Published: 22 March 2025
Online AM: 04 February 2025
Accepted: 27 January 2025
Revised: 04 December 2024
Received: 16 July 2024

Citation in BibTeX format

RESEARCH-ARTICLE

Gradual Drift Detection in Process Models Using Conformance Metrics

VÍCTOR GALLEGO-FONTENLA, University of Santiago de Compostela, Santiago de Compostela, A Coruna, Spain

PEDRO GAMALLO-FERNÁNDEZ, University of Santiago de Compostela, Santiago de Compostela, A Coruna, Spain

JUAN CARLOS VIDAL, University of Santiago de Compostela, Santiago de Compostela, A Coruna, Spain

MANUEL LAMA, University of Santiago de Compostela, Santiago de Compostela, A Coruna, Spain

Gradual Drift Detection in Process Models Using Conformance Metrics

VÍCTOR GALLEGO-FONTENLA, PEDRO GAMALLO-FERNANDEZ, JUAN C. VIDAL, and MANUEL LAMA, Centro Singular de Investigación en Tecnoloxías Intelixentes, Universidade de Santiago de Compostela, Santiago de Compostela, Spain

Changes, planned or unexpected, are common during the execution of real-life processes. Detecting these changes is a must for optimizing the performance of organizations running such processes. Most of the algorithms present in the *state-of-the-art* focus on the detection of sudden changes, leaving aside other types of changes. In this article, we will focus on the automatic detection of gradual drifts, a special type of change, in which the cases of two models overlap during a period of time. The proposed algorithm relies on conformance checking metrics to carry out the automatic detection of the changes, performing also a fully automatic classification of these changes into sudden or gradual. The approach has been validated with a synthetic dataset consisting of 120 logs with different distributions of changes, getting better results in terms of detection and classification accuracy, delay, and change region overlapping than the main *state-of-the-art* algorithms.

CCS Concepts: • **Information systems** → *Information systems applications*;

Additional Key Words and Phrases: Business Processes, Concept drift, Gradual drift, Process mining Conformance checking

Associate Editor: Xiaohui Yu

ACM Reference format:
Víctor Gallego-Fontenla, Pedro Gamallo-Fernandez, Juan C. Vidal, and Manuel Lama. 2025. Gradual Drift Detection in Process Models Using Conformance Metrics. *ACM Trans. Knowl. Discov. Data.* 19, 3, Article 71 (March 2025), 40 pages.
https://doi.org/10.1145/3716169

Víctor Gallego-Fontenla is also with Intelligent Systems Group (GSI), Electronics and Computer Science Department (DEC), Universidade de Santiago de Compostela.
The work from Pedro Gamallo-Fernandez was supported by the Spanish Ministry of Science, Innovation, and Universities (Grant PRE2021-097271). This work has received financial support from the Consellería de Educación, Ciencia, Universidades e Formación Profesional (Accreditation 2019–2022 ED431G-2019/04), the European Regional Development Fund (ERDF), which acknowledges the CiTIUS—Centro Singular de Investigación en Tecnoloxías Intelixentes da Universidade de Santiago de Compostela as a Research Center of the Galician University System, and the Spanish Ministry of Science, Innovation and Universities (Grants PDC2021-121072-C21, PID2020-112623GB-I00, TED2021-130374B-C21, and PID2023-149549NB-I00).
Authors' Contact Information: Víctor Gallego-Fontenla, Centro Singular de Investigación en Tecnoloxías Intelixentes, Universidade de Santiago de Compostela, Santiago de Compostela, Spain; e-mail: victorjose.gallego@usc.es; Pedro Gamallo-Fernandez (corresponding author), Centro Singular de Investigación en Tecnoloxías Intelixentes, Universidade de Santiago de Compostela, Santiago de Compostela, Spain; e-mail: pedro.gamallo.fernandez@usc.es; Juan C. Vidal, Centro Singular de Investigación en Tecnoloxías Intelixentes, Universidade de Santiago de Compostela, Santiago de Compostela, Spain; e-mail: juan.vidal@usc.es; Manuel Lama, Centro Singular de Investigación en Tecnoloxías Intelixentes, Universidade de Santiago de Compostela, Santiago de Compostela, Spain; e-mail: manuel.lama@usc.es.

ACM 1556-472X/2025/3-ART71
https://doi.org/10.1145/3716169

1 Introduction

While the day-to-day operations of organizations normally remain unchanged over time, certain processes are susceptible to change due to internal or external factors. Regulatory shifts, changes in availability of different resources, or new consumption patterns, for example, can influence the structure of the processes that shape these operations. At best, these changes will be carried out intentionally, so the organization will be aware of their presence. But usually, these changes arise without prior warning and may be overlooked. It is in these cases when it becomes important to have methods that allow organizations to detect these changes in order to explicitly adapt their processes. Otherwise the organization might take decisions on the basis of outdated knowledge and, consequently, get unexpected results.

These changes, also called *drifts*, are categorized into four types, depending on their temporal distribution [12]: sudden drifts (Figure 1(a)), when they occur at an exact moment in time; gradual drifts (Figure 1(b)), when the change is prolonged over time, with the original and the modified behavior overlapping for a period of time; recurrent drifts (Figure 1(c)), when the changes repeat from time to time; and incremental drifts (Figure 1(d)), when successive changes with an intermediate behavior repeat until the model stabilizes. Despite being a widely explored field in descriptive analysis, it should be noted that dealing with concept drift in process mining faces an additional challenge: the data—execution traces from a process—may contain partial information about the process structure that can only be correctly understood when looking at the whole set of traces. For example, the presence of a choice structure can only be extracted by observing several traces, since the information present in a single trace will only represent a specific path. This can happen also with other control structures, such as loops or concurrent blocks.

This article focuses on the detection of gradual drifts, in which two versions of the process coexist for a certain period of time until one of them completely replaces the other. This type of change is challenging in process mining, since whenever a change is made to the process, traces already in execution can continue with the previous version of the model, while new traces will be generated by the already modified model, so both versions will coexist for a certain period of time. Thus, proposals that deal with gradual drift in process mining must ideally have a series of desirable features:

- *F1.* *Automatically distinguish between gradual and sudden drifts*, so it is not known *a priori* which kind of changes might be found in the log. Note that there is no possible confusion between gradual drifts and the other two types of drifts, since in recurrent drifts we deal with sudden or gradual drifts that are periodically repeated, while incremental drifts are a sequence of sudden or gradual drifts that constitute a single change.
- *F2.* Deal with different *distributions of traces* during the occurrence of gradual changes such as linear, exponential, random, or constant.
- *F3.* Detect gradual changes with a *low delay*, so a change can be precisely localized in time.
- *F4.* Have a *high accuracy*, detecting all the changes minimizing the **false positives (FPs)** and **false negatives (FNs)**.

Several authors have proposed a variety of approaches to the detection of sudden concept drift in process mining [4, 5, 8, 10, 16, 17, 25, 29, 47]. In contrast, as an exhaustive *state-of-the-art* survey on concept drift adoption [34] shows, only a few proposals allow automatic detection and classification of gradual changes, and their results still provide room for improvement. These proposals follow three main approaches for performing the drift detection: (i) extracting the trace features to abstract its behavior and then detect changes in these features [4, 18, 20, 21, 26, 29, 44]; (ii) generating process models and compute its features to detect changes [14, 36, 45]; or (iii) a combination of

Source	Trace
M_1	A B C D
M_1	A B C D
M_1	A B C D
M_1	A B C D
M_1	A B C D
M_1	A B C D
M_2	A B D C
M_2	A B D C
M_2	A B D C
M_2	A B D C
M_2	A B D C
M_2	A B D C

(a) Sudden drift

Source	Trace
M_1	A B C D
M_1	A B C D
M_1	A B C D
M_2	A B D C
M_1	A B C D
M_1	A B C D
M_2	A B D C
M_2	A B D C
M_1	A B C D
M_2	A B D C
M_2	A B D C
M_2	A B D C

(b) Gradual drift

Source	Trace
M_1	A B C D
M_1	A B C D
M_1	A B C D
M_1	A B C D
M_2	A B D C
M_2	A B D C
M_2	A B D C
M_2	A B D C
M_1	A B C D
M_1	A B C D
M_1	A B C D
M_1	A B C D

(c) Recurrent drift

Source	Trace
M_1	A B C D
M_1	A B C D
M_1	A B C D
M_2	A B D C
M_2	A B D C
M_2	A B D C
M_3	A B D C E
M_3	A B D C E
M_3	A B D C E
M_4	A D C E
M_4	A D C E
M_4	A D C E

(d) Incremental drift

Fig. 1. Types of concept drift based on their occurrence over time.

both approaches [23, 24, 37]. However, these approaches present some drawbacks. Some of them address either sudden or gradual drifts, but not both at the same time [4, 26], while other ones do not even distinguish between sudden and gradual drifts [14, 18, 20, 21, 29, 37]. In other cases, only a subset of the possible change patterns is detected [36], or the detection must be manually confirmed by the user [45]. Furthermore, some approaches support only one distribution of traces during the changes, usually the linear distribution [4, 23, 24]. Finally, most of these approaches have been only tested with 5 or less logs [4, 18, 21, 26, 36, 45]—only [37] and [20] have used 18 and 68 logs, respectively, in their validation—so it is difficult to assess the quality of each approach.

In this article, we present **Conformance-based gRadual drIft detEction algoRithm (*CRIER*)**, an algorithm that can detect gradual and sudden changes in event logs based on the change of conformance checking metrics—fitness and precision—over a sliding window. *CRIER* computes these metrics for the successive sliding windows from the starts until the end of the log. The evolution of these conformance metrics is evaluated using linear regressions and hypothesis testing, so that a potential change is detected when the metric values change significantly. In order to identify gradual drifts, detected change points are analyzed. If the behavior contained between two consecutive points is a combination of the behavior present before the first point—detected as a change in fitness—and the behavior after the second one—detected as a change in precision—then the change is classified as gradual.

The main contributions of the proposed approach are:

(1) *The formal demonstration of the impact of gradual drifts on fitness and precision metrics*. We demonstrated that a gradual drift in the control flow of a process inevitably leads to two distinct effects: (i) a drop in fitness at the onset of the drift, caused by the emergence of new behavior not previously supported by the model, and (ii) a decline in precision towards the end of the drift, due to the disappearance of previously observed behavior that is no longer present. This finding verifies the soundness of our approach by confirming that these predictable conformance patterns can be systematically detected and correlated with process model drift.

(2) *The design of an algorithm for gradual drift detection using conformance metrics combined with regression analysis and statistical testing*. Our approach leverages conformance metrics calculated from a process model discovered directly from the event data to detect changes. By applying regression analysis and statistical tests to these metrics, we identify drifts based

on the expectation that the slope should remain close to zero in the absence of changes. Regardless of the exact values of the metrics, their stability around a consistent value indicates process stability, while deviations signal potential drifts. This provides a robust, automated method for detecting process changes without depending on any specific change pattern found in the log.

CRIER has been tested using a dataset with 120 synthetic event logs with different traces distributions. The results show that our approach improves significantly the scores obtained by the main approaches of the *state-of-the-art*, making possible the treatment of sudden and gradual drifts in logs without any user intervention.

The remainder of the article is structured as follows. In Section 2 we present the main approaches to gradual concept drift detection in process mining from the *state-of-the-art*. In Section 3 we define a set of concepts that define the framework of the approach. In Section 4 we present the formal proof of the hypothesis behind the proposed algorithm. In Section 5 we detail how *CRIER* is able to detect both sudden and gradual changes automatically. In Section 6 we present the validation of our approach and how it outperforms the main algorithms from the literature. Finally, in Section 7 we present our conclusions and outline our future work.

2 Related Work

Concept drift is a traditional problem in descriptive analysis that studies the change over time of a target variable in order to minimize the error prediction. Traditional change detection algorithms, such as ADWIN [2], DDM [11], ECCD [32], or more recent approaches like Type-LDD [46], are designed to work with numeric time series, not being directly applicable in the context of process mining due to the inherent complexity of the input data: Execution traces composed of a sequence of process activities performed by a set of resources. Therefore, in the following, we only explore approaches designed specifically to detect changes in process models through the data generated during the process execution. Also, we will only focus on the drift detection task and not on the adaptation to the detected changes, so approaches like the ones presented in [30] or [31] are also out of the scope of this work. In the field of process mining, concept drift is identified as an area of significant interest due to the changing nature of business processes [40]. However, most of the existing approaches focus on the detection of sudden changes, paying little attention to other types of changes. An exhaustive survey on the topic [34] identifies a total of 36 scientific publications and 22 different approaches of which only 5 can handle gradual changes. This trend has continued in recent years.

A common approach to gradual change detection in process mining is the transformation of logs into time series by extracting different features from the traces. These features make it possible to obtain useful information contained in the traces, simplifying the data to be processed. Based on this idea, in [4], authors present an approach that computes features from follow/precede relations and use a statistical hypothesis test to check if there are changes in the features over time, specifically over two consecutive fixed size windows. Examples of these features are: the number of activities that always, sometimes and never follow/precede a given activity; the number of sequences of size n that start with an activity α_1 and contain an activity α_2 in each trace; the significance of an activity α_1 following/preceding an activity α_2 with a maximum distance of n in each trace; and so on. The main drawback of this approach is that it does not distinguish between sudden and gradual changes and it deals only with linear distributions for gradual changes. Additionally, it requires the user to have an advanced knowledge of the process, including which features to select, the activities that can change, or the statistical test to be used in order to process the log. Furthermore, the approach is not extensively tested, being validated only with two synthetic logs and a real one,

without providing any quantitative assessment of the results, although they seem to be promising. An extension to this approach is presented by [26], where the authors propose the use of non-consecutive windows, leaving a gap between them, in order to increase the difference between features in gradual environments in order to reduce the FNs. In this extension, the restriction of gradual drifts distribution is removed, so the algorithm can detect also changes that are not linear. Yet, the approach fails to provide a solution able to identify if a change is sudden or gradual, requires even more knowledge from the user to set other parameters—as the size of the gap between the windows—and still does not provide quantitative metrics in the validation, which is performed only over two logs.

Similarly, in [28], authors use a window-based strategy to sample data over time. For each window, process features describing the process behavior over time are extracted, such as the transition probability between two activities, the transition frequency, the transition average time, and so on. The values of each process feature are represented in an $n \times n$ transition matrix, where n is the number of activities and each value refers to a transition between two activities according to the event log. Then, all transition matrices are combined into a single transformed transition matrix containing all process features. The transformed transition matrices of two different windows are compared using various strategies—as statistical tests or thresholds—detecting a concept drift when significant differences can be observed in the values of one or more process features. However, this approach does not distinguish between sudden and gradual changes, and it requires *a priori* knowledge of the process by the user to select process features and parameters such as window size or thresholds. Moreover, although the approach demonstrates promising results in terms of F-score, it has been compared only with a baseline based on adaptive windows [24]. In [29], authors extend the experimentation of the same approach, including more logs from different processes. Nevertheless, the focus remains on the detection of sudden changes and comparisons are made only with the same baseline as in [28].

In [18] the log is transformed into a time series using a feature called *r-measure* that compares the relation matrices between two consecutive windows. Then, the algorithm looks for outliers in these time series, and determines that a change exists when one is detected. As the previous proposals, the main drawback of this approach is that it does not distinguish sudden and gradual drifts and it is prone to mix up changes with outliers—abnormal executions. Moreover, it is also not extensively tested, being evaluated against four synthetic logs, where they obtain very promising results in terms of F_{score}, with values between 0.6 and 0.95 depending on the used parameters.

In [20] the log is transformed in a stream of events and a reference window is built. Each time the reference window moves, a new event is peeked, and if it introduces new behavior, it is treated as a candidate drift point. The process behavior is represented through directly-follows relations, applying statistical tests when a candidate drift point is detected. If the new behavior causes a significant difference, then the candidate drift point is marked as an actual drift point. Although it is a robust approach to the noise in event logs, it does not distinguish between sudden or gradual changes. Moreover, the specific parameter settings employed can result in significantly diverse detection results; hence it is necessary to manually configure the values for each event log. The validation was conducted on 68 synthetic event logs, resulting in F-score values between 0.85 and 0.88.

In [21], the authors propose the use of an agglomerative hierarchical clustering over the traces to detect changes. For the clustering, traces are transformed to feature vectors that abstract the trace behavior. Specifically, the maximal repeat [3] and the starting timestamp of the trace are used as features. Once the clusters are generated, they define a change as the point in time where one cluster ends and a new one starts. However, this method is very dependent on the number of clusters, which should be fixed by the user and equal to the number of changes present in

the log. Regarding the experimentation, the approach is only validated using three logs, with an accuracy—calculated as the sum of **true positives (TPs)** and true negatives, and divided by the total number of traces—between 57% and 100%.

In [44], an initial set of traces is selected and activity relationship pairs are extracted from this set, called **initial baseline (*IB*)**. Then the activity relationship matrix is constructed and the behavior of subsequent traces is compared to the *IB*. If some features present in *IB* appear or disappear in the current process, it is considered that there is a drift and the *IB* is updated to repeat the procedure again. The drifts detected are classified as sudden changes, gradual changes, or a combination of both—what the authors refer to as nested changes—using *behavior replacement*. Although the results in terms of F-score seem promising, no quantitative values are provided, and the experimentation is performed on only three synthetic logs of different sizes from the same process. Furthermore, it is necessary to manually define certain parameters, such as the size of the *IB* set or the number of traces used to compare the behavior of the current process.

An alternative approach to gradual change detection is to discover process models from the event log over time. Each model discovered represents the behavior of the process up to a specific point in time, allowing the identification of change points. This approach is used in [36], where an algorithm that deals with a stream of events is proposed, allowing the detection of changes in incomplete traces. Using a window of a given size, different versions of the process model are discovered—the *process history*—and the fitness is computed against the last mined model. In this approach, a change is present when the fitness of a trace is below a threshold. Thus, when a trace does not fit the last discovered model, a new one is mined using only the unfitting traces. To prevent FPs, each model also receives a score, and a model is considered to change only when that score is over a threshold. Finally, changes are classified in sudden, gradual, recurring and incremental using both two new thresholds and the process history. The main drawback of this approach is that it requires the user to have a deep knowledge of the problem in order to tune the hyperparameters used in the detection—the window size and four different thresholds. Furthermore, the algorithm cannot detect all structural change patterns, e.g., the transformation of an optional parallel structure to an exclusive choice where the order of some activities is enforced. Also, the approach has not been extensively tested and no quantitative measures are provided about the goodness of the results.

In [45] authors propose a method that is also based on the idea of using the process model for detecting changes. This approach splits the log in n windows of size m. Then, a set of declarative rules is extracted from the complete log and their confidence is computed with respect to each window, obtaining a multivalued time series. Finally, a traditional concept drift detection algorithm—namely, the PELT algorithm—in combination with a hierarchical clustering technique is used to detect changes in the resulting time series. The main drawback with this approach is that the classification of changes in sudden or gradual is not automatic, but it must be performed visually by the user. Regarding the validation, the algorithm shows promising results for the detection, but it is only tested with four synthetic and two real logs.

An alternative approach to discovering the process model, based on trees, is presented in [14]. This method extracts event data from a window of observation and stores it in an efficient data structure called *prefix-trees*, which are then converted into continuous values using PCA-based methods. The distribution of the data between two windows is compared using ADWIN to detect concept drifts. Once again, the main drawback with this approach is that it does not distinguish between sudden and gradual changes. Moreover, it requires knowledge from the user to set parameters such as the number of trees per window or the number of events per tree. The experimentation is performed on six synthetic event logs, without including all types of changes.

Another very interesting proposal, halfway between the extraction of features and the use of a process model, is the one presented by [23, 24]. In this approach, the behavior of the traces is

abstracted using *partial-ordered-runs*—i.e., a graph representation of the traces. Once the behavior is abstracted, two consecutive sliding windows are used and, by means of a statistical test, it is checked if the content of these windows is significantly different. Once the changes are detected, a classification is performed to guess if they represent a sudden or a gradual drift. In this classification, a statistical test checks if the combination of traces between two consecutive changes represents a linear combination of the traces before and after the first and second changes, respectively. This is one of the most thoroughly validated proposals, tested with multiple logs and evaluated using well-known metrics, as F_{score} and *delay*. The main drawback of this approach is that it can be very sensitive to changes in the frequencies of the relations, which may lead to the detection of FPs. Also, the algorithm can only detect gradual changes that are due to a linear distribution of the traces between two models, which may not be the case in real logs.

Finally, [37] propose an online approach where they use a clustering approach over graph distances. The approach takes as input an event stream, and updates the corresponding trace graph every time a new event is received. When they have enough traces, a process model is discovered by generating two weighted graphs: In one of them, the weight of the arcs represents the frequency of the transitions between activities; and in the other one, the average time between activities is the arc weight. Then, a distance between the trace graph and these two weighted graphs is computed, and these distances are grouped using an online density-based clustering algorithm—namely, DenStream. Regarding the validation, the approach is tested using 18 synthetic logs, but the only metric provided is the number of changes detected. The main drawback of this method is that it mixes control flow and behavioral changes. Also, it requires a lot of hyperparameters to be tuned by the user in order to obtain good results. Finally, as many of the presented proposals, it supports the detection over logs with gradual changes, but does not distinguish sudden from gradual changes.

Table 1 summarizes the pros and contras of the main *state-of-the-art* proposals. A more in-depth comparison of the approaches can be found in [34]. As a summary, there is no approach that provides a fully automated solution for detecting sudden and gradual changes, supporting all types of trace distributions and with an exhaustive validation that demonstrates its high accuracy and low delay. *CRIER* is designed to overcome these problems, minimizing FPs and negatives in change detection while maximizing the coverage of the detected change region with acceptable delay values. To achieve this, we propose the utilization of a process model discovered directly from the observed executions, ensuring that it accurately reflects the behavior recorded in the event log. This model is monitored using fitness and precision metrics, allowing for the detection of changes when the model quality degrades. The use of a process model and conformance metrics enables the capture of the inherent complexity of the traces without oversimplifying or making assumptions about the event log behavior, thereby allowing verification that subsequent executions conform to the expected.

3 Preliminaries

In this section, we introduce some terms necessary to understand the concept drift detection problem and our approach. The proposed method takes an event log as input and tries to detect changes in the observed behavior.

Definition 3.1 (Event). Given the set of activities $\mathcal{A}$ that conform a process, an event ε can be defined as the execution of an activity $\alpha \in \mathcal{A}$ at a given instant t in the context of a process instance c. The activity name $\varepsilon.a$, the timestamp $\varepsilon.t$ and the process instance identifier—often referred to as *case*—$\varepsilon.c$ are the only mandatory attributes of an event, which can also have other generic attributes, such as the resources that perform the activities, or domain-specific attributes, understood as variables whose values are modified in the activity execution.

Table 1. Approaches for Gradual Drift Detection and Their Performance from the Accuracy and Delay Perspectives

	Approach	Pros	Cons
Li et al. [18]	*Model-to-log alignment*	Promising accuracy scores	Tested only with four logs. Prone to mixing up changes with outliers
Stertz and Rinderle-Ma [36]	*Model-to-log alignment*	Can detect changes even in incomplete traces. Classifies all types of drifts	Fails to detect some control flow change patterns. Lacks an extensive validation. No quantitative quality measurements are reported
Yeshchenko et al. [45]	*Model-to-log alignment*	Promising accuracy scores	Does not provide an automated solution but requires the visual identification of the drifts by the user. Lacks of an extensive validation
Xu et al. [44]	*Model-to-log alignment*	Automatically classifies sudden and gradual drifts. Promising accuracy scores	The features used for building the *IB* can overlook some change patterns. Lacks of an extensive validation
Lu et al. [20]	*Model-to-log alignment*	Tested over a variety of logs. Promising accuracy scores	Does not distinguish between sudden and gradual drifts. Hard to fine-tune
Luengo and Sepúlveda [21]	*Clustering*	Robust to the presence of noise	Requires the user to specify the number of drifts to find. Lack of an extensive validation
Tavares et al. [37]	*Clustering*	Robust to the presence of noise	Does not distinguish between sudden and gradual drifts. Mixes control flow and behavioral drifts. Needs a lot of hyperparameters to be manually tuned
Bose et al. [4]	*Windowing plus statistical analysis*	The use of multiple different features allows for adapting to specific change patterns	Does not distinguish between sudden and gradual drifts. Only deals with linear distributions for gradual changes
Martjushev et al. [26]	*Windowing plus statistical analysis*	Increased robustness in gradual drift detection by using non-contiguous windows	Does not distinguish between sudden and gradual drifts. Requires a deep knowledge about the process to fine tune the parameters
Maaradji et al. [23, 24]	*Windowing plus statistical analysis*	Thoroughly validated. Automatically classifies drifts in sudden or gradual	Can be very sensitive to changes in frequencies. When detecting gradual drifts, only deals with linear changes
Neto et al. [28, 29]	*Windowing plus statistical analysis*	Promising accuracy scores. The use of multiple different features allows for adapting to specific change patterns	Does not distinguish between sudden and gradual drifts. Requires a deep knowledge about the process choose the correct features
Huete et al. [14]	*Windowing plus statistical analysis*	The use of PCA methods should increase the robustness of the approach in noisy environments	Does not distinguish between sudden and gradual drifts. Lacks of an extensive validation. Requires a deep knowledge about the process choose the correct features

Definition 3.2 (Trace). Given the full set of events $\mathcal{E}$ recorded from the execution of a process, a trace τ can be defined as the ordered sequence of all the events belonging to the same process instance $\tau.c$, where the order is defined by the timestamp of the events:

$$\tau = \langle \varepsilon_0, \ldots, \varepsilon_n \rangle : \forall i \in [0, n),\ \forall j \in (i, n],\ \varepsilon_i, \varepsilon_j \in \mathcal{E} \rightarrow (\varepsilon_i.c = \varepsilon_j.c) \wedge (\varepsilon_i.t < \varepsilon_j.t).$$

We denote as B_τ the behavior observed in a trace τ, represented as an ordered sequence of activities:

$$B_\tau = \langle \alpha_0, \ldots, \alpha_n \rangle : \forall \epsilon_i \in \tau \rightarrow \alpha_i = \epsilon_i.a.$$

Definition 3.3 (Log). A log can be defined as a sequence of traces $L = \langle \tau_0, \ldots, \tau_n \rangle$ where each trace represents a different process instance $\nexists \tau_i, \tau_j \in L :\ \tau_i.c = \tau_j.c$ and the order is given by the timestamp of the last event in each trace. We denote as $B_L = \{B_{\tau_0}, \ldots, B_{\tau_n}\}$ the behavior observed in the log L. This behavior is composed by the set of distinct behaviors captured in the traces of the log. Two different traces τ_i and τ_j such that $\tau_i.c \neq \tau_j.c$ can have the same behavior $B_{\tau_i} = B_{\tau_j}$ if the ordered sequence of activities for τ_i is the same as for τ_j. The size of B_L will always be smaller than or, at most, equal to the size of the log $|B_L| \leq |L|$.

Figure 2 shows an example of an event log with 3 different traces, 15 events, 5 activities, and 6 resources. In this example, the behavior observed in the log B_L is

$$B_L = \left\{ \begin{array}{l} \langle \text{Lock feature, Check restrictions, Build part, Integration test, Quality test} \rangle \\ \langle \text{Lock feature, Interview customer, Build part, Quality test, Integration test} \rangle \end{array} \right\},$$

being the first of the sequences the behavior observed in cases #aaa and #aab and the second one the behavior for the case #aac.

Case	Timestamp	Activity	Resource
#aaa	01/10/2021 08:01	Lock feature	Phoebe
#aaa	01/10/2021 08:53	Check restrictions	Phoebe
#aab	01/10/2021 11:40	Lock feature	Rachel
#aac	01/10/2021 09:12	Lock feature	Ross
#aac	01/10/2021 09:33	Interview customer	Ross
#aac	01/10/2021 11:48	Build part	Ross
#aab	01/10/2021 11:49	Check restrictions	Rachel
#aaa	01/10/2021 08:57	Build part	Phoebe
#aab	01/10/2021 16:18	Build part	Rachel
#aac	01/10/2021 12:16	Quality test	Monica
#aaa	01/10/2021 13:45	Integration test	Chandler
#aab	01/10/2021 17:23	Integration test	Joey
#aaa	01/10/2021 13:37	Quality test	Monica
#aac	01/10/2021 16:22	Integration test	Joey
#aab	01/10/2021 17:35	Quality test	Monica

Fig. 2. Example of a log with 15 events belonging to 3 different traces. It records the execution of 5 different activities by 6 resources.

The relations and the dependencies between the different activities that conform a process can be represented using a *process model*, which can be represented with a directed graph that ideally describes the behavior observed in the execution log. In this article, we use Petri nets to represent the process models because they provide, simultaneously, a powerful mathematical formalism and a simple graphical representation of these models. However, the approach is not tied to this formalism, so any other language capable of capturing the semantics of the model could be employed.

Definition 3.4 (Labeled Petri Net, Workflow Net). A labeled Petri net is a bipartite graph that can be defined by a tuple $N = (P, T, F, \lambda)$, where:

- P is a finite set of places;
- T is a finite set of transitions;
- $P \cap T = \emptyset$;
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs; and
- $\lambda : T \to \mathcal{A}$ is a function that assigns to each transition $t \in T$ an activity $a \in \mathcal{A} \cup \emptyset$.

Let $x \in T \cup P$ be a node from the Petri net. We denote ${}^\bullet x = \{y \mid (y, x) \in F\}$ as the set of inputs of x, and $x^\bullet = \{y \mid (x, y) \in F\}$ as the set of outputs of x. A labeled Petri net is a *workflow net* [38] if and only if:

- $\exists! in \in P : {}^\bullet in = \emptyset$, i.e., there exists a single place in the net with no inputs;
- $\exists! out \in P : out^\bullet = \emptyset$, i.e., there exists a single place in the net with no outputs;
- Adding to the net a transition t^* such that ${}^\bullet t^* = out$ and $t^{*\bullet} = in$ results in a strongly connected graph, i.e., all $t \in T$ are in a path from *in* to *out*.

The state of a Petri net, namely a *marking*, is a function $m : P \to \mathbb{N}$ that indicates the number of tokens contained in the place $p \in P$. In a workflow net, the initial marking, denoted as m_0, is the one in which only *in* contains a token. Let $t \in T$ be a transition from the Petri net. We say t is *enabled* when $\forall p \in {}^\bullet t \to m(p) > 0$, i.e., when all its input places have at least one token. When an enabled transition is fired, it consumes a token from each input place and produces a new token in each output place.

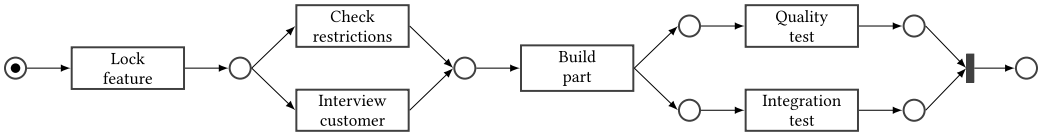


Fig. 3. Petri net capturing the behavior observed in the log from Figure 2.

We call B_N to the set of all possible paths—i.e., all the unique possible sequences of transition labels—from *in* to *out* in the workflow net N. For the rest of this article, when referring to Petri nets, we assume we are talking about workflow nets. Figure 3 shows an example Petri net, where circles represent places, rectangles represent transitions, and a black dot represents a token. In this example, the behavior of the model is

$$B_N = \left\{ \begin{array}{l} \langle \text{Lock feature, Check restrictions, Build part, Quality test, Integration test} \rangle \\ \langle \text{Lock feature, Check restrictions, Build part, Integration test, Quality test} \rangle \\ \langle \text{Lock feature, Interview customer, Build part, Quality test, Integration test} \rangle \\ \langle \text{Lock feature, Interview customer, Build part, Integration test, Quality test} \rangle \end{array} \right\}.$$

The quality of a process models can be evaluated with respect to a log using *conformance checking metrics*. In this article, we will use two of these metrics as the basis of the algorithm for detecting gradual drifts: *fitness* and *precision*.

Definition 3.5 (Fitness Metric). Fitness measures the fraction of behavior observed in the log that is captured by the model [6]:

$$\gamma(L, N) = \frac{|B_L \cap B_N|}{|B_L|}.$$

In the *state-of-the-art* have been proposed a multitude of techniques for fitness computation [27], that vary in determining the degree of compliance of the model with respect to the log traces [1, 7, 39]. For example, a model with a behavior that only differs in a single activity with respect to a trace with eight activities can have a fitness of 0 if the metric only considers perfect matches or it can have a fitness of 7/8 if the metric accounts for the activities that do not deviate from the behavior supported by the model. In this article, we have opted for a metric that computes the percentage of log traces that are fully supported by the model:

$$\gamma(L, N) = \frac{|\{\tau : \tau \in L \wedge B_\tau \in B_N\}|}{|L|}.$$

For example, considering the log and the model from Figures 2 and 3, respectively:

$$\gamma(L, N) = \frac{|L|}{|L|} = 1$$

because all traces from the log are supported by the model. This metric is quite restrictive when calculating the fitness, but for concept drift detection we just need an estimator of how good the model captures the behavior observed in the log. As a counterpart, by using this metric we obtain a high performance, which is important since it will be calculated repeatedly as the log is processed.

Definition 3.6 (Precision Metric). Precision measures the fraction of allowed behavior that is observed in the log [6]:

$$\rho(L, N) = \frac{|B_L \cap B_N|}{|B_N|}.$$

In the case of precision, many implementations have been also proposed in the *state-of-the-art* [7, 33, 39, 42]. Although we could have used any of those implementations, in *CRIER* we consider a custom metric as an estimator of the precision evolution since our algorithm only needs to know whether the precision has changed or not—i.e., it does not need to know the exact value of the precision. Specifically, this metric checks the percentage of direct relations between activities from the model observed in the traces:

$$\rho(L, N) = \frac{\left|\overset{L}{\longrightarrow} \cap \overset{N}{\longrightarrow}\right|}{\left|\overset{N}{\longrightarrow}\right|},$$

where $\overset{L}{\longrightarrow}$ are the pairs of consecutive activities found in the traces, and $\overset{N}{\longrightarrow}$ are the directly connected pairs of activities in the model. For example, using the log and the model from Figures 2 and 3, respectively:

$$\overset{L}{\longrightarrow} = \left\{\begin{array}{l}(Lock\ feature \rightarrow Chech\ restrictions),\\ (Lock\ feature \rightarrow Interview\ customer),\\ (Chech\ restrictions \rightarrow Build\ part),\\ (Interview\ customer \rightarrow Build\ part),\\ (Build\ part \rightarrow Integration\ test),\\ (Build\ part \rightarrow Quality\ test),\\ (\text{Integration test} \rightarrow \text{Quality test}),\\ (\text{Quality test} \rightarrow \text{Integration test})\end{array}\right\}, \overset{N}{\longrightarrow} = \left\{\begin{array}{l}(Lock\ feature \rightarrow Chech\ restrictions),\\ (Lock\ feature \rightarrow Interview\ customer),\\ (Chech\ restrictions \rightarrow Build\ part),\\ (Interview\ customer \rightarrow Build\ part),\\ (Build\ part \rightarrow Integration\ test),\\ (Build\ part \rightarrow Quality\ test)\end{array}\right\},$$

and, therefore, $\rho(L, N) = 6/6 = 1$. But if the relation $(Lock\ feature \rightarrow Chech\ restrictions)$ is not observed in the log, $\rho(L, N) = 5/6 = 0.83$, showing that the estimator decreases when the precision also decreases.

For a more extensive analysis of different conformance metrics and their performance in drift detection, please check [10]. In this article, the authors empirically found that the use of simplified metrics in combination with the Inductive Miner discovery algorithm not only achieved the best values but also significantly reduced the execution time, resulting in the best tradeoff between results and performance.

To capture the evolution of the process over time we use a sliding window, which is a sublog containing only the most recent traces at a given instant, being one of the main methods used as a basis for concept drift detection algorithms [2].

Definition 3.7 (Sliding Window). The sliding window of size n over a log L is defined as $\omega_i(L, n) = \langle \tau_{i-n}, \ldots, \tau_i \rangle$, where its content is conformed by the most recent n traces present in the log at instant i. When a new trace is read from the log, the window slides one position, so the oldest trace is forgotten and the new trace is incorporated. This way, the algorithm only takes into account the most recent information for detecting the changes.

The main objective of this article is the identification of changes in the process structure over time and its classification in sudden and gradual drifts. One of the greatest challenges is differentiating between a real change and an anomalous execution. To accomplish this objective and address this challenge, we use the concept of drift candidate.

Definition 3.8 (Drift Candidate). A drift candidate can be defined as a potential change in the structure of the process that has to be confirmed later. Given two consecutive windows ω_i and ω_{i+1}, and two process models $N_i = (P_{N_i}, T_{N_i}, F_{N_i}, \lambda_{N_i})$ and $N_{i+1} = (P_{N_{i+1}}, T_{N_{i+1}}, F_{N_{i+1}}, \lambda_{N_{i+1}})$ discovered from each of these windows using the same discovery algorithm, we say that window ω_i is a drift candidate when $T_{N_i} \neq T_{N_{i+1}} \vee F_{N_i} \neq F_{N_{i+1}}$. A drift is confirmed only after several successive windows are marked as drift candidates and, in that moment, the change is pinpointed to the specific trace that triggered the change—the first trace of the first candidate in the case of fitness, or the last trace of the first window in the case of precision.

Although there are several types of concept drifts, in this article we focus on the detection of gradual drifts and on the distinction with sudden drifts.

Definition 3.9 (Gradual Drift, Sudden Drift). A gradual drift is defined as a change where the behavior before the change does not disappear suddenly, but coexists with the one after the change for a period of time, while vanishing until it is no longer observed. Given two time instants t_1 and t_2 that bound the change, two models can be discovered, $N_{<t_1}$ and $N_{>t_2}$, that capture the behavior before and after those instants. For a change to be considered as gradual, four conditions must be satisfied:

(1) Behavior captured by the models discovered before t_1 and after t_2 are different: $B_{N_{<t_1}} \neq B_{N_{>t_2}}$.
(2) Some of the behavior observed between t_1 and t_2 is captured by the model discovered before t_1: $B_{L_{[t_1,t_2]}} \cap B_{N_{<t_1}} \neq \emptyset$.
(3) Some of the behavior observed between t_1 and t_2 is captured by the model discovered after t_2: $B_{L_{[t_1,t_2]}} \cap B_{N_{>t_2}} \neq \emptyset$.
(4) All of the behavior observed between t_1 and t_2 is captured either by the model discovered before t_1, by the model discovered after t_2 or by both: $B_{L_{[t_1,t_2]}} \subseteq (B_{N_{<t_1}} \cup B_{N_{>t_2}})$.

If any of these conditions are not met, the change is considered a sudden change, where the old behavior is replaced instantly by a new one.

4 Gradual Drift Detection Using Conformance Checking

A gradual change is defined between two instants t_1 and t_2, where the process has a behavior B_1 before t_1, a different behavior B_2 after t_2, while B_1 and B_2 coexist in $[t_1, t_2]$ (see Definition 3.9).

Theorem 4.1. *All gradual drifts are characterized by a fitness change in t_1 and by a precision change in t_2.*

Proof. Consider a log L, with a gradual change between instants t_1 and t_2, in which the new behavior is first observed at t_1 and some old behavior is less and less frequently observed, until it disappears completely at t_2. Let us consider three disjoint sets of behavior:

— B_c, which contains the behavior that appears throughout the whole duration of the log L and which can be empty;

— B_p, which contains the previous behavior of L that disappears between t_1 and t_2 and which cannot be empty; and
— B_n, which contains the new behavior that starts to appear after t_1 and replaces B_p after t_2 and which also cannot be empty.

Using these two instants t_1 and t_2, we can split the log L in three sublogs. Let $L_{<t_1}$ be the log containing all the traces before t_1, $L_{[t_1,t_2]}$ be the log containing the traces between t_1 and t_2, and $L_{>t_2}$ be the log with the remaining traces after t_2. Similarly, we can define the corresponding behaviors for these logs as $B_{L_{<t_1}} = B_c \cup B_p$, $B_{L_{[t_1,t_2]}} = B_c \cup B_p \cup B_n$, and $B_{L_{>t_2}} = B_c \cup B_n$, respectively, and the reference models $N_{<t_1}$, $N_{[t_1,t2]}$, and $N_{>t_2}$ that can be discovered from the traces of these logs, respectively. The values for fitness and precision before the change, i.e., $L_{<t_1}$, can be computed as follows:

$$\gamma(L_{<t_1}, N_{<t_1}) = \frac{|B_{L_{<t_1}} \cap B_{N_{<t_1}}|}{|B_{L_{<t_1}}|} = \frac{|(B_c \cup B_p) \cap B_{N_{<t_1}}|}{|B_c \cup B_p|} = \frac{|(B_c \cap B_{N_{<t_1}}) \cup (B_p \cap B_{N_{<t_1}})|}{|B_c \cup B_p|}$$
$$\rho(L_{<t_1}, N_{<t_1}) = \frac{|B_{L_{<t_1}} \cap B_{N_{<t_1}}|}{|B_{N_{<t_1}}|} = \frac{|(B_c \cup B_p) \cap B_{N_{<t_1}}|}{|B_{N_{<t_1}}|} = \frac{|(B_c \cap B_{N_{<t_1}}) \cup (B_p \cap B_{N_{<t_1}})|}{|B_{N_{<t_1}}|},$$

which can be simplified since B_c and B_p are disjoint sets:

$$\gamma(L_{<t_1}, N_{<t_1}) = \frac{|B_c \cap B_{N_{<t_1}}| + |B_p \cap B_{N_{<t_1}}|}{|B_c| + |B_p|} = \frac{|B_c \cap B_{N_{<t_1}}|}{|B_c| + |B_p|} + \frac{|B_p \cap B_{N_{<t_1}}|}{|B_c| + |B_p|}$$
$$\rho(L_{<t_1}, N_{<t_1}) = \frac{|B_c \cap B_{N_{<t_1}}| + |B_p \cap B_{N_{<t_1}}|}{|B_{N_{<t_1}}|} = \frac{|B_c \cap B_{N_{<t_1}}|}{|B_{N_{<t_1}}|} + \frac{|B_p \cap B_{N_{<t_1}}|}{|B_{N_{<t_1}}|}.$$

Similarly, fitness and precision values in the change interval $[t_1, t_2]$ with respect to this same model $N_{<t_1}$ can be computed as follows:

$$\begin{aligned}\gamma(L_{[t_1,t_2]}, N_{<t_1}) &= \frac{|B_{L_{[t_1,t_2]}} \cap B_{N_{<t_1}}|}{|B_{L_{[t_1,t_2]}}|} = \frac{|(B_c \cup B_p \cup B_n) \cap B_{N_{<t_1}}|}{|B_c \cup B_p \cup B_n|} \\ &= \frac{|B_c \cap B_{N_{<t_1}}|}{|B_c| + |B_p| + |B_n|} + \frac{|B_p \cap B_{N_{<t_1}}|}{|B_c| + |B_p| + |B_n|} + \frac{|B_n \cap B_{N_{<t_1}}|}{|B_c| + |B_p| + |B_n|} \\ \rho(L_{[t_1,t_2]}, N_{<t_1}) &= \frac{|B_{L_{[t_1,t_2]}} \cap B_{N_{<t_1}}|}{|B_{N_{<t_1}}|} = \frac{|(B_c \cup B_p \cup B_n) \cap B_{N_{<t_1}}|}{|B_{N_{<t_1}}|} \\ &= \frac{|B_c \cap B_{N_{<t_1}}|}{|B_{N_{<t_1}}|} + \frac{|B_p \cap B_{N_{<t_1}}|}{|B_{N_{<t_1}}|} + \frac{|B_n \cap B_{N_{<t_1}}|}{|B_{N_{<t_1}}|}.\end{aligned}$$

This equation can be simplified since B_n only starts to appear after t_1, i.e., $B_{N_{<t_1}} \cap B_n = \emptyset$:

$$\gamma(L_{[t_1,t_2]}, N_{<t_1}) = \frac{|B_c \cap B_{N_{<t_1}}|}{|B_c| + |B_p| + |B_n|} + \frac{|B_p \cap B_{N_{<t_1}}|}{|B_c| + |B_p| + |B_n|}$$
$$\rho(L_{[t_1,t_2]}, N_{<t_1}) = \frac{|B_c \cap B_{N_{<t_1}}|}{|B_{N_{<t_1}}|} + \frac{|B_p \cap B_{N_{<t_1}}|}{|B_{N_{<t_1}}|}.$$

Moreover, since $B_p \neq \emptyset$ and $B_n \neq \emptyset$, $|B_c| + |B_p| + |B_n| > |B_c| + |B_p|$. Thus, $\gamma(L_{<t_1}, N_{<t_1}) > \gamma(L_{[t_1,t_2]}, N_{<t_1})$ but $\rho(L_{<t_1}, N_{<t_1}) = \rho(L_{[t_1,t_2]}, N_{<t_1})$, confirming our hypothesis that fitness decreases at the beginning of a gradual drift, but precision remaining unaltered.

Once the beginning of the drift has been detected, a new model $N_{[t_1,t_2]}$ is discovered using the log $L_{[t_1,t_2]}$. Then, the conformance metrics can be computed using Definitions 3.5 and 3.6 as follows:

$$\gamma(L_{[t_1,t_2]}, N_{[t_1,t_2]}) = \frac{|B_{L_{[t_1,t_2]}} \cap B_{N_{[t_1,t_2]}}|}{|B_{L_{[t_1,t_2]}}|} = \frac{|B_c \cap B_{N_{[t_1,t_2]}}|}{|B_c| + |B_p| + |B_n|} + \frac{|B_p \cap B_{N_{[t_1,t_2]}}|}{|B_c| + |B_p| + |B_n|} + \frac{|B_n \cap B_{N_{[t_1,t_2]}}|}{|B_c| + |B_p| + |B_n|}$$
$$\rho(L_{[t_1,t_2]}, N_{[t_1,t_2]}) = \frac{|B_{L_{[t_1,t_2]}} \cap B_{N_{[t_1,t_2]}}|}{|B_{N_{[t_1,t_2]}}|} = \frac{|B_c \cap B_{N_{[t_1,t_2]}}|}{|B_{N_{[t_1,t_2]}}|} + \frac{|B_p \cap B_{N_{[t_1,t_2]}}|}{|B_{N_{[t_1,t_2]}}|} + \frac{|B_n \cap B_{N_{[t_1,t_2]}}|}{|B_{N_{[t_1,t_2]}}|}.$$

We can then proceed to define the values after t_2, where B_p has totally disappeared:

$$\gamma(L_{>t_2}, N_{[t_1,t_2]}) = \frac{|B_{L_{>t_2}} \cap B_{N_{[t_1,t_2]}}|}{|B_{L_{>t_2}}|} = \frac{|B_c \cap B_{N_{[t_1,t_2]}}|}{|B_c| + |B_n|} + \frac{|B_n \cap B_{N_{[t_1,t_2]}}|}{|B_c| + |B_n|}$$
$$\rho(L_{>t_2}, N_{[t_1,t_2]}) = \frac{|B_{L_{>t_2}} \cap B_{N_{[t_1,t_2]}}|}{|B_{N_{[t_1,t_2]}}|} = \frac{|(B_c \cap B_{N_{[t_1,t_2]}})|}{|B_{N_{[t_1,t_2]}}|} + \frac{|B_n \cap B_{N_{[t_1,t_2]}}|}{|B_{N_{[t_1,t_2]}}|}.$$

Nothing can be affirmed with respect to fitness, but we can conclude that $\rho(L_{[t_1,t_2]}, N_{[t_1,t_2]}) > \rho(L_{>t_2}, N_{[t_1,t_2]})$, i.e., precision will decrease at the end of the gradual drift.

A situation in which a gradual change starts with a change in precision at t_1 is impossible, since the change in precision necessarily implies a sudden change in the model structure. Since this proof is straightforward, we did not include it in this article—$B_n = \emptyset$, since no new behavior is added after t_1, otherwise a change in fitness should be present, and $B_{L_{<t1}} = B_{L_{[t1,t2]}}$, which contradicts Definition 3.8. □

This demonstration proves that, regardless of the specific control flow drift patterns present in the log, any change should be detected as long as the appropriate conformance metrics are monitored. This establishes a sound foundation for detecting process drifts, offering significant flexibility in identifying previously unseen changes.

Unlike other methods that primarily depend on features derived from directly-follows relations, *CRIER* leverages the entire process structure through a process model and its conformance metrics. This enables it to capture a broader range of control flow constructs, including concurrencies and loops, which are often misrepresented or overlooked by approaches based solely on directly-follows relations. This flexibility enhances the reliability and generalizability of our approach in real-world

dynamic environments and ensures robust detection without being constrained by predefined patterns or feature dependencies.

5 Algorithm for Gradual Drift Detection

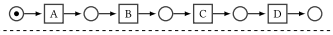
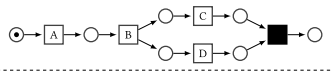
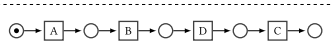
The premise underlying the operation of *CRIER* is that, using fitness and precision metrics, gradual drifts can be detected with high accuracy. However, to better understand how the algorithm works, we illustrate the gradual drift detection with the example of Figure 4. The example is based on a sliding window of size 4. First, traces are read until the window is full. Then, traces from the first full window ω_4 are mined with a discovery algorithm to obtain the model that describes its behavior. Then, the window is shifted trace by trace and for each shift the fitness and precision—which are initially 1—are checked to detect whether they change. In the window ω_9 the fitness decreases while the precision does not change, so τ_9 is labeled as a drift candidate. The fitness decrease lasts for three windows, which means that traces from τ_{10} to τ_{12} are also drift candidates, so trace τ_9 is finally labeled as a drift. As a consequence, a new model is discovered at ω_{12}, starting a new cycle of the algorithm, until a change in fitness or precision is detected in the next windows. In this example, a precision decrease is detected for the windows ω_{20}, ω_{21}, ω_{22}, and ω_{23}, confirming a drift in trace τ_{17} and, therefore, a gradual drift between traces τ_9 and τ_{17}.

The key advantage of our approach lies in its ability to capture the full expressiveness of the process model, including complex constructs such as concurrencies and loops, which are often missed by *state-of-the-art* methods relying on features derived from directly-follows relations [4, 14, 24, 26, 36, 44]. By monitoring drift candidates using conformance metrics and regression analysis, our method is robust against noise, reducing the likelihood of FPs. Furthermore, drift detection and classification are fully automated, requiring minimal user intervention after selecting the window size, unlike in other *state-of-the-art* approaches [4, 14, 20, 21, 28, 37, 45]. This level of automation significantly simplifies the deployment and use of *CRIER* in dynamic environments where continuous monitoring is essential.

5.1 Algorithm Description

Algorithm 1 shows the pseudocode of the *CRIER* algorithm. The only mandatory inputs are the event log and a minimum window size. As part of the initialization phase, the algorithm creates two empty lists for the confirmed drifts, one for sudden and one for gradual drifts (line 2) and sets the initial window index $i = 1$ (line 4). As the algorithm is based on a sliding window ω_i (Definition 3.7), in this initialization phase its optimal size n is computed (line 3) using the function AdjustWindow (Algorithm 2, lines 1–15). The adjustment of the window size is based on the comparison of the behavior observed in three consecutive windows. In this step, we start with a size n' initialized as the minimum window size n and discover three models from three consecutive windows of size n' (Algorithm 2, lines 5–7). If the three models capture the same behavior (Algorithm 2, line 8), the size of each window n' is incremented by n' (Algorithm 2, line 9) and the process starts again. The procedure finishes when one of the three models has a different behavior or when the condition $3 * n' < |L|$ fails, meaning that it is not possible to discover three consecutive models, returning the last n' as the adjusted optimal window size (Algorithm 2, line 14).

Once the size is adjusted, the window is populated with the first n remaining traces from the initial index. Then, the first process model is discovered (line 6) using the content from this ω_i window. This model is stored in a list N which contains the set of models that will be discovered for each detected drift.

ID	Trace	Window	Window fitness	Window precision	Actions	Model
τ_1	$ABCD$	$\omega_1 = \langle \tau_1 \rangle$	–	–	Window is not full. Read new trace	
τ_2	$ABCD$	$\omega_2 = \langle \tau_1, \tau_2 \rangle$	–	–	Window is not full. Read new trace	
τ_3	$ABCD$	$\omega_3 = \langle \tau_1, \tau_2, \tau_3 \rangle$	–	–	Window is not full. Read new trace	
τ_4	$ABCD$	$\omega_4 = \langle \tau_1, \tau_2, \tau_3, \tau_4 \rangle$	$\gamma(N_1, \omega_4) = 1.00$	$\rho(N_1, \omega_4) = 1.00$	Discover model for ω_4	N_1 
τ_5	$ABCD$	$\omega_5 = \langle \tau_2, \tau_3, \tau_4, \tau_5 \rangle$	$\gamma(N_1, \omega_5) = 1.00$	$\rho(N_1, \omega_5) = 1.00$	No drift candidate detected	
τ_6	$ABCD$	$\omega_6 = \langle \tau_3, \tau_4, \tau_5, \tau_6 \rangle$	$\gamma(N_1, \omega_6) = 1.00$	$\rho(N_1, \omega_6) = 1.00$	No drift candidate detected	
τ_7	$ABCD$	$\omega_7 = \langle \tau_4, \tau_5, \tau_6, \tau_7 \rangle$	$\gamma(N_1, \omega_7) = 1.00$	$\rho(N_1, \omega_7) = 1.00$	No drift candidate detected	
τ_8	$ABCD$	$\omega_8 = \langle \tau_5, \tau_6, \tau_7, \tau_8 \rangle$	$\gamma(N_1, \omega_8) = 1.00$	$\rho(N_1, \omega_8) = 1.00$	No drift candidate detected	
τ_9	$ABDC$	$\omega_9 = \langle \tau_6, \tau_7, \tau_8, \tau_9 \rangle$	$\boldsymbol{\gamma(N_1, \omega_9) = 0.75}$	$\rho(N_1, \omega_9) = 1.00$	Drift candidate on ω_9 (by fitness)	
τ_{10}	$ABCD$	$\omega_{10} = \langle \tau_7, \tau_8, \tau_9, \tau_{10} \rangle$	$\boldsymbol{\gamma(N_1, \omega_{10}) = 0.75}$	$\rho(N_1, \omega_{10}) = 1.00$	Drift candidate on ω_{10} (by fitness)	
τ_{11}	$ABDC$	$\omega_{11} = \langle \tau_8, \tau_9, \tau_{10}, \tau_{11} \rangle$	$\boldsymbol{\gamma(N_1, \omega_{11}) = 0.50}$	$\rho(N_1, \omega_{11}) = 1.00$	Drift candidate on ω_{11} (by fitness)	
τ_{12}	$ABCD$	$\omega_{12} = \langle \tau_9, \tau_{10}, \tau_{11}, \tau_{12} \rangle$	$\boldsymbol{\gamma(N_1, \omega_{12}) = 0.50}$	$\rho(N_1, \omega_{12}) = 1.00$	Drift candidate on ω_{12} (by fitness) Confirm drift at τ_9 (by fitness) Discover model for ω_{12}	N_2 
τ_{13}	$ABDC$	$\omega_{13} = \langle \tau_{10}, \tau_{11}, \tau_{12}, \tau_{13} \rangle$	$\gamma(N_2, \omega_{13}) = 1.00$	$\rho(N_2, \omega_{13}) = 1.00$	No drift candidate detected	
τ_{14}	$ABCD$	$\omega_{14} = \langle \tau_{11}, \tau_{12}, \tau_{13}, \tau_{14} \rangle$	$\gamma(N_2, \omega_{14}) = 1.00$	$\rho(N_2, \omega_{14}) = 1.00$	No drift candidate detected	
τ_{15}	$ABDC$	$\omega_{15} = \langle \tau_{12}, \tau_{13}, \tau_{14}, \tau_{15} \rangle$	$\gamma(N_2, \omega_{15}) = 1.00$	$\rho(N_2, \omega_{15}) = 1.00$	No drift candidate detected	
τ_{16}	$ABCD$	$\omega_{16} = \langle \tau_{13}, \tau_{14}, \tau_{15}, \tau_{16} \rangle$	$\gamma(N_2, \omega_{16}) = 1.00$	$\rho(N_2, \omega_{16}) = 1.00$	No drift candidate detected	
τ_{17}	$ABDC$	$\omega_{17} = \langle \tau_{14}, \tau_{15}, \tau_{16}, \tau_{17} \rangle$	$\gamma(N_2, \omega_{17}) = 1.00$	$\rho(N_2, \omega_{17}) = 1.00$	No drift candidate detected	
τ_{18}	$ABDC$	$\omega_{18} = \langle \tau_{15}, \tau_{16}, \tau_{17}, \tau_{18} \rangle$	$\gamma(N_2, \omega_{18}) = 1.00$	$\rho(N_2, \omega_{18}) = 1.00$	No drift candidate detected	
τ_{19}	$ABDC$	$\omega_{19} = \langle \tau_{16}, \tau_{17}, \tau_{18}, \tau_{19} \rangle$	$\gamma(N_2, \omega_{19}) = 1.00$	$\rho(N_2, \omega_{19}) = 1.00$	No drift candidate detected	
τ_{20}	$ABDC$	$\omega_{20} = \langle \tau_{17}, \tau_{18}, \tau_{19}, \tau_{20} \rangle$	$\gamma(N_2, \omega_{20}) = 1.00$	$\boldsymbol{\rho(N_2, \omega_{20}) = 0.66}$	Drift candidate on ω_{20} (by precision)	
τ_{21}	$ABDC$	$\omega_{21} = \langle \tau_{18}, \tau_{19}, \tau_{20}, \tau_{21} \rangle$	$\gamma(N_2, \omega_{21}) = 1.00$	$\boldsymbol{\rho(N_2, \omega_{21}) = 0.66}$	Drift candidate on ω_{21} (by precision)	
τ_{22}	$ABDC$	$\omega_{22} = \langle \tau_{19}, \tau_{20}, \tau_{21}, \tau_{22} \rangle$	$\gamma(N_2, \omega_{22}) = 1.00$	$\boldsymbol{\rho(N_2, \omega_{22}) = 0.66}$	Drift candidate on ω_{22} (by precision)	
τ_{23}	$ABDC$	$\omega_{23} = \langle \tau_{20}, \tau_{21}, \tau_{22}, \tau_{23} \rangle$	$\gamma(N_2, \omega_{23}) = 1.00$	$\boldsymbol{\rho(N_2, \omega_{23}) = 0.66}$	Drift candidate on ω_{23} (by precision) Confirm drift at τ_{17} (by precision) Discover model for ω_{23}	N_3 
τ_{24}	$ABDC$	$\omega_{24} = \langle \tau_{21}, \tau_{22}, \tau_{23}, \tau_{24} \rangle$	$\gamma(N_3, \omega_{24}) = 1.00$	$\rho(N_3, \omega_{24}) = 1.00$	No drift candidate detected	
τ_{25}	$ABDC$	$\omega_{25} = \langle \tau_{22}, \tau_{23}, \tau_{24}, \tau_{25} \rangle$	$\gamma(N_3, \omega_{25}) = 1.00$	$\rho(N_3, \omega_{25}) = 1.00$	No drift candidate detected	
τ_{26}	$ABDC$	$\omega_{26} = \langle \tau_{23}, \tau_{24}, \tau_{25}, \tau_{26} \rangle$	$\gamma(N_3, \omega_{26}) = 1.00$	$\rho(N_3, \omega_{26}) = 1.00$	No drift candidate detected	
τ_{27}	$ABDC$	$\omega_{27} = \langle \tau_{24}, \tau_{25}, \tau_{26}, \tau_{27} \rangle$	$\gamma(N_3, \omega_{27}) = 1.00$	$\rho(N_3, \omega_{27}) = 1.00$	No drift candidate detected	

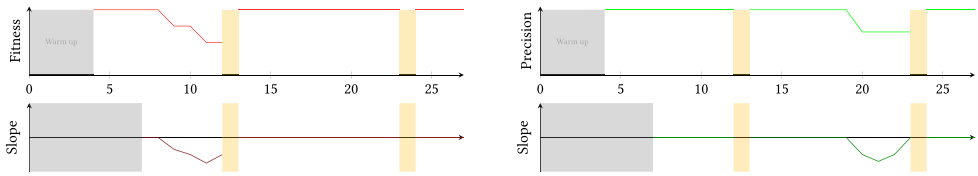


Fig. 4. Example of a gradual change where some previously unobserved behavior starts to replace the previous one at τ_9, which is no longer observed after trace τ_{16}. Graphs show the evolution of fitness and precision metrics, and the slope of the regression for each conformance metric. Drift points are marked in yellow.

After this initialization, the drift detection step is executed (lines 8–42). This step is composed of a loop that repeats until no traces remain unprocessed in the log. Thus, as part of the initialization performed after each change detection, two empty lists are created (line 9), Γ and P, for storing the fitness and precision, respectively. Also, two more lists, D^{Γ} and D^{P}, are initialized (line 10). These lists will later be populated with Booleans indicating if the successive windows are marked as drift candidates. Finally, a flag used for indicating if a change has been detected is initialized (line 11).

Once this initialization is completed, the main detection loop begins (lines 12–41), where the detection is performed based on the trends in the values of the conformance metrics. This loop processes the remaining windows, starting at index i, until a new change is detected or to the end of the log if no change is present (line 12). For each window ω_i the fitness and the precision of the last discovered model $N_{|N|}$ are computed, and their values are appended to Γ and P, respectively (lines 13 and 14). Then, using the function IdentifyDriftCandidate from Algorithm 2, the window ω_i

Algorithm 1: *CRIER*

Inputs: an event log L, minimum size of the sliding window n'
Outputs: a set of traces (for sudden changes)/segments of the log (for gradual changes) causing drifts

```
1:  procedure ConceptDriftDetection(L, n')
2:    D^S, D^G ← [ ] //confirmed sudden and gradual drifts
3:    n ←AdjustWindow(n', ⟨τ_1, ..., τ_|L|⟩)
4:    i ← n
5:    ω_i ← ⟨τ_{i−n}, ..., τ_i⟩
6:    N ← [discover(ω_i)] //save the model in the model history
7:    τ*, τ' ← τ_1
8:    while i < |L| do
9:      Γ, P ← [ ] //fitness and precision measures (Def. 3.5 and Def. 3.6)
10:     D^Γ, D^P ← [ ] //drift candidates (fitness and precision)
11:     flag ← FALSE
12:     while (i < |L|) ∧ ¬flag do
13:       Γ ← Γ :: γ(ω_i, N_|N|) //append current fitness
14:       P ← P :: ρ(ω_i, N_|N|) //append current precision
15:       D^Γ ← D^Γ :: IdentifyDriftCandidate(n, Γ, D^Γ)
16:       D^P ← D^P :: IdentifyDriftCandidate(n, P, D^P)
17:       if ConfirmDrift(n, D^Γ) ∨ ConfirmDrift(n, D^P) then//change confirmed
18:         if ConfirmDrift(n, D^Γ) then
19:           τ* ← τ_{i−n} //confirmed drift in the last trace from the first candidate
20:         else if ConfirmDrift(n, D^P) then
21:           τ* ← τ_{i−2n} //confirmed drift in the first trace from the first candidate
22:         end if
23:         flag ← TRUE
24:         L' ← ⟨τ', ..., τ*⟩ //store the sublog between confirmed drifts
25:         n ← AdjustWindow(n', ⟨τ_{i+1}, ..., τ_|L|⟩)
26:         i ← i + n
27:         ω_i ← ⟨τ_{i−n}, ..., τ_i⟩
28:         N ← N :: discover(ω_i) //append current model to model history
29:         if |N| > 2 ∧ (∃τ ∈ L' : B_τ ∈ B_{N_|N|−2}) ∧ (∃τ ∈ L' : B_τ ∈ B_{N_|N|}) ∧ (∀τ ∈ L' : B_τ ∈ B_{N_|N|−2} ∪ B_{N_|N|}) then
30:           τ' ← D^S_|D^S|
31:           D^S ← {d ∈ D^S : d ≠ τ'}
32:           D^G ← D^G :: [τ', τ*) //a gradual change
33:         else
34:           D^S ← D^S :: τ* //a sudden change
35:         end if
36:         τ' ← τ*
37:       else
38:         i ← i + 1
39:         ω_i ← ⟨τ_{i−n}, ..., τ_i⟩
40:       end if
41:     end while
42:   end while
43:   return D^S ∪ D^G
44: end procedure
```

is classified or not as a drift candidate (lines 15–16). To carry out this classification step, a simple linear regression—using the ordinary least squares method—is computed over the metric values (Algorithm 2, line 17). If there are enough data for computing the regression and the slope of the fitted line is different from 0 with enough statistical confidence—p-value < 0.05 in the t-test, where H_0 states that the slope of the regression is 0, or the slope is 0 and the previous window has been marked as a drift candidate, the window ω_i will be marked as a drift candidate. To finalize this detection step the function ConfirmDrift from Algorithm 2 is executed (line 29). This function is in charge of checking whether a drift candidate persists over time, becoming a real change, or, on the contrary, only represents a noisy trace. A drift candidate is confirmed as a real change if the last n windows have been also marked as drift candidates (Algorithm 2, lines 23–26).

Algorithm 2: Auxiliary Functions

```
1:  function AdjustWindow(n, L)
2:      N_1, N_2, N_3 ← ∅
3:      n' ← n
4:      while 3n < |L| do
5:          N_1 ← discover(⟨τ_0, ..., τ_{n'}⟩)
6:          N_2 ← discover(⟨τ_{n'}, ..., τ_{2n'}⟩)
7:          N_3 ← discover(⟨τ_{2n'}, ..., τ_{3n'}⟩)
8:          if B_{N_1} = B_{N_2} = B_{N_3} then
9:              n' ← increment(n')
10:         else
11:             return n'
12:         end if
13:     end while
14:     return n'
15: end function

16: function IdentifyDriftCandidate(n, data, D)
17:     ϒ   ← regress({data_{|data|-n}, ..., data_{|data|}})
18:     m^< ← ϒ.slope < 0 ∧ ϒ.confidence < 0.05
19:     m^> ← ϒ.slope > 0 ∧ ϒ.confidence < 0.05
20:     m^= ← (¬ m^<) ∧ (¬ m^>)
21:     return (|data| > n) ∧ (m^< ∨ m^> ∨ (m^= ∧ (D_{|D|} = true)))
22: end function

23: function ConfirmDrift(n, D)
24:     d' ← ∀d ∈ {D_{|D|-n}, ..., D_{|D|}} : d = true
25:     return (|D| ≥ n) ∧ d'
26: end function
```

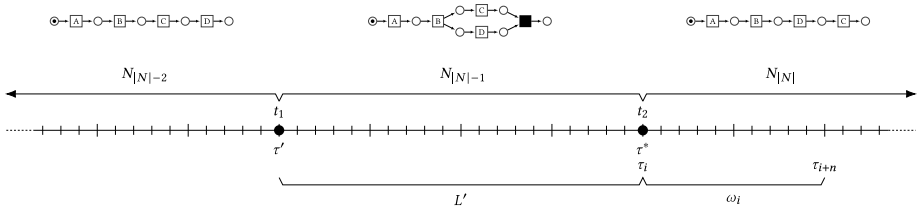


Fig. 5. Drift classification phase in *CRIER*.

Once the change is confirmed, it should be pinpointed in time to a specific trace. In case the drift comes from a change in fitness, the trace causing the drift is the last trace from the first window identified as a candidate—i.e., the first trace causing a change in the metric. In the case of a change in precision, the trace causing the drift is the first trace from the first window identified as candidate—i.e., the first trace from the first window causing a change in precision. Then, the flag indicating a detection is updated (line 23) and the sublog L' between confirmed drifts is stored for later. The drift must then be classified as sudden or gradual. Figure 5 illustrates this part of the algorithm. The first step is to update both the index at which the next window will start and its optimal size (lines 25 and 26). Then, the new window is populated with the traces and the model describing the behavior contained in this window is discovered (lines 27 and 28). This model is stored in the model list. Since gradual changes are delimited by two drifts, the last three models of this list—$N_{|N|-2}$, $N_{|N|-1}$, and $N_{|N|}$—are used to check the conditions that determine whether the drift is gradual or not. Note that the model $N_{|N|-1}$ corresponds with the sublog in which the confirmed drift has been detected. These conditions are the following (line 29):

(1) There are more than two models in the model list ($|N| > 2$).

(2) At least, one trace from L' is supported by $N_{|N|}$.
(3) At least, one trace from L' is supported by $N_{|N|-2}$.
(4) The behavior from every trace in L' is part of the behavior of $N_{|N|-2}$ or $N_{|N|}$.

If L' fulfills these four requirements, the change is classified as gradual, indicating that the drift starts at the trace where the last sudden drift $D^S{}_{|D^S|}$ was detected and lasts until τ^*. Furthermore, $D^S{}_{|D^S|}$ is removed from the sudden drift list (line 31), and appended to the list D^Γ of classified drifts (line 32). On the other hand, if the candidate cannot be classified as gradual because it does not meet any of the requirements, the change at τ^* is classified as sudden (line 34). Finally, the index i is incremented by 1 (line 38), the sliding window is updated to this index (line 39), and the cycle starts again.

5.2 Complexity Analysis

The computational complexity analysis is crucial for understanding the scalability of the proposed algorithm and determining the maximum manageable size of the input data. In real-world processes, traces can grow significantly, both in volume and frequency. Therefore, analyzing how the algorithm performance scales with data size is essential to ensure its practical applicability in real environments.

5.2.1 Space Complexity. In terms of spatial complexity, the memory usage of *CRIER* is bounded by the window size. Thus, the algorithm has a $O(size(\omega))$ spatial complexity. By processing the windows independently, *CRIER* does not need to store the entire log in memory, meaning there is no strict limit on the maximum log size that can be processed, as long as the log is read efficiently and the window size is kept within acceptable limits. For example, in an experiment with windows of 100 traces, only the last 100 traces need to be held in memory at any given time. Therefore, the spatial cost for a log with 1,000 traces will be the same as for a log with 1,000,000 traces.

5.2.2 Computational Complexity. The computational cost of *CRIER* can be decomposed into three main components:

(1) The complexity of the algorithm for process model discovery, $O_{discovery}$, which is executed every time a drift is detected.
(2) The complexity of the conformance metrics, $O_{fitness}$ and $O_{precision}$, which are computed for each window, growing linearly with the number of traces, i.e., with the log size.
(3) The complexity of the linear regression, $O_{regression}$, that is computed again for each window, and that also grows linearly with the size of the log.

Considering this, the computational complexity of *CRIER* would be

$$O_{CRIER} = O_{discovery} \times \text{number of drifts} + (O_{fitness} + O_{precision} + O_{regression}) \times \text{number of windows}.$$

As the number of windows grows linearly with the log size, and in the worst case, the number of drifts also grows linearly—one new drift for each new trace, the complexity of *CRIER* is $O_{CRIER} = O(size(log)) \sim O(n)$, where n is the number of traces.

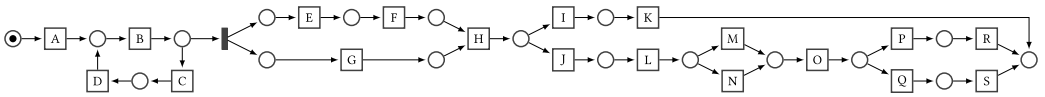


Fig. 6. Petri net model used in the validation of the algorithm, where activity names are shortened for better understandability.

6 Experimentation

In this section, we present the evaluation of *CRIER* using synthetic datasets for multiple change patterns and change distributions. Also, in Section 6.5 we present the results of executing *CRIER* over real-life datasets.

6.1 Validation Data

Concept drift algorithms are validated with synthetic data generated from real processes, in which changes are introduced at a specific moment and with a specific duration, since there are no real logs in which these change regions are identified. In this article, *CRIER* has been validated with synthetic logs generated from a loan granting process, which is the *de-facto* benchmark used to validate concept drift approaches in process mining [8, 20, 23, 28, 35, 44, 45, 47]. Its model, represented in Figure 6, consists of 19 different activities structured according to typical control constructs such as sequences, parallels, and choices [9].

From this base model, and applying the patterns of change presented in [43], modified models have been generated using the methodology described in [23]. These patterns can be classified into three categories (Table 2(a)): (i) changes involving the insertion of new behavior—marked with *I*; (ii) changes involving the optionalization of a part of the model—marked with *O*; and (iii) changes involving the restructuring and rearrangement of some parts of the model—marked with *R*. In addition, logs combining the different patterns have also been generated. Only the patterns or patterns combination that can generate a gradual change—i.e., those that add some new, unobserved behavior, forcing a change in fitness—have been applied to create the validation models, as they are the only ones that can generate a gradual drift (Table 2(b)). Note that the other patterns—specifically, `cm`, `cb`, `lp`, `cd`, and `pl`—produce models that only would change the precision and, therefore, no gradual changes could be generated from these models (see Theorem 4.1). As a result of this procedure, 10 derived models are created. These models are used to generate the trace logs with which the approach is validated since each one is the result of applying the gradual change that takes place from the base model. For this generation, a **cumulative probability function (*cdf*)** selects the model—base or derived—that will generate each trace according to a given probability distribution (P). Specifically, the following procedure is applied to create the validation logs:

(1) The base model M_1—i.e., the loan application process model, the modified model M_2—i.e., one of the 10 derived models, and the probability distribution P are chosen.
(2) A block of 500 traces corresponding to model M_1 is generated.
(3) While the *cdf* of P is below the stopping criterion—$cdf(P) \leq 0.999$, a model is chosen between M_1 and M_2 with probabilities $1 - cdf(P)$ and $cdf(P)$, respectively, and a new trace is generated from the selected model. Note that this stopping criterion is due to the *cdf* for some distributions being asymptotic to 1.
(4) Repeat from step 2 interchanging M_1 and M_2.

Table 2. Change Patterns Applied to the Original Model from Figure 6 and Resulting Models

(a) Change patterns.

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

(b) Derived models.

Model code	Applied change patterns
cp	cp
pm	pm
re	re
rp	rp
rw	rw
cf	cf
OIR	lp + re + cd
ORI	lp + pl + re
RIO	cf + cp + cb
ROI	pl + lp + rp

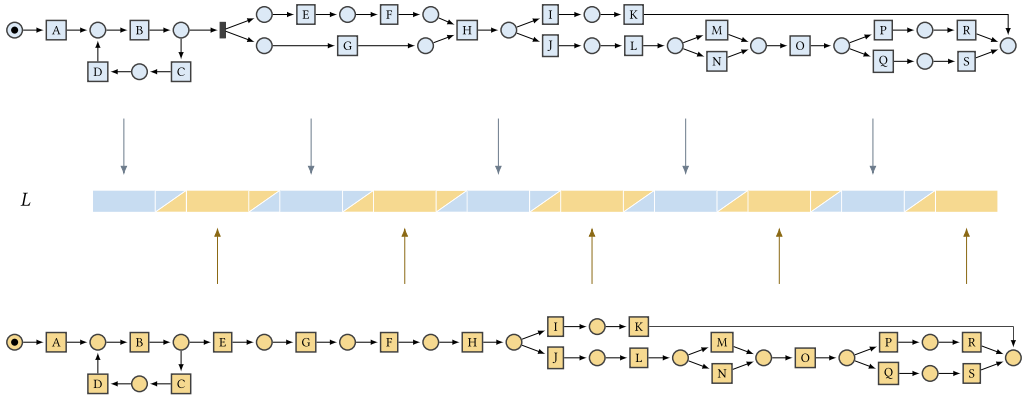


Fig. 7. Log generation example.

Figure 7 shows an example of log generation where the base model is combined with a model derived from applying the pm pattern by using a linear probability distribution.

In our experimentation, four different probability functions have been considered: (i) *linear*, where the frequency of the new behavior increment linearly when new traces are observed; (ii) *Gaussian*, where the change starts to appear slowly and accelerates to end with the old behavior fading away slowly as well; (iii) *exponential*, where the frequency of the new behavior grows very fast at the beginning and slows down as new traces are observed; and (iv) *constant*, where the old and the new behavior have a constant probability of appearing that is not modified during a number of traces. For these distributions, several *cdfs* have been applied with different configuration parameters, for a total of 12 distinct probability distributions, as Figure 8 shows. In summary, 120 synthetic logs[1] have been generated, using 10 different patterns and 12 probability distributions. Table 3 summarizes the features of the validation logs, with the log size, the change points, and the distributions applied.

It is important to highlight that, although we have strived for thoroughness and rigorously tested simple patterns, it is practically impossible to cover all possible combinations of patterns and drift distributions. On the pattern side, *CRIER* is designed to be pattern-agnostic, meaning its

[1]The generated syntetic logs are available at https://gitlab.citius.usc.es/ProcessMining/logs/-/tree/master/drift/gradual

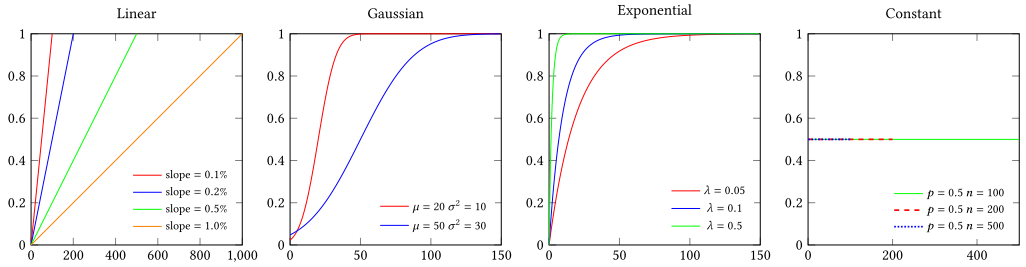


Fig. 8. Cumulative distribution functions for the applied distributions in the log generation.

Table 3. Summary of Validation Logs and Their Change Points, Where t_1 and t_2 Are the Temporal Points between Which the Gradual Change Takes Place

Distribution	Log size		Change regions								
			Drift 1	Drift 2	Drift 3	Drift 4	Drift 5	Drift 6	Drift 7	Drift 8	Drift 9
Linear (slope = 0.1%)	14,000	t_1	500	2,000	3,500	5,000	6,500	8,000	9,500	11,000	12,500
		t_2	1,500	3,000	4,500	6,000	7,500	9,000	10,500	12,000	13,500
Linear (slope = 0.2%)	9,500	t_1	500	1,500	2,500	3,500	4,500	5,500	6,500	7,500	8,500
		t_2	1,000	2,000	3,000	4,000	5,000	6,000	7,000	8,000	9,000
Linear (slope = 0.5%)	6,800	t_1	500	1,200	1,900	2,600	3,300	4,000	4,700	5,400	6,100
		t_2	700	1,400	2,100	2,800	3,500	4,200	4,900	5,600	6,300
Linear (slope = 1.0%)	5,900	t_1	500	1,100	1,700	2,300	2,900	3,500	4,100	4,700	5,300
		t_2	600	1,200	1,800	2,400	3,000	3,600	4,200	4,800	5,400
Gaussian ($\mu = 20, \sigma^2 = 10$)	5,459	t_1	500	1,051	1,602	2,153	2,704	3,255	3,806	4,357	4,908
		t_2	551	1,102	1,653	2,204	2,755	3,306	3,857	4,408	9,959
Gaussian ($\mu = 50, \sigma^2 = 30$)	6,287	t_1	500	1,143	1,786	2,429	3,072	3,715	4,358	5,001	5,644
		t_2	643	1,286	1,929	2,572	3,215	3,858	4,501	5,144	5,787
Exponential ($\lambda = 0.05$)	6,251	t_1	500	1,139	1,778	2,417	3,056	3,695	4,334	4,973	5,612
		t_2	639	1,278	1,917	2,556	3,195	3,834	4,473	5,112	5,751
Exponential ($\lambda = 0.1$)	5,630	t_1	500	1,070	1,640	2,210	2,780	3,350	3,920	4,490	5,060
		t_2	570	1,140	1,710	2,280	2,850	3,420	3,990	4,560	5,130
Exponential ($\lambda = 0.5$)	5,126	t_1	500	1,014	1,528	2,042	2,556	3,070	3,584	4,098	4,612
		t_2	514	1,028	1,542	2,056	2,570	3,084	3,598	4,112	4,626
Constant (p = 0.5, $n = 100$)	5,900	t_1	500	1,100	1,700	2,300	2,900	3,500	4,100	4,700	5,300
		t_2	600	1,200	1,800	2,400	3,000	3,600	4,200	4,800	5,400
Constant (p = 0.5, $n = 200$)	6,800	t_1	500	1,200	1,900	2,600	3,300	4,000	4,700	5,400	6,100
		t_2	700	1,400	2,100	2,800	3,500	4,200	4,900	5,600	6,300
Constant (p = 0.5, $n = 500$)	9,500	t_1	500	1,500	2,500	3,500	4,500	5,500	6,500	7,500	8,500
		t_2	1,000	2,000	3,000	4,000	5,000	6,000	7,000	8,000	9,000

performance does not rely on specific change patterns, reducing the potential impact of not testing every possible combination.

However, when it comes to drift distributions, these can significantly affect the performance of the algorithm. To address this, we have conducted extensive experiments covering a wide range of scenarios, including edge cases such as abrupt or very gradual drift onset and offset. Despite our best efforts, we acknowledge that there may still be situations not covered in these experiments.

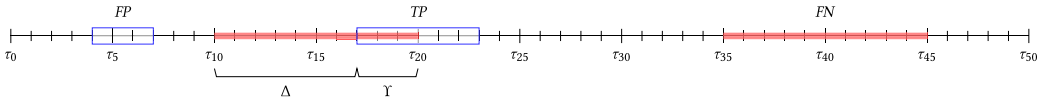


Fig. 9. Example of classification of results over a log with two gradual changes. Real drift regions are marked in red. Detected drift regions are outlined in blue.

6.2 Metrics

When evaluating the performance of a drift detection solution, there are two main metrics used in the *state-of-the-art*. The first metric, called F_{score}, calculates the harmonic mean between *precision* and *recall*, allowing to evaluate how reliable the results are, based on the number of TP/FPs and negatives. For gradual changes, a TP is any change whose detection area overlaps with a real change not previously detected, while a FP is any change that does not correspond to a region of real change, or that matches a region of change previously detected. In addition, a FN is a region of change that does not overlap with anyone of the detected changes. Considering this, the F_{score} metric is defined as follows:

$$F_{score} = \frac{2 \times precision \times recall}{precision + recall}$$
$$precision = \frac{TP}{TP + FP}$$
$$recall = \frac{TP}{TP + FN}.$$

The second metric, called *delay* (Δ), measures how late a drift is reported from the actual occurrence until its detection. In the case of gradual changes, since we are dealing with areas of change and not single points, we measure the delay in detecting the beginning of the region:

$$\Delta(d^R, d^D) = |\min(d^R) - \min(d^D)|,$$

where d^R and d^D are intervals indicating a real and a detected drift region, respectively.

Finally, and to better reflect the goodness of the results when dealing with gradual changes, these two metrics are complemented with a third one: the *change region overlapping* (ϒ). This metric evaluates the percentage of the actual region of change that has been detected by the algorithm, reflecting how well the duration of the changes is captured:

$$\Upsilon(d^R, d^D) = \frac{|d^R \cap d^D|}{|d^R|},$$

where d^R and d^D are intervals indicating a real and a detected drift region, respectively.

Figure 9 shows an example of how these metrics are applied in the evaluation of the results. This example presents a log with two gradual changes, one between traces τ_{10} and τ_{20}, and the other one between traces τ_{35} and τ_{45}. The algorithm detects two changes, one between traces τ_4 and τ_7, classified as a FP as no real change happened in this time interval, and one between traces τ_{17} and τ_{23}, classified as a TP because it overlaps a real change. In this second detection, the delay would be seven traces—from trace τ_{10} to trace τ_{17}. On the other hand, the change region overlapping would be 30%, since only 3 traces out of the 10 that constitute the region of change are detected. Finally, the second change, between traces τ_{35} and τ_{45}, will be classified as a FN, since the algorithm has not detected any change in that region.

Table 4. Evidence Levels Proposed by [15] for the Bayes Factors

Bayes factor $BF_{H_iH_j}$					Interpretation
		$BF_{H_iH_j}$	>	100	Extreme evidence for H_i
100	>	$BF_{H_iH_j}$	>	30	Very strong evidence for H_i
30	>	$BF_{H_iH_j}$	>	10	Strong evidence for H_i
10	>	$BF_{H_iH_j}$	>	3	Moderate evidence for H_i
3	>	$BF_{H_iH_j}$	>	1	Anecdotal evidence for H_i
		$BF_{H_iH_j}$	=	1	No evidence
1	>	$BF_{H_iH_j}$	>	0.3333	Anecdotal evidence for H_j
0.3333	>	$BF_{H_iH_j}$	>	0.1	Moderate evidence for H_j
0.1	>	$BF_{H_iH_j}$	>	0.0333	Strong evidence for H_j
0.0333	>	$BF_{H_iH_j}$	>	0.01	Very strong evidence for H_j
0.01	>	$BF_{H_iH_j}$			Extreme evidence for H_j

Furthermore, results of the different algorithms have been also evaluated using a *Bayesian hypothesis test*. We opted for a Bayesian approach instead of a traditional **null hypothesis significance test (NHST)** [22] because:

(1) *NHST* does not provide any certainty about the validity of the null hypothesis. Thus, if the hypothesis is not rejected, it can only be stablished that there is no evidence to reject it, but never to accept it as such.
(2) *NHST* does not estimate the probability of the hypotheses to be valid, so comparing the algorithms is harder.

In this article, we have performed a **BAyesian INformative hypothesis evaluation (BAIN)** [13] to analyze the experimentation. BAIN is based on the use of *Bayes factors* to compare the conditional probability between two competing hypotheses. In a simplified form, a *Bayesian factor* $BF_{H_iH_j}$ is the ratio of the probability of the hypothesis H_i to the probability of the hypothesis H_j. For example, $BF_{H_iH_j} = 5$ would indicate that the hypothesis H_i is five times more likely than hypothesis H_j. The most frequent interpretation of these Bayes factors was proposed by [15], where the Bayes factor values are classified in 11 levels, from *no evidence* to *extreme evidence*. Table 4 shows the different levels of evidence defined in this classification. For all the hypothesis proposed for the test, symbols >, =, and < refer to the algorithm obtaining better, equal, or worst results. All tests have been executed using the software package JASP [19].

6.3 Setup

The results of *CRIER*[2] have been compared with the three best publicly available *state-of-the-art* algorithms: the proposal from *Martjushev et al.* [26]—available as a plugin of the process mining platform ProM,[3] *ProDrift*,[4] [23] and the approach from *Lu et al.* [20].[5] Note that *ProDrift* has not been run for logs with Gaussian, exponential, and constant distributions since authors explicitly state that it only deals with linear changes. The rest of the approaches identified in the *state-of-the-art* have not been tested as the source code of their implementations are not available. The specific configurations used for these three algorithms were as follows:

[2]The implementation of CRIER is available at https://github.com/ProcessMiningUSC/CRIER
[3]https://www.promtools.org/
[4]https://kodu.ut.ee/~dumas/tools/ProDrift2.5.zip
[5]https://github.com/bearlu1996/ProcessDrifts

Table 5. Average F_{score}, Δ, and ϒ Values for the Logs with Gradual Changes

		CRIER			*Martjushev et al.*			*ProDrift*			*Lu et al.*		
		F_{score}	Δ	ϒ	F_{score}	Δ	ϒ	F_{score}	Δ	ϒ	F_{score}	Δ	ϒ
Linear	*slope* = 0.1%	0.6481	57.4677	69.66%	0.4873	416.6944	17.70%	0.5712	252.6599	37.89%	0.4036	117.8907	26.59%
	slope = 0.2%	0.9146	25.0048	83.31%	0.5090	148.8261	22.73%	0.3706	243.4345	15.58%	0.3576	186.4861	13.07%
	slope = 0.5%	0.9742	18.1222	79.52%	0.4886	187.6234	18.31%	0.0333	406.5000	2.22%	0.5331	46.0765	29.66%
	slope = 1.0%	0.9737	11.9873	69.57%	0.2844	137.8463	14.84%	0.0500	433.6667	2.24%	0.7186	20.5751	39.97%
Gaussian	$\mu = 20$ $\sigma^2 = 10$	0.9882	8.7806	63.50%	0.2264	98.5952	16.03%	–	–	–%	0.8749	12.7694	51.03%
	$\mu = 50$ $\sigma^2 = 30$	0.9847	11.5921	61.59%	0.4567	161.3414	19.66%	–	–	–%	0.6173	20.1121	23.11%
Exponential	$\lambda = 0.05$	0.5591	8.5742	31.11%	0.3646	158.9043	17.37%	–	–	–%	0.6218	29.5625	11.37%
	$\lambda = 0.10$	0.8999	5.6948	50.21%	0.3036	139.5498	22.03%	–	–	–%	0.7428	14.2912	23.84%
	$\lambda = 0.50$	0.4584	2.8296	27.34%	0.2093	109.9016	21.52%	–	–	–%	0.8150	2.6347	30.16%
Constant	p = 0.5 $n = 100$	1.0000	6.4333	83.98%	0.3333	160.0795	16.53%	–	–	–%	0.8332	14.3706	70.37%
	p = 0.5 $n = 200$	1.0000	4.8333	94.03%	0.5090	220.1310	18.10%	–	–	–%	0.1401	104.0556	7.51%
	p = 0.5 $n = 500$	1.0000	5.3778	97.07%	0.6125	147.2569	33.04%	–	–	–%	0.1034	165.9167	3.81%

The colors highlight the best performing approach for each metric.

(1) *Martjushev et al.*: local features using the follows relations for all pairs of activities; *J-measure* with a window size of 10; adaptive window with a minimum size of 50 and a maximum of 500; a step of 1, and an automatic gap size; and comparing windows with the *Kolmogorov-Smirnov* test with a p-value of 0.4—the default value specified by the authors.
(2) *ProDrift*: trace-based detection, with a fixed-size window of 100 traces.
(3) *Lu et al.*: window size of 1,500 and number of consecutive statistical tests set at the window size divided by 2—default values specified by the authors.

6.4 Results

Table 5 shows the average results for each algorithm. *CRIER* performs, on average, better than the other models for all datasets and for all metrics, except for exponential logs, where *Lu et al.* exceeds in some cases. For linear logs, *CRIER* improves F_{score} and Δ values as the slope grows, exceeding 0.9 in F_{score} and with delays below 20 traces. Additionally, *CRIER* demonstrates a notable difference in terms of coverage, with values around 70% or higher, while the other approaches remain below 40%. In the case of Gaussian logs, *CRIER* clearly outperforms *Martjushev et al.* and *Lu et al.* for all metrics, obtaining almost a perfect F_{score}, delays of about 10 traces and a coverage of more than 60%. For exponential logs, *CRIER* obtains the best results when $\lambda = 0.10$ with a F_{score} 0.15 points better than the second best proposal—*Lu et al.*, Δ three times lower and ϒ two times higher. However, with $\lambda = 0.50$, *Lu et al.* improves the F_{score} by 0.35 points and also obtains slightly better Δ and ϒ. For constant logs, *CRIER* outperforms again *Martjushev et al.* and *Lu et al.*, achieving a perfect F_{score}, delays under seven traces on average and ϒ more than 80%. The results for each type of distribution are presented in detail in Appendix A.

Table 6 shows the findings of the BAIN test performed over the results obtained by all approaches. In this test, the hypothesis H_1 considers that *CRIER* achieves better results than the other *state-of-the-art* approaches—i.e., *CRIER* > *Martjushev et al.* and *CRIER* > *ProDrift* and *CRIER* > *Lu et al.*, while H_1^c considers the contrary hypothesis. As Table 6 shows, the Bayes factor $BF_{H_1H_1^c}$ for all metrics in linear and constant distributions is greater than 100, meaning that there is an extreme evidence for H_1, i.e., *CRIER* is better than *ProDrift*, *Martjushev et al.*, and *Lu et al.* In Gaussian logs, the test shows that there is an extreme evidence for H_1 in terms of F_{score} and ϒ, but for the Δ the most probable hypothesis is H_5—*CRIER = Lu et al.*—with a value between 3 and 10, meaning that there is a moderate evidence that *CRIER* and *Lu et al.* are equivalents. Finally, in exponential distributions the results of the test are more diverse, showing an extreme evidence for H_1 in terms of ϒ but a moderate evidence for H_3—*CRIER < Lu et al.*—and H_5 in terms of F_{score} and Δ, respectively.

Table 6. BAIN Test Results for the Validation Logs, Where $BF_{H_iH_i^c}$ Shows the Bayesian Factor for H_i against Its Complementary $H_i^c = \neg H_i$

		$BF_{H_iH_i^c}$		
		F_{score}	Λ	Υ
Linear logs	H_1: *CRIER* > *Martjushev et al.* and *CRIER* > *ProDrift* and *CRIER* > *Lu et al.*	8.96×10^{9}	2.85×10^{2}	8.96×10^{9}
	H_2: *CRIER* < *Martjushev et al.*	3.40×10^{-12}	8.19×10^{-15}	3.4×10^{-12}
	H_3: *CRIER* < *ProDrift*	5.96×10^{-23}	3.21×10^{-13}	5.96×10^{-23}
	H_4: *CRIER* < *Lu et al.*	1.79×10^{-9}	1e-2	1.72×10^{-9}
	H_5: *CRIER* = *Martjushev et al.*	4.36×10^{-10}	1.1×10^{-12}	4.36×10^{-10}
	H_6: *CRIER* = *ProDrift*	1.08×10^{-20}	3.18×10^{-11}	1.08×10^{-20}
	H_7: *CRIER* = *Lu et al.*	1.99×10^{-7}	4.61×10^{-1}	1.99×10^{-7}
Gaussian logs	H_1: *CRIER* > *Martjushev et al.* and *CRIER* > *Lu et al.*	1.73×10^{4}	3.44	5.99×10^{7}
	H_2: *CRIER* < *Martjushev et al.*	4.42×10^{-22}	4.69×10^{-12}	1.22×10^{-18}
	H_3: *CRIER* < *Lu et al.*	1.75×10^{-4}	5.45×10^{-1}	1.33×10^{-7}
	H_4: *CRIER* = *Martjushev et al.*	5.89×10^{-20}	4.15×10^{-10}	1.48×10^{-16}
	H_5: *CRIER* = *Lu et al.*	9e-3	5.1	9.69×10^{-6}
Exponential logs	H_1: *CRIER* > *Martjushev et al.* and *CRIER* > *Lu et al.*	2.42×10^{-1}	4.86	8.93×10^{2}
	H_2: *CRIER* < *Martjushev et al.*	2.38×10^{-7}	7.16×10^{-16}	6.69×10^{-4}
	H_3: *CRIER* < *Lu et al.*	8.8	3.84×10^{-1}	2e-3
	H_4: *CRIER* = *Martjushev et al.*	2.09×10^{-5}	9.02×10^{-14}	3.9×10^{-2}
	H_5: *CRIER* = *Lu et al.*	2.99	5.49	1.03×10^{-1}
Constant logs	H_1: *CRIER* > *Martjushev et al.* and *CRIER* > *Lu et al.*	1.48×10^{10}	6.41×10^{2}	2.25×10^{13}
	H_2: *CRIER* < *Martjushev et al.*	5.16×10^{-10}	3.9×10^{-15}	6.75×10^{-28}
	H_3: *CRIER* < *Lu et al.*	5.7×10^{-14}	3.00×10^{-3}	1.91×10^{-25}
	H_4: *CRIER* = *Martjushev et al.*	5.43×10^{-8}	4.7×10^{-13}	1.25×10^{-25}
	H_5: *CRIER* = *Lu et al.*	7.24×10^{-12}	1.4×10^{-1}	3.36×10^{-23}

The most likely hypothesis is shown shaded in blue.

As part of our experimentation, we also recorded the execution times of *CRIER* for each dataset. The average execution times for each distribution are reported in Table 7. The detailed results confirm the findings from the complexity analysis in Section 5.2: The time follows a linear relation with the size of the log. This is clearly seen when checking the average time per trace: No matter the distribution, it always takes between 1.84 and 1.99 ms to evaluate each trace. This demonstrates that *CRIER* performance remains consistent across various drift scenarios, with execution time scaling predictably based on the volume of traces processed, regardless of the complexity or nature of the changes.

6.5 Real-Life Datasets

Additional experiments were conducted using real-life event logs to evaluate the practical applicability of *CRIER* in real-world environments. Since these logs lack annotations and therefore have no known ground truth, they cannot be used for comparing different approaches. Nevertheless, manual validation of the results provides valuable insights into *CRIER* robustness and applicability in real operational contexts.

For this purpose, the BPIC 2020 [41] dataset was utilized. This dataset consists of five distinct log files documenting 2 years of travel expense claims from a Dutch university. Each log corresponds to a specific sub-process: domestic travel claims, international travel claims, travel permission requests, pre-payment of travel costs, and payment requests. The first year includes events from two departments, while in the second year, the process expanded to encompass the entire university.

Table 8 summarizes the results obtained from these logs. *CRIER* successfully detected a drift occurring between late 2017 and early 2018 in all logs, which coincides with the expansion of the

Table 7. Execution Times for *CRIER*

		Log size	Avg. execution time (s)	Avg. time per trace (ms)
Linear	*slope* = 0.1%	14,000	26.8799	1.9199
Linear	*slope* = 0.2%	9,500	17.9835	1.8930
Linear	*slope* = 0.5%	6,800	12.7490	1.8748
Linear	*slope* = 1.0%	5,900	11.3507	1.9238
Gaussian	$\mu = 20 \quad \sigma^2 = 10$	5,459	10.6949	1.9591
Gaussian	$\mu = 50 \quad \sigma^2 = 30$	6,287	12.3591	1.9658
Exponential	$\lambda = 0.05$	6,251	11.5780	1.8521
Exponential	$\lambda = 0.10$	5,630	11.1719	1.9843
Exponential	$\lambda = 0.50$	5,126	10.0708	1.9646
Constant	$p = 0.5 \quad n = 100$	5,900	11.3331	1.9208
Constant	$p = 0.5 \quad n = 200$	6,800	12.5410	1.8442
Constant	$p = 0.5 \quad n = 500$	9,500	18.2111	1.9169

Times shown represent the average execution time for all change patterns for each distribution.

Table 8. Drifts Detected by *CRIER* in the BPIC 2020 Dataset

Log		Detected drifts							
Domestic declarations	Start	29 June 2017	**18 December 2017**	15 February2018	12 April 2018	25 June 2018	10 October 2018	26 November 2018	12 December 2018
	End	7 September 2017	**22 January 2018**	22 February 2018	24 April 2018	28 June 2018	22 October 2018	3 December 2018	13 December 2018
International declarations	Start	10 August 2017	**16 April 2018**	17 May 2018					
	End	21 September 2017	**26 April 2018**	31 May 2018					
Permit log	Start	31 July 2017	**1 March 2018**	24 May 2018	18 October 2018	12 November 2018			
	End	14 September 2017	**29 March 2018**	14 June 2018	25 October 2018	20 November 2018			
Prepaid travel cost	Start		**11 January 2018**						
	End		**12 February 2018**						
Request for payment	Start		**11 January 2018**						
	End		**23 January 2018**						

Each column represents a drift. In bold, the drift that matches the one corresponding to the extension of the process to the whole university.

process to cover the entire university. However, in three of the five logs, *CRIER* detected additional drifts. This is likely due to the window size used by *CRIER*, which is based on a fixed number of traces rather than a specific period of time. The significant increase in the arrival rate of instances across all these processes—going from around 100 traces per month to over 600 in some cases—leads the window to capture a smaller timeframe, resulting in drift detection at a higher level of granularity. Upon further analysis, process models before and after each detected drift reveal distinct control flow changes. The evolution of these processes can be found in Appendix B, where the differences between the multiple process versions can be clearly observed. If the managers had been aware of these drifts, they could have made decisions regarding the process execution. Therefore, these detections likely reflect granularity issues rather than outright FPs.

These findings underscore the importance of selecting an appropriate window size in dynamic real-world environments to enhance the accuracy of the drift analysis in rapidly evolving processes.

6.6 Discussion

As shown previously, *CRIER* clearly outperforms the current *state-of-the-art* approaches in most of the logs, with no strong evidence of being inferior in the remaining cases. In particular, as it is shown in Table 1, for linear logs, *ProDrift* has a higher accuracy for some logs—`cf`, `cp`, `rp`, `ORI`, and `RIO`, with *slope* of 0.1%, and `pm` with *slopes* of 0.1% and 0.2%—in which the traces of the new behavior are added at a slower rate at the beginning of the change, while *CRIER* has clearly the best performance with higher insertion rates. This low frequency forces the conformance checking metrics to change slowly, as the behavior causing the change appears rarely, which in turn led

the slope of the regression to stay invariant until the behavior is more frequent, resulting in a late/early confirmation of the change depending on whether we are at the beginning or at the end of the change. Consequently, *CRIER* obtains high delays in detecting the start of the change, and premature detections at the end of the change, having a high number of FPs that lead to a decrease in accuracy. In fact, the results in terms of Δ demonstrate that the lower the *slope*, the higher the delay in *CRIER*, obtaining higher delays than *Lu et al.* in some logs with the slowest changes—`cf`, `pm`, `re`, `ORI`, and `OIR`, with *slope* of 0.1%, and `cf`, `re`, and `rp`, with *slope* of 0.2%. However, with respect to the average values of the metrics, *CRIER* is the best positioned since *ProDrift* and *Lu et al.* fail to detect many drifts—not being able to detect any changes in 25 and 4 logs out of 40, respectively—while *Martjushev et al.* obtains many FPs, with a F_{score} lower than 0.5.

As it is shown in Table 2, in logs with Gaussian distributions, the results obtained by *CRIER* are clearly better than those of *Martjushev et al.* and *Lu et al.*, with F_{score} of 1.0 in 17 of 20 logs. These logs are characterized by relatively fast changes, but with both a slow start and termination. The results are consistent with what was previously established for the linear changes: As these are relatively fast changes—slightly more than 150 traces in the slowest case—the slope of the regression changes abruptly, facilitating the detection of changes. The values of Δ and Υ obtained by *Martjushev et al.* are particularly indicative. The slow start and termination of the new behavior led to smaller changes in the distributions of the features, making the statistical test employed by this approach unable to detect the drifts and causing the beginning of the changes to be detected late—increasing Δ values—while their termination is detected prematurely—resulting in low Υ values. In the case of *Lu et al.*, the directly-follows relations used to represent process behaviors are able to detect most of the changes, with also a F_{score} of 1.0 in 7 of 20 logs. However, the results are lower in the rest, especially in the logs with `RIO` change patterns, where only one drift out of nine is detected. The bidirectional search used allows to correctly detect the beginning of the drifts in its forward pass, resulting in a Δ similar to that of *CRIER*, but it anticipates the end of the drift in its backward pass, obtaining a much lower Υ.

Something similar happens for logs with constant distribution. In this case, *CRIER* obtains the best results, with a perfect average F_{score} and an average Υ of around 90%. This is because, as soon as the change starts, half of the observed behavior is part of the new model. Consequently, conformance metrics change very fast, which means that the slope of the regression is significantly modified and changes are detected almost instantly. For *Martjushev et al.*, as it is shown in Table 4, there are several logs in which no change is detected—`rp` and `sw` for all *cdfs*. The poor performance of this algorithm for the same change patterns may be caused by the selected features since they are not able to capture such changes. Perhaps selecting other features that might be more sensitive to this type of changes would improve the results; however a more in-depth analysis should be performed to confirm this intuition. For *Lu et al.*, the approach also obtains a perfect F_{score} in seven cases, but it fails to detect any change in most of the logs with `cf`, `cp`, `pm`, `sw`, and `RIO`. Again, the forward pass of the bidirectional search allows to detect the starting point of the drifts with a low delay, but the backward pass anticipates the end of the drift, reducing the Υ obtained.

Finally, for logs with exponential changes, the results of *CRIER* are quite degraded when compared to the other distributions. These results are due to two distinct situations: On the one hand, in logs with $\lambda = 0.5$, changes are so fast—only 14 traces—that *CRIER* is not able to detect them as gradual, leading to FNs; and, on the other hand, in logs with $\lambda = 0.05$, changes have such long tails that they are terminated prematurely, and consequently, a high number of FPs are detected in traces that belong to the actual change but have not been detected as part of the drift. This behavior of the algorithm is reflected by the values of Δ and Υ. All changes are detected very close where they begin—with Δ below 10 traces in almost all cases, but Υ values remain around 20%, indicating that only the initial part of the change region is detected. In addition, logs in which *CRIER* obtains

the lowest Υ coincide with those with the worst F_{score}, confirming this hypothesis. In these cases, *Lu et al.* outperforms *CRIER*. Some modifications that could address this limitation involve the use of dynamic adaptation mechanisms for the window size, adjusting its size dynamically to accommodate the long tails and ensure that enough historical data are available to detect changes accurately, or the fine-tuning of the thresholds used for detecting changes to force the algorithm to wait for more drift candidates to confirm a change.

This limitation is common among most algorithms designed to detect gradual changes, as distinguishing rare traces from random noise purely through statistical methods becomes increasingly difficult in these scenarios.

To provide a specific example, let us consider the exponential distribution with $\lambda = 0.05$ from the experiments. By the 60th trace in the drift, there is still a 5% probability that a trace originates from the original model. After observing 100 traces, this probability only reduces to 1%. It takes approximately 140 traces for the probability to drop below 0.1%. During this extended period, although the probability of observing traces from the original model is low, it remains non-zero. Consequently, *CRIER* may interpret the absence of these traces over time as the completion of the change, leading to a premature detection of the end of the drift.

Notably, this challenge is not exclusive to exponential distributions. Similar issues can arise with linear distributions characterized by extremely shallow slopes. When changes progress slowly over an extended period, the frequency of traces originating from the new or old model may become too subtle for the algorithm to reliably distinguish. This increases the risk of prematurely deciding that the change has ended.

These cases highlight that the challenge lies in differentiating genuine signals of change from low-frequency deviations, regardless of the underlying distribution type. Addressing this issue requires enhanced detection strategies that go beyond traditional statistical approaches to better handle rare but significant trace patterns in both exponential and linear contexts.

7 Conclusions and Future Work

In this article, we presented *CRIER*, an offline gradual-drift detection algorithm. *CRIER* is based on the hypothesis that change detection and classification can be addressed by analyzing how the fitness and precision of the process models—the old model and the new one after the change—vary from the beginning to the end of the gradual change. Specifically, the hypothesis is that at the beginning there is a modification of the fitness, keeping the precision, while at the end the precision changes. This hypothesis has been mathematically demonstrated.

The approach has been validated using a synthetic dataset with 120 logs that present different change patterns and gradual change distributions, from linear to Gaussian, exponential and constant. The experiments show that *CRIER* outperforms, in general, the results of the main *state-of-the-art* approaches in terms of accuracy (F_{score}); delay (Δ), so drifts are detected faster; and change region overlapping (Υ), so the time interval during which two processes coexist is clearly identified.

Furthermore, *CRIER* is more robust for that change distributions in which the traces of the new model start to arise with a higher frequency at the beginning of the gradual drift, such as Gaussian and constant distributions. Nonetheless, our approach has proven to be more consistent even when the former conditions are not present such as when the drift follows a linear distribution with a low slope, being able to detect all the change patterns with low delay Δ and high Υ.

However, the algorithm has some limitations. First, the assumption that two sudden drifts bound a gradual change can be challenged if two changes overlap in time. In such cases, the algorithm may produce false detections that would need to be refined in a post-processing stage. Additionally, the window mechanism used is defined by a fixed number of traces. While this approach performs well in environments where the arrival rate of new traces is relatively constant, it can lead to

performance degradation in scenarios with highly uneven trace arrival rates over time. Lastly, the algorithm performs poorest when changes begin or end very slowly, as in the case of exponential changes.

Future work includes the exploration of alternative strategies for defining and updating windows, aiming to improve performance in scenarios where trace arrival rates vary significantly or changes present long tails, as seen in exponential logs. The use of dynamic or adaptive windowing methods could better handle these situations, reducing FPs and improving the detection of gradual or slow-onset changes. Additionally, we plan to tackle the remaining task related to the treatment of changes in process mining: the description of the changes and how they affect the process and the context in which they are executed. However, this task represents a much greater challenge, since most of the time the causes of the drifts are not directly captured in the data present in the logs.

References

[1] Arya Adriansyah. 2014. *Aligning Observed and Modeled Behaviour*. Ph.D. Dissertation. Department of Mathematics and Computer Science. DOI: https://doi.org/10.6100/IR770080

[2] Albert Bifet and Ricard Gavaldà. 2007. Learning from time-changing data with adaptive windowing. In *Proceedings of the 7th SIAM International Conference on Data Mining (SDM '07)*. SIAM, 443–448. DOI: https://doi.org/10.1137/1.9781611972771.42

[3] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. 2009. Trace clustering based on conserved patterns: Towards achieving better process models. In *Proceedings of the 2009 International Conference on Business Process Management Workshops (BPM '09)*. Springer, 170–181. DOI: https://doi.org/10.1007/978-3-642-12186-9_16

[4] R. P. Jagadeesh Chandra Bose, Wil M. P. van der Aalst, Indre Zliobaite, and Mykola Pechenizkiy. 2014. Dealing with concept drifts in process mining. *IEEE Transactions on Neural Networks and Learning Systems* 25 (2014), 154–171. DOI: https://doi.org/10.1109/TNNLS.2013.2278313

[5] Josep Carmona and Ricard Gavaldà. 2012. Online techniques for dealing with concept drift in process mining. In *Proceedings of the 11th International Symposium on Advances in Intelligent Data Analysis (IDA '12)*. Springer, 90–102. DOI: https://doi.org/10.1007/978-3-642-34156-4_10

[6] Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. 2018. *Conformance Checking—Relating Processes and Models*. Springer. DOI: https://doi.org/10.1007/978-3-319-99414-7

[7] Massimiliano de Leoni and Wil M. P. van der Aalst. 2013. Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In *Proceedings of the 2013 International Conference on Business Process Management (BPM '13)*. Springer, 113–129. DOI: https://doi.org/10.1007/978-3-642-40176-3_10

[8] Rafael Gaspar de Sousa, Antonio Carlos Meira Neto, Marcelo Fantinato, Sarajane Marques Peres, and Hajo Alexander Reijers. 2024. Integrated detection and localization of concept drifts in process mining with batch and stream trace clustering support. *Data & Knowledge Engineering* 149 (2024), 102253. DOI: https://doi.org/10.1016/J.DATAK.2023.102253

[9] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. 2018. *Fundamentals of Business Process Management* (2nd. ed.). Springer. DOI: https://doi.org/10.1007/978-3-662-56509-4

[10] Victor Gallego-Fontenla, Juan Vidal, and Manuel Lama. 2021. A conformance checking-based approach for sudden drift detection in business processes. *IEEE Transactions on Services Computing* 16, 1 (2021), 13–26. DOI: https://doi.org/10.1109/TSC.2021.3120031

[11] João Gama, Pedro Medas, Gladys Castillo, and Pedro Pereira Rodrigues. 2004. Learning with drift detection. In *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence (SBIA '04)*. Springer, 286–295. DOI: https://doi.org/10.1007/978-3-540-28645-5_29

[12] João Gama, Indre Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Computing Surveys* 46 (2014), 44:1–44:37. DOI: https://doi.org/10.1145/2523813

[13] Xin Gu, Joris Mulder, and Herbert Hoijtink. 2018. Approximated adjusted fractional Bayes factors: A general method for testing informative hypotheses. *British Journal of Mathematical and Statistical Psychology* 71 (2018), 229–261. DOI: https://doi.org/10.1111/bmsp.12110

[14] Jesús Huete, Abdulhakim Ali Qahtan, and Marwan Hassani. 2023. PrefixCDD: Effective online concept drift detection over event streams using prefix trees. In *Proceedings of the 47th IEEE Annual Computers, Software, and Applications Conference (COMPSAC '23)*. IEEE, 328–333. DOI: https://doi.org/10.1109/COMPSAC57700.2023.00051

[15] Harold Jeffreys. 1998. *The Theory of Probability*. Oxford University Press, Oxford. DOI: https://doi.org/10.1126/science.92.2395.479.b

[16] Sylvio Barbon Junior, Gabriel Marques Tavares, Victor G. Turrisi da Costa, Paolo Ceravolo, and Ernesto Damiani. 2018. A framework for human-in-the-loop monitoring of concept-drift detection in event log stream. In *Proceedings of the Web Conference 2018 (WWW '18)*. ACM, 319–326. DOI: https://doi.org/10.1145/3184558.3186343

[17] Fatemeh Khojasteh, Behshid Behkamal, Mohsen Kahani, and Mahsa Khorasani. 2023. Trace2Vec-CDD: A framework for concept drift detection in business process logs using trace embedding. *Computer and Knowledge Engineering* 6, 1 (2023), 71–79. DOI: https://doi.org/10.22067/CKE.2023.79099.1070

[18] Tianyang Li, Ting He, Zhongjie Wang, Yufeng Zhang, and Dian-Hui Chu. 2017. Unraveling process evolution by handling concept drifts in process mining. In *Proceedings of the 2017 IEEE International Conference on Services Computing (SCC '17)*. IEEE Computer Society, 442–449. DOI: https://doi.org/10.1109/SCC.2017.63

[19] Jonathon Love, Ravi Selker, Maarten Marsman, Tahira Jamil, Damian Dropmann, Josine Verhagen, Alexander Ly, Quentin F. Gronau, Martin Šmíra, Sacha Epskamp, et al. 2019. JASP: Graphical statistical software for common statistical designs. *Journal of Statistical Software* 88 (2019), 1–17. DOI: https://doi.org/10.18637/jss.v088.i02

[20] Yang Lu, Qifan Chen, and Simon K. Poon. 2021. A robust and accurate approach to detect process drifts from event streams. In *Proceedings of the 2021 International Conference on Business Process Management (BPM '21)*. Springer, 383–399. DOI: https://doi.org/10.1007/978-3-030-85469-0_24

[21] Daniela Luengo and Marcos Sepúlveda. 2011. Applying clustering in process mining to find different versions of a business process that changes over time. In *Proceedings of the 2011 International Conference on Business Process Management Workshops (BPM '11)*. Springer, 153–158. DOI: https://doi.org/10.1007/978-3-642-28108-2_15

[22] Alexander Ly, Angelika Stefan, Johnny van Doorn, Fabian Dablander, Don van den Bergh, Alexandra Sarafoglou, Simon Kucharskỳ, Koen Derks, Quentin F. Gronau, Akash Raj, et al. 2020. The Bayesian methodology of Sir Harold Jeffreys as a practical alternative to the *p*-value hypothesis test. *Computational Brain & Behaviour* 3 (2020), 153–161. DOI: https://doi.org/10.1007/s42113-019-00070-x

[23] Abderrahmane Maaradji, Marlon Dumas, Marcello La Rosa, and Alireza Ostovar. 2017. Detecting sudden and gradual drifts in business processes from execution traces. *IEEE Transactions on Knowledge and Data Engineering* 29 (2017), 2140–2154. DOI: https://doi.org/10.1109/TKDE.2017.2720601

[24] Abderrahmane Maaradji, Marlon Dumas, Marcello La Rosa, and Alireza Ostovar. 2015. Fast and accurate business process drift detection. In *Proceedings of the 2015 International Conference on Business Process Management (BPM '15)*. Springer, 406–422. DOI: https://doi.org/10.1007/978-3-319-23063-4_27

[25] Fabrizio Maria Maggi, Andrea Burattin, Marta Cimitile, and Alessandro Sperduti. 2013. Online process discovery to detect concept drifts in LTL-based declarative process models. In *Proceedings of the On the Move to Meaningful Internet Systems Conferences (OTM '13)*. Springer, 94–111. DOI: https://doi.org/10.1007/978-3-642-41030-7_7

[26] J. Martjushev, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. 2015. Change point detection and dealing with gradual and multi-order dynamics in process mining. In *Proceedings of the 14th International Conference on Perspectives in Business Informatics Research (BIR '15)*. Springer, 161–178. DOI: https://doi.org/10.1007/978-3-319-21915-8_11

[27] Jorge Munoz-Gama. 2016. *Conformance Checking and Diagnosis in Process Mining—Comparing Observed and Modeled Processes*. Springer. DOI: https://doi.org/10.1007/978-3-319-49451-7

[28] Antonio Carlos Meira Neto, Rafael Gaspar de Sousa, Marcelo Fantinato, and Sarajane Marques Peres. 2023. Towards a transition matrix-based concept drift approach: Experiments on the detection task. In *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS '23)*. SCITEPRESS, 361–372. DOI: https://doi.org/10.5220/0011843600003467

[29] Antonio Carlos Meira Neto, Rafael Gaspar de Sousa, Marcelo Fantinato, and Sarajane Marques Peres. 2024. Revisiting the transition matrix-based concept drift approach: Improving the detection task reliability through additional experimentation. *SN Computer Science* 5, 1 (2024), 188. DOI: https://doi.org/10.1007/S42979-023-02536-Z

[30] Vincenzo Pasquadibisceglie, Annalisa Appice, Giovanna Castellano, Nicola Fiorentino, and Donato Malerba. 2023. STARDUST: A novel process mining approach to discover evolving models from trace streams. *IEEE Transactions on Services Computing* 16 (2023), 2970–2984. DOI: https://doi.org/10.1109/TSC.2022.3215502

[31] Stephen Pauwels and Toon Calders. 2021. Incremental predictive process monitoring: The next activity case. In *Proceedings of the 19th International Conference in Business Process Management (BPM '21)*, Vol. 12875. Springer, 123–140. DOI: https://doi.org/10.1007/978-3-030-85469-0_10

[32] Gordon J. Ross, Niall M. Adams, Dimitris K. Tasoulis, and David J. Hand. 2012. Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters* 33, 2 (2012), 191–198. DOI: https://doi.org/10.1016/J.PATREC.2011.08.019

[33] Anne Rozinat and Wil M. P. van der Aalst. 2008. Conformance checking of processes based on monitoring real behaviour. *Information Systems* 33 (2008), 64–95. DOI: https://doi.org/10.1016/j.is.2007.07.001

[34] Denise Maria Vecino Sato, Sheila Cristiana De Freitas, Jean Paul Barddal, and Edson Emilio Scalabrin. 2021. A survey on concept drift in process mining. *ACM Computing Surveys* 54 (2021), 189:1–189:38. DOI: https://doi.org/10.1145/3472752

[35] Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. 2017. Detecting concept drift in processes using graph metrics on process graphs. In *Proceedings of the 9th Conference on Subject-Oriented Business Process Management (S-BPM ONE '17)*. ACM, 1–10.

[36] Florian Stertz and Stefanie Rinderle-Ma. 2018. Process histories—Detecting and representing concept drifts based on event streams. In *Proceedings of the On the Move to Meaningful Internet Systems Conferences (OTM '18)*. Springer, 318–335. DOI: https://doi.org/10.1007/978-3-030-02610-3_18

[37] Gabriel Marques Tavares, Paolo Ceravolo, Victor G. Turrisi da Costa, Ernesto Damiani, and Sylvio Barbon Junior. 2019. Overlapping analytic stages in online process mining. In *Proceedings of the 2019 IEEE International Conference on Services Computing (SCC '19)*. IEEE, 167–175. DOI: https://doi.org/10.1109/SCC.2019.00037

[38] Wil M. P. van der Aalst. 1998. The application of Petri nets to workflow management. *Journal of Circuits, Systems and Computers* 8 (1998), 21–66. DOI: https://doi.org/10.1142/S0218126698000043

[39] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. 2012. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2 (2012), 182–192. DOI: https://doi.org/10.1002/widm.1045

[40] Wil M. P. van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose R. P., Peter van den Brand, Ronald Brandtjen, Joos C. A. M. Buijs, et al. 2011. Process mining manifesto. In *Proceedings of the 2011 International Conference on Business Process Management Workshops (BPM '11)*. Springer, 169–194. DOI: https://doi.org/10.1007/978-3-642-28108-2_19

[41] Boudewijn van Dongen. 2020. BPI Challenge 2020. DOI: https://doi.org/10.4121/UUID:52FB97D4-4588-43C9-9D04-3604D4613B51

[42] Seppe K. L. M. vanden Broucke, Jochen De Weerdt, Jan Vanthienen, and Bart Baesens. 2014. Determining process model precision and generalization with weighted artificial negative events. *IEEE Transactions on Knowledge and Data Engineering* 26 (2014), 1877–1889. DOI: https://doi.org/10.1109/TKDE.2013.130

[43] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. 2008. Change patterns and change support features—Enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering* 66 (2008), 438–466. DOI: https://doi.org/10.1016/j.datak.2008.05.001

[44] Jiuyun Xu, Yue Zhang, and Qiang Duan. 2023. Concept drift detection and localization framework based on behaviour replacement. *Applied Intelligence* 53, 13 (2023), 16776–16796. DOI: https://doi.org/10.1007/S10489-022-04341-2

[45] Anton Yeshchenko, Claudio Di Ciccio, Jan Mendling, and Artem Polyvyanyy. 2019. Comprehensive process drift detection with visual analytics. In *Proceedings of the 38th International Conference on Conceptual Modeling (ER '19)*. Springer, 119–135. DOI: https://doi.org/10.1007/978-3-030-33223-5_11

[46] Hang Yu, Jinpeng Li, Jie Lu, Yiliao Song, Shaorong Xie, and Guangquan Zhang. 2024. Type-LDD: A type-driven lite concept drift detector for data streams. *IEEE Transactions on Knowledge and Data Engineering* 36, 12 (2024), 9476–9489. DOI: https://doi.org/10.1109/TKDE.2023.3344602

[47] Canbin Zheng, Lijie Wen, and Jianmin Wang. 2017. Detecting process concept drifts from event logs. In *Proceedings of the On the Move to Meaningful Internet Systems Conferences (OTM '17)*. Springer, 524–542. DOI: https://doi.org/10.1007/978-3-319-69462-7_33

Appendices

A Detailed Experiments Results

This appendix contains the detailed results for the experiments of *CRIER* for the synthetic logs of the loan application process.

A.1 Linear Logs

Table A1 shows the results for lineal gradual changes with slopes of 0.1%, 0.2%, 0.5%, and 1%. *CRIER* clearly outperforms the other proposals by a margin of more than 0.3 in terms of average F_{score}, with *ProDrift* obtaining better values for 6 logs out of 10 with 0.1% slope and *Lu et al.* matching or even exceeding the best results in 3 logs out of 4 with `ROI` change patterns. However, when changes happen at a faster rate—i.e., the slope of the linear combination increases, *ProDrift* performance significantly drops, detecting almost no changes, as is the case for *Lu et al.* in `RIO` logs. *CRIER* achieves better results also in terms of delay (Δ), with values typically bellow 50 traces and always clearly smaller than the obtained by *Martjushev et al.* and *ProDrift*, and also better than *Lu et al.*, except for five logs with 0.1% slope—`cf`, `pm`, `re`, `ORI`, `RIO`—where the later achieves lower delays. Finally, in terms of Υ, *CRIER* obtains better results than the rest of the algorithms in all cases but six—three for *ProDrift* and three for *Lu et al.*

Table A1. F_{score}, Δ, and ϒ Values for the Logs with Linear Gradual Changes with Slopes of 0.1%, 0.2%, 0.5%, and 1.0%

		CRIER			*Martjushev et al.*			*ProDrift*			*Lu et al.*		
		F_{score}	Δ	ϒ	F_{score}	Δ	ϒ	F_{score}	Δ	ϒ	F_{score}	Δ	ϒ
cf	*slope* = 0.1%	0.8000	45.5000	79.86%	0.6207	477.1111	27.90%	0.8750	240.7143	66.28%	0.3077	1.5000	22.19%
	slope = 0.2%	1.0000	14.4444	90.93%	0.7826	100.4444	47.42%	0.3636	240.0000	9.00%	0.3333	10.5000	11.16%
	slope = 0.5%	1.0000	19.8889	78.67%	0.7273	219.1250	18.00%	0.0000	-	0.00%	0.2857	92.5000	3.94%
	slope = 1.0%	1.0000	15.3333	66.22%	0.3810	182.2500	10.11%	0.0000	-	0.00%	0.7500	23.8333	25.33%
cp	*slope* = 0.1%	0.7619	42.7500	79.23%	0.7200	364.5556	29.44%	0.8750	245.4286	64.96%	0.4000	333.6667	22.18%
	slope = 0.2%	1.0000	24.7778	87.29%	0.8000	137.7500	41.80%	0.7143	235.7500	24.42%	0.2667	249.5000	0.09%
	slope = 0.5%	1.0000	18.1111	77.94%	0.7368	243.2857	30.44%	0.0000	-	0.00%	0.6250	36.4000	28.44%
	slope = 1.0%	1.0000	11.2222	70.33%	0.4444	280.7500	8.89%	0.0000	-	0.00%	0.7143	16.0000	27.56%
pm	*slope* = 0.1%	0.3448	105.6000	43.72%	0.5000	283.3333	15.09%	0.8750	216.2857	68.73%	0.3333	9.5000	0.04%
	slope = 0.2%	0.8000	38.3750	74.80%	0.6154	139.7500	25.84%	0.8235	203.1667	42.67%	0.0000	-	0.00%
	slope = 0.5%	0.9474	27.6667	78.28%	0.5000	170.3333	29.78%	0.3333	406.5000	22.00%	0.3333	100.5000	4.39%
	slope = 1.0%	0.7368	15.4286	54.00%	0.0000	-	0.00%	0.5000	433.6667	22.44%	0.5714	37.3333	8.22%
re	*slope* = 0.1%	0.8571	27.6667	82.39%	0.5806	337.7500	24.82%	0.4615	288.3333	20.26%	0.5714	0.0000	44.30%
	slope = 0.2%	1.0000	16.7778	88.20%	0.7500	157.1111	34.47%	0.0000	-	0.00%	0.3333	7.0000	21.80%
	slope = 0.5%	1.0000	10.3333	82.39%	0.4762	193.6000	16.78%	0.0000	-	0.00%	0.6667	46.6000	36.06%
	slope = 1.0%	1.0000	8.1111	68.78%	0.1905	41.0000	12.00%	0.0000	-	0.00%	1.0000	6.2222	83.00%
rp	*slope* = 0.1%	0.5385	60.5714	56.50%	0.2000	881.0000	1.32%	0.8750	236.1429	63.11%	0.5333	253.7500	33.28%
	slope = 0.2%	0.9474	41.6667	81.89%	0.0000	-	0.00%	0.5000	243.3333	12.91%	0.4615	8.3333	22.22%
	slope = 0.5%	0.9474	12.5556	71.94%	0.0000	-	0.00%	0.0000	-	0.00%	0.7143	37.6000	32.00%
	slope = 1.0%	1.0000	12.0000	68.89%	0.0000	-	0.00%	0.0000	-	0.00%	0.7143	17.2000	38.33%
sw	*slope* = 0.1%	0.7273	54.0000	79.32%	0.0000	-	0.00%	0.0000	-	0.00%	0.5714	250.2500	33.44%
	slope = 0.2%	1.0000	21.1111	86.33%	0.0000	-	0.00%	0.0000	-	0.00%	0.3077	250.0000	0.07%
	slope = 0.5%	1.0000	13.3333	81.00%	0.0000	-	0.00%	0.0000	-	0.00%	0.7500	29.6667	49.44%
	slope = 1.0%	1.0000	10.1111	66.44%	0.0000	-	0.00%	0.0000	-	0.00%	0.8750	10.1429	56.78%
OIR	*slope* = 0.1%	0.6087	45.4286	70.71%	0.5455	474.1111	26.09%	0.0000	-	0.00%	0.5882	208.6000	43.92%
	slope = 0.2%	0.6364	14.7143	70.18%	0.6667	173.1250	37.71%	0.0000	-	0.00%	0.4444	499.0000	11.20%
	slope = 0.5%	1.0000	8.2222	90.56%	0.6957	186.1250	13.44%	0.0000	-	0.00%	0.5556	28.2000	32.00%
	slope = 1.0%	1.0000	11.6667	80.56%	0.5600	60.8571	27.00%	0.0000	-	0.00%	0.5556	17.4000	31.22%
ORI	*slope* = 0.1%	0.4444	81.5000	56.19%	0.5000	435.8889	26.08%	0.8750	325.1429	54.56%	0.2857	3.5000	22.11%
	slope = 0.2%	0.7619	27.6250	77.40%	0.5385	208.7143	22.62%	0.3636	341.5000	3.36%	0.4286	332.6667	11.18%
	slope = 0.5%	0.9474	34.4444	73.94%	0.6087	209.1429	17.00%	0.0000	-	0.00%	0.4000	28.0000	18.39%
	slope = 1.0%	1.0000	11.4444	68.67%	0.4348	31.4000	29.44%	0.0000	-	0.00%	0.8235	25.2857	40.33%
RIO	*slope* = 0.1%	0.6364	80.2857	67.39%	0.3636	209.5000	10.09%	0.8750	216.5714	69.41%	0.0000	-	0.00%
	slope = 0.2%	1.0000	36.7778	83.76%	0.2000	79.0000	7.58%	0.9412	196.8571	56.84%	0.0000	-	0.00%
	slope = 0.5%	1.0000	27.6667	78.22%	0.2000	32.0000	11.11%	0.0000	-	0.00%	0.0000	-	0.00%
	slope = 1.0%	1.0000	17.2222	70.78%	0.5000	145.0000	33.33%	0.0000	-	0.00%	0.1818	47.0000	2.22%
ROI	*slope* = 0.1%	0.7619	31.3750	81.29%	0.8421	287.0000	26.34%	0.0000	-	0.00%	0.4444	0.2500	44.39%
	slope = 0.2%	1.0000	13.7778	92.36%	0.7368	194.7143	34.51%	0.0000	-	0.00%	1.0000	134.8889	53.00%
	slope = 0.5%	0.9000	9.0000	82.28%	0.9412	247.3750	46.28%	0.0000	-	0.00%	1.0000	15.2222	91.94%
	slope = 1.0%	1.0000	7.3333	81.00%	0.3333	223.6667	22.89%	0.0000	-	0.00%	1.0000	5.3333	86.67%
Average	*slope* = 0.1%	0.6481	57.4677	69.66%	0.4873	416.6944	17.70%	0.5712	252.6599	37.89%	0.4036	117.8907	26.59%
	slope = 0.2%	0.9146	25.0048	83.31%	0.5090	148.8261	22.73%	0.3706	243.4345	15.58%	0.3576	186.4861	13.07%
	slope = 0.5%	0.9742	18.1222	79.52%	0.4886	187.6234	18.31%	0.0333	406.5000	2.22%	0.5331	46.0765	29.66%
	slope = 1.0%	0.9737	11.9873	69.57%	0.2844	137.8463	14.84%	0.0500	433.6667	2.24%	0.7186	20.5751	39.97%

The colors highlight the best performing approach for each metric.

Table A2. F_{score}, Δ, and Υ Values for the Logs with Gaussian Gradual Changes with ($\mu = 20, \sigma^2 = 10$) and ($\mu = 50, \sigma^2 = 30$)

		CRIER			*Martjushev et al.*			*Lu et al.*		
		F_{score}	Δ	Υ	F_{score}	Δ	Υ	F_{score}	Δ	Υ
cf	$\mu = 20$ $\sigma^2 = 10$	0.9412	10.8750	61.22%	0.2727	25.6667	20.92%	0.8889	11.2500	40.09%
	$\mu = 50$ $\sigma^2 = 30$	1.0000	9.7778	70.78%	0.5000	255.6000	6.84%	0.7143	26.7500	14.92%
cp	$\mu = 20$ $\sigma^2 = 10$	1.0000	8.2222	67.97%	0.2105	170.0000	0.44%	1.0000	11.2222	57.52%
	$\mu = 50$ $\sigma^2 = 30$	1.0000	8.6667	63.33%	0.7368	283.0000	18.80%	0.7143	15.4000	21.68%
pm	$\mu = 20$ $\sigma^2 = 10$	1.0000	10.6667	55.56%	0.3636	168.5000	22.22%	0.7059	12.6667	35.51%
	$\mu = 50$ $\sigma^2 = 30$	1.0000	14.8889	66.36%	0.6154	94.2500	44.44%	0.4286	40.0000	4.66%
re	$\mu = 20$ $\sigma^2 = 10$	1.0000	6.4444	63.40%	0.2000	55.5000	12.42%	1.0000	4.4444	68.19%
	$\mu = 50$ $\sigma^2 = 30$	1.0000	10.0000	62.94%	0.4000	176.0000	9.25%	0.6250	19.6000	27.89%
rp	$\mu = 20$ $\sigma^2 = 10$	1.0000	7.1111	64.49%	0.0000	-	0.00%	1.0000	7.1111	63.18%
	$\mu = 50$ $\sigma^2 = 30$	1.0000	9.8889	60.22%	0.0000	-	0.00%	0.7143	15.0000	20.67%
sw	$\mu = 20$ $\sigma^2 = 10$	1.0000	12.2222	63.83%	0.0000	-	0.00%	1.0000	10.2222	52.29%
	$\mu = 50$ $\sigma^2 = 30$	0.9474	8.1111	59.75%	0.0000	-	0.00%	0.7143	10.2000	28.28%
OIR	$\mu = 20$ $\sigma^2 = 10$	1.0000	6.4444	64.05%	0.5000	5.3333	61.44%	1.0000	6.8889	54.03%
	$\mu = 50$ $\sigma^2 = 30$	1.0000	25.6667	61.93%	0.5000	72.6667	26.34%	0.5882	14.8000	23.15%
ORI	$\mu = 20$ $\sigma^2 = 10$	1.0000	8.6667	62.31%	0.2727	11.6667	29.85%	1.0000	10.1111	58.39%
	$\mu = 50$ $\sigma^2 = 30$	1.0000	6.1429	47.40%	0.6087	141.7143	20.44%	0.7143	8.8000	23.00%
RIO	$\mu = 20$ $\sigma^2 = 10$	0.9412	10.3750	60.57%	0.0000	-	0.00%	0.1538	49.0000	0.44%
	$\mu = 50$ $\sigma^2 = 30$	1.0000	15.4444	64.34%	0.3636	13.0000	21.13%	0.1818	32.0000	2.56%
ROI	$\mu = 20$ $\sigma^2 = 10$	1.0000	6.7778	69.28%	0.4444	253.5000	17.86%	1.0000	4.7778	69.72%
	$\mu = 50$ $\sigma^2 = 30$	0.9000	7.3333	68.07%	0.8421	254.5000	36.52%	0.7778	18.5714	23.11%
Average	$\mu = 20$ $\sigma^2 = 10$	0.9882	8.7806	63.50%	0.2264	98.5952	16.03%	0.8749	12.7694	51.03%
	$\mu = 50$ $\sigma^2 = 30$	0.9847	11.5921	61.59%	0.4567	161.3414	19.66%	0.6173	20.1121	23.11%

The colors highlight the best performing approach for each metric.

A.2 Gaussian Logs

Table A2 shows the results for the detection of gradual changes in logs that follow two Gaussian distributions. *CRIER* clearly outperforms *Martjushev et al.* in F_{score}, Δ and Υ, obtaining values very close to 1 in F_{score}, delays always lower than 25 traces for almost all the logs, and Υ of more than 60%. *Lu et al.* also obtains a perfect F_{score} for 7 logs and delays lower than 50—with slightly better results than *CRIER* in 5 logs out of 16, but its performance decreases in the rest of the cases.

A.3 Exponential Logs

Table A3 shows the results for the detection of gradual changes in logs that follow three exponential distributions with λ set to 0.05, 0.1, and 0.5. *CRIER* also obtains the best results in terms of Δ, with values below 10 traces, clearly lower than those of *Lu et al.* and *Martjushev et al.*, except for $\lambda = 0.50$ where both *CRIER* and *Lu et al.* obtain average delays of less than three traces. In terms of F_{score},

Table A3. F_{score}, Δ, and Υ Values for the Logs with Exponential Gradual Changes with λ Set to 0.05, 0.1, and 0.5

		CRIER			*Martjushev et al.*			*Lu et al.*		
		F_{score}	Δ	Υ	F_{score}	Δ	Υ	F_{score}	Δ	Υ
cf	$\lambda = 0.05$	0.5000	6.0000	23.02%	0.3333	205.2500	11.99%	0.5333	5.0000	9.03%
	$\lambda = 0.10$	1.0000	5.2222	60.00%	0.2857	38.6667	11.43%	0.5882	5.2000	6.98%
	$\lambda = 0.50$	0.3636	2.0000	19.05%	0.2857	9.0000	33.33%	0.7778	2.5714	31.75%
cp	$\lambda = 0.05$	0.4615	8.8333	12.39%	0.5000	297.8000	15.83%	0.6250	8.2000	9.83%
	$\lambda = 0.10$	0.9412	6.3750	53.81%	0.2222	300.5000	6.51%	0.8235	14.1429	22.86%
	$\lambda = 0.50$	0.3636	3.0000	17.46%	0.1111	285.0000	0.79%	0.8889	5.7500	23.81%
pm	$\lambda = 0.05$	0.4167	9.8000	25.82%	0.2000	166.0000	11.11%	0.4286	14.3333	6.24%
	$\lambda = 0.10$	1.0000	8.7778	49.68%	0.2000	217.0000	11.11%	0.5714	19.2500	6.03%
	$\lambda = 0.50$	0.2000	3.0000	8.73%	0.0000	-	0.00%	0.4706	3.2500	19.84%
re	$\lambda = 0.05$	0.6364	7.4286	34.69%	0.5000	92.6667	22.14%	0.7778	46.1429	21.58%
	$\lambda = 0.10$	1.0000	3.7778	57.46%	0.4348	47.6000	17.14%	1.0000	2.5556	57.46%
	$\lambda = 0.50$	0.7143	2.8000	44.44%	0.1905	7.5000	11.11%	1.0000	1.5556	46.03%
rp	$\lambda = 0.05$	0.4800	8.3333	17.43%	0.0000	-	0.00%	0.5556	11.4000	10.15%
	$\lambda = 0.10$	0.9412	4.2500	52.70%	0.0000	-	0.00%	0.8889	18.2500	19.05%
	$\lambda = 0.50$	0.0000	-	0.00%	0.0000	-	0.00%	0.7778	3.4286	15.87%
sw	$\lambda = 0.05$	0.2963	11.5000	15.51%	0.0000	-	0.00%	0.7778	31.1667	11.11%
	$\lambda = 0.10$	0.7000	5.7143	38.73%	0.0000	-	0.00%	0.7778	14.4286	29.21%
	$\lambda = 0.50$	0.8000	3.3333	50.79%	0.0000	-	0.00%	1.0000	1.1250	50.00%
OIR	$\lambda = 0.05$	1.0000	5.2222	69.06%	0.6154	50.3750	41.25%	0.8889	55.3750	11.43%
	$\lambda = 0.10$	1.0000	3.7778	60.48%	0.6667	20.8889	61.27%	1.0000	28.2222	25.08%
	$\lambda = 0.50$	0.6154	2.5000	36.51%	0.6667	13.1111	99.21%	1.0000	2.2222	37.30%
ORI	$\lambda = 0.05$	0.5000	9.8333	29.42%	0.5600	52.0000	29.58%	0.6316	30.0000	9.67%
	$\lambda = 0.10$	0.6000	6.5000	35.40%	0.5000	22.1429	53.02%	0.7778	24.1429	10.63%
	$\lambda = 0.50$	0.3636	2.5000	18.25%	0.4167	11.2000	49.21%	1.0000	4.8889	26.19%
RIO	$\lambda = 0.05$	0.8000	12.1250	46.44%	0.2000	160.0000	11.11%	0.0000	-	0.00%
	$\lambda = 0.10$	0.9412	8.1250	56.67%	0.2000	233.0000	11.11%	0.0000	-	0.00%
	$\lambda = 0.50$	0.3636	3.5000	16.67%	0.2000	192.0000	11.11%	0.2353	1.0000	19.05%
ROI	$\lambda = 0.05$	0.5000	6.6667	29.26%	0.7368	247.1429	25.34%	1.0000	64.4444	22.30%
	$\lambda = 0.10$	0.8750	4.4286	46.98%	0.5263	236.6000	38.10%	1.0000	2.4286	44.29%
	$\lambda = 0.50$	0.8000	2.8333	53.17%	0.2222	251.5000	22.22%	1.0000	0.5556	49.21%
Average	$\lambda = 0.05$	0.5591	8.5742	31.11%	0.3646	158.9043	17.37%	0.6218	29.5625	11.37%
	$\lambda = 0.10$	0.8999	5.6948	50.21%	0.3036	139.5498	22.03%	0.7428	14.2912	23.84%
	$\lambda = 0.50$	0.4584	2.8296	27.34%	0.2093	109.9016	21.52%	0.8150	2.6347	31.92%

The colors highlight the best performing approach for each metric.

Lu et al. outperforms *CRIER* in most of the logs—20 out of 30, with better average results for $\lambda = 0.05$ and $\lambda = 0.50$, but lower for $\lambda = 0.10$. *CRIER* achieves also the best results of Υ, that are, on average, twice better than *Martjushev et al.* and *Lu et al.* However, it is worth noting that Υ values are usually lower with the exponential distribution, if compared with the other distributions.

Table A4. F_{score}, Δ, and ϒ Values for the Logs with Constant Gradual Changes with p = 0.5 and n Set to 100, 200, and 500

			CRIER			*Martjushev et al.*			*Lu et al.*		
			F_{score}	Δ	ϒ	F_{score}	Δ	ϒ	F_{score}	Δ	ϒ
cf	p = 0.5	n = 100	1.0000	5.4444	84.33%	0.3158	230.0000	2.22%	0.7059	27.8333	35.78%
	p = 0.5	n = 200	1.0000	6.2222	93.00%	0.7273	172.5000	26.06%	0.0000	-	0.00%
	p = 0.5	n = 500	1.0000	6.2222	97.36%	0.9474	127.2222	46.69%	0.0000	-	0.00%
cp	p = 0.5	n = 100	1.0000	5.6667	84.33%	1.0000	263.5556	30.44%	0.7778	26.4286	52.22%
	p = 0.5	n = 200	1.0000	5.1111	92.22%	0.8889	287.7500	28.72%	0.0000	-	0.00%
	p = 0.5	n = 500	1.0000	4.8889	96.56%	1.0000	125.7778	53.64%	0.0000	-	0.00%
pm	p = 0.5	n = 100	1.0000	4.5556	86.44%	0.0000	-	0.00%	0.6667	15.0000	35.44%
	p = 0.5	n = 200	1.0000	4.4444	94.72%	0.0000	-	0.00%	0.0000	-	0.00%
	p = 0.5	n = 500	1.0000	6.6667	97.67%	0.6154	81.7500	37.73%	0.0000	-	0.00%
re	p = 0.5	n = 100	1.0000	3.2222	84.22%	0.1053	93.0000	0.78%	1.0000	2.2222	95.44%
	p = 0.5	n = 200	1.0000	3.5556	93.39%	0.5714	230.1667	17.33%	0.1429	175.0000	1.28%
	p = 0.5	n = 500	1.0000	3.0000	96.49%	0.7273	213.7500	37.71%	0.1176	1.0000	0.09%
rp	p = 0.5	n = 100	1.0000	17.1111	63.11%	0.0000	-	0.00%	1.0000	5.2222	90.11%
	p = 0.5	n = 200	1.0000	5.2222	90.94%	0.0000	-	0.00%	0.0000	-	0.00%
	p = 0.5	n = 500	1.0000	5.8889	96.89%	0.0000	-	0.00%	0.1667	499.0000	0.02%
sw	p = 0.5	n = 100	1.0000	4.6667	87.33%	0.0000	-	0.00%	1.0000	2.6667	91.89%
	p = 0.5	n = 200	1.0000	4.2222	94.44%	0.0000	-	0.00%	0.0000	-	0.00%
	p = 0.5	n = 500	1.0000	6.8889	96.29%	0.0000	-	0.00%	0.0000	-	0.00%
OIR	p = 0.5	n = 100	1.0000	2.7778	89.22%	0.6154	39.7500	49.44%	1.0000	2.7778	94.67%
	p = 0.5	n = 200	1.0000	3.0000	95.06%	0.5455	191.8333	19.22%	0.1250	1.0000	0.06%
	p = 0.5	n = 500	1.0000	3.1111	96.73%	0.8571	198.7778	38.87%	0.1176	1.0000	0.24%
ORI	p = 0.5	n = 100	1.0000	5.4444	86.56%	0.4545	88.6000	25.89%	1.0000	5.2222	77.56%
	p = 0.5	n = 200	1.0000	3.5556	96.06%	0.8571	232.5556	19.61%	0.1333	196.0000	0.17%
	p = 0.5	n = 500	1.0000	3.8889	97.24%	0.6667	256.4444	40.60%	0.0000	-	0.00%
RIO	p = 0.5	n = 100	1.0000	12.1111	83.67%	0.0000	-	0.00%	0.1818	55.0000	2.78%
	p = 0.5	n = 200	1.0000	10.0000	93.11%	0.5000	171.6667	33.06%	0.0000	-	0.00%
	p = 0.5	n = 500	1.0000	9.8889	97.40%	0.3636	21.0000	20.22%	0.0000	-	0.00%
ROI	p = 0.5	n = 100	1.0000	3.3333	90.89%	0.8421	245.5714	42.22%	1.0000	1.3333	96.22%
	p = 0.5	n = 200	1.0000	3.0000	96.33%	1.0000	254.4444	44.94%	1.0000	44.2222	66.11%
	p = 0.5	n = 500	1.0000	3.3333	98.36%	0.9474	153.3333	54.98%	0.6316	162.6667	33.91%
Average	p = 0.5	n = 100	1.0000	6.4333	83.98%	0.3333	160.0795	16.53%	0.8332	14.3706	70.37%
	p = 0.5	n = 200	1.0000	4.8333	94.03%	0.5090	220.1310	18.10%	0.1401	104.0556	7.51%
	p = 0.5	n = 500	1.0000	5.3778	97.07%	0.6125	147.2569	33.04%	0.1034	165.9167	3.81%

The colors highlight the best performing approach for each metric.

A.4 Constant Logs

Table A4 shows the results for the detection of gradual changes in logs that follow the three different constant distributions. In these logs, *CRIER* obtained the best results, with a perfect F_{score} for all the cases, average Δ of less than 10 traces, and ϒ values above 80%. *Lu et al.* also obtains a perfect F_{score} for seven logs, but it significantly deteriorates its results in the rest, especially in some logs with n = 200 and n = 500—`cf`, `cp`, `pm`, `sw`, and `RIO`, where it is not able to detect any change. In terms of Δ and ϒ, the results show a large variability, with values between 1 and 500 traces and 0% and 96%, respectively. On the contrary, *Martjushev et al.* achieves average Δ values between 140

and 220 traces, Υ values always below 60%, and average F_{score} values that do not reach 0.6, with also several cases in which no change is detected—as `rp` and `sw`.

B BPIC Models

This appendix contains the graphical representation for the models discovered after each drift detected in the BPIC 2020 logs.

B.1 Domestic Declarations

Figures B1–B9 depict the different versions of the domestic declarations process.

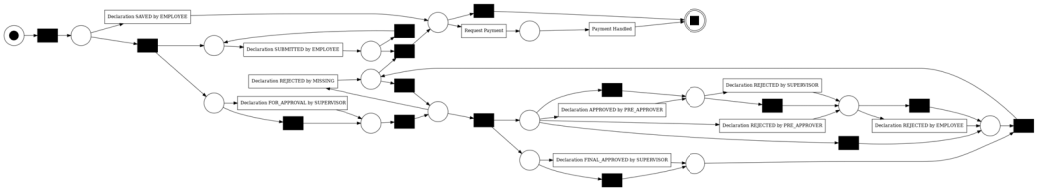


Fig. B1. Original process model for *Domestic declarations* until 29 June 2017.

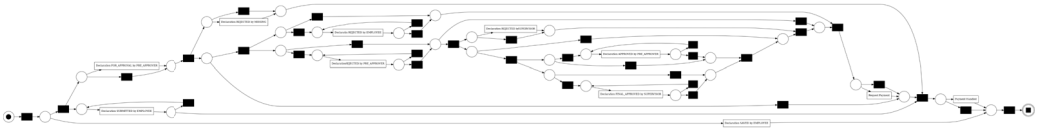


Fig. B2. Process model for *Domestic declarations* between 7 September 2017 and 18 December 2017.

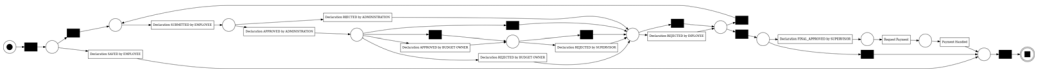


Fig. B3. Process model for *Domestic declarations* between 22 January 2018 and 15 February 2018.

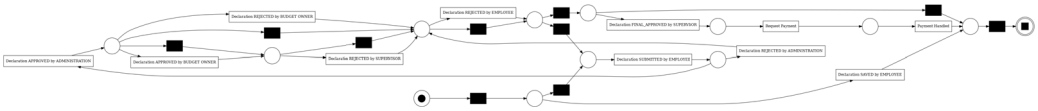


Fig. B4. Process model for *Domestic declarations* between 22 February 2018 and 12 April 2018.

Fig. B5. Process model for *Domestic declarations* between 24 April 2018 and 25 June 2018.

B.2 International Declarations

Figures B10–B13 depict the different versions of the international declarations process.

Fig. B6. Process model for *Domestic declarations* between 28 June 2018 and 10 October 2018.

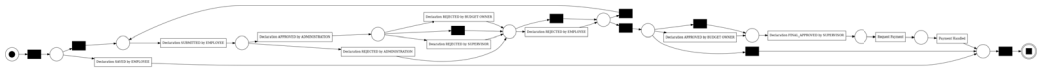


Fig. B7. Process model for *Domestic declarations* between 22 October 2018 and 26 November 2018.

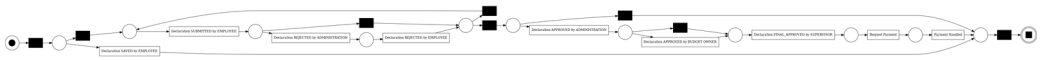


Fig. B8. Process model for *Domestic declarations* between 3 December 2018 and 12 December 2018.

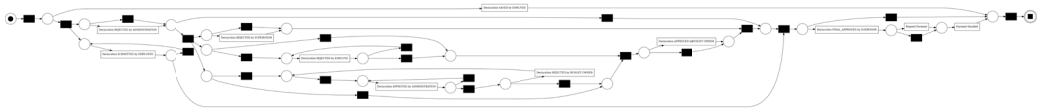


Fig. B9. Final process model for *Domestic declarations* after 13 December 2018.

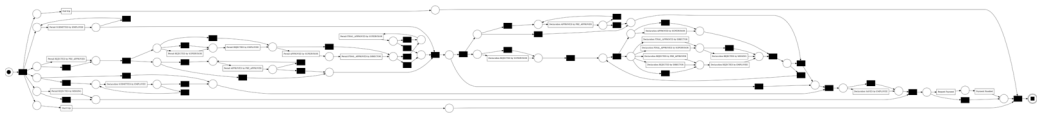


Fig. B10. Original process model for *International declarations* until 10 August 2017.

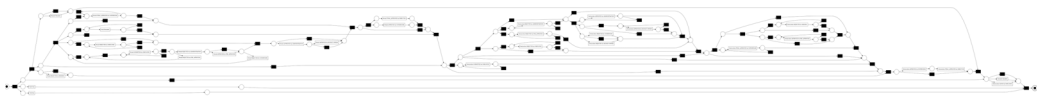


Fig. B11. Process model for *International declarations* between 21 September 2017 and 16 April 2018.

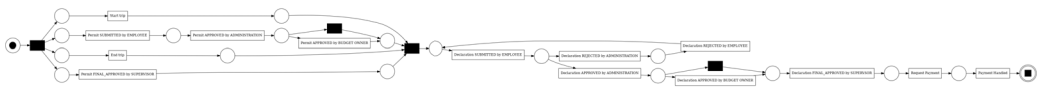


Fig. B12. Process model for *International declarations* between 26 April 2018 and 7 May 2018.

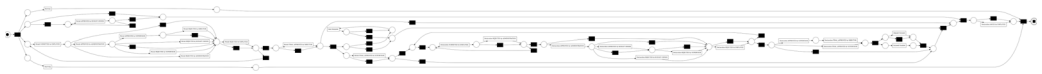


Fig. B13. Final process model for *International declarations* after 31 May 2018.

B.3 Permit

Figures B14–B19 depict the different versions of the permit process.

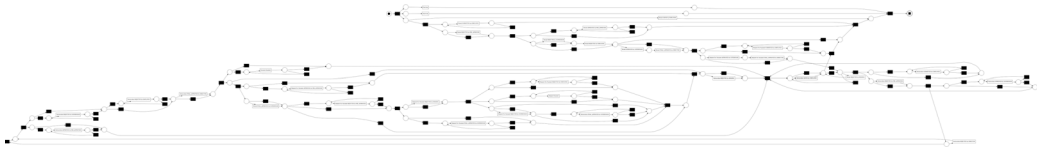


Fig. B14. Original process model for *Permit* until 31 July 2017.

Fig. B15. Process model for *Permit* between 14 September 2017 and 1 March 2018.

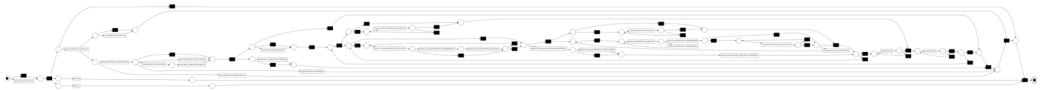


Fig. B16. Process model for *Permit* between 29 March 2018 and 24 May 2018.

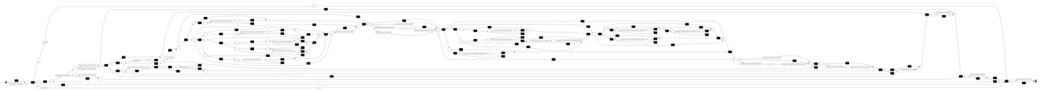


Fig. B17. Process model for *Permit* between 14 June 2018 and 18 October 2018.

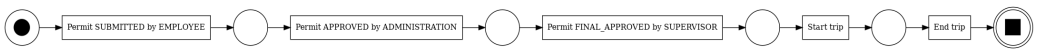


Fig. B18. Process model for *Permit* between 25 October 2018 and 12 November 2018.

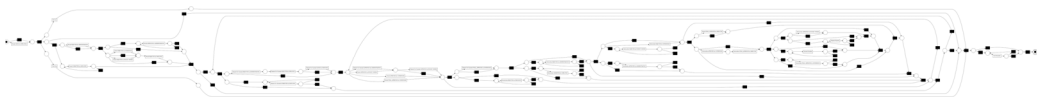


Fig. B19. Final process model for *Permit* after 20 November 2018.

B.4 Prepaid Travel Cost

Figures B20 and B21 depict the different versions of the prepaid travel cost process.

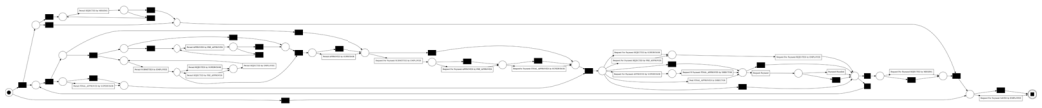


Fig. B20. Original process model for *Prepaid travel cost* until 11 January 2018.

B.5 Request for Payment

Figures B22 and B23 depict the different versions of the request for payment process.

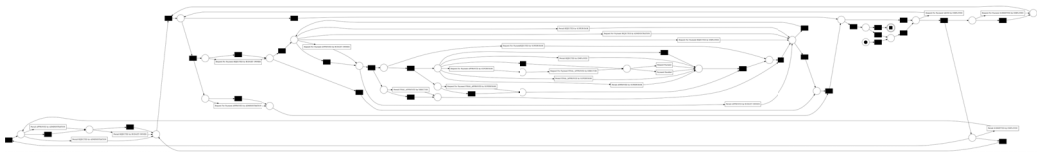


Fig. B21. Final process model for *Prepaid travel cost* after 12 February 2018.

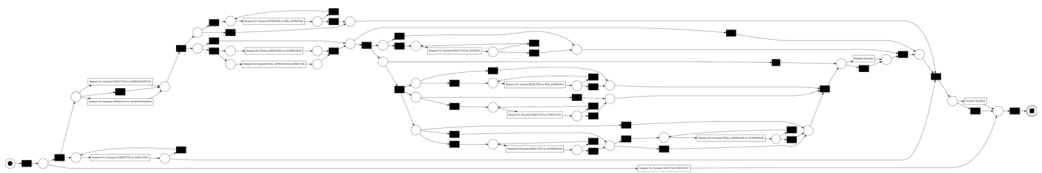


Fig. B22. Original process model for *Request for payment* until 11 January 2018.

Fig. B23. Final process model for *Request for payment* after 23 January 2018.

Received 16 July 2024; revised 4 December 2024; accepted 27 January 2025