This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2021.3120031, IEEE Transactions on Services Computing

A Conformance Checking-based Approach for Sudden Drift Detection in Business Processes

Víctor Gallego-Fontenla, Juan C. Vidal, *Member, IEEE,* and Manuel Lama

Abstract— Real life business processes change over time, in both planned and unexpected ways. The detection of these changes is crucial for organizations to ensure that the expected and the real behavior are as similar as possible. These changes over time are called concept drifts and its detection is a big challenge in process mining since the inherent complexity of the data makes difficult distinguishing between a change and an anomalous execution. In this paper, we present *C2D2* (Conformance Checking-based Drift Detection), a new approach to detect sudden control-flow changes in the process models from event traces. *C2D2* combines discovery techniques with conformance checking methods to perform an offline detection. Our approach has been validated with a synthetic benchmarking dataset formed by 68 logs, showing an improvement in the accuracy while maintaining a very low delay in the drift detection.

Index Terms—Business Processes, Concept drift, Process mining, Conformance checking-based detection

1 Introduction

Real-life processes are not immutable. Instead, they evolve to adapt to changes in their context, as new regulations or new consumption patterns. Changes can be planned by the organization, but also happen unexpectedly. In the first case, the impact on the process can be computed and minimized. But in the second case, it may lead to wrong decisions because of outdated information. Thus, organizations should put in place prevention measures to detect when something is running differently from planned to reduce this negative impact. These unforeseen changes over time are known as concept drifts, which is one of the challenges presented in the *Process Mining Manifesto* [1].

Changes can be classified based on their distribution over time [2]: (i) sudden drifts (Figure 1a), which means that the new concept replaces the previous one; (ii) gradual drifts (Figure 1b), where the new and the old concepts coexist for some time; and (iii) incremental drifts (Figure 1c), when the transition from the oldest concept to the newest one passes through some intermediate states that are, usually, some kind of combination from both. Furthermore, when changes can be repeated over time, periodically switching between concepts, the change is classified as a recurrent drift (Figure 1d). In this paper, we focus on sudden drift detection.

In addition, based on how data are processed [3], concept drift can be: (i) offline, when change detection is made *post-mortem*, being all data available from the beginning, and (ii) online, when change detection is made *on-the-fly*, and new data are processed just when it is generated. In this paper we focus on offline concept drift, which additionally faces two challenges: a) the inherent complexity of process models, that can contain and combine different structures such as sequences, loops, parallel branches and choices; and b) the distinction between a change and an outlier which is not always clear [2], [4], [5], [6] and may depend on the application domain and the context of the detection. For example, when analyzing a sales process, the changes caused by the increase in customers on Black Friday can be considered a drift if we analyze the data on a weekly time frame. But if we analyze the data for a whole year to get an overall perspective, these changes can be considered as outliers, because they last for a very short time.

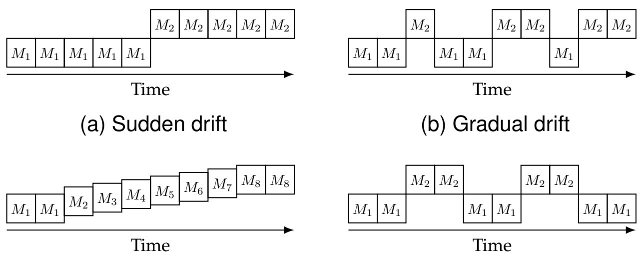
Although some authors have proposed different approaches for concept drift detection in process mining [3], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], identifying all possible change patterns [22] with a short delay, allowing organizations to know exactly when the change took place and helping in the identification of the reasons that caused the change, are still a challenge. In addition, many of the existing approaches can only detect some change patterns, which makes them less suitable for real use, as some of the drifts will remain unknown to the organizations. Another issue in some proposals is their high dependence on the end-user, who is required to have some *a-priori* knowledge of the process structure or skills to identify accurately the drift within a set of possibilities.

To address the aforementioned issues, in this paper we present *C2D2*, a novel and fully automatic approach based on discovery and conformance checking techniques for offline detection of sudden concept drifts in the control-flow of process models. The method starts by defining a reference window, that will serve as a ground truth. Then traces are processed by a discovery algorithm to extract the corresponding process model. To process the remaining traces, the window is slid over the log, updating conformance metrics related to that process model. With these conformance values a regression is computed, and when the measurements decrease significantly a drift is detected. The underlying idea is that the value from the conformance metrics computed over the reference model and the new traces should decrease when the latter comes from a modified process, being this enough to determine if a change exists or not with a low

- *Víctor Gallego-Fontenla, Juan C. Vidal and Manuel Lama are with the Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Galicia, Spain. Juan C. Vidal is also with the Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Galicia, Spain. E-mail: {victorjose.gallego, juan.vidal, manuel.lama}@usc.es*

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2021.3120031, IEEE Transactions on Services Computing



(a) Sudden drift (b) Gradual drift

(c) Incremental drift (d) Recurring drift

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

delay. Specifically, the main contribution of this paper is the use of conformance metrics, in particular, fitness and precision, to detect changes in processes, which is a novel and unexplored approach so far. Namely, we propose the use of fitness metrics to detect changes that include traces with behavior not supported by the current process model, and the use of precision metrics to detect changes that imply behavior from the model disappearing from the real executions.

C2D2 has been tested using a dataset with 68 synthetic event logs. The results have been compared with the ones obtained by the methods available in the state of the art. *C2D2* has proved to be better at the accuracy level, getting better F_{score}. In addition, *C2D2* gets very low delays, identifying changes closer to the point in which they happened. Getting good values for both metrics is important for organizations for minimizing the number of unidentified changes and for reacting as soon as possible to those changes.

The remainder is structured as follows. In Section 2 we analyze the main approaches to concept drift analysis. In Section 3 we define a set of terms necessary to understand correctly our approach. In Section 4 a formal proof of the hypothesis for sudden concept drift detection is presented. In Section 5 we detail our method for offline control-flow process concept drift detection. In Section 6 we present the experimentation performed to validate 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

Although process mining is a rather active research field, concept drift analysis has not received much attention until recently. It is worth noting that, although the method proposed in this paper focuses on offline detection, online approaches are also considered in the following analysis, because they can be easily adapted to detect this type of change by simulating an online environment from the complete event log.

In [7], the authors propose a method for online concept drift detection using a polyhedron-based log representation. Then, they monitor the probability that a trace falls into that polyhedron using the ADWIN algorithm [23]. The main drawback of this approach is that it can only detect the presence of a change, but it does not give any information about when it happened.

Online detection is also addressed in [8], where the authors discover a probabilistic process model that, given an activity, assigns a probability to every possible successor, and check how these probabilities evolve throughout the complete log using statistical hypothesis tests. Although the method identifies drifts in most cases, small changes in less likely activities generate changes in the probabilities that can stay undetected.

In [9] the authors propose an online approach based on the extraction of histograms from traces and then use a clustering algorithm to generate groups of similar traces. A change is triggered when a new cluster appears. An important drawback of this approach is that events order is not accounted for. Thus, it can only detect the addition or removal of new activities, but not the changes in the precedence relations between them.

In [3], the authors use a fixed-size window over some features extracted from the follows/precedes relations present in traces, and statistical hypothesis tests to evaluate whether these features have changed significantly. The weak point of this method is that it requires a lot of interaction from the user, including previous knowledge of the process model and the areas where the changes can be located. An extension of this work has been proposed in [11], where the authors implement a recursive bisectioning approach. Specifically, they take the traces that are involved in a drift detection and recursively split them into halves, intending to automatically localize the change. A drawback of this approach is that it still requires the user to know the possible changes to obtain good results. A similar solution is presented in [12], where the authors propose the usage of event class correlation as a feature, and apply statistical hypothesis tests to detect changes. However, it fails in detecting some change patterns such as the changes in the execution order of activities.

Another approach followed by some authors is the usage of clustering techniques to detect the drift. In [14], the authors cluster traces using the distance between pairs of activities. However, this approach does not support models with loops. Moreover, the distance can ignore certain change patterns depending on how many activities are affected by the change. In [15], the authors extend a trace clustering algorithm [24] adding a time dimension to force clusters to include only consecutive traces, and thus be able to detect changes. Their approach highly depends on the number of clusters, fixed by the user, and only obtain good results when the number of clusters is equal to the number of changes. In [16], the authors use a Markov clustering algorithm over different time windows to detect changes, but the approach does not focus on the control-flow perspective. Instead, multiple viewpoints of the process are taken into account simultaneously, mixing control-flow changes with behavioral and resource changes.

Another interesting approach, called *ProcessDrift*, is proposed in [17], where the authors transform traces into *partial-ordered-runs* and then apply a statistical hypothesis test over two windows (one for reference and one for detection) to detect changes. The main drawback of this approach lies in its sensitivity to changes in the frequencies of certain relations present in the log, which may lead to false positives in the detection. A related method is presented in [21], where the authors focus on detecting the change at the event level instead of at trace level. Specifically, they extract the α^+ relations from two consecutive adaptive windows

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2021.3120031, IEEE Transactions on Services Computing

of events, and then, applying a statistical test, namely the G-test, compare the relations distribution of these two windows. This allows the detection even with unfinished executions, and reduce the detection delay. The drawback of this approach is that it requires high amounts of traces to be able to detect changes, being possible to ignore them when they are close to each other.

In [18], [19] the authors apply graph metrics to detect changes. In [18], the authors compare the eigenvectors and the eigenvalues of undirected weighted graphs representing the log at different instants. In this graph, each vertex represents a trace. The edges weight is the similarity between the vertex (traces) it connects. However, this method needs a huge amount of traces, being unable to detect changes in logs with less than $2{,}000$ traces. In [19] the authors compare models over time using graph features, such as the node degree, the graph density or the occurrence of nodes and edges. However, this approach does not perform well in processes with loops.

In [20] the authors present *TPCDD*, a method that transforms the event log into a relation matrix using direct succession and weak order relations, where each column represents a trace and each row a relation. Then, based on the trend of these relations, it generate candidate drift points. These points are clustered using DBSCAN, to group candidates that belong to the same drift point. This approach relies heavily in the user defining a correct radius for the DBSCAN algorithm, potentially getting a high number of false positives when it is too low and a high number of false negatives when it is too high.

In [25], authors propose an algorithm for detecting sudden drifts in event streams using relation frequency maps and an adaptive window. They propose the use of an ADWIN with different distances between these frequency matrices, so a change would be detected if two consecutive frequency maps are different enough. The main drawback with this approach lies in choosing a good distance metric that serves to detect all types of drift in any context.

A similar approach to the proposed one is presented in [26], where the authors perform the drift discovery over an event stream using a sliding window and process histories. A process history is a collection of every process model used to represent the behaviour in the event stream along time. For detecting changes they compute the fitness between the last known model from the process history and the trace for the current event, considering that a trace fits a model if the computed fitness is over a threshold. If the trace that is being processed does not fit the last known model, they discover a new one by using only the unfitting traces from the window. To avoid false positives due to anomalous executions, they also assign a score to the model, and these positives are considered viable only if the computed score is over a threshold. Finally, detected changes are classified based on the models present in the process history, using two thresholds. The drawbacks of this approach are that it requires the end-user to provide multiple parameters (the window size and 4 different thresholds), and that it can not detect all types of change in the model, as when optional parallel paths from the model change to an exclusive choice.

Finally, an interesting job is presented in [27], where multiple configurations for [3] and [17] are tested in a real life scenario, showing the complexity of the concept drift detection in process mining.

Table 1
An example of a process log.

Case	Timestamp	Activity	Resource	Cost
1	01-01-2010 10:00	A	User 1	10
1	01-01-2010 11:30	B	User 2	4
1	01-01-2010 11:40	C	User 3	6
1	01-01-2010 15:00	D	User 3	11
1	02-01-2010 08:00	E	User 1	7
1	02-01-2010 09:00	G	User 1	5
2	01-01-2010 12:00	A	User 3	12
2	01-01-2010 12:10	C	User 2	2
2	02-01-2010 07:25	B	User 2	17
2	02-01-2010 13:15	D	User 1	18
2	02-01-2010 13:25	E	User 3	18
2	02-01-2010 14:00	G	User 2	1
3	01-01-2010 11:45	A	User 3	4
3	01-01-2010 12:30	B	User 1	7
3	03-01-2010 10:00	C	User 3	13
3	03-01-2010 17:25	D	User 1	1
3	03-01-2010 17:30	F	User 2	117
3	03-01-2010 17:35	G	User 3	3

With *C2D2* we take the aforementioned issues and try to minimize them to improve the results of the process drift detection. The method removes any user interaction in the drift detection, requiring only a minimum window size to be specified. Moreover, the method can detect all change patterns independently of the process structure. Furthermore, the method is designed to identify drifts with low delay, minimizing the detection of false negatives and positives.

3 Preliminaries

Below we present some concepts needed to understand the proposed method. The method takes an event log of a process and tries to detect the changes in the execution of that process over time.

Definition 1 (Event). An event ε represents the execution of the activity α in the context of a process. Events have some mandatory attributes such as the activity, the execution case or the execution timestamp. They can also have optional attributes, such as the resource that performed the activity, the variables that were modified or the location.

Definition 2 (Trace). A trace is an ordered sequence of events $\tau = \langle \varepsilon_1, ..., \varepsilon_n \rangle$ where every event belongs to the same execution case.

Definition 3 (Log). A log is defined as an ordered collection of traces $L = \langle \tau_1, ..., \tau_n \rangle$ where each trace represents one execution of the process. The size of the log, denoted as $|L|$, represents the number of traces in that log.

Table 1 shows an example of a log, where each row is an event, and dotted lines separate different traces. In addition to the mandatory attributes, the log has information about who performed the activity and the cost of executing that activity. For clarity, in the rest of the paper we represent traces as a sequence of ordered activities, without showing the rest of the attributes, and logs as collections of traces, ordered by the timestamp of their last event.

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

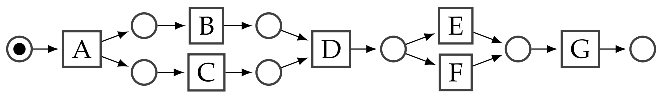


Figure 2. Example of a Petri net representing a process model.

A process model is a graph that describes the log behavior, that is, a graph that can replay the log traces. A process model contains a representation of the coordination between the process activities, through sequences, parallels, choices and so on. In this paper we formalize process models using Petri nets.

Definition 4 (**Petri net**)**.** A Petri net is a tuple $N = (P, T, F)$, where:

- P is a finite set of places
- T is a finite set of transitions;
- $P \cap T = \emptyset$; and
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs.

Given $x \in T \cup P$, the set ${}^\bullet x = \{y \mid (y, x) \in F\}$ is the set of inputs of x, and $x^\bullet = \{y \mid (x, y) \in F\}$ the set of outputs of x. Given a Petri net $N = (P, T, F)$, a marking of N is a mapping $M : P \to \mathbb{N}$, where $\mathbb{N}$ is the number of tokens in the place.

Processes usually have a unique start place $s \in P$ which has no inputs (${}^\bullet s = \emptyset$) and a unique end place $f \in P$ which has no outputs ($f^\bullet = \emptyset$). The initial marking of the Petri net M_0 contains only the initial place $M_0(s) = 1 \wedge \forall q \neq s \in P : M_0(q) = 0$. For a transition t to be fired, all its input places must contain at least one token ($\forall p \in {}^\bullet t : M(p) \geq 1$). When t is executed, it consumes a token from each of its inputs and puts a token in every of its outputs. Petri nets can be depicted as bipartite graphs, being transitions represented as rectangles and places as circles. A black bullet into a place represents a token.

Figure 2 shows a Petri net example. In this example, the process is conformed by the activities A, B, C, D, E, F and G. In real executions, A must be executed first. Then, B and C can be executed in any order. After these two activities are finished, D is executed. Then, exclusively one of E or F must be executed. Finally, G is executed and the process execution finishes.

The quality of a process model N with respect to a log L can be estimated comparing the allowed and the observed behaviour through some well established metrics such as fitness and precision.

Definition 5 (**Fitness metric**)**.** Given a log L and a process model N, the fitness can be defined as a function $\gamma : L \times N \to \mathbb{R}$ which represents the fraction of the observed behaviour that is captured by the model [28]. Fitness can be represented by the following expression, where B represents the allowed behaviour of N:

$$\gamma = \frac{|L \cap B|}{|L|} \tag{1}$$

For instance, every trace in the log from Table 1 fits perfectly in the model depicted in Figure 2, because they can be fully executed from start to end. Conversely, traces $\langle A, B, D, E, F, G\rangle$ or $\langle A, C, D, E, F, G\rangle$ do not fit the model because D can not be executed without executing both B and C previously and, furthermore, E and F can not appear both in the same trace.

Definition 6 (**Precision metric**)**.** Given a log L and a process model N, the precision can be defined as a function $\rho : L \times N \to \mathbb{R}$ which measures the fraction of the allowed behaviour that is observed in the log [28]. Precision can be defined by the next expression, where B represents the allowed behaviour of N:

$$\rho = \frac{|L \cap B|}{|B|} \tag{2}$$

For instance, if we compute the precision of the model depicted in Figure 2 against cases 1 and 2 from the log in Table 1, the value will be lower than when using the full log, because the model allows for executing alternatively E or F, but in the first two traces only E is observed.

Although there exist multiple ways to compute these metrics, these approaches can be grouped in two main groups [29]:

1) Metrics based on replaying the log over the model, like in [30], where each trace is re-executed over the model in order to detect discrepancies.
2) Metrics based on the alignment between the log and the model, like in [31] and [32], where an alignment is computed between the expected and the observed behaviour (namely, the one supported by the model and the one presented in the log).

Despite using Petri nets for modeling the processes, not all discovery algorithms use this representation. Indeed, the literature contains multiple ways for representing a process model (e.g., process trees or heuristics nets). However, these representations can be translated to an equivalent Petri net without loosing information. Moreover, all the conformance metrics used in this paper need a Petri net as its input [33]. Using Petri nets as an intermediate representation of the processes allows *C2D2* to be agnostic with respect to the chosen discovery and conformance metrics.

It is worth mentioning that, although the proposed algorithm is agnostic to a specific metric, results will depend, to a large extent, on the ability of the metric to stabilize over time. In order to study the impact that the choice of a particular metric may have, tests have been carried out using metrics that are well established in the state of the art. Specifically, the following metrics were used: Alignment Based Fitness [31] and Negative Event Recall [34] for fitness; and Advanced Behavioural Appropriateness [35] and Negative Event Precision [34] for precision. In addition, two new metrics are proposed (one for fitness and one for precision). The proposed metrics are better suited for this problem (Section 5.3) and, as we will see below, obtain better results in change detection, in addition of having a lower computational complexity.

One of the objectives in this paper is the identification of changes in the process structure over time. This is done exploring the traces generated by the process and assessing if the more recent traces are product of the same process model. The concept of a sliding window captures this latter set of traces.

Definition 7 (**Sliding window**)**.** Given a log L and an integer $n \leq |L|$, a sliding window of size n over the log L can be defined as the sublog that at instant i contains the last n

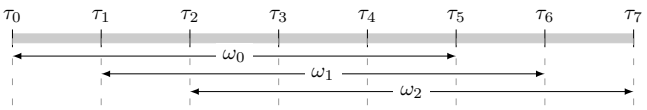


Figure 3. Example of the sliding window behaviour.

traces, denoted by $\omega_i = \langle \tau_i, ..., \tau_{i+n} \rangle$. When a new trace is read from the log, i is incremented, so the oldest trace from the window is forgotten, and the new one is added at the end of the window ($\omega_{(i+1)} = \langle \tau_{(i+1)}, ..., \tau_{(i+1)+n} \rangle$).

Figure 3 shows an example log and the behaviour of a six-sized sliding window over it. At instant 0, the sliding window ω_0 contains traces τ_0 to τ_5. When a new trace is read from the log, i is incremented, so the oldest trace in the window (τ_0) is forgotten and the new trace (τ_6) is added to the window. This behaviour continues until the full log has been read.

The structural evolution of the process over time, e.g., to adapt to new context circumstances or organization needs, is known as a drift candidate.

Definition 8 (Drift candidate). Let $\omega_i = \langle \tau_i, ..., \tau_{i+n} \rangle$ and $\omega_{(i+1)} = \langle \tau_{(i+1)}, ..., \tau_{(i+1)+n} \rangle$ be two consecutive windows over a log. Let $N_i = (P, T, F)$ and $N_{i+1} = (P^*, T^*, F^*)$ be the models describing the behaviour observed in ω_i and $\omega_{(i+1)}$ respectively, mined using a discovery algorithm. We say that the trace τ_{i+n} is a drift candidate when any of the following conditions is satisfied:

- $T \neq T^*$, which means that there are different activities in both models
- $F \neq F^*$, which means that the conections between activities have changed.

A drift candidate indicates a potential change point in the log, which has to be confirmed ex post. To prevent false positives due to the presence of noise or anomalous data, a trace τ_j will be considered as a confirmed change if it was marked as a drift candidate and several of the next traces are also marked as drift candidates.

4 Conformance Checking Based Drift Detection

The algorithm proposed in this paper is based on the assumption that changes in the model structure (drifts) can be detected through changes in fitness or precision.

Theorem 1. *Fitness detects changes related to unsupported behaviour that is being observed, but it can not detect fitting behaviour that is disappearing from the log.*

Proof. Let us suppose a log L, a model M, and let B be the behaviour supported by M. Also, let us suppose that every trace in the log is supported by the model.

$$\forall \tau \in \mathrm{L} : \tau \in \mathrm{B} \implies \mathrm{L} \subseteq \mathrm{B} \implies \mathrm{L} \cap \mathrm{B} = \mathrm{L}$$

If we replace the former property in Eq. (1) and Eq. (2), then:

$$\gamma(L, N) = \frac{|L|}{|L|} \qquad \rho(L, N) = \frac{|L|}{|B|}$$

Now, let us suppose we observe a new trace τ^* that is not part of the supported behaviour, that is, $\tau^* \notin \mathrm{B}$. If we add τ^* to L, Eq. (1) and Eq. (2) can be written as:

$$\gamma(L \cup \tau^*, N) = \frac{|(L \cup \tau^*) \cap B|}{|L \cup \tau^*|} = \frac{|(L \cap B) \cup (\tau^* \cap B)|}{|L \cup \tau^*|}$$

$$\rho(L \cup \tau^*, N) = \frac{|(L \cup \tau^*) \cap B|}{|B|} = \frac{|(L \cap B) \cup (\tau^* \cap B)|}{|B|}$$

If we apply the assumption that every trace is supported by the model to the former equation then:

$$\gamma(L \cup \tau^*, N) = \frac{|L \cup \emptyset|}{|L \cup \tau^*|} = \frac{|L|}{|L| + 1}$$

$$\rho(L \cup \tau^*, N) = \frac{|L \cup \emptyset|}{|B|} = \frac{|L|}{|B|}$$

Thus, $\gamma(L, N) > \gamma(L \cup \tau^*, N)$ and $\rho(L, N) = \rho(L \cup \tau^*, N)$, i.e., when a new behaviour arises in the log, fitness can detect the change but precision can not. □

Theorem 2. *Precision detects changes related to the fitting behaviour that is disappearing from the log, but it can not detect new unsupported behaviour that is being observed.*

Proof. Let us suppose now a trace τ^*, which behaviour is unique, disappears from the log. In this situation, Eq. (1) and Eq. (2) are equivalent to:

$$\gamma(L \setminus \tau^*, N) = \frac{|(L \setminus \tau^*) \cap B|}{|L \setminus \tau^*|} = \frac{|(L \cap B) \setminus \tau^*|}{|L \setminus \tau^*|} = \frac{|L \setminus \tau^*|}{|L \setminus \tau^*|}$$

$$\rho(L \setminus \tau^*, N) = \frac{|(L \setminus \tau^*) \cap B|}{|B|} = \frac{|(L \cap B) \setminus \tau^*|}{|B|} = \frac{|L \setminus \tau^*|}{|B|}$$

Since all the behaviour in the log is supported by the model, the equations can be simplified to:

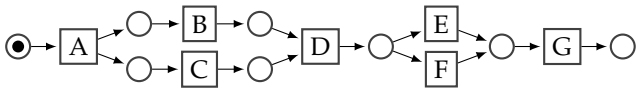
$$\gamma(L \setminus \tau^*, N) = \frac{|L \setminus \tau^*|}{|L \setminus \tau^*|} = \frac{|L| - 1}{|L| - 1}$$

$$\rho(L \setminus \tau^*, N) = \frac{|L \setminus \tau^*|}{|B|} = \frac{|L| - 1}{|B|}$$

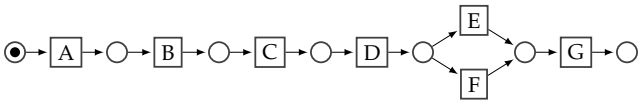
Therefore, $\gamma(L, N) = \gamma(L \setminus \tau^*, N)$ and $\rho(L, N) > \rho(L \setminus \tau^*, N)$, i.e., when some behaviour disappears from the log the fitness can not detect changes but precision can. □

Corollary. *Fitness and precision separately can not detect all possible changes in the process structure, but a combination of both can.*

To illustrate these theorems, let us suppose the models N_1 and N_2 depicted in Figure 4a and Figure 4b. The difference between both models is that activities B and C are in parallel in N_1, but in sequence in N_2. Let us also suppose that the process N_1 changes to N_2 at instant $i = 8$, which log is represented in Figure 4c, henceforth denoted as L_1. Traces τ_1 to τ_8 correspond to the execution of N_1 and traces τ_9 to τ_{16} correspond to N_2. In L_1, the concurrent execution of activities B and C becomes a sequence from τ_9 onwards. After this change, traces are still replayable, so the fitness remains unaltered. However, no trace in the window contains the path $A \to C \to D$ from τ_9 onwards, so the precision falls. This can be seen in Figure 4e, where precision falls because the model allows more behaviour than is present in the traces, but fitness remains unaltered.



(a) Model N_1.



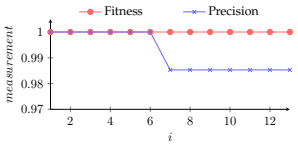
(b) Model N_2.

ID	Trace
τ_1	A B C D E G
τ_2	A C B D E G
τ_3	A B C D F G
τ_4	A C B D F G
τ_5	A B C D E G
τ_6	A C B D E G
τ_7	A B C D F G
τ_8	A C B D F G
τ_9	A B C D E G
τ_{10}	A B C D F G
τ_{11}	A B C D E G
τ_{12}	A B C D F G
τ_{13}	A B C D E G
τ_{14}	A B C D F G
τ_{15}	A B C D E G
τ_{16}	A B C D F G

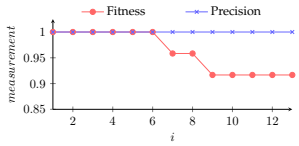
(c) Log generated using N_1 as the initial process and N_2 as the modified one.

ID	Trace
τ_1	A B C D E G
τ_2	A B C D F G
τ_3	A B C D E G
τ_4	A B C D F G
τ_5	A B C D E G
τ_6	A B C D F G
τ_7	A B C D E G
τ_8	A B C D F G
τ_9	A B C D E G
τ_{10}	A C B D E G
τ_{11}	A B C D F G
τ_{12}	A C B D F G
τ_{13}	A B C D E G
τ_{14}	A C B D E G
τ_{15}	A B C D F G
τ_{16}	A C B D F G

(d) Log generated using N_2 as the initial process and N_1 as the modified one.



(e) Fitness and precision evolution when using a window of size 4, log from Figure 4c and reference model N_1



(f) Fitness and precision evolution when using a window of size 4, log from Figure 4d and reference model N_2.

Figure 4. Measurements evolution for two logs with different changes.

Let us now suppose a different change, from model N_2 to N_1, at the same time instant, which log is represented in Figure 4d, henceforth denoted as L_2. Traces τ_1 to τ_8 are generated by N_2 while traces τ_9 to τ_{16} by N_1. In L_2, B and C, originally in sequence, are in parallel from τ_9 onwards. This change can not be detected using precision (the model does not generate more behavior than the present in the log), but it can be detected though fitness, since τ_{10}, τ_{12}, τ_{14} and τ_{16} can not be replayed by N_2. This situation is represented in Figure 4f, where precision remains unchanged, but fitness falls in the 7$^{\text{th}}$ iteration of the algorithm.

5 Algorithm

In this section we will detail how changes in the structure of a process can be identified when significant variations in the conformance are detected when comparing incoming traces and the process model.

Algorithm 1 (*C2D2*) performs drift detection based on a sliding window (Def. 7) whose optimal size is automatically adjusted at the beggining and after a drift is confirmed (line 9). Thus, the only input of the algorithm is a minimum window size, aside from the event log. The algorithm starts by initializing a list D of traces that are confirmed drifts (line 2), and the initial window index $i = 0$ (line 3).

From lines 4 to 23, the main loop will identify and confirm the drifts of the process. In this loop, the list Γ will store the fitness of a model N that is discovered from the traces of a window. For instance, Γ_i will contain the fitness of the model N with respect to the traces of the window ω_i. Similarly, the list P will store the precision measurements. In addition, the lists D^Γ and D^P store, at index i, a boolean that indicates whether the last trace from window ω_i has been marked as a drift candidate or not for fitness and precision, respectively. For instance, D^Γ_i is set to true when the trace τ_{i+n} has been marked as a drift candidate. In line 9, the optimal sliding window size is calculated from the remaining traces that have not been processed (more details in Section 5.2), and then the sliding window ω_i and the model that describes the behavior observed in this window are obtained (lines 10 and

Algorithm 1 Conformance Checking-based Drift Detection

Inputs: an event log L and a minimum window size $min\_ws < |L|$
Outputs: a list of trace causing drift D

1: **procedure** ConceptDriftDetection(L, min_ws)
2: D $\leftarrow$ [ ] *//confirmed drift traces*
3: $i \leftarrow 0$
4: **while** $i < |L|$ **do**
5: $\Gamma \leftarrow$ [ ] *//fitness measures (Def. 5)*
6: P $\leftarrow$ [ ] *//precision measures (Def. 6)*
7: $D^\Gamma \leftarrow$ [ ] *//drift candidates (fitness)*
8: $D^P \leftarrow$ [ ] *//drift candidates (precision)*
9: n $\leftarrow$ AdjustWindow($min\_ws, \langle \tau_i, ..., \tau_{|L|} \rangle$)
10: $\omega_i \leftarrow \langle \tau_i, ..., \tau_{i+n} \rangle$
11: N $\leftarrow discover(\omega_i)$
12: **while** $((i+n) < |L|) \wedge (\tau_{i+n-1} \notin$ D) **do**
13: $\Gamma \leftarrow \Gamma :: \gamma(\omega_i, N)$ *//append current fitness*
14: P $\leftarrow$ P $:: \rho(\omega_i, N)$ *//append current precision*
15: $D^\Gamma \leftarrow D^\Gamma ::$ IdentifyDriftCandidate(n, Γ, D^Γ)
16: $D^P \leftarrow D^P ::$ IdentifyDriftCandidate(n, P, D^P)
17: **if** ConfirmDrift(n, D^Γ, D^P) **then**
18: D $\leftarrow$ D $:: \tau_{i+n}$
19: **end if**
20: $i \leftarrow i + 1$
21: $\omega_i \leftarrow \langle \tau_i, ..., \tau_{i+n} \rangle$
22: **end while**
23: **end while**
24: **return** D
25: **end procedure**

26: **function** IdentifyDriftCandidate(n, $data$, D^*)
27: $\Upsilon \leftarrow regress(\{data_{|data|-(n/2)}, ..., data_{|data|}\})$
28: $m^< \leftarrow \Upsilon.slope < 0 \wedge \Upsilon.confidence < 0.05$
29: $m^> \leftarrow \Upsilon.slope > 0 \wedge \Upsilon.confidence < 0.05$
30: $m^= \leftarrow (\neg m^<) \wedge (\neg m^>)$
31: **return** $(|data| > n/2) \wedge (m^< \vee m^> \vee (m^= \wedge (D^*_{|D^*|} = true)))$
32: **end function**

33: **function** ConfirmDrift(n, D^Γ, D^P)
34: $d^\Gamma \leftarrow \forall d \in \{D^\Gamma_{|D^\Gamma|-n}, ..., D^\Gamma_{|D^\Gamma|}\} : d = true$
35: $d^P \leftarrow \forall d \in \{D^P_{|D^P|-n}, ..., D^P_{|D^P|}\} : d = true$
36: **return** $(|D^\Gamma| \geq n \wedge d^\Gamma) \vee (|D^P| \geq n \wedge d^P)$
37: **end function**

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2021.3120031, IEEE Transactions on Services Computing

11). These four lists, the sliding window ω_i and the model N are reinitialized whenever a drift has been confirmed.

The inner loop (lines 12 to 22) performs the detection of drifts based on conformance measurements. This loop iterates from the index i, corresponding to the window ω_i, until the end of the log as long as no drift has been confirmed (line 12). In each iteration i, the fitness and precision of the model N are computed with respect to the sliding window ω_i (lines 13 and 14). Traces are identified as a drift candidate (lines 15 and 16) by computing in function IDENTIFYDRIFTCANDIDATE a linear regression over the values of the lists D^Γ and D^P (lines 26 to 32) and then by checking if the slope of the fitted function is statistically different from zero (more details in Section 5.1).

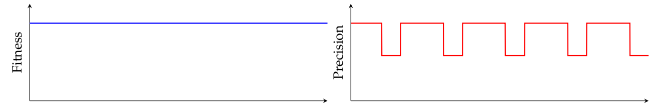
If the trace τ_{i+n} is identified as a drift candidate, the function CONFIRMDRIFT checks if this drift persist over time (line 17). This function checks that the last n traces have been classified as drift candidates either for fitness (line 34) or precision (line 35). This allows the method to prevent false positives due to the existence of temporal falls in the metrics caused by outlier traces. Once the candidate is confirmed as a real drift, trace τ_{i+n} is added to the list of confirmed drifts D (line 18), the window slides one position, reading a new trace from the log (line 21), and the algorithm loops back to the initialization phase.

5.1 Drift detection

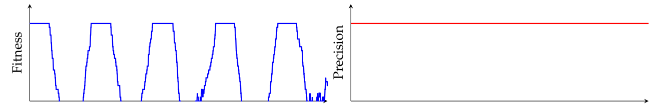
The detection mechanism is listed in lines 26 to 32 of Algorithm 1 (function IDENTIFYDRIFTCANDIDATE). As a first step, a simple linear regression [36] is computed over the last $n/2$ measurements for both fitness and precision (line 27). To calculate this regression, the ordinary least squares method has been used, which minimizes the sum of squares of the difference between the real and the predicted values of the dependent variable (i.e., the fitness/precision value). Also, a statistical test over the regression slope has been performed. Namely a t-test with $(n/2)-2$ degrees of freedom. The null hypothesis (H_0) states that the slope of the regression is equal to zero. The minimum significance level has been set at 0.05. When H_0 is rejected (i.e. $\Upsilon.\mathit{confidence} < 0.05$) we asume that enough evidences exist to accept the slope value $\Upsilon.\mathit{slope}$. Otherwise, we can not assume that the slope value is different from 0.

There are three possible situations:

1) The regression slope is negative (line 28): metrics get lower values, so more traces are not replayable for fitness or, conversely, more paths of the model are not contained in traces for precision. Thus the window is marked as a drift candidate.
2) The regression slope is positive (line 29): metrics get higher values, since more traces are replayable for fitness or, conversely, more paths of the model are contained in traces for precision. Thus the window is marked as a drift candidate.
3) The regression slope is zero (line 30): no change in conformance metrics, i.e., the window does not present any drift. In this case, a drift can also be detected, but only if the previous window was marked as a drift candidate.



(a) Log 1 (`pl`). Precision falls every time a change happens but fitness remain unaltered.



(b) Log 2 (`cb`). Fitness falls but precision remains unaltered in every change.

Figure 5. Evolution of fitness and precision metrics when computed over a sliding window of 100 traces on logs `pl` and `cb`.

An example of this behaviour is depicted in Figure 5. This example shows the drift detection using the logs `pl` and `cb`, that will be described in Section 6.1, which contain a change every 250 traces. In the case of `pl` (Figure 5a), two fragments that are originally executed in a concurrent form are transformed into a sequential execution, which should imply a reduction in precision but not in fitness. In the case of `cb` (Figure 5b), a fragment is transformed from mandatory to skippable, which should imply a reduction in fitness but not in precision.

5.2 Adjusting the Window Size

When some behaviour appears in some traces but not in the model, they can be initially considered as outliers. But when this behaviour persists for a long time, it can be flagged as a change, having the organization an opportunity to enhance its process. Something similar happens when some behaviour is no longer observed in the log. A path of the process that is not present during a short period of time can be seen as a temporary exception. But if this behaviour is absent for a long time, some optimizations can be made to improve the process performance. Hence, small windows will detect less durable changes, while larger window sizes will detect changes that persist.

Adjusting the window size for processing the log is not a trivial task. A small window would led to multiple false detections, due to the window not containing enough information to describe the process executed at a given instant. On the other hand, a big window would not detect some changes, because the reference window will contain traces from before and after the change. Thus, a good balance between the two options is essential. To adjust the window size, *C2D2* uses an approach based on the comparison of models from consecutive sublogs (Algorithm 2). We start with three empty models (line 2) and a window size n', that is initialized to the minimum window size n (line 3). Then, three new models for three consecutive sublogs are discovered with the same discovery algorithm used for the detection (lines 5-7). If these three discovered models are equal, we increment the window size and try again (lines 8-10). Else, if any of the discovered models differ from the rest, the procedure finishes and the last n' is used as the window size. By default, we use a minimum window size of $1\,\%$ and an increment of $0.1\,\%$ of the log size.

Figure 6 shows why three consecutive models are required. Let us consider the log in Figure 4d, that presents a

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Algorithm 2 Automatic window size optimizer

Inputs: an event log L and a minimum window size $n < |L|$
Outputs: the optimal window size for processing L

```
1: function AdjustWindow(n, L)
2:     N1, N2, N3 ← ∅
3:     n' ← n
4:     while N1 = N2 = N3 do
5:         N1 ← discover(⟨τ0, ..., τn'⟩)
6:         N2 ← discover(⟨τn', ..., τ2n'⟩)
7:         N3 ← discover(⟨τ2n', ..., τ3n'⟩)
8:         if N1 = N2 = N3 then
9:             n' ← increment(n')
10:        end if
11:    end while
12:    return n'
13: end function
```

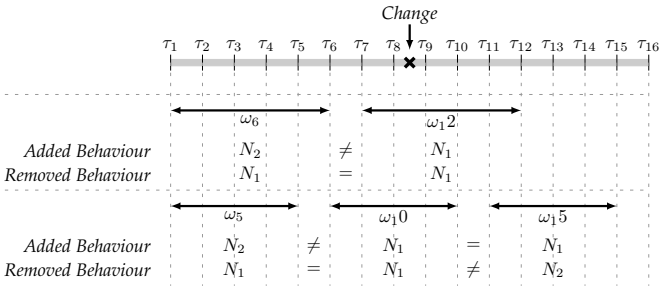
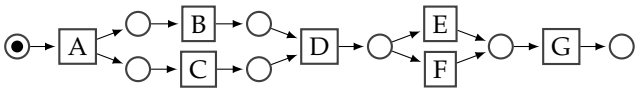


Figure 6. Behaviour of the adaptive window in the example from Figure 4. Log correspond with the one in Figure 4c and Figure 4d for removing and adding behaviour, respectively. N_1 and N_2 refer respectively to the models in Figure 4a and Figure 4b.

change between traces τ_8 and τ_9. The change consist in some behaviour being added to the process (the execution of B before C is replaced by a concurrent execution of these two activities). In this case, using two windows, one that contains behaviour from before the change and one that contains behaviour from both before and after the change, is enough because the models will be different. Consider now the log in Figure 4c, that also contains a change between traces τ_8 and τ_9. This time, the change consists in some behaviour being removed from the process (the execution of C before B disappears from the log). In this case, when using just two windows, the model discovered with traces from both pre- and post-drift traces reflects no changes, because the missing path is present in the pre-drift traces used for discovery. In this scenario three windows, with their respective models, are required: one to depict the behaviour of the process before the change, one to detect both pre- and post-drift behaviour, which will be the same as in the pre-drift case, and one to represent the behaviour after the change.

5.3 Custom Fitness and Precision

Traditional fitness and precision metrics are designed to assess the global quality of a model. These metrics use different approaches to compute fitness and precision in a reliable way, giving each trace a score in a continuous scale depending on how well they conform to the model, rather than following a discrete approach where traces can only get a binary rating for Conformance. However, *C2D2* use them to detect structural changes in the execution of a process. In this



Log
A B C D E G
A C B D E G
A B C D E G
A C B D E G
A C B D E G
A B C D E G
A B C D E G
A C B D E G
A B C D E G

$$OLP = \langle (A \to B)(A \to C)(B \to D)(C \to D) (D \to E)(D \to F)(E \to G)(F \to G) \rangle$$

$$DFR = \langle (A \to B)(A \to C)(B \to C)(B \to D) (C \to B)(C \to D)(D \to E)(E \to G) \rangle$$

$$PC = 1 - \frac{|\langle (D \to F)(F \to G) \rangle|}{8} = 0.75$$

Figure 7. *PC* computation example.

paper, we propose two simpler fitness and precision metrics aside from the well-established metrics from the state of the art. These two approaches have much lower computational complexity and were designed to detect changes in simple and noise-free logs.

In the case of fitness, we use the percentage of replayable traces (Eq. 1). This approach is not particularly useful for measuring the quality of a model, since it equally penalizes traces that do not fit the model and those that deviate slightly from it. Despite this, it can be used to estimate changes in fitness, since a change in the percentage of traces that can be replayed in the model always leads to a change in the metric value.

For precision, the following approach is used:

$$PC = 1 - \frac{|OLP \setminus DFR|}{|OLP|} \tag{3}$$

where:

- A set of one-length paths (*OLP*) is extracted from the model. An *OLP* is a pair of activities that are directly connected in the process model, without any other activity in between.
- A set of directly-follows relations (*DFR*) is extracted from the log. A *DFR* is a pair os activities that appear one after the other in the log, without any activity in between.
- Operator $\setminus$ is the difference between two sets.

Eq. (3) does not measure the precision *per se*, but the change in the precision. The moment a *OLP* stops appearing in the log is indicative that some path of the model has disappeared. The proposed approach returns 1 when all the supported behavior of the model appear at least once in the log, and 0 otherwise, i.e., when none of the behavior supported by the model appears in the log. The computation of this metric is illustrated with an example in Figure 7.

6 Experimentation

Concept drift algorithms are assessed based on two quality measures: F_{score} (4a), which is an accuracy metric computed as the harmonic mean between precision (4b) and recall (4c); and delay (henceforth Δ), which is the distance between the point when the change really happened and when it is detected.

$$F_{score} = \frac{2 \times precision \times recall}{precision + recall} \tag{4a}$$

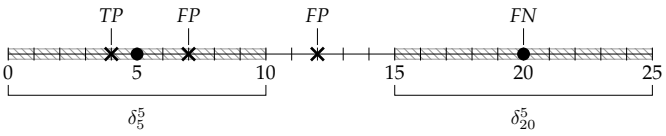


Figure 8. Change results classification in *TP*, *FP* and *FN*. A dot represents a real change. A cross represents a detection. Shadowed in the neighborhood.

Table 2
Simple change patterns from [22] applied to the original model.

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

$$precision = \frac{TP}{TP + FP} \tag{4b}$$

$$recall = \frac{TP}{TP + FN} \tag{4c}$$

To classify the detected changes as *true positives* (*TP*), *false positives* (*FP*) or *false negatives* (*FN*), we use a threshold ε, that represents the error tolerance of the quality measures, and a neighborhood δ_i^ε, defined as the interval between $i - \varepsilon$ and $i + \varepsilon$. Let a change happen at instant i. This change is classified as a *TP* only when it is detected in δ_i^ε. When no change is detected in δ_i^ε it is classified as *FN*. Finally, all changes detected in δ_i^ε where a previous change has been already detected are classified as a *FP*, as well as the ones detected outside any δ^ε. Figure 8 shows an example with two real changes (d_5, at instant 5, and d_{20}, at instant 20), and three detections, at instants 4 (c_4), 7 (c_7) and 12 (c_{12}), using a $\varepsilon = 5$. In this example, c_4 is classified as a *TP*, because it lies in the neighborhood of d_5; c_7 is classified as a *FP*, because, despite being in the neighborhood of d_5, another change has been detected previously; c_{12} is classified too as a *FP*, in this case for being detected outside any neighborhood δ^5; and finally, d_{20} is classified as a *FN* since no change is detected in its neighborhood.

The algorithm implementation is published online and available to researchers as a REST API[1].

6.1 Validation Data

Our proposal has been tested with three models extracted from the literature [37], [38], [39], which describe a loan application process, a hospital emergency ward process, and a central venous catheter process, respectively. These models are usually part of benchmarks for concept drift detection, process discovery and conformance ckecking. A set of sythetic logs for each one of the process models have been generated using the methodoloy and change patterns described in [17]. This is the most extended methodology [19], [20], [40] for generating datasets when validating sudden concept drift detection algorithms in process mining. For each process, we generated a dataset composed of 68 logs: 17 with 2,500 traces, 17 with 5,000, 17 with 7,500 and 17 with 10,000. The original dataset from [17] contains 4 more logs (one for each of the sizes), but they have been discarded because its drifts (changing the frequency of the branches in a choice construct) are not control-flow drifts, but behavioural ones. It should be noted that, for simplicity and space limitation, the explanation of the concept drift tests will only describe the results of the loan application process. Detailed results of the other two models can be found in the supplemetary material of this paper.

The Petri net corresponding to the loan application process is depicted in Figure 9. To generate the 17 modified models, 11 simple change patterns from [22] are applied to the original process. The applied patterns are collected in Table 2. These changes can imply an insertion (labeled as *I*), an optionalization (labeled as *O*) or a resequentialization (labeled as *R*). For each simple change pattern a different model is generated. The remaining 6 models are generated by applying a combination of simple change patterns, picking one change from each of the previously named classes.

Once all the models are available, the logs are generated simulating executions of those processes. The original log is then combined with the modified ones to generate logs with drifts. The final log is composed joining alternatively sublogs from both the original model and the modified ones. Each drifting log presents a change every 10 % of its final size. A log generation example is represented in Figure 10. Two logs (L_1 and L_2) with different models are split in 5 sublogs with equal sizes (L_1^1 to L_1^5 and L_2^1 to L_2^5). This sublogs are combined alternatively into a log L, with size $|L_1| + |L_2|$.

6.2 Impact of Discovery Algorithm in Fitness and Precision Metrics during Detection

In order to check the impact of the discovery algorithm and the conformance metrics in *C2D2* performance, different configurations have been tested:

1) Discovery algorithms: Inductive Miner (*IM*) [41] and Heuristics Miner (*HM*) [42], which are two of the most used methods for discovering models from event logs. No algorithm based on evolutionary computation has been selected because it would increase the computational complexity significantly.
2) Fitness metrics: Alignment Based Fitness (*AF*) [43], Negative Event Recall (*NR*) [34] and the percentage of completely replayable traces (*RT*) from Section 5.3.
3) Precision metrics: Advanced Behavioural Appropriateness (*ABA*) [35], Negative Event Precision (*NP*) [34] and precision change assessment (*PC*) from Section 5.3.

The key when choosing a discovery algorithm and a pair of fitness and precision metrics is to obtain a combination that allows the results of the regression to stabilize around a value, so the slope is zero while there are no changes. If we focus on fitness, reaching a constant value in absence of changes

1. https://tec.citius.usc.es/concept-drift-api/swagger-ui.html

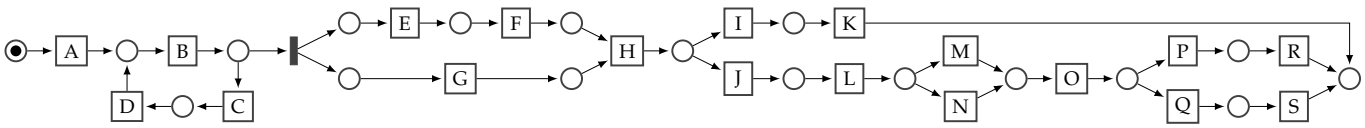


Figure 9. Petri net for the original model [17] used to generate logs. Activity names are shortened for better understandability.

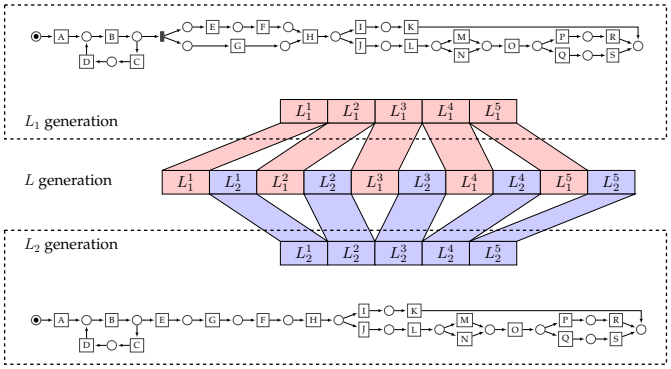


Figure 10. Log generation example.

is easier if we use a discovery algorithm that ensures traces replayability, such as Inductive Miner. However, if we use a discovery algorithm that does not ensure traces replayability, such as Heuristics Miner, the selected metric should take this into account, or the obtained values will not stabilize, so more false positives will be notified. This can be seen in the experiments in Table 3, where the results of Inductive Miner do not change significantly no matter which fitness metric is used. On the contrary, when we use Heuristics Miner, best results are obtained when a more robust fitness metric is used (e.g., alignments based one vs. percentage of replayable traces).

On the other hand, if we focus on precision, the chosen discovery algorithm has less impact, as none of the discovery algorithms used in the experimentation ensures a perfect precision. When computing precision, situations in which a path from the model is only present in few traces (or even in none) are quite common, so the metric can oscilate a lot. With the use of the PC precision metric this is partially addressed, because this metric does not take into account how many times a path occurs but only its presence. This forces the metric to converge quickly, so the regression slopes are near to zero earlier, and change more abruptly when the path disappears competely from the executions.

Taking into account the former results, the experiments in the following sections have been performed using *IM* algorithm, *RT* fitness and *PC* precision.

6.3 Comparison with Other Process Drift Detection Algorithms

In this section, *C2D2* algorithm is compared with *Trace-Based ProDrift* (*PD-T*) [17], *Event-Based ProDrift* (*PD-E*) [21] and *TPCDD* [20], the three sudden process drift detection algorithms with best results in the state of the art. Specifically, we used the following configurations:

1) *PD-T* with an *adaptive window* and an initial size of 50;
2) *PD-E* with an *adaptive window* and an initial size of 50. *Relation noise filter threshold* was set to 0 % and *sensitivity* to *very high*, as the authors recommend for analyzing synthetic logs without noise;
3) *TPCDD* with minimum window size set to 100 and *DBSCAN radius* to 10.

Table 3
Mean Δ and F_{score} for every tested configuration over the 2,500 traces logs. Window size has been fixed to 100 traces. Best results are shadowed in a darker grey and runners-up in a lighter one. Error tolerance has been set to a 5 % of the log size.

			F_{score}	Δ
IM	$\gamma = AF$	$\rho = ABA$	0.9028	4.0915
		$\rho = NP$	0.7985	43.6243
		$\rho = PC$	0.9969	3.6471
	$\gamma = NR$	$\rho = ABA$	0.9028	4.0915
		$\rho = NP$	0.7864	41.5318
		$\rho = PC$	0.9969	3.5948
	$\gamma = RT$	$\rho = ABA$	0.9425	4.0663
		$\rho = NP$	0.7955	41.7028
		$\rho = PC$	0.9969	3.5948
HM	$\gamma = AF$	$\rho = ABA$	0.9327	7.4608
		$\rho = NP$	0.7365	36.1014
		$\rho = PC$	0.9789	4.4412
	$\gamma = NR$	$\rho = ABA$	0.9402	10.3758
		$\rho = NP$	0.7427	36.8831
		$\rho = PC$	0.9750	5.3828
	$\gamma = RT$	$\rho = ABA$	0.4724	6.0812
		$\rho = NP$	0.7025	73.5175
		$\rho = PC$	0.7176	2.9829

IM	Inductive Miner
HM	Heuristics Miner
AF	Alignment Based Fitness
NR	Negative Event Recall
RT	Percentage of Replayable Traces
ABA	Advanced Behavioural Appropriateness
NP	Negative Event Precision
PC	Precision Change Assessment

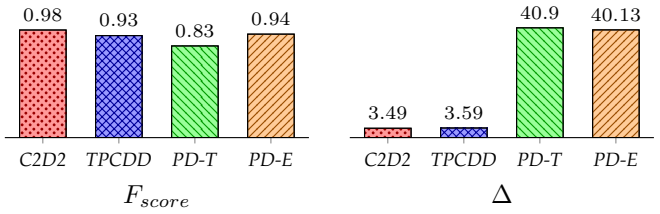


Figure 11. Mean F_{score} and Δ values for *C2D2*, *TPCDD*, *PD-T* and *PD-E*.

Table 4 to Table 7 show the detailed results. *C2D2* outperforms the rest of algorithms in terms of F_{score} in all the cases, except in `OIR` log, getting always the best average value. Moreover, *C2D2* is also the second best in terms of delay, very close to *TPCDD*, and being all the values in the same order of magnitude.

For the 2,500 trace logs, *C2D2* obtains a F_{score} of 1.0 in

Table 4
Mean F_{score} and Δ values for each algorithm using logs with $2{,}500$ traces.

Log	C2D2		TPCDD		PD-T		PD-E	
	F_{score}	Δ	F_{score}	Δ	F_{score}	Δ	F_{score}	Δ
cb	1.0000	6.2222	1.0000	5.5556	0.2000	90.0000	1.0000	78.4444
cd	1.0000	1.7778	1.0000	0.7778	0.0000	—	1.0000	42.4444
cf	1.0000	3.8889	1.0000	1.4444	1.0000	40.0000	1.0000	49.7778
cm	1.0000	5.8889	1.0000	11.5556	0.7143	66.0000	0.9412	95.1250
cp	1.0000	2.8889	1.0000	1.2222	1.0000	45.2222	1.0000	39.0000
lp	1.0000	3.6667	0.9474	10.0000	1.0000	52.7778	0.9412	36.5000
sw	1.0000	2.8889	1.0000	1.0000	1.0000	43.5556	0.9412	38.1250
pl	1.0000	1.7778	1.0000	1.3333	0.9412	36.0000	0.9412	51.3750
pm	1.0000	3.2222	1.0000	3.3333	0.8000	49.6667	1.0000	48.4444
re	1.0000	2.0000	1.0000	1.1111	1.0000	19.8889	0.9412	49.8750
rp	1.0000	3.0000	1.0000	1.2222	1.0000	45.2222	1.0000	65.0000
IOR	1.0000	2.6667	1.0000	2.0000	1.0000	38.7778	1.0000	63.3333
IRO	1.0000	6.0000	1.0000	2.7778	1.0000	52.3333	1.0000	43.0000
OIR	0.9000	2.6667	1.0000	0.6667	0.6154	29.5000	0.9412	15.7500
ORI	1.0000	2.5556	1.0000	0.6667	1.0000	50.5556	1.0000	35.0000
RIO	1.0000	4.6667	1.0000	10.8889	0.9412	51.1250	0.9412	53.8750
ROI	1.0000	2.0000	1.0000	0.7778	1.0000	32.3333	0.9412	40.1250
AVG	0.9941	3.3987	0.9969	3.3137	0.8360	46.4349	0.9723	49.7173

Table 5
Mean F_{score} and Δ values for each algorithm using logs with $5{,}000$ traces.

Log	C2D2		TPCDD		PD-T		PD-E	
	F_{score}	Δ	F_{score}	Δ	F_{score}	Δ	F_{score}	Δ
cb	1.0000	6.7778	0.9000	6.5556	0.7143	57.4000	1.0000	99.6667
cd	1.0000	1.2222	0.9474	0.7778	0.0000	—	0.9412	2.5555
cf	1.0000	3.3333	0.9474	9.2222	1.0000	25.7777	1.0000	50.1111
cm	1.0000	7.7778	0.9474	6.1111	0.7143	51.8000	0.8750	117.5714
cp	1.0000	3.5556	0.9000	5.4444	1.0000	33.3333	1.0000	66.2222
lp	1.0000	2.0000	0.9000	4.1111	1.0000	48.4444	1.0000	45.3333
sw	1.0000	2.7778	0.9474	0.6667	1.0000	29.6667	1.0000	21.3333
pl	1.0000	1.2222	0.9474	0.7778	0.0000	—	1.0000	51.2222
pm	1.0000	3.6667	0.9474	3.4444	0.9412	37.0000	1.0000	25.8889
re	1.0000	2.0000	0.9474	1.5556	1.0000	19.1111	0.9474	16.7778
rp	1.0000	2.8889	0.9000	0.4444	1.0000	28.2222	0.9412	48.7500
IOR	1.0000	2.2222	0.9474	2.7778	1.0000	24.2222	1.0000	52.0000
IRO	1.0000	5.4444	0.9000	1.7778	1.0000	49.4444	0.9412	31.2500
OIR	0.9474	2.0000	0.9000	1.0000	1.0000	26.6667	0.9412	0.0000
ORI	1.0000	3.2222	0.9000	1.1111	1.0000	36.1111	0.9412	22.7500
RIO	1.0000	5.2222	0.9474	1.7778	1.0000	45.2222	1.0000	59.7778
ROI	1.0000	2.0000	0.9474	0.4444	1.0000	26.0000	0.9412	7.5000
AVG	0.9969	3.3725	0.9279	2.8235	0.8453	35.8948	0.9692	42.2771

all logs, except in `OIR`, where it returns two false positives. *TPCDD* also obtains a perfect F_{score}, except in `lp`. In this case the error is due to a false negative (i.e., a change that remains undetected). Both *PD-T* and *PD-E* have much worse results, having 7 and 8 cases where they does not detect all changes. In fact, *PD-T* even is unable to detect any change in `cd`.

For logs with $5{,}000$, $7{,}500$ and $10{,}000$ traces, *C2D2* gets similar results, and only do not have a perfect F_{score} in `OIR`, again because of false positives and negatives. However, *TPCDD* behaves much worse, being unable to get a perfect F_{score} in any log. The same happens to *PD-E* in logs with $7{,}500$ and $10{,}000$ traces, where it can not detect all changes correctly. *PD-T* gets worse F_{score} in most of the logs, and, in addition, is unable to detect any change in `cd` and `pl`, for logs with $5{,}000$ and $7{,}500$ traces, and in `cd` and `cm`, for logs with $10{,}000$ traces.

In terms of Δ, *C2D2* and *TPCDD* obtain similar results for all the logs, being able to detect all changes always with less than 10 traces of delay. In comparison with *PD-T* and *PD-E*, *C2D2* Δ are always an order of magnitude below, thus being able to detect changes closer to the point where they really happened.

As a summary, Figure 11 shows the mean F_{score} and the mean Δ for each algorithm. As can be seen, *C2D2* outperforms every other algorithm in terms of mean F_{score} and Δ. Additional experiments supporting these results and conclusions can be found in the supplementary material.

6.3.1 Statistical tests

F_{score} results (including those from the supplementary material) have been evaluated using a statistical test in

Table 6
Mean F_{score} and Δ values for each algorithm using logs with $7{,}500$ traces.

Log	C2D2		TPCDD		PD-T		PD-E	
	F_{score}	Δ	F_{score}	Δ	F_{score}	Δ	F_{score}	Δ
cb	1.0000	7.7778	0.9474	16.6667	1.0000	68.6667	0.9474	70.2222
cd	1.0000	2.1111	0.9474	1.0000	0.0000	—	0.9474	2.2222
cf	1.0000	2.7778	0.9474	1.5556	1.0000	22.3333	0.9474	37.5556
cm	1.0000	5.3333	0.9474	4.4444	0.9412	83.8750	0.8889	78.6250
cp	1.0000	3.2222	0.9000	1.3333	1.0000	33.2222	0.9474	26.5556
lp	1.0000	1.4444	0.7500	1.2222	1.0000	52.8889	0.7200	34.6667
sw	1.0000	2.6667	0.8571	0.8889	1.0000	32.4444	0.9000	16.1111
pl	1.0000	2.1111	0.9474	1.6667	0.0000	—	0.9474	30.8889
pm	1.0000	3.1111	0.9474	2.6667	1.0000	43.4444	0.9474	14.6667
re	1.0000	2.0000	0.9474	0.5556	1.0000	21.4444	0.9474	27.5556
rp	1.0000	2.5556	0.8571	30.6667	1.0000	29.8889	0.9000	50.5556
IOR	1.0000	4.3333	0.9474	10.7778	0.9000	34.5556	0.9474	53.2222
IRO	1.0000	4.2222	0.9000	2.7778	1.0000	53.4444	0.9474	14.3333
OIR	0.5000	2.0000	0.7826	0.3333	1.0000	69.7778	0.9474	0.0000
ORI	1.0000	2.3333	0.8571	0.8889	1.0000	33.1111	0.9000	36.8889
RIO	1.0000	4.2222	0.8571	2.4444	1.0000	40.6667	0.9000	47.6667
ROI	1.0000	2.0000	0.9474	1.0000	1.0000	31.3333	0.9474	7.8888
AVG	0.9706	3.1895	0.8993	4.7582	0.8730	43.4065	0.9194	32.3309

Table 7
Mean F_{score} and Δ values for each algorithm using logs with $10{,}000$ traces.

Log	C2D2		TPCDD		PD-T		PD-E	
	F_{score}	Δ	F_{score}	Δ	F_{score}	Δ	F_{score}	Δ
cb	1.0000	9.0000	0.8571	18.2222	0.5000	41.6667	0.8571	84.4444
cd	1.0000	1.5556	0.9474	1.0000	0.0000	—	0.9000	3.3333
cf	1.0000	3.8889	0.9474	5.8889	1.0000	29.6667	0.9474	45.8889
cm	1.0000	8.5556	0.9000	7.6667	0.0000	—	0.9474	97.7778
cp	1.0000	4.4444	0.9000	2.3333	1.0000	33.5556	0.9000	30.7778
lp	1.0000	2.3333	0.8182	8.1111	1.0000	47.1111	0.8182	32.8889
sw	1.0000	3.6667	0.9474	1.5556	1.0000	33.4444	0.9000	4.6667
pl	1.0000	1.5556	0.9474	1.1111	0.2000	84.0000	0.9474	30.7778
pm	1.0000	3.6667	0.9474	2.1111	0.7500	31.3333	0.9474	9.0000
re	1.0000	2.0000	0.9000	0.7778	1.0000	17.7778	0.9000	23.0000
rp	1.0000	3.7778	0.8182	1.3333	1.0000	31.2222	0.8571	46.2222
IOR	1.0000	2.7778	0.9000	3.0000	1.0000	27.7778	0.9474	59.5556
IRO	1.0000	7.6667	0.9474	2.1111	0.8750	50.1429	0.8571	18.0000
ORI	1.0000	3.8889	0.8182	1.1111	1.0000	32.6667	0.9000	54.6666
OIR	0.1333	2.0000	0.7200	0.2222	1.0000	37.7778	0.9474	2.7778
RIO	1.0000	5.8889	0.7500	2.1111	0.8889	41.0000	0.8182	63.0000
ROI	1.0000	2.0000	0.9474	0.4444	1.0000	20.6667	0.9000	8.4444
AVG	0.9490	4.0392	0.8831	3.4771	0.7773	37.8540	0.8995	36.1895

order to confirm the quality of the proposed method. A *non-parametric one-vs-all* test has been executed using the STAC tool [44]. Namely, *Friedman's Aligned Ranks* test with a significance level of 0.05 has been used. The test result confirms that H_0 is rejected, thus algorithms do not converge to the same mean, being *C2D2* statistically the best in terms of F_{score}. Table 8 shows the rankings of the algorithms. In this case, *C2D2* clearly outscores the other algorithms.

A post-hoc test (namely *Holm-Bonferroni* post-hoc test), a pairwise comparison, was also performed. Results are shown in Table 9. For all the cases, H_0 is rejected, meaning that no algorithm is able to equal *C2D2* in F_{score} values. Finally, tests were also conducted for Δ values. *C2D2* and *TPCDD* are tied as the best-performing algorithms. *Friedman's Aligned Ranks* for Δ is depicted in Table 10. Both *C2D2* and *TPCDD* obtain similar scores, thus rejecting the hypothesis that one of the algorithms outperforms the other. Table 11 shows the post-hoc test for Δ. As we can see in the last row, the null hypothesis H_0 between *C2D2* and *TPCDD* cannot be rejected, which is consistent with the scores obtained by *Friedman's Aligned Ranks*.

7 Conclusions and Future Work

In this paper we presented *C2D2*, an approach to the *offline* detection of *sudden control-flow drifts* in process mining. *C2D2* drift detection is supported by the assumption that conformance checking measures are suitable to detect control-flow drifts. Specifically, we argue that fitness and precision are complementary metrics in concept drift, and while fitness is useful to identify traces that are not supported by the model, precision looks for behaviour not present in the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2021.3120031, IEEE Transactions on Services Computing

Table 8
F_{score} one-vs-all statistical test results.

Rank	Algorithm
220.45597	*C2D2*
278.38679	*TPCDD*
293.05031	*PD-E*
482.10692	*PD-T*

Table 9
F_{score} post-hoc statistical test results. H_0 indicates that the mean of the results of each pair of algorithms is equal.

Algorithms	p_{value}	Result
C2D2 vs. *PD-T*	$< 1 \times 10^{-5}$	H_0 is rejected
C2D2 vs. *PD-E*	0.00085	H_0 is rejected
C2D2 vs. *TPCDD*	0.00494	H_0 is rejected

Table 10
Δ one-vs-all statistical test results.

Rank	Algorithm
195.32317	*C2D2*
202.64329	*TPCDD*
393.67378	*PD-E*
522.35976	*PD-T*

Table 11
Δ post-hoc statistical test results. H_0 indicates that the mean of the results of each pair of algorithms is equal.

Algorithms	p_{value}	Result
C2D2 vs. *PD-T*	$< 1 \times 10^{-5}$	H_0 is rejected
C2D2 vs. *PD-E*	$< 1 \times 10^{-5}$	H_0 is rejected
C2D2 vs. *TPCDD*	0.72651	H_0 cannot be rejected

window of traces. Related to this, we propose the usage of two new metrics, one for fitness and one for precision, that have a low computational complexity to detect changes in models.

Our approach has been validated against 3 synthetic benchmarking dataset, each one consisting of 68 logs, outperforming the best concept drift algorithms in terms of accuracy (F_{score}) while maintaining a minimum delay (Δ). Finally, a statistical test over the results of all algorithms confirmed that the presented solution is statistically better in terms of accuracy.

As future work, we plan to extend the algorithm in order to deal with other types of change, and study the usage of memory mechanisms in order to be able to classify them. We plan also to extend the method to be executed in *online* environments, where the requirements in terms of computational complexity are different.

Acknowledgements

The work from Víctor J. Gallego was supported by the Ministerio de Educación, Cultura y Deporte (grant FPU17/05138, co-funded by the European Regional Development Fund - ERDF program); the Consellería de Educación, Universidade e Formación Profesional (accreditation 2019-2022 ED431G-2019/04) and 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. This paper was also supported by the Spanish Ministerio de Ciencia e Innovación under the project PID2020-112623GB-I00.

References

[1] W. M. P. van der Aalst *et al.*, "Process Mining Manifesto," in *Proceedings of the 2011 International Business Process Management Workshops (BPM 2011), Part I*. Springer, 2011, pp. 169–194.
[2] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A Survey on Concept Drift Adaptation," *ACM Computing Surveys*, vol. 46, pp. 44:1–44:37, 2014.
[3] R. P. J. C. Bose, W. M. P. van der Aalst, I. Zliobaite, and M. Pechenizkiy, "Dealing With Concept Drifts in Process Mining," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, pp. 154–171, 2014.
[4] C. C. Aggarwal, *Outlier Analysis*. Springer, 2013.
[5] D. Sargun and C. E. Koksal, "Separating an outlier from a change," in *53rd Asilomar Conference on Signals, Systems, and Computers (ACSCC 2019)*, 2019.
[6] Z. Li, H. Ma, and Y. Mei, "A unifying method for outlier and change detection from data streams based on local polynomial fitting," in *Proceedings of the 11th Conference on Advances in Knowledge Discovery and Data Mining (PAKDD 2007)*, 2007.
[7] J. Carmona and R. Gavaldà, "Online Techniques for Dealing with Concept Drift in Process Mining," in *Proceedings of the 11th International Symposium on Advances in Intelligent Data Analysis (IDA 2012)*. Springer, 2012, pp. 90–102.
[8] P. Weber, B. Bordbar, and P. Tiño, "Real-Time Detection of Process Change using Process Mining," in *Proceedings of the 2011 Imperial College Computing Student Workshop (ICCSW 2011)*. Imperial College London, 2011, pp. 108–114.
[9] S. B. Junior, G. M. Tavares, V. G. T. da Costa, P. Ceravolo, and E. Damiani, "A Framework for Human-in-the-loop Monitoring of Concept-drift Detection in Event Log Stream," in *Proceedings of The Web Conference 2018, (WWW 2018)*. ACM, 2018, pp. 319–326.
[10] F. M. Maggi, A. Burattin, M. Cimitile, and A. Sperduti, "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 2013) - Confederated International Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE 2013*. Springer, 2013, pp. 94–111.
[11] J. Martjushev, R. P. J. C. Bose, and W. M. P. van der Aalst, "Change Point Detection and Dealing with Gradual and Multi-order Dynamics in Process Mining," in *Proceedings of the 14th International Conference on Perspectives in Business Informatics Research (BIR 2015)*. Springer, 2015, pp. 161–178.
[12] M. K. M. V., L. Thomas, and A. Basava, "Capturing the Sudden Concept Drift in Process Mining," in *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data (ATAED 2015)*. CEUR-WS, 2015, pp. 132–143.
[13] T. Li, T. He, Z. Wang, Y. Zhang, and D. Chu, "Unraveling Process Evolution by Handling Concept Drifts in Process Mining," in *Proceedings of the 2017 IEEE International Conference on Services Computing (SCC 2017)*. IEEE Computer Society, 2017, pp. 442–449.
[14] R. Accorsi and T. Stocker, "Discovering Workflow Changes with Time-Based Trace Clustering," in *Proceedings of the First International Symposium in Data-Driven Process Discovery and Analysis (SIMPDA 2011)*. Springer, 2011, pp. 154–168.
[15] D. Luengo and M. Sepúlveda, "Applying Clustering in Process Mining to Find Different Versions of a Business Process That Changes over Time," in *Proceedings of the 2011 International Business Process Management Workshops (BPM 2011), Part I*. Springer, 2011, pp. 153–158.
[16] B. Hompes, J. C. A. M. Buijs, W. M. P. van der Aalst, P. Dixit, and H. Buurman, "Detecting Change in Processes Using Comparative Trace Clustering," in *Proceedings of the 5th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2015)*. CEUR-WS, 2015, pp. 95–108.
[17] A. Maaradji, M. Dumas, M. L. Rosa, and A. Ostovar, "Detecting

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Sudden and Gradual Drifts in Business Processes from Execution Traces," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, pp. 2140–2154, 2017.

[18] G. T. Lakshmanan, P. T. Keyser, and S. Duan, "Detecting changes in a semi-structured business process through spectral graph analysis," in *Proceedings of the 27th International Conference on Data Engineering Workshops (ICDE 2011)*. IEEE Computer Society, 2011, pp. 255–260.

[19] A. Seeliger, T. Nolle, and M. Mühlhäuser, "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 2017)*. ACM, 2017, pp. 1–10.

[20] C. Zheng, L. Wen, and J. Wang, "Detecting Process Concept Drifts from Event Logs," in *Proceedings of the On the Move to Meaningful Internet Systems Conferences (OTM 2017) - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Part I*. Springer, 2017, pp. 524–542.

[21] A. Ostovar, A. Maaradji, M. L. Rosa, A. H. M. ter Hofstede, and B. F. van Dongen, "Detecting Drift from Event Streams of Unpredictable Business Processes," in *Proceedings of the 35th International Conference on Conceptual Modeling, (ER 2016)*. Springer, 2016, pp. 330–346.

[22] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features - Enhancing flexibility in process-aware information systems," *Data and Knowledge Engineering*, vol. 66, pp. 438–466, 2008.

[23] A. Bifet and R. Gavaldà, "Learning from Time-Changing Data with Adaptive Windowing," in *Proceedings of the 7th SIAM International Conference on Data Mining (SDM 2007)*. SIAM, 2007, pp. 443–448.

[24] R. P. J. C. Bose and W. M. P. van der Aalst, "Trace Clustering Based on Conserved Patterns: Towards Achieving Better Process Models," in *Proceedings of the 2009 International Business Process Management Workshops (BPM 2009)*. Springer, 2009, pp. 170–181.

[25] M. Hassani, "Concept Drift Detection Of Event Streams Using An Adaptive Window," in *Proceedings of the 33rd International Conference on Modelling and Simulation (ECMS 2019)*. European Council for Modeling and Simulation, 2019, pp. 230–239.

[26] F. Stertz and S. Rinderle-Ma, "Process Histories - Detecting and Representing Concept Drifts Based on Event Streams," in *Proceedings of the On the Move to Meaningful Internet Systems Conferences (OTM 2018) - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018, Part I*. Springer, 2018, pp. 318–335.

[27] N. J. Omori, G. M. Tavares, P. Ceravolo, and S. B. Jr., "Comparing concept drift detection with process mining tools," in *Proceedings of the XV Brazilian Symposium on Information Systems (SBSI 2019)*. ACM, 2019, pp. 31:1–31:8.

[28] J. Carmona, B. F. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking - Relating Processes and Models*. Springer, 2018.

[29] J. Munoz-Gama, *Conformance Checking and Diagnosis in Process Mining - Comparing Observed and Modeled Processes*. Springer, 2016.

[30] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, pp. 182–192, 2012.

[31] A. Adriansyah, "Aligning observed and modeled behavior," Ph.D. dissertation, Department of Mathematics and Computer Science, 2014.

[32] M. de Leoni and W. M. P. van der Aalst, "Aligning Event Logs and Process Models for Multi-perspective Conformance Checking: An Approach Based on Integer Linear Programming," in *Proceedings of the 11th International Conference on Business Process Management, (BPM 2013)*. Springer, 2013, pp. 113–129.

[33] S. K. L. M. vanden Broucke, J. D. Weerdt, J. Vanthienen, and B. Baesens, "A comprehensive benchmarking framework (CoBeFra) for conformance analysis between procedural process models and event logs in ProM," in *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2013)*. IEEE Computer Society, 2013, pp. 254–261.

[34] S. K. L. M. vanden Broucke, J. D. Weerdt, J. Vanthienen, and B. Baesens, "Determining Process Model Precision and Generalization with Weighted Artificial Negative Events," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, pp. 1877–1889, 2014.

[35] A. Rozinat and W. M. P. van der Aalst, "Conformance Checking of Processes Based on Monitoring Real Behavior," *Information Systems*, vol. 33, pp. 64–95, 2008.

[36] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis*. John Wiley & Sons, 2012.

[37] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.

[38] F. Mannhardt, M. de Leoni, H. A. Reijers, and W. M. P. van der Aalst, "Data-driven process discovery - revealing conditional infrequent behavior from event logs," in *Proceedings of the 29th International Conference in Advanced Information Systems Engineering (CAiSE 2017)*, 2017.

[39] J. Munoz-Gama, R. R. de la Fuente, M. M. Sepúlveda, and R. R. Fuentes, "Conformance checking challenge 2019 (ccc19)," 2019.

[40] A. Yeshchenko, C. D. Ciccio, J. Mendling, and A. Polyvyanyy, "Comprehensive process drift detection with visual analytics," in *Proceedings of the 38th International Conference on Conceptual Modeling, (ER 2019)*. Springer, 2019, pp. 119–135.

[41] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering Block-Structured Process Models from Event Logs - A Constructive Approach," in *Proceedings of the 34th International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2013)*. Springer, 2013, pp. 311–329.

[42] A. J. M. M. Weijters and J. T. S. Ribeiro, "Flexible Heuristics Miner (FHM)," in *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*. IEEE Computer Society, 2011, pp. 310–317.

[43] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst, "Conformance checking using cost-based fitness analysis," in *Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, (EDOC 2011)*. IEEE Computer Society, 2011, pp. 55–64.

[44] I. Rodríguez-Fdez, A. Canosa, M. Mucientes, and A. Bugarín, "STAC: A Web Platform for the Comparison of Algorithms Using Statistical Tests," in *Proceedings of the 2015 IEEE International Conference on Fuzzy Systems, (FUZZ-IEEE 2015)*. IEEE Computer Society, 2015, pp. 1–8.

Víctor Gallego-Fontenla is working towards the Ph.D degree at CiTIUS, University of Santiago de Compostela, Spain. He received the BSc degree in computer science from the University of Stantiago de Compostela in 2015, and a MSc degree in artificial intelligence from the International University Menéndez Pelayo (UIMP) in 2017. His main research interests include process mining and the analysis of changes in bussiness processes.

Dr. Juan C. Vidal received the B.Eng. degree in computer science from the University of A Coruña, A Coruña, Spain, in 2000 and his Ph.D. degree in Artificial Intelligence from the University of Santiago de Compostela (USC), Santiago de Compostela, Spain, in 2010. He is currently associate researcher at the Research Center in Intelligent Technology (CiTIUS), USC. His main research interests include linguistic data process mining, fuzzy logic, machine learning and linguistic summarization of data with fuzzy operators.

Dr. Manuel Lama Penín is an Associate Professor of Computer Science and Artificial Intelligence of the University of Santiago de Compostela (USC) and senior researcher of the Intelligent Technologies Center of the USC (CiTIUS). He is author/co-author of 140 scientific papers and has participated in more than 30 R&D projects. His research interests focus on the development of process mining techniques in massive data environments and the application of these techniques to solve real problems.