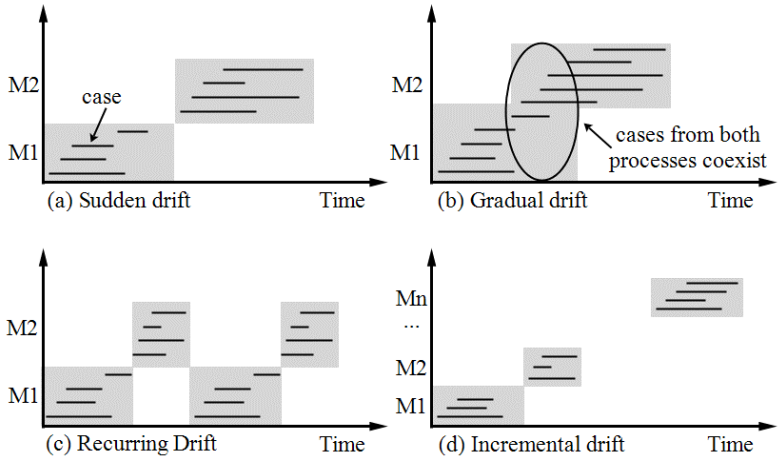


Figure 6.5. Different types of concept drifts in processes from a workflow perspective. Adapted from (Sato et al., 2022).

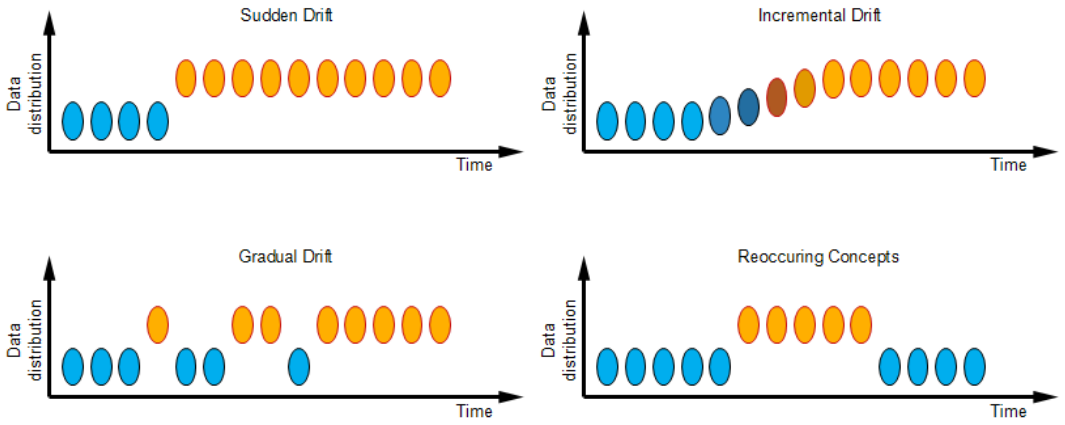


Figure 6.6. Different types of concept drifts in processes from a time perspective. Adapted from (Sato et al., 2022).

Authors proposed distinct approaches for concept drift detection in processes to balance good detection accuracy with few false alarms. However, the accuracy of the detections is sensitive to the parameter configuration, and defining a good value for the request parameters is still a challenge (Sato et al., 2022). Most concept drift detection approaches apply a windowing strategy to split the event log into sub-logs to detect the drift in the control-flow perspective, and the definition of the window size critically affects the accuracy of the detection. Besides this dependency of the parameter configuration, fewer methods have been proposed for dealing with concept drifts in the time perspective.

(Brockhoff et al., 2020) calculated the Earth Mover's Distance (EMD) using trace descriptors derived from sub-logs extracted using a sliding fixed-size window. The user can then inspect the EMD values calculated for a window size to identify potential drifts. This approach can detect drifts in the time perspective (service and sojourn times); however, the window size parameter is critical to the accuracy of the detected drifts. Furthermore, the author only experimentally evaluated the approach using one synthetic log.

CDESF is another concept drift detection approach for the time perspective (Mora et al., 2020; Tavares et al., 2019). It derives the graph-distance trace (GDtrace) and graph-distance time (GDtime) by comparing the new trace (obtained from the arrived event) and the process graph (PMG) normalized, representing the current behavior – the method updates it after the end of the time horizon (TH) defined by a parameter, which is a sliding window. Then, using DenStrean, it clusters the feature vector [GDtrace, GDtime] and outputs a drift plot showing the position of the detected drifts in case new core micro-clusters are detected. If a new behavior is dense enough to form a new core micro-cluster, it indicates a drift. The definition of the TH parameter is still a challenge because it affects the accuracy of the detected drifts; a larger TH will miss drifts that may occur inside the interval, and in real-world event logs, the interval between drifts probably will not be fixed.

The Tesseract method (Richter & Seidl, 2017), is an online temporal drift detection. The main advantage of this method is that it calculates a significance score based on the calculated times (time interval between activities), indicating how far a new observation is from the mean value. The two decay factors applied in the method allow the user to define how much the method relies on detecting significant drift alerts but increases the detection delay and makes it hard to detect short-term anomalies. However, the implementation of the method only detected drifts in the time interval between activities.

There is still a significant gap regarding proposals that integrate process mining, concept drift detection, and industrial maintenance in the literature. However, the potential for integrating these tools in industrial maintenance becomes clear when we analyze the proposals and results presented by the authors. Considering these methods allow identifying: (i) performance loss, (ii) variations in production cycle times, and (iii) changes in the sequencing of process activities; it is plausible to relate these indicators to anomalies in the machinery, indicating a malfunction and the consequent onset of a failure. Therefore, this is the premise of our proposal. Further, we believe a method for concept drift detection using information about the time perspective of the production process, e.g., sojourn activity time, can provide valuable information for the CBM, considering that different variations on the time perspective of the process may be related to the need of maintenance tasks.

6.2.4 Concept Drift and Manufacturing Processes

Concept drift detection has also been applied in the data mining area, usually when a supervised learning model loses accuracy because the relation between the input data and the target variable changes over time (Gama et al., 2014). The main difference from the detection in the context of processes is that the target variable is more straightforward than categorical or continuous values in a vector (Sato et al., 2022). In many cases, changes are difficult to detect and measure; therefore, many studies have been carried out with proposals for different and varied detection methods (Sun et al., 2020). (J. Lu et al., 2019) carried out a literature review to help understand these techniques, considering three main components: detection, understanding, and adaptation.

Even though concept drift detection based on data variables has been applied in manufacturing processes, few proposals have been aimed at industrial maintenance (Zenisek et al., 2019), especially in CBM. As much as some companies have the necessary technology to monitor equipment conditions and store data properly, one of the biggest problems in CBM raised by (C. C. Lin et al., 2019) is the imbalance present in these data since the data that indicate potential failures are a minority within the huge datasets generated, making anomaly detection a non-trivial task. The authors present a proposal for

joint learning between concept drift and imbalance data, creating new classifiers using the SMOTE method and improving detection with the LFR (Linear Four Rates) method.

(Jayaratne et al., 2021) present a method for continuous drift detection in industrial cyber-physical systems (CPS), using closed-loop incremental machine learning. The algorithm presented is based on the unsupervised IKASL learning approach and was applied to a synthetic database obtained with the Streaming Ensemble Algorithm (SEA). Afterward, two experiments were conducted on two industrial CPS applications. However, drift detection was conducted only on activity monitoring and energy consumption data streams. Further, the authors point out future opportunities to be able to determine causalities of the concept drift and, in this case, it may also be useful in maintenance. The authors in (Altendeitering & Dübler, 2020) also present learning techniques for drift detection to apply in Industry 4.0 scenarios. However, the authors also claim that there are still difficulties given the small number of samples or resources for effective solutions.

In (Zenisek et al., 2019), the authors present a drift prediction method based on Machine Learning (ML) for Predictive Maintenance (PdM). The author uses Linear Regression (LR), Random Forest Regression (RF), and Symbolic Regression (SR) algorithms after a series of preprocessing steps that include filters and the consolidation of data from different monitoring sources. A case study was carried out on radial fans, monitoring vibration, speed, temperature, pressure, etc. However, the method's effectiveness cannot be fully verified due to the absence of adequate measurements of the equipment deterioration progression. In (Jimenez-Cortadi et al., 2020), the authors present an improved method, structuring all the steps (acquisition, preprocessing, processing, analysis, and decision making) to transform a preventive maintenance policy into predictive maintenance. However, as much as the process includes drift detection, RUL estimates, and diagnostics and prognostics for decision making, it was applied to a single monitored parameter: spindle load. The authors do not address the concepts of proactive maintenance actions, P-F curve, or strategies for CBM policies.

6.3 PROPOSAL

This section presents the IPDD concepts used in our approach, the proposed framework, and the details for model validation and application in the CBM. The shortcomings perceived in the literature and pointed out by the authors themselves are considered in this paper, as well as the lack of a more careful and well-defined integration between the elements listed above provides a relevant contribution from the use of the concept drift detection within CBM policies.

6.3.1 INTERACTIVE PROCESS DRIFT DETECTION FOR DETECTING TIME DRIFTS

The Interactive Process Drift Detection (IPDD) is a generic framework for process drift detection (Sato, Barddal, et al., 2021). According to the implementation, the framework can detect process drifts in the different perspectives of the process models. This paper developed a new instantiation of the framework for detecting time drifts in the activity cycle-time, based on an adaptive windowing approach. We also combined the detection of time drifts with control-flow drifts, which indicate a change in the structure of the process model in our implementation, activities, or paths added or removed.

• **Windowing strategy.** We collected each activity's sojourn time from the traces (sorted based on the first activity's timestamp). For each activity, we derived a time series, which is a sequence of ordered data points $T_{activity} =< t_1, t_2, \dots, t_n >$, containing all the sojourn time values in seconds for the specific activity, extracted from each trace reporting the activity – using the PM4Py framework[41] (Berti et al., 2019). For every time series $T_{activity}$, we applied the ADWIN change detector, which detects concept drift from data sequences that may vary with time, providing rigorous guarantees of performance as bounds on the rates of false positives and false negatives (Bifet & Gavaldà, 2007). Then, we split the event log based on the ADWIN detected drifts for each derived $T_{activity}$ providing distinct window cuts for each activity reporting a temporal drift. ADWIN receives the $\delta \in (0,1)$ parameter as input, which affects the sensitivity of the detection, i.e., a higher δ will increase the rate of false positives (Bifet & Gavaldà, 2007). We used the *scikit-multiflow*[42] implementation of the ADWIN detector (Montiel et al., 2018) to obtain the change points.

• **Process discovery.** We applied the PM4Py framework (Berti et al., 2019) to discover the directly-follows graph (DFG) with the frequencies of activities and paths.

• **Model-to-model comparison.** We calculated the nodes and edges similarity between the models derived from each window (Sato, Barddal, et al., 2021). If one of the metrics indicates a drift, we also reported it as drift in the control-flow perspective.

• **Evaluation.** We calculated the F-Score metric as suggested in (Sato et al., 2022) for evaluating the detected drifts for the non-stationary synthetic event logs. We calculated the false positive rate (FPR) for evaluating the false alarms for the stationary synthetic event log.

---

[41] https://pm4py.fit.fraunhofer.de/

[42] https://scikit-multiflow.github.io/

The implementation of the IPDD with the option of detecting temporal drifts in the activity's sojourn time is available for download[43].

6.3.2 IPDD APPLIED TO INDUSTRIAL MAINTENANCE

The present work considers the concepts of CBM and uses the P-F curve as a support tool to define the proactive maintenance actions according to the IPDD drift detections. In this way, a new drift detection will provide information that can be compared with the historical data of previous detections, allowing to assess whether or not the currently detected drift is related to the beginning of a potential failure. Furthermore, we make it clear that the role of the expert manager's knowledge will always be essential in the evaluation of results for decision-making.

We present the P-F curve elements in more detail in Figure 6.7. Any maintenance action taken after point F will be within the reactive domain interval; time-based and condition-based actions between points P and F (P-F interval) correspond to the corrective domain but are already part of preventive and predictive maintenance policies (Bousdekis et al., 2020). Finally, even needing prediction methods, actions taken before point P will be in the proactive domain, allowing the execution of maintenance closer to the moment considered ideal when a failure starts (Prajapati & Ganesan, 2013).

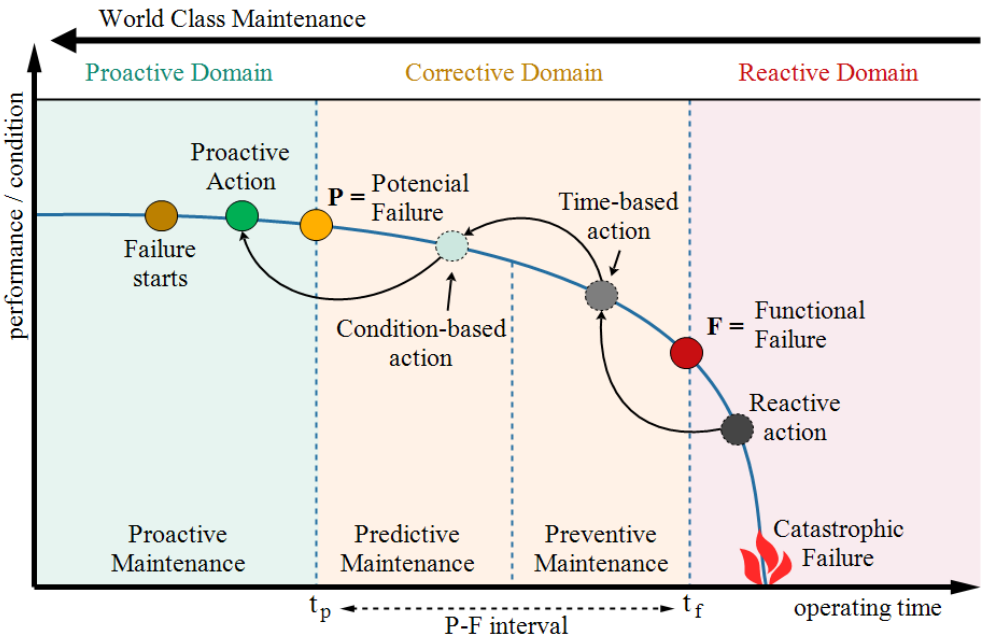


Figure 6.7. Maintenance strategies linked to the P-F curve concepts.

[43] https://github.com/denisesato/InteractiveProcessDriftDetectionFW

Figure 6.8 presents all the steps of the framework proposed in this paper. The IPDD is responsible for processing data from different sources related to the production system, e.g., event logs from the manufacturing process, temperature, and vibration data. Both manufacturing process data (event log) and equipment condition monitoring parameters, e.g., vibration, temperature, and noise level, can be input into the tool, identifying existing patterns in these data and pointing out any drift that may occur.

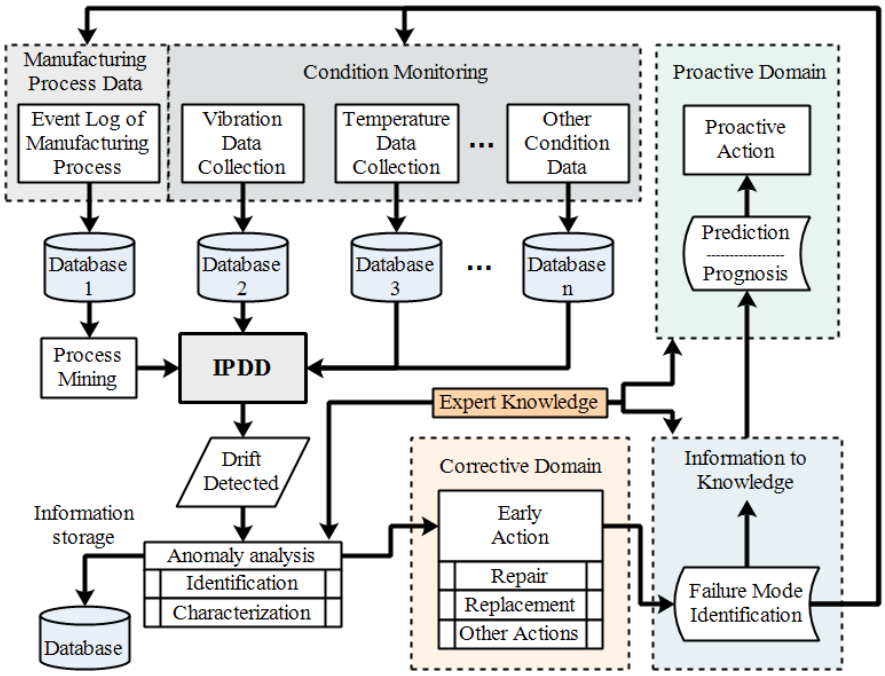


Figure 6.8. Proposal for the application of IPDD in Condition-Based Maintenance.

So that we can make any decision from this point on, we need to identify which anomaly was detected and store the information in a database. Expert judgment is essential at this stage of application, given that current mathematical models still suffer from the difficulty of identifying the cause of a particular anomaly in a clear, objective, and assertive manner. This information is confirmed in the literature (Bennane & Yacout, 2009), where many cited authors claim that almost identical variations of a given parameter may indicate different potential failures in the equipment. With information about the anomaly related to the detected drift, it is possible to carry out early maintenance actions within the corrective domain (time-based preventive maintenance rescheduling or condition-based maintenance). Further, we may collect information related to the equipment's failure modes and compare it with the detected anomalies. This comparison is performed to verify the quality and hit rate related to the anomaly identification. From the drift detection and failures historical data, such information can be used with expert knowledge to obtain predictions and prognoses to promote proactive maintenance actions. The main idea is to refine the knowledge at each detection cycle, enabling safer maintenance decision making.

Considering that the manufacturing systems and the equipment that compose them have different behaviors, it is safe to say that the variations of the monitored parameters are also different. Therefore, the sensitivity of the IPDD model will be distinct for each monitored process or equipment. Thus, the models can be improved over time by adjusting the δ parameter, always aiming at reducing the false positive rate (FPR). An evaluation of the results based on historical data of drift detection, failure occurrences, and maintenance interventions can be carried out for adjustments and constant refinement of the model.

6.3.3 SYNTHETIC DATASET VALIDATION

We apply IPDD in different artificial manufacturing datasets containing drifts in the time perspective with different sizes and configurations. The drift occurred in the sojourn time (activity duration) of the activity “Machine Working”, and, in all cases, we simulated an increase in the activity’s sojourn time, indicating a possible problematic situation in the production line. Thus, we only report the time drifts detected in the “Machine Working” activity. Table 6.2 describes the configuration of the event logs; some configurations combine a period of stability (no increase in the sojourn time) followed by one or more periods where the time increases. The change point indicates where the drift starts, i.e., the trace index where the sojourn time of the activity starts increasing. The interval of values indicates the amplitude randomly selected for stationarity and the period of increasing sojourn time. We applied a linearly increasing function with a slope (“Increase function” column).

Table 6.2. Description of the synthetic event logs.

Dataset	**Characteristic**	**Size (traces)**	**Interval of values (in seconds)**	**Change points**	**Increase function**
ST	Stationary	250	60	-	-
DR	Drift without maintenance stop	500	1,000	349	+1% for each trace, after trace 349
DR_MS	Drift with maintenance stop	250	60	0;26; 100; 148; 215	+1% for each trace, with random return to original value
DR_MS_ST	Drift with maintenance stop combined with stationary periods	2,500	3	205; 858; 1246; 1,555; 2,006	Random stationary period; +1% for each trace after random triggering of the increment; random return to original value
TD	Temperature Data	2,500	3	351; 475; 575; 751; 805; 1,210; 1,212; 1,350; 1,520; 1,600; 1,997; 2,189	Random stationary period; +1% for each trace after random triggering of the increment; random return to original value

Table 6.3 shows the F-score – Equation 1 – and the False Positive Rate (FPR) – Equation 4 – using different values of δ parameter from the ADWIN detector. We can observe that we can tune the parameter based on the detected drifts and the false alarms. Figure 6.9 shows the activity's sojourn time (activity duration) for each event log, which the detected drifts (vertical red lines) for the selected configuration of δ parameter.

$$F - score = 2 \times \frac{precision \times recall}{precision + recall} \quad (1)$$

$$precision = \frac{TP}{TP + FP} \quad (2)$$

$$recall = \frac{TP}{TP + FN} \quad (3)$$

$$FPR = \frac{FP}{FP + TN} \quad (4)$$

The F-score (Equation 1) evaluates the accuracy of the detected drifts (dd) compared to the real ones (rd) using the harmonic mean between precision and recall. We considered a true positive (TP) when a detected drift is reported in the interval $[rd, rd + et]$, where et defines an error tolerance, as suggested in (Sato et al., 2022). A false positive (FP) is a detected drift not in the interval $[rd, rd + et]$, and a false negative FN is a real drift not considered for any TPs. In our experiments, we set the $et = 100$.

The FPR (Equation 4) indicates the probability that a false alarm will be raised. The FPR is important when the cost of incorrectly identifying a drift is high, and in our case, we apply it to confirm that increasing the δ parameter will also increase the number of false alarms. Therefore, there should be a trade-off between detecting the drift as soon as it occurred (by increasing δ) and the cost of generating too many false alarms. Even the ST dataset may report a drift when increasing the δ to 1. A true negative (TN) is any trace not informed as a real drift and not detected as a drift by IPDD.

Table 6.3. F-score and FPR for the synthetic datasets using different δ configurations.

Dataset	**F-score**				**FPR**			
	$\boldsymbol{\delta = 0.05}$	$\boldsymbol{\delta = 0.1}$	$\boldsymbol{\delta = 0.3}$	$\boldsymbol{\delta = 1}$	$\boldsymbol{\delta = 0.05}$	$\boldsymbol{\delta = 0.1}$	$\boldsymbol{\delta = 0.3}$	$\boldsymbol{\delta = 1}$
ST	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.004
DR	1.000	1.000	0.500	0.400	0.000	0.000	0.004	0.006
DR_MS	0.750	0.750	0.888	0.600	0.000	0.000	0.000	0.008
DR_MS_ST	0.417	0.417	0.370	0.303	0.006	0.006	0.007	0.010
TD	0.417	0.417	0.350	0.295	0.008	0.008	0.010	0.012

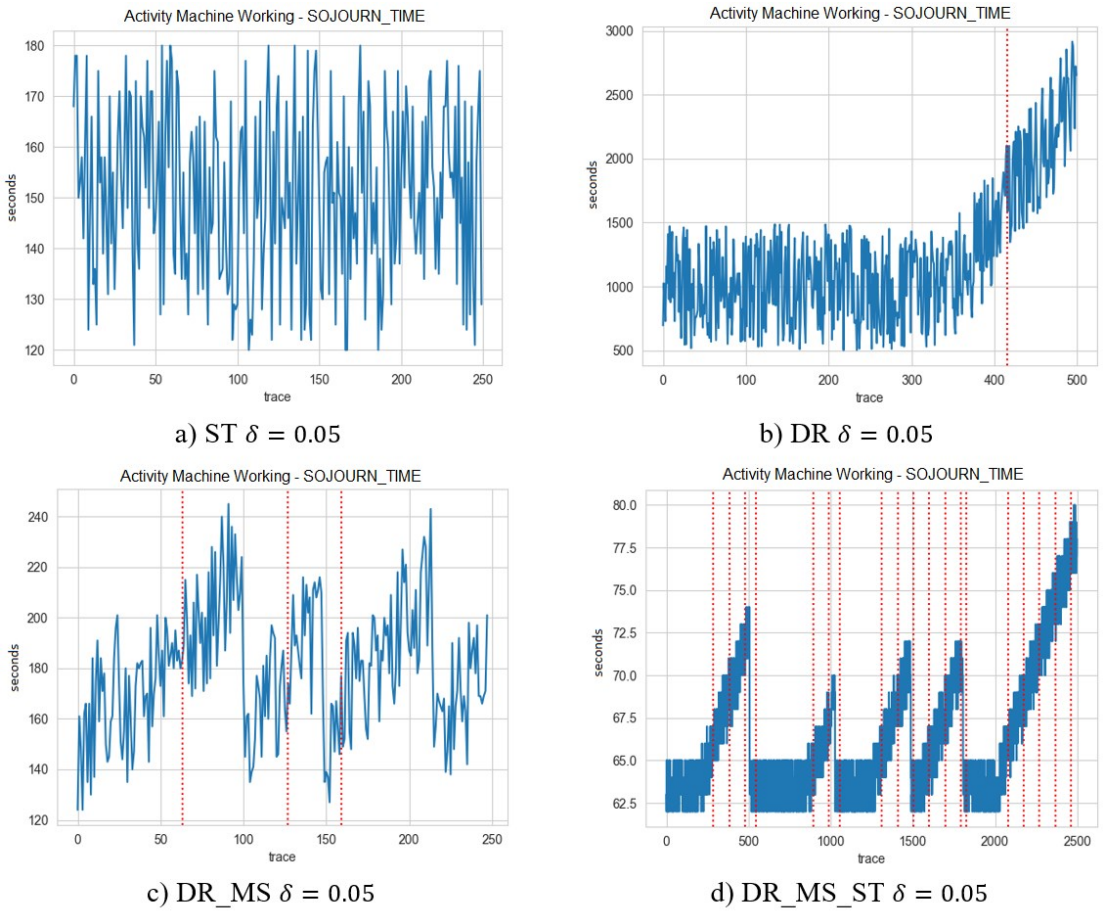


Figure 6.9. Change points detected for each selected scenario.

Figure 6.9 (a) has no change points, which was expected as it is a stationary dataset. The process represented in Figure 6.9 (b) has only one change point, which was identified by the IPDD as expected. Likewise, the tool identified the changes present in the process in Figure 6.9 (c). Although, when comparing Table 6.3 with Figure 6.9 (d), we can see that there are many change points detected for the DR_MS_ST scenario and the F-score is the lowest for all δ values (F-score = [0.417; 0.417; 0.370; 0.303]), as well the false positive rate is the highest one (FPR = [0.006; 0.006; 0.007; 0.010]). However, it is important to note that we consider the model validation step. In other words, there are still no actions (maintenance interventions) being performed after the detection of the first change point to stop the increase. Therefore, the values continue to be incremented, and, naturally, the IPDD continues to periodically detect new change points after the first detection in a given series. Thus, even if it seems counterintuitive, given the prior knowledge of the synthetic scenario, we can say that such behavior of the IPDD only demonstrates its drift detection potential.

We also perform the model validation based on detecting a temperature parameter change, shown in Figure 6.10. This synthetic dataset represents random increases in equipment temperature, and the return to normal values (from 49 to 52°C) is obtained after

simulated maintenance occurrences in the equipment. The vertical red lines indicate the detection of drift, and the model's functioning is similar to that applied to the duration of the process activities.

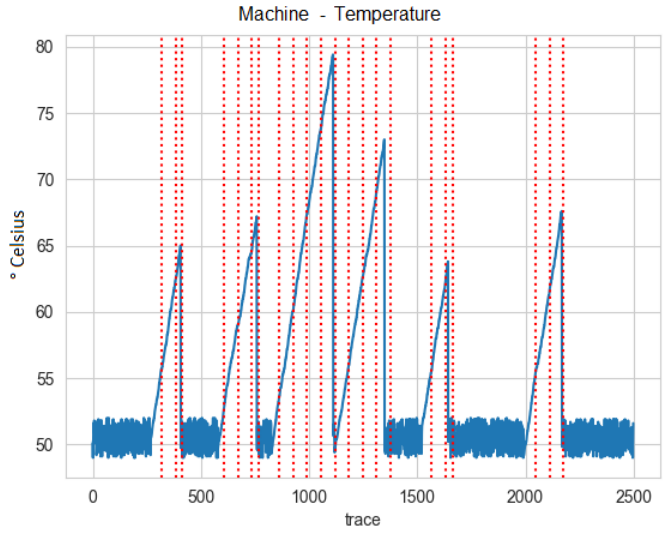


Figure 6.10. Drift detection in the temperature parameter (TD) with $\delta = 0.05$.

6.4 CASE STUDY

The IPDD was applied to a real-life dataset containing 29,998 records in 7,479 distinct cases (about one month of production) from a manufacturing process performed by a CNC lathe machine in one automobile industry in Brazil. In this process, each unit produced by the equipment corresponds to an instance (trace in the log). The event log contains the activity "Machine Working", which reports the time the machine effectively produces the product, and it is the aim of our study. This activity's mean sojourn time and the median sojourn time are 2m 26s. Table 6.4 shows the statistics of the dataset used, presenting the meaning of each term and the maximum, minimum, and mean duration, in seconds, of each activity or event. Figure 6.11 shows the number of occurrences of each activity or event.

Table 6.4. Real-life dataset statistics.

Meaning	**Max Duration (sec)**	**Min Duration (sec)**	**Mean duration (sec)**
Machine loading	43	25	34
Short stoppage	25	15	20
Machine working	171	126	146
Finished Part	-	-	-
Awaiting maintenance	6046	3482	5172
Equipment failure	-	-	-
Electrical maintenance	6328	6328	6328
Mechanical maintenance	5981	5341	5648
Preventive maintenance	2066	2046	2056
Readjustments	484	136	301

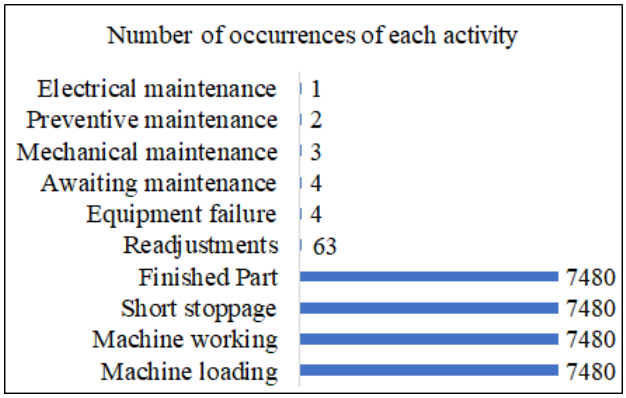


Figure 6.11. The number of occurrences of each activity.

As shown in Figure 6.12, the IPDD method detected and reported a process drift (vertical red lines) in the time duration of the activity “Machine Working” in the traces [543; 798; 1,150; 1,597; 2,077; 2,941; 3,581; 3,997; 4,445; 5,117; 5,276; 5,596; 5,820; 6,556; 6,971]. Note that drifts were also detected where there was a reduction in the duration of the activity “Machine Working”. This reduction in the production activity duration usually occurs after a maintenance activity, as shown in Figure 6.13. Therefore, as we are interested in detecting anomalies to carry out proactive maintenance actions, we will only consider drift detection during the increase in the cycle times since the reduction of activities duration and cycle times is precisely one of the expected improvements of this method. In this way, the descent drifts detected soon after the maintenance event will be disregarded in the traces [798; 1,597; 2,941; 4,445; 5,276; 5,820].

Figure 6.12. Time drift detection in the production activity for the real-life dataset with $\delta = 0.05$.

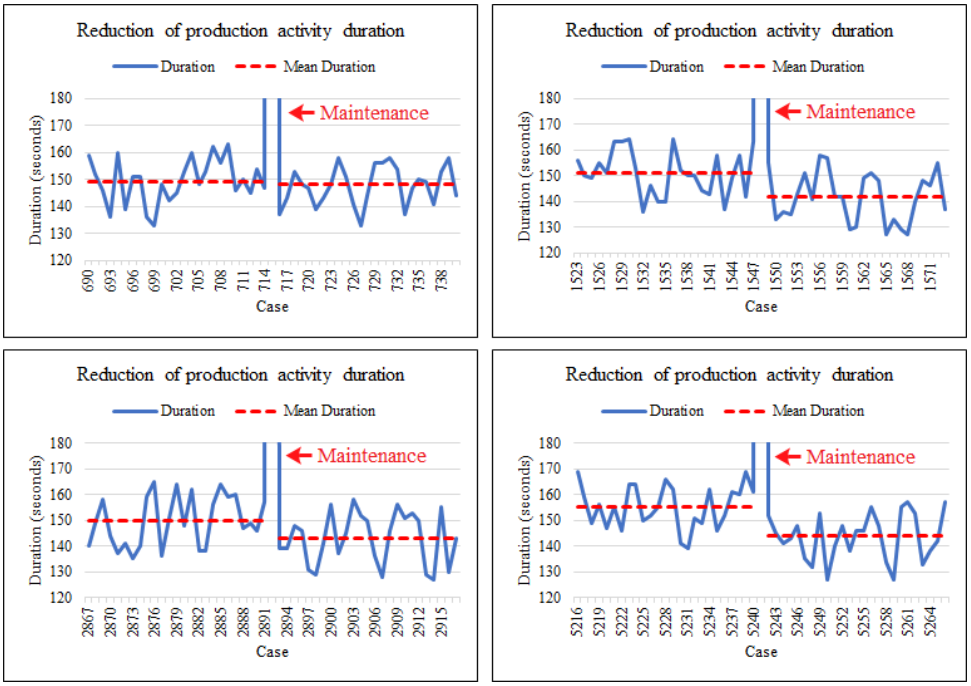


Figure 6.13. Reduction in the mean duration of production activity after maintenance occurrence.

To analyze the results, we performed the calculations listed below and shown in Figure 6.14.

- $\bar{M}_{m\to d}$: Mean duration of production activities performed between a maintenance event and the next drift detection.
- $\bar{M}_{d\to f}$: Mean duration of production activities performed between a drift detection and the next failure or maintenance event (whichever comes first).
- $L_P = \left(\bar{M}_{d\to f} - \bar{M}_{m\to d}\right) \times m$: Performance losses (according to the time unit used) will be considered as the difference between the two previous variables multiplied by the number of machine activity occurrences (m) between a drift detection and the next failure or maintenance event (whichever comes first), since the maintenance actions proposed by the IPDD method include interventions right after drift detections.
- $L_{Total} = \sum L_P$: The total of performance losses is the sum of all periods calculated for L_P used as a parameter to measure the efficiency of the IPDD.

The event log interval analyzed has 4 occurrences of equipment failure (cases [714; 1,545; 5,235; 6,846]), followed by 1 electrical maintenance (case [714]) and 3 mechanical maintenances (cases [1545; 5,235; 6,846]). In addition, it also has the occurrence of 2 time-based preventive maintenance (cases [2,887; 5,779]). For the analysis of the proposed method, we will pay attention to the drifts detected slightly before the 4 failures (cases [543;

1,150; 5,117; 6,556]) and the 2 preventive maintenance (cases [2,077; 5,596]). Table 6.5 details this information.

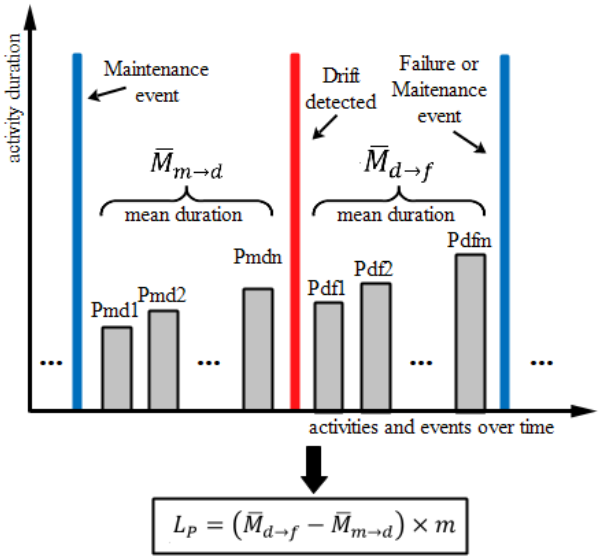


Figure 6.14. Results evaluation method for the IPDD case study.

Table 6.5. Statistics of equipment failures, maintenance occurrences and detected drifts.

Event	Case where the event occurred	Case where drift was detected
Equipment failure (electrical maintenance)	714	543
Equipment failure (mechanical maintenance)	1,545	1,150
Preventive maintenance	2,887	2,077
Equipment failure (mechanical maintenance)	5,235	5,117
Preventive maintenance	5,779	5,596
Equipment failure (mechanical maintenance)	6,846	6,556

Using process mining techniques, we identified the cases where failures and maintenance events occurred. Then, according to the evaluation method described in Figure 6.14 we calculated the mean durations between drifts and failures or maintenance. Additionally, we also calculate the amount (L_P) corresponding to increases in production times, from the case of drift detection to the case of a failure or preventive maintenance event. Table 6.6 presents these values and all cases referring to the respective intervals.

Table 6.6. Cases with equipment failures, maintenance occurrences, detected drifts, and performance losses.

Last maintenance event on case	Parts produced ($P_{m\to d}$)	$\bar{M}_{m\to d}$ (sec)	Drift detected on case	Parts produced ($P_{d\to f}$)	$\bar{M}_{d\to f}$ (sec)	Failure (F) or maintenance event (M) on case	L_P in minutes
-	543	143	543	171	149	714 (F)	17,1
714	436	146	1,150	396	149	1,545 (F)	19,8
1,545	532	143	2,077	810	149	2,887 (M)	81,0
2,887	2,230	148	5,117	118	154	5,235 (F)	11,8
5,235	362	144	5,596	183	148	5,779 (M)	12,2
5,779	777	143	6,556	290	147	6,846 (F)	19,3

The total sum of machine activity durations above the mean considered normal (between a maintenance event and a drift detection) was approximate 161.2 minutes ($L_P = 161.2\ secs$), or about 2.69 hours, within the 1-month interval considered in the event log (276 hours of total production). This corresponds to about 1.01% of the production time, as shown in Figure 6.15. Considering the individual percentages of each period (Figure 6.16), all of them exceeded 2% of losses due to a drop in performance, with three reaching close to 4%.

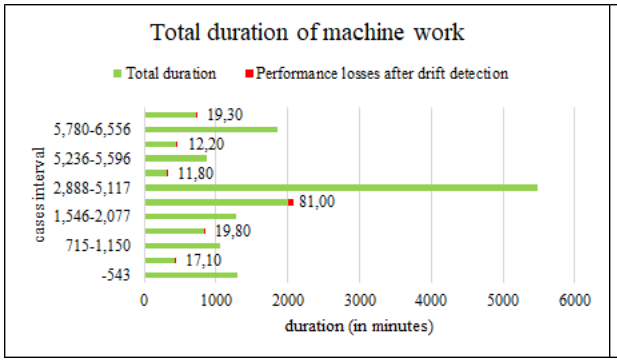


Figure 6.15. Total duration of machine work.

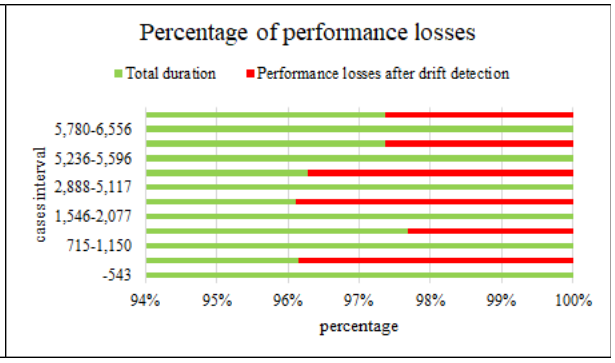


Figure 6.16. Percentage of performance losses.

Considering the performance improvements that the real-life dataset itself presents after the maintenance occurrences, it is safe to say that such improvements could still be achieved, even if the times when the maintenance took place were changed. Therefore, applying the information gathered in this case study to the proposal of this paper, the six maintenance interventions considered could be anticipated, as shown in Table 6.7, with estimates of significant improvements in the production performance.

Table 6.7. Suggested anticipations for maintenance occurrences in the case study.

Maintenance occurrence		Estimated improvement for the period		Total estimated improvement
Real date and time	**Suggested anticipation date**	**Minutes**	**Percentage**	
May 11 11:44 PM	May 11 02:01 PM	17.1	3.86%	0.10% (0.10%)
May 14 01:18 AM	May 13 02:38 AM	19.8	2.32%	0.12% (0.22%)
May 17 08:35 AM	May 15 09:44 AM	81.0	3.88%	0.49% (0.71%)
May 22 10:01 PM	May 22 03:09 PM	11.8	3.73%	0.07% (0.78%)
May 24 08:34 AM	May 23 09:35 PM	12.2	2.63%	0.07% (0.86%)
May 26 08:15 PM	May 26 03:56 AM	19.3	2.64%	0.15% (1.01%)

The drifts in the cases [3,581; 3.997; 6,971] were not addressed since the process has readjustment activities, which could be considered one autonomous maintenance, as they also influence, although more subtly, the machine performance. In this way, a more profound expert judgment would be necessary to evaluate the detected anomalies in these cases. However, the non-treatment of these drifts did not impact the results presented.

6.5 CONCLUSION

This paper presents a new approach to concept drift detection, called Interactive Process Drift Detection (IPDD), for application in condition-based maintenance policies. Elements of the P-F curve were used as support parameters so that we could suggest proactive maintenance actions based on drift detections. The proposed method was validated through its application in five different synthetic datasets, and a case study was conducted on a real-life dataset.

Several deficiencies related to the applicability of effective methods in CBM are pointed out in the literature. One of them is the scarcity of adequate real-life datasets, making it impossible to evaluate and validate the methods better. These gaps are also noticed when information on concept drift detection is sought, directly impacting the proposed models' results. Further, proposals that work with the integration of concept drift and CBM are still rare, making the challenge of proposing effective methods in this area even more difficult.

However, as much as we have faced all these difficulties mentioned, the IPDD proved to be very promising in assisting industrial maintenance decision-making. Three points can be highlighted in the contributions of this proposal:

- The models can be easily configured, changing the sensitivity so that the FPR is reduced and, thus, presenting reliable detections.
- The use of process mining techniques to detect process drift and changes in cycle times and machine activities duration allows these elements to also be used as

indicators of equipment condition, as long as it is possible to relate these drifts to potential failures.

- We can integrate different models using the IPDD, one for each parameter to be monitored, with its different sensitivity settings. Applying them in parallel provides robust monitoring of machinery conditions.

It is important to note that the suggested maintenance must occur immediately to achieve the estimated improvements. We are not considering the waiting times for it to occur. Further, the lack of adequate real-life datasets prevented us from conducting a more in-depth and detailed evaluation of this method. In addition, it is directly dependent on technologies not yet present in all companies, e.g., effective sensing systems for collecting data from machinery, reliable expert knowledge, skilled labor for data processing, and subsequent application of proposed methods. The windowing strategy and δ parameter also need to be rigorously tested during model validation, directly impacting drift detection. Another point we list as an opportunity for future work is the need to investigate prediction techniques and models that can be integrated with the results obtained from the IPDD, enabling more effective and reliable proactive maintenance actions. Finally, the application of this method in CBM can also be very promising if we consider the possibility of creating a multi-criteria decision-making model (MCDM) or a framework for maintenance decision-making that uses the IPDD information.

6.6 SECTION NOTES

This section describes the application of the extension of IPDD for handling time and data drifts in condition-based maintenance policies. This extension was not originally one objective of this thesis; however, the application scenario appears as a practical problem where we could validate the effectiveness of the developed tool in a real situation. The IPDD application in condition-based maintenance policies has been developed in partnership with the Graduate Program in Production and Systems Engineering (PUCPR).

7 CONCLUSIONS AND FUTURE WORK

Handling concept drift detection in process models raises different challenges. The first one is that despite the different approaches for detecting process drifts developed, most have the detection's accuracy sensitive to the parameter configuration. Some methods aim to reduce the number of parameters, minimizing this situation (A Maaradji et al., 2017; Abderrahmane Maaradji et al., 2015; Yeshchenko et al., 2021). However, the evaluation of the accuracy of proposed approaches lacks an objective experimental protocol with public datasets and clearly defined metrics, which characterizes another challenge for drift detection. This thesis extensively evaluates three tools (*IPDD*, Apromore ProDrift, and VDD) for sudden drift detection using two synthetic datasets and two metrics: F-score and mean delay. The applied experimental protocol and the defined metrics can be applied to compare other future approaches. We also publicize the two synthetic datasets, their process models (Petri nets), and a source code for simulating event logs containing sudden drifts (using distinct intervals between drifts). Even with the experimental protocol for synthetic datasets, there is a lack of validation methods for real-life event logs, where the ground truth about the drifts is unknown.

To verify the applicability of *IPDD* in real scenarios, we applied the *Adaptive IPDD trace by trace* in a real-life event log from a ticketing management process of an Italian Company[44] . We also performed a case study application of *Adaptive IPDD for time and data drifts* in condition-based maintenance policies. Based on both applications, we believe that the tool can be applied to different contexts of processes to identify process drifts. However, the validation of the detected drifts in real situations cannot follow objective metrics such as F-score or mean delay because we do not *a priori* know if a change occurred and when. In these cases, other strategies may be applied for validating the detected drifts, such as: monitoring metrics of the process (KPIs - Key Performance Indicators) or validating

It is essential to highlight that the current tools usually neglect to discover the changing process, described earlier as the absence of a simple visualization of the evolution of the process. Only VDD handles this issue; however, we obtained a low detection accuracy in the synthetic datasets evaluation. *IPDD* contributes to characterizing the process drifts based on similarity metrics between the process versions over time. The web interface implemented in *IPDD* provides a simple visualization of the process models using DFGs, where the user can visually compare the process versions supported by the description of

[44] https://doi.org/10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb

the change. Furthermore, showing the different versions of the process applied by *IPDD* is not the same as splitting the event log based on the change points and mining the process models. As change point detection is a reactive task, the change point is reported after the real drift. Thus, mining the models based on the reported change point derives models mixing traces before and after the drift.

The results from VDD highlight another interesting challenge related to the applied experimental protocol; the chosen datasets or metrics sometimes bias the reported results. The Apromore ProDrift, which is a relatively “old” approach (papers from 2015 to 2020), still shows the analysis more reliable considering the experimental protocol adopted (two evaluation metrics, different window sizes, and evaluation considering the complete dataset).

Most approaches still detect process drifts offline. Some approaches based on an online setting consider a stream of traces, which do not correctly represent the online situation. Another important aspect is that when dealing with event streams, we can consider also live event streams, where the event is added to the stream where it occurs. The identified online approaches deal only with event streams by reading the events ordered by their timestamps and handling incomplete traces, and sometimes this is not clearly defined.

Besides *IPDD* contributes with an adaptive approach based on the ADWIN change detector, its accuracy is still related to the window size parameter. However, the performed experiments showed that using a small window with *Adaptive IPDD trace by trace* can provide promising results. Also, discovering the changing process may help business analysts validate the drift detection results. Furthermore, *IPDD* provided a generic framework architecture that can be enhanced for other windowing approaches, similarity metrics, etc.

The limitations of the implemented *IPDD* approaches are:

- IPDD does not detect behavioral control-flow drifts;
- If the user sets a window size larger than the interval between drifts, the drifts are not accurately detected;
- The *Adaptive IPDD* processing time increases if the process model is complex because of the fitness and precision calculation;
- Data drift detection (reported in Section 6) is only available for the CLI or massive user interface.

For future work, we plan to extend *IPDD* to include an adaptive approach based on quality metrics using a window of adaptive size for the precision dimension. We also plan to evaluate other change detector algorithms for drift detection. Another necessary extension is to detect concept drifts online. Furthermore, as we have defined an experimental protocol for

comparing different tools, we plan to compare *IPDD* with other experimental tools which provide that source code, e.g., DOA (Zellner et al., 2020), CDESF Toolkit (Mora et al., 2020; Tavares et al., 2019), and LCDD (L. Lin et al., 2020). Another future work is to evaluate the user interface of the IPDD to verify its simplicity and usability for showing the process drifts.

REFERENCES

Accorsi, Rafael, Stocker, T., Accorsi, R., Stocker, T., Workflow, D., Clustering, T. T., Accorsi, R., & Stocker, T. (2012). Discovering Workflow Changes with Time-Based Trace Clustering. In K. Aberer, E. Damiani, & T. Dillon (Eds.), *Data-Driven Process Discovery and Analysis* (pp. 154–168). Springer Berlin Heidelberg. https://doi.org/https://doi.org/10.1007/978-3-642-34044-4_9

Accorsi, Riccardo, Manzini, R., Pascarella, P., Patella, M., & Sassi, S. (2017). Data Mining and Machine Learning for Condition-based Maintenance. *Procedia Manufacturing*, *11*, 1153–1161. https://doi.org/10.1016/J.PROMFG.2017.07.239

Adams, J. N., van Zelst, S. J., Quack, L., Hausmann, K., van der Aalst, W. M. P., & Rose, T. (2021). A Framework for Explainable Concept Drift Detection in Process Mining. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *12875 LNCS*, 400–416. https://doi.org/10.1007/978-3-030-85469-0_25

Akkiraju, R., & Ivan, A. (2010). Discovering Business Process Similarities: An Empirical Study with SAP Best Practice Business Processes. In P. P. Maglio, M. Weske, J. Yang, & M. Fantinato (Eds.), *Service-Oriented Computing* (pp. 515–526). Springer Berlin Heidelberg.

Altendeitering, M., & Dübler, S. (2020). Scalable Detection of Concept Drift: A Learning Technique Based on Support Vector Machines. *Procedia Manufacturing*, *51*, 400–407. https://doi.org/10.1016/J.PROMFG.2020.10.057

Ankerst, M., Breunig, M. M., Kriegel, H. P., & Sander, J. (1999). OPTICS: Ordering Points to Identify the Clustering Structure. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, *28*(2), 49–60. https://doi.org/10.1145/304181.304187

Barbon Junior, S., Tavares, G. M., da Costa, V. G. T., Ceravolo, P., & Damiani, E. (2018). A Framework for Human-in-the-loop Monitoring of Concept-drift Detection in Event Log Stream. *WWW '18: Companion Proceedings of the The Web Conference 2018*, *2*, 319–326. https://doi.org/10.1145/3184558.3186343

Batyuk, A, Oityshyn, V. V, & Verhun, V. (2018). Software Architecture Design of the Real-Time Processes Monitoring Platform. *2018 IEEE Second International Conference on Data Stream Mining Processing (DSMP)*, 98–101. https://doi.org/10.1109/DSMP.2018.8478589

Batyuk, Anatoliy, & Voityshyn, V. (2020). Streaming process discovery method for semi-structured business processes. *International Conference on Data Stream Mining and Processing, DSMP 2020*, 444–448. https://doi.org/10.1109/DSMP47368.2020.9204201

Becker, M., & Laue, R. (2012). A comparative survey of business process similarity measures. *Computers in Industry*, *63*(2), 148–167. https://doi.org/10.1016/j.compind.2011.11.003

Bengtsson, M., & Lundström, G. (2018). On the importance of combining "the new" with "the old" – One important prerequisite for maintenance in Industry 4.0. *Procedia Manufacturing*, *25*, 118–125. https://doi.org/10.1016/J.PROMFG.2018.06.065

Bennane, A., & Yacout, S. (2009). LAD-CBM; new data processing tool for diagnosis and prognosis in condition-based maintenance. *Journal of Intelligent Manufacturing*, *23*(2), 265–275. https://doi.org/10.1007/S10845-009-0349-8

Berti, A., & van der Aalst, W. (2019). Reviving token-based replay: Increasing speed while improving diagnostics. In C (Ed.), *International Workshop on Algorithms & Theories for the Analysis of Event Data 2019 (ATAED 2019)* (Vol. 2371, pp. 87–103). CEUR-WS.

Berti, A., Van Zelst, S. J., & van der Aalst, W. M. P. (2019). Process Mining for Python (PM4Py): Bridging the Gap Between Process-and Data Science. *Proceedings of the ICPM Demo Track 2019, 1st International Conference on Process Mining (ICPM 2019)*. http://python.org

Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. *Proceedings of the 7th SIAM International Conference on Data Mining*, 443–448. https://doi.org/10.1137/1.9781611972771.42

Blum, F. R. (2015). Metrics in process discovery. *Technical Report TR/DCC*, *2015–6*, 1–21.

Bose, R. P. J. C., & van der Aalst, W. M. P. (2010). Trace clustering based on conserved patterns: Towards achieving better process models. *Lecture Notes in Business Information Processing*, *43 LNBIP*, 170–181. https://doi.org/10.1007/978-3-642-12186-9_16

Bose, R. P. J. C., van der Aalst, W. M. P., Zliobaite, I., & Pechenizkiy, M. (2014). Dealing With Concept Drifts in Process Mining. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(1), 154–171. https://doi.org/10.1109/TNNLS.2013.2278313

Bose, R. P. J. C., van der Aalst, W. M. P., Žliobaite, I., & Pechenizkiy, M. (2011). Handling concept drift in process mining. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6741 LNCS*, 391–405. https://doi.org/10.1007/978-3-642-21640-4_30

Bousdekis, A., Apostolou, D., & Mentzas, G. (2020). Predictive Maintenance in the 4th Industrial Revolution: Benefits, Business Opportunities, and Managerial Implications. *IEEE Engineering Management Review*, *48*(1), 57–62. https://doi.org/10.1109/EMR.2019.2958037

Bousdekis, A., Magoutas, B., Apostolou, D., & Mentzas, G. (2015). A proactive decision making framework for condition-based maintenance. *Industrial Management and Data Systems*, *115*(7), 1225–1250. https://doi.org/10.1108/IMDS-03-2015-0071/FULL/PDF

Breuniq, M. M., Kriegel, H. P., Ng, R. T., & Sander, J. (2000). LOF: Identifying density-based local outliers. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, *29*(2), 93–104. https://doi.org/10.1145/335191.335388

Brockhoff, T., Uysal, M. S., & van der Aalst, W. M. P. (2020). Time-aware Concept Drift Detection Using the Earth Mover's Distance. *2nd International Conference on Process Mining (ICPM)*, 33–40. https://doi.org/10.1109/ICPM49681.2020.00016

Buijs, J. C. A. M., van Dongen, B. F., & van der Aalst, W. M. P. (2014). Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. *International Journal of Cooperative Information Systems*, *23*(1), 1–39. https://doi.org/10.1142/S0218843014400012

Burattin, A, Cimitile, M., Maggi, F. M., & Sperduti, A. (2015). Online Discovery of Declarative Process Models from Event Streams. *IEEE Transactions on Services Computing*, *8*(6), 833–846. https://doi.org/10.1109/TSC.2015.2459703

Burattin, A, Sperduti, A., & van der Aalst, W. M. P. (2014). Control-flow discovery from event streams. *2014 IEEE Congress on Evolutionary Computation (CEC)*, 2420–2427. https://doi.org/10.1109/CEC.2014.6900341

Burattin, Andrea, Cimitile, M., & Maggi, F. M. (2015). Lights, camera, action! Business process movies for online process discovery. In F. Fournier & J. and Mendling (Eds.), *Business Process Management Workshops (BPM 2014)* (pp. 408--419). Springer International Publishing. https://doi.org/10.1007/978-3-319-15895-2_34

Buttazzo, G. C. (2011). Hard Real-Time Computing Systems. In *Journal of Chemical Information and Modeling* (3rd ed., Vol. 24). Springer US. https://doi.org/10.1007/978-1-4614-0676-1

Cao, F., Estert, M., Qian, W., & Zhou, A. (2006). Density-Based Clustering over an Evolving Data Stream with Noise. *Proceedings of the 2006 SIAM International Conference on Data Mining*, 328–339. https://doi.org/10.1137/1.9781611972764.29

Cao, Y., Subramaniam, V., & Chen, R. (2012). Performance evaluation and enhancement of multistage manufacturing systems with rework loops. *Computers & Industrial Engineering*, *62*(1), 161–176. https://doi.org/10.1016/J.CIE.2011.09.004

Carmona, J., & Gavaldà, R. (2012). *Online Techniques for Dealing with Concept Drift in*

Process Mining (pp. 90–102). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-34156-4_10

Ceravolo, G. M. T. S. B. J. . P. (2019). *Synthetic Event Streams*. IEEE Dataport. https://doi.org/10.21227/2kxd-m509

Ceravolo, P., Marques Tavares, G., Junior, S. B., & Damiani, E. (2020). Evaluation Goals for Online Process Mining: a Concept Drift Perspective. *IEEE Transactions on Services Computing*. https://doi.org/10.1109/TSC.2020.3004532

Choueiri, A. C., & Portela Santos, E. A. (2021). Multi-product scheduling through process mining: bridging optimization and machine process intelligence. *Journal of Intelligent Manufacturing*, *32*(6), 1649–1667. https://doi.org/10.1007/S10845-021-01767-2/FIGURES/16

Choueiri, A. C., Sato, D. M. V., Scalabrin, E. E., & Santos, E. A. P. (2020). An extended model for remaining time prediction in manufacturing systems using process mining. *Journal of Manufacturing Systems*, *56*, 188–201. https://doi.org/10.1016/j.jmsy.2020.06.003

Conforti, R., Rosa, M. L., & Hofstede, A. H. M. t. (2017). Filtering Out Infrequent Behavior from Business Process Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, *29*(2), 300–314. https://doi.org/10.1109/TKDE.2016.2614680

Crc, H. (2014). *DATA CLUSTERING Algorithms and Apllications* (C. C. Aggarwal & C. K. Reddy (eds.)).

De Medeiros, A. K. A., van Dongen, B. F., van der Aalst, W. M. P., & Weijters, A. J. M. M. (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*, 151–165. https://doi.org/10.1007/978-3-540-30188-2_12

De Sousa, R. G., Peres, S. M., Fantinato, M., & Reijers, H. A. (2021). Concept drift detection and localization in process mining: An integrated and efficient approach enabled by trace clustering. *Proceedings of the ACM Symposium on Applied Computing*, 364–373. https://doi.org/10.1145/3412841.3441918

Denisov, V., Elena, B., & D, F. (2018). BPIC'2018: Mining Concept Drift in Performance Spectra of Processes. In *International Workshop on Business Process Intelligence 2018*. https://doi.org/10.4121/uuid:3301445f-95e8-4ff0-981f1f204972

Ditzler, G., Roveri, M., Alippi, C., & Polikar, R. (2015). Learning in Nonstationary Environments: A Survey. *IEEE Computational Intelligence Magazine*, *10*(4), 12–25. https://doi.org/10.1109/MCI.2015.2471196

Do, P., Voisin, A., Levrat, E., & Iung, B. (2015). A proactive condition-based maintenance strategy with both perfect and imperfect maintenance actions. *Reliability Engineering & System Safety*, *133*, 22–32. https://doi.org/10.1016/J.RESS.2014.08.011

Dries, A., & Rückert, U. (2009). Adaptive concept drift detection. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *2*(5–6), 311–327. https://doi.org/10.1002/SAM.10054

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 226–231. www.aaai.org

Fan, B., Andersen, D. G., Kaminsky, M., & Mitzenmacher, M. D. (2014). Cuckoo Filter: Practically Better Than Bloom. *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, 75–88. https://doi.org/10.1145/2674005.2674994

Frank, E., & Witten, I. H. (1998). Using a permutation test for attribute selection in decision trees. *15th International Conference on Machine Learning*, 152–160. https://researchcommons.waikato.ac.nz/handle/10289/1506

Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A Survey on Concept Drift Adaptation. *ACM Comput. Surv.*, *46*(4), 44:1--44:37. https://doi.org/10.1145/2523813

Garcia, C. dos S., Meincheim, A., Faria Junior, E. R., Dallagassa, M. R., Sato, D. M. V., Carvalho, D. R., Santos, E. A. P., & Scalabrin, E. E. (2019). Process mining techniques and applications – A systematic mapping study. *Expert Systems with Applications*, *133*, 260–295. https://doi.org/10.1016/j.eswa.2019.05.003

Gulledge, T., Hiroshige, S., & Iyer, R. (2010). Condition-based Maintenance and the product improvement process. *Computers in Industry*, *61*(9), 813–832. https://doi.org/10.1016/J.COMPIND.2010.07.007

Günther, C. W., & Rozinat, A. (2012). Disco: Discover Your Processes. *Proceedings of the Demonstration Track of the 10th International Conference on Business Process Management (BPM 2012)*.

Günther, C. W., & van der Aalst, W. M. P. (2007). Fuzzy mining - Adaptive process simplification based on multi-perspective metrics. *International Conference on Business Process Management BPM 2007: Business Process Management*, *4714 LNCS*, 328–343. https://doi.org/10.1007/978-3-540-75183-0_24

Hassani, M. (2019). Concept drift detection of event streams using an adaptive window. *Proceedings - European Council for Modelling and Simulation, ECMS*, *33*(1), 230–239. https://doi.org/10.7148/2019-0230

Hompes, B., van der Aalst, W. M. P., & Dixit, P. (2017). Detecting Changes in Process Behavior Using Comparative Case Clustering. *Data-Driven Process Discovery and Analysis (SIMPDA 2015)*, *244*(November 2018), 0–22. https://doi.org/10.1007/978-3-319-53435-0

Huang, D. T. J., Koh, Y. S., Dobbie, G., & Bifet, A. (2015). Drift detection using stream volatility. *Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2015)*, *9284*, 417–432. https://doi.org/10.1007/978-3-319-23528-8_26

Huynh, K. T., Barros, A., & Bérenguer, C. (2015). Multi-level decision-making for the predictive maintenance of k-out-of-n:F deteriorating systems. *IEEE Transactions on Reliability*, *64*(1), 94–117. https://doi.org/10.1109/TR.2014.2337791

Impedovo, A., Mignone, P., Loglisci, C., & Ceci, M. (2020). Simultaneous Process Drift Detection and Characterization with Pattern-Based Change Detectors. *International Conference on Discovery Science (DS2020)*, *12323 LNAI*, 451–467. https://doi.org/10.1007/978-3-030-61527-7_30

Janssenswillen, G., Depaire, B., Swennen, M., Jans, M., & Vanhoof, K. (2019). bupaR: Enabling reproducible business process analysis. *Knowledge-Based Systems*, *163*, 927–930. https://doi.org/10.1016/j.knosys.2018.10.018

Jardine, A. K. S., Lin, D., & Banjevic, D. (2006). A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing*, *20*(7), 1483–1510. https://doi.org/10.1016/J.YMSSP.2005.09.012

Jayaratne, D., De Silva, D., Alahakoon, D., & Yu, X. (2021). Continuous detection of concept drift in industrial cyber-physical systems using closed loop incremental machine learning. *Discover Artificial Intelligence*, *1*(1), 1–13. https://doi.org/10.1007/S44163-021-00007-Z

Jimenez-Cortadi, A., Irigoien, I., Boto, F., Sierra, B., & Rodriguez, G. (2020). Predictive maintenance on the machining process and machine tool. *Applied Sciences (Switzerland)*, *10*(1). https://doi.org/10.3390/APP10010224

Killick, R., Fearnhead, P., & Eckley, I. A. (2012). Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, *107*(500), 1590–1598. https://doi.org/10.1080/01621459.2012.737745

Kitchenham, B., & Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. In *Technical report* (Vol. 2, Issue 3).

Kurniati, A. P., McInerney, C., Zucker, K., Hall, G., Hogg, D., & Johnson, O. (2020). Using a multi-level process comparison for process change analysis in cancer pathways. *International Journal of Environmental Research and Public Health*, *17*(19), 1–16. https://doi.org/10.3390/ijerph17197210

Kurniati A.P., McInerney C., Zucker K., Hall G., Hogg D., J. O. (2019). A Multi-level Approach for Identifying Process Change in Cancer Pathways. *International Conference on Business Process Management BPM 2019: Business Process Management Workshops*, 595–607.

Kurscheidt Netto, R. J., Santos, E. A. P., Loures, E. F. R., & Pécora Jr., J. E. (2015). Discovering Bayesian networks using process mining: an application in manufacturing. *XXI International Conference on Industrial Engineering and Operations Management*, *367*.

Kurscheidt, R. J., Santos, E. A. P., De, E., Loures, F. R., Pecora, J. E., & Cestari, J. M. A. P. (2015). A Methodology for Discovering Bayesian Networks Based on Process Mining. *Industrial and Systems Engineering Research Conference*.

La Rosa, M., Reijers, H. A., van der Aalst, W. M. P., Dijkman, R. M., Mendling, J., Dumas, M., & García-Bañuelos, L. (2011). APROMORE: An advanced process model repository. *Expert Systems with Applications*, *38*(6), 7029–7040. https://doi.org/10.1016/j.eswa.2010.12.012

Leemans, S. J. J., Fahland, D., & van der Aalst, W. M. P. (2013). Discovering block-structured process models from event logs - A constructive approach. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7927 LNCS*, 311–329. https://doi.org/10.1007/978-3-642-38697-8_17

Lin, C. C., Deng, D. J., Kuo, C. H., & Chen, L. (2019). Concept drift detection and adaption in big imbalance industrial IoT data using an ensemble learning method of offline classifiers. *IEEE Access*, *7*, 56198–56207. https://doi.org/10.1109/ACCESS.2019.2912631

Lin, L., Wen, L., Lin, L., Pei, J., & Yang, H. (2020). LCDD: Detecting Business Process Drifts Based on Local Completeness. *IEEE Transactions on Services Computing*, *14*(8), 1–1. https://doi.org/10.1109/TSC.2020.3032787

Liu, J., Song, B., & Zhang, Y. (2018). Competing failure model for mechanical system with multiple functional failures: *Advances in Mechanical Engineering*, *10*(5), 2018. https://doi.org/10.1177/1687814018773155

Liu, N., Huang, J., & Cui, L. (2018). A Framework for Online Process Concept Drift Detection from Event Streams. *2018 IEEE International Conference on Services Computing (SCC)*, 105–112. https://doi.org/10.1109/SCC.2018.00021

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2019). Learning under Concept Drift: A Review. *IEEE Transactions on Knowledge and Data Engineering*, *31*(12), 2346–2363. https://doi.org/10.1109/TKDE.2018.2876857

Lu, Y. (2017). Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration*, *6*, 1–10. https://doi.org/10.1016/j.jii.2017.04.005

Luengo, D., & Sepúlveda, M. (2012). Applying Clustering in Process Mining to Find Different Versions of a Business Process that Changes over Time. In F. Daniel, K. Barkaoui, & S. Dustdar (Eds.), *Business Process Management Workshops* (pp. 153–158). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-28108-2_15

Maaradji, A, Dumas, M., Rosa, M. L., & Ostovar, A. (2017). Detecting Sudden and Gradual Drifts in Business Processes from Execution Traces. *IEEE Transactions on Knowledge and Data Engineering*, *29*(10), 2140–2154. https://doi.org/10.1109/TKDE.2017.2720601

Maaradji, Abderrahmane, Dumas, M., La Rosa, M., & Ostovar, A. (2015). Fast and Accurate Business Process Drift Detection. *International Conference on Business Process Management BPM 2016: Business Process Management*, 406–422.

https://doi.org/10.1007/978-3-319-23063-4_27

Maggi, F. M., Burattin, A., Cimitile, M., & Sperduti, A. (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), 94–111. https://doi.org/10.1007/978-3-642-41030-7_7

Manku, G. S., & Motwani, R. (2002). Approximate Frequency Counts over Data Streams. In P. A. Bernstein, Y. E. Ioannidis, R. Ramakrishnan, & D. Papadias (Eds.), *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases* (pp. 346–357). Morgan Kaufmann. https://doi.org/https://doi.org/10.1016/B978-155860869-6/50038-X

Manoj Kumar, M. V., Thomas, L., & Annappa, B. (2015). Capturing the sudden concept drift in process mining. *International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2015*, *1371*(January), 132–143.

Martino, G. D. S., Navarin, N., & Sperduti, A. (2013). A Lossy Counting Based Approach for Learning on Streams of Graphs on a Budget. *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 1294–1301.

Martjushev, J., Bose, R. P. J. C., & van der Aalst, W. M. P. (2015). Change Point Detection and Dealing with Gradual and Multi-Order Dynamics in Process Mining. *International Conference on Business Informatics Research*, 1–15. https://doi.org/10.1007/978-3-319-21915-8_11

Mehta, P., Werner, A., & Mears, L. (2015). Condition based maintenance-systems integration and intelligence using Bayesian classification and sensor fusion. *Journal of Intelligent Manufacturing*, *26*(2), 331–346. https://doi.org/10.1007/S10845-013-0787-1/FIGURES/17

Mermillod, M., Bugaiska, A., & Bonin, P. (2013). The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology*, *4*, 504. https://doi.org/10.3389/fpsyg.2013.00504

Montiel, J., Read, J., Bifet, A., & Abdessalem, T. (2018). Scikit-multiflow: A Multi-output Streaming Framework. *Journal of Machine Learning Research*, *19*. https://doi.org/10.5555/3291125.3309634

Mora, D., Ceravolo, P., Damiani, E., & Tavares, G. M. (2020). The CDESF toolkit: An introduction. *ICPM Doctoral Consortium and Tool Demonstration Track 2020*, *2703*(October), 47–50. https://github.com/gbrltv/cdesf2

Muñoz-Gama, J., & Carmona, J. (2010). A Fresh Look at Precision in Process Conformance. *International Conference on Business Process Management (BPM 2010)*, *6336 LNCS*, 211–226. https://doi.org/10.1007/978-3-642-15618-2_16

Omori, N. J., Tavares, G. M., Ceravolo, P., & Barbon, S. (2019). Comparing Concept Drift Detection with Process Mining Tools. *SBSI'19: Proceedings of the XV Brazilian Symposium on Information Systems*, 1–8. https://doi.org/10.1145/3330204.3330240

Ostovar, A., Leemans, S. J. J., & Rosa, M. La. (2020). Robust Drift Characterization from Event Streams of Business Processes. *ACM Transactions on Knowledge Discovery from Data*, *14*(3), 1–57. https://doi.org/10.1145/3375398

Ostovar, A., Maaradji, A., La Rosa, M., & ter Hofstede, A. H. M. (2017). Characterizing Drift from Event Streams of Business Processes. *International Conference on Advanced Information Systems Engineering CAiSE 2017: Advanced Information Systems Engineering*, 210–228. https://doi.org/10.1007/978-3-319-59536-8_14

Ostovar, A., Maaradji, A., La Rosa, M., ter Hofstede, A. H. M., & van Dongen, B. F. V. (2016). Detecting Drift from Event Streams of Unpredictable Business Processes. *Conceptual Modeling: 35th International Conference, ER 2016 (Lecture Notes in Computer Science)*, 330–346. https://doi.org/10.1007/978-3-319-46397-1_26

Parikh, R. J. (1966). On Context-Free Languages. *Journal of the Association for Computing*

Machinery, *13*(4), 570–581.

Pauwels, S., & Calders, T. (2019). An anomaly detection technique for business processes based on extended dynamic Bayesian networks. *Proceedings of the ACM Symposium on Applied Computing*, *Part F1477*(April), 494–501.

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2014). A Design Science Research Methodology for Information Systems Research. *Https://Doi.Org/10.2753/MIS0742-1222240302*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Peng, Y., Dong, M., & Zuo, M. J. (2010). Current status of machine prognostics in condition-based maintenance: a review. *The International Journal of Advanced Manufacturing Technology*, *50*(1), 297–313. https://doi.org/10.1007/S00170-009-2482-0

Prajapati, A., & Ganesan, S. (2013). Application of Statistical Techniques and Neural Networks in Condition-Based Maintenance. *Quality and Reliability Engineering International*, *29*(3), 439–461. https://doi.org/10.1002/QRE.1392

Redlich, D., Molka, T., Gilani, W., Blair, G., & Rashid, A. (2014). Scalable dynamic business process discovery with the constructs competition miner. *International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA 2014)*, *1293*(December), 91–107.

Richter, F., Maldonado, A., Zellner, L., & Seidl, T. (2020). OTOSO: Online Trace Ordering for Structural Overviews. *Process Mining Workshops - 1st International Workshop on Streaming Analytics for Process Mining (SA4PM'20)*, 189–192.

Richter, F., & Seidl, T. (2019). Looking into the TESSERACT: Time-drifts in event streams using series of evolving rolling averages of completion times. *Information Systems*, *84*, 265–282. https://doi.org/10.1016/j.is.2018.11.003

Richter, F., & Seidl, T. (2017). TESSERACT: Time-Drifts in Event Streams Using Series of Evolving Rolling Averages of Completion Times. *International Conference on Business Process Management BPM 2017: Business Process Management*, *84*, 289–305. https://doi.org/10.1007/978-3-319-65000-5_17

Rozinat, A., & van der Aalst, W. M. P. (2008). Conformance checking of processes based on monitoring real behavior. *Information Systems*, *33*(1), 64–95. https://doi.org/10.1016/j.is.2007.07.001

Ruschel, E., Rocha Loures, E. de F., & Santos, E. A. P. (2021). Performance analysis and time prediction in manufacturing systems. *Computers & Industrial Engineering*, *151*, 106972. https://doi.org/10.1016/J.CIE.2020.106972

Ruschel, E., Santos, E. A. P., & Loures, E. de F. R. (2017). Industrial maintenance decision-making: A systematic literature review. *Journal of Manufacturing Systems*, *45*, 180–194. https://doi.org/10.1016/J.JMSY.2017.09.003

Ruschel, E., Santos, E. A. P., & Loures, E. de F. R. (2020). Establishment of maintenance inspection intervals: an application of process mining techniques in manufacturing. *Journal of Intelligent Manufacturing*, *31*(1), 53–72. https://doi.org/10.1007/S10845-018-1434-7

Sato, D. M. V., Barddal, J. P., & Scalabrin, E. E. (2021). Interactive Process Drift Detection Framework. *ICAISC 2021: Artificial Intelligence and Soft Computing*, *12855 LNAI*, 192–204. https://doi.org/10.1007/978-3-030-87897-9_18

Sato, D. M. V., De Freitas, S. C., Barddal, J. P., & Scalabrin, E. E. (2022). A Survey on Concept Drift in Process Mining. *ACM Computing Surveys*, *54*(9), 1–38. https://doi.org/10.1145/3472752

Sato, D. M. V., Fontana, R. M., Barddal, J. P., & Scalabrin, E. E. (2021). Interactive Process Drift Detection: A Framework for Visual Analysis of Process Drifts (Extended Abstract). *ICPM 2021 Doctoral Consortium and Demo Track 2021*. https://youtu.be/8feKd6jr8Gs

Schonenberg, H., Mans, R., Russell, N., Mulyar, N., & van der Aalst, W. M. P. (2008). Process flexibility: A survey of contemporary approaches. *International Workshop on*

Cooperation and Interoperability, Architecture and Ontology Workshop on Enterprise and Organizational Modeling and Simulation CIAO! 2008, EOMAS 2008: Advances in Enterprise Engineering, *10 LNBIP*, 16–30. https://doi.org/10.1007/978-3-540-68644-6_2

Schubert, E., Weiler, M., & Kriegel, H.-P. (2014). SigniTrend: Scalable Detection of Emerging Topics in Textual Streams by Hashed Significance Thresholds. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 871–880. https://doi.org/10.1145/2623330.2623740

Seeliger, A., Nolle, T., & Mühlhäuser, M. (2017). Detecting concept drift in processes using graph metrics on process graphs. *Proceedings of the 9th Conference on Subject-Oriented Business Process Management, S-BPM ONE '17*, *Part F1271*. https://doi.org/10.1145/3040565.3040566

Shin, J. H., & Jun, H. B. (2015). On condition based maintenance policy. *Journal of Computational Design and Engineering*, *2*(2), 119–127. https://doi.org/10.1016/J.JCDE.2014.12.006

Stertz, F., & Rinderle-Ma, S. (2019). Detecting and Identifying Data Drifts in Process Event Streams Based on Process Histories. *Information Systems Engineering in Responsible Information Systems. CAiSE 2019.*, *350*(628), 133–144. https://doi.org/10.1007/978-3-030-21297-1

Stertz, F., & Rinderle-Ma, S. (2018). Process Histories - Detecting and Representing Concept Drifts Based on Event Streams. In H. Panetto, C. Debruyne, H. A. Proper, C. A. Ardagna, D. Roman, & R. Meersman (Eds.), *On the Move to Meaningful Internet Systems. {OTM} 2018 Conferences - Confederated International Conferences: CoopIS, C&TC, and {ODBASE} 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part {I}* (Vol. 11229, pp. 318–335). Springer. https://doi.org/10.1007/978-3-030-02610-3_18

Sun, Z., Tang, J., Qiao, J., & Cui, C. (2020). Review of Concept Drift Detection Method for Industrial Process Modeling. *Chinese Control Conference, CCC*, *2020-July*, 5754–5759. https://doi.org/10.23919/CCC50068.2020.9189106

Tang, D., Yu, J., Chen, X., & Makis, V. (2015). An optimal condition-based maintenance policy for a degrading system subject to the competing risks of soft and hard failure. *Computers and Industrial Engineering*, *83*, 100–110. https://doi.org/10.1016/j.cie.2015.02.003

Tavares, G. M., Ceravolo, P., Da Costa, V. G. T., Damiani, E., & Junior, S. B. (2019). Overlapping analytic stages in online process mining. *IEEE International Conference on Services Computing (SCC 2019)*, 167–175. https://doi.org/10.1109/SCC.2019.00037

Tian, Z., Lin, D., & Wu, B. (2012). Condition based maintenance optimization considering multiple objectives. *Journal of Intelligent Manufacturing*, *23*(2), 333–340. https://doi.org/10.1007/s10845-009-0358-7

Truong, C., Oudre, L., & Vayatis, N. (2020). Selective review of offline change point detection methods. *Signal Processing*, *167*, 107299. https://doi.org/https://doi.org/10.1016/j.sigpro.2019.107299

van der Aalst, W. M. P. (2016). Process mining: Data science in action. In *Process Mining: Data Science in Action*. Springer. https://doi.org/10.1007/978-3-662-49851-4

van der Aalst, W. M. P., Adriansyah, A., de Medeiros, A. K. A., Arcieri, F., Baier, T., Blickle, T., Bose, J. C., van den Brand, P., Brandtjen, R., Buijs, J., Burattin, A., Carmona, J., Castellanos, M., Claes, J., Cook, J., Costantini, N., Curbera, F., Damiani, E., de Leoni, M., … Wynn, M. (2011). Process Mining Manifesto. *International Conference on Business Process Management BPM 2011: Business Process Management Workshops*, *99*, 169–194. https://doi.org/10.1007/978-3-642-28108-2_19

van der Aalst, W. M. P., Pesic, M., & Schonenberg, H. (2009). Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, *23*(2), 99–113. https://doi.org/10.1007/s00450-009-0057-9

van Der Aalst, W. M. P., Rubin, V., Verbeek, H. M. W., van Dongen, B. F., Kindler, E., & Günther, C. W. (2010). Process mining: A two-step approach to balance between

underfitting and overfitting. *Software and Systems Modeling*, *9*(1), 87–111. https://doi.org/10.1007/s10270-008-0106-z

van der Aalst, W. M. P., van Dongen, B. F., Günther, C., Rozinat, A., Verbeek, H. M. W., & Weijters, A. J. M. M. (2009). Prom: The process mining toolkit. *BPM 2009 Demonstration Track*, *489*, 1–4. https://research.tue.nl/en/publications/prom-6-the-process-mining-toolkit

van Eck, M. L., Lu, X., Leemans, S. J. J., & van der Aalst, W. M. P. (2015). PM2: A Process Mining Project Methodology. *International Conference on Advanced Information Systems Engineering*, 297–313. https://doi.org/10.1007/978-3-319-19069-3_19

van Glabbeek, R., & Goltz, U. (1989). Equivalence notions for concurrent systems and refinement of actions (extended abstract). *Proceedings 14th Symposium on Mathematical Foundations of Computer Science, MFCS '89*, *379*, 237–248. https://dl.acm.org/doi/10.5555/645719.663398

van Zelst, S. J., van Dongen, B. F., & van der Aalst, W. M. P. (2018). Event stream-based process discovery using abstract representations. *Knowledge and Information Systems*, *54*(2), 407–435. https://doi.org/10.1007/s10115-017-1060-2

Veldman, J., Wortmann, H., & Klingenberg, W. (2011). Methodology and theory typology of condition based maintenance. *Journal of Quality in Maintenance Engineering*, *17*(2), 183–202. https://doi.org/10.1108/13552511111134600

Voisin, A., Levrat, E., Cocheteux, P., & Iung, B. (2010). Generic prognosis model for proactive maintenance decision support: Application to pre-industrial e-maintenance test bed. *Journal of Intelligent Manufacturing*, *21*(2), 177–193. https://doi.org/10.1007/s10845-008-0196-z

Weber, B., Reichert, M., & Rinderle-Ma, S. (2008). Change patterns and change support features - Enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering*, *66*(3), 438–466. https://doi.org/10.1016/j.datak.2008.05.001

Weber, P., Tino, P., & Bordbar, B. (2012). Process Mining in Non-Stationary Environments. *European Symposium on Artificial Neural Networks, ESANN 2012*, *April*.

Weidlich, M., Mendling, J., & Weske, M. (2011). Efficient consistency measurement based on behavioral profiles of process models. *IEEE Transactions on Software Engineering*, *37*(3), 410–429. https://doi.org/10.1109/TSE.2010.96

Weijters, A. J. M. M., van der Aalst, W. M. P., & Alves De Medeiros, A. K. (2006). *Process Mining with the HeuristicsMiner Algorithm*. Technische Universiteit Eindhoven. https://research.tue.nl/en/publications/process-mining-with-the-heuristicsminer-algorithm

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*, 1–10. https://doi.org/10.1145/2601248.2601268

Yang, H., Wen, L., & Wang, J. (2012). An Approach to Evaluate the Local Completeness of an Event Log. *2012 IEEE 12th International Conference on Data Mining*, 1164–1169. https://doi.org/10.1109/ICDM.2012.66

Yeshchenko, A., Bayomie, D., Gross, S., & Mendling, J. (2020). Visualizing Business Process Evolution. In S. Nurcan, I. Reinhartz-Berger, P. Soffer, & J. Zdravkovic (Eds.), *Enterprise, Business-Process and Information Systems Modeling* (pp. 185–192). Springer International Publishing.

Yeshchenko, A., Ciccio, C. Di, Mendling, J., & Polyvyanyy, A. (2019). Comprehensive Process Drift Analysis with the Visual Drift Detection Tool. *CEUR Workshop Proceedings*, 108--112.

Yeshchenko, A., Di Ciccio, C., Mendling, J., & Polyvyanyy, A. (2021). Visual Drift Detection for Sequence Data Analysis of Business Processes. *IEEE Transactions on Visualization and Computer Graphics*, 1–1. https://doi.org/10.1109/TVCG.2021.3050071

Yeshchenko, A., Di Ciccio, C., Mendling, J., & Polyvyanyy, A. (2019). Comprehensive

Process Drift Detection with Visual Analytics. *Conceptual Modeling. ER 2019. Lecture Notes in Computer Science*, *November*, 119–135. https://doi.org/https://doi.org/10.1007/978-3-030-33223-5_11

Yeshchenko, A., Mendling, J., Ciccio, C. Di, & Polyvyanyy, A. (2020). VDD: A Visual Drift Detection System for Process Mining. *ICPM 2020 Doctoral Consortium and Tool Demonstration Track*, 31–34. https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460

Zellner, L., Richter, F., Sontheim, J., Maldonado, A., & Seidl, T. (2020). Concept Drift Detection on Streaming Data with Dynamic Outlier Aggregation. In Springer (Ed.), *Process Mining Workshops - 1st International Workshop on Streaming Analytics for Process Mining (SA4PM'20)* (pp. 189–192). Springer. https://icpmconference.org/2020/

Zenisek, J., Holzinger, F., & Affenzeller, M. (2019). Machine learning based concept drift detection for predictive maintenance. *Computers and Industrial Engineering*, *137*, 106031. https://doi.org/10.1016/j.cie.2019.106031

Zha, H., Wang, J., Wen, L., Wang, C., & Sun, J. (2010). A workflow net similarity measure based on transition adjacency relations. *Computers in Industry*, *61*(5), 463–471. https://doi.org/10.1016/j.compind.2010.01.001

Zheng, C., B, L. W., Wang, J., Wen, L., & Wang, J. (2017). Detecting Process Concept Drifts from Event Logs. In H. Panetto, C. Debruyne, W. Gaaloul, M. Papazoglou, A. Paschke, C. A. Ardagna, & R. Meersman (Eds.), *On the Move to Meaningful Internet Systems. OTM 2017.* (Issue November, pp. 524–542). Springer International Publishing. https://doi.org/10.1007/978-3-319-69462-7

Zhou, Y., Lin, T. R., Sun, Y., Bian, Y., & Ma, L. (2015). An effective approach to reducing strategy space for maintenance optimisation of multistate series-parallel systems. *Reliability Engineering and System Safety*, *138*, 40–53. https://doi.org/10.1016/j.ress.2015.01.018