Author/s:
Yeshchenko, A;Di Ciccio, C;Mendling, J;Polyvyanyy, A

Title:
Comprehensive Process Drift Detection with Visual Analytics

Date:
2019-10-15

Citation:
Yeshchenko, A., Di Ciccio, C., Mendling, J. & Polyvyanyy, A. (2019). Comprehensive Process Drift Detection with Visual Analytics. 38th International Conference, ER 2019, 11788, pp.119-135. Springer. https://doi.org/10.1007/978-3-030-33223-5_11.

Persistent Link:
https://hdl.handle.net/11343/258888

Comprehensive Process Drift Detection with Visual Analytics

Anton Yeshchenko[1] [0000−0002−5346−8358], Claudio Di Ciccio[1] [0000−0001−5570−0475], Jan Mendling[1] [0000−0002−7260−524X], and Artem Polyvyanyy[2] [0000−0002−7672−1643]

[1] Vienna University of Economics and Business, Vienna, Austria
{anton.yeshchenko,claudio.di.ciccio,jan.mendling}@wu.ac.at
[2] The University of Melbourne, Parkville, VIC, 3010, Australia
artem.polyvyanyy@unimelb.edu.au

Abstract. Recent research has introduced ideas from concept drift into process mining to enable the analysis of changes in business processes over time. This stream of research, however, has not yet addressed the challenges of drift categorization, drilling-down, and quantification. In this paper, we propose a novel technique for managing process drifts, called Visual Drift Detection (VDD), which fulfills these requirements. The technique starts by clustering declarative process constraints discovered from recorded logs of executed business processes based on their similarity and then applies change point detection on the identified clusters to detect drifts. VDD complements these features with detailed visualizations and explanations of drifts. Our evaluation, both on synthetic and real-world logs, demonstrates all the aforementioned capabilities of the technique.

Keywords: Process mining · Process drifts · Declarative process models

1 Introduction

The availability of data has extended conceptual modeling as a research field of manually created models with automatic techniques for generating models from data. Process mining is one of these recent extensions that is concerned with providing transparency of how the businesses operate based on real-world event data. Process discovery algorithms have proven to be highly effective in generating process models from data of stable behavior [1]. However, many processes are not stable but are subject to various forms of change over time. In data mining, such change over time is called a *drift*. A drift is a concept that process mining has addressed only to a limited extent so far.

Recent works have focused on integrating ideas from research on concept drift from data mining into process mining [7,12,22,26,18]. The arguably most advanced technique is proposed in [14], where Maaradji et al. present a framework for detecting process drifts based on tracking behavioral relations over time using statistical tests. A strength of this approach is its statistical soundness and ability to identify a rich set of drifts, which makes it a suitable tool for verifying if an intervention at a known point in time has resulted in an assumed change of behavior. However, in practice, the existence of different types of drifts in a business process is not known beforehand, and the analysts are interested in distinguishing what has and what has not changed over time. This need calls for a more fine-granular analysis.

In this paper, we present a novel technique for process drift detection, called Visual Drift Detection (VDD), which addresses the identified research gap. More specifically,

our technique facilitates the *visual interpretation* [25] of process drifts founded in the formal rigor of temporal logic of DECLARE constraints [2,10] and time series analysis [6]. Key strengths of our technique are clustering, i.e., grouping, of declarative behavioral constraints that exhibit similar trends of changes over time and automatic detection of changes, i.e., drift points. These features allow us to detect and explain drifts that would otherwise sneak undetected by other techniques. The paper presents an evaluation that demostrates these capabilities.

The remainder of the paper is structured as follows. Section 2 illustrates the problem of process drift detection and formulates five requirements for its analysis. Then, Section 3 states the preliminaries. Section 4 presents our drift detection technique, while Section 5 evaluates the technique using synthetic and real-world benchmark data. Finally, Section 6 summarizes the results and concludes with an outlook on future research.

2 Process Drift Analysis

This section discusses and motivates the problem of process drift analysis (Section 2.1), and specifies requirements for its solution (Section 2.2).

2.1 Motivating example

Various logs of real-world business process executions have been recently made available for research. As an example, consider the log of the Italian process for handling the collection of road ticket fines [16]. This process starts with a ticket being issued. In the best case, which covers a third of all the cases, the fine is directly paid. In roughly half of the other cases, a fine notification is sent to the accused driver. Some of these drivers appeal, while some ignore the notice, such that a considerable share of cases sees a penalty being added. Partially, these are further appealed, paid or eventually sent for credit collection. The authority is now interested in this question: Has the process of handling road ticket fines, specifically for the accused drivers, changed over time, and which parts of the process now work differently than in the past?

The described problem is typical for many domains. The objective is to explain the change of the system's behavior in a dynamically changing non-stationary environment based on some *hidden context* [11]. In this setting, a *concept drift* is a change of the conditional distribution of the output given a specific input. Research in data mining and machine learning distinguishes techniques for uncovering drifts in an *online* or *offline* manner [23], with applications in prediction and fraud detection.

In process mining, *process drift* is a notion for analyzing changes of business processes over time. Classical process mining techniques have implicitly assumed that logs are not sensitive to time in terms of systematic change [1]. Sampling-based techniques explicitly build on this assumption for generating a process model with a subset of the event log data [5]. A significant challenge for adopting concept drift for process mining is to represent behavior in a time-dependent way. The approach reported in [14] uses causal dependencies and tracks them over time windows. The specific challenge is to not only spot a drift but also to classify it. Figure 1 shows established drift classes from data mining. Next, we use the example of the road ticket fines process to illustrate the potential causes of drifts.

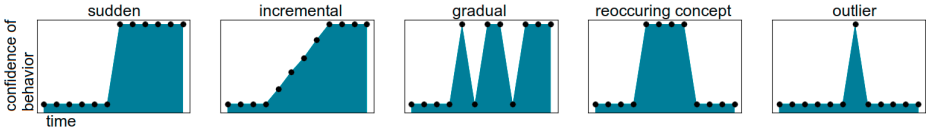


Fig. 1: Different types of drifts, cf. Fig. 2 in [11]; note that an outlier is not a drift.

A *sudden drift* is typically caused by an intervention. A new law could eliminate the right of an accused driver to lodge a second appeal. As a result, we would not see second appeal events in our log in the future. An *incremental drift* might result from a stepwise introduction of self-service terminals for paying fines at toll stations. A *gradual drift* may yield from a new policy to show less indulgence with drivers who marginally violated speeding rules. Finally, a *reoccurring drift* might result from specific measures taken in the holiday season from June to August, like flagging down drivers directly on the highway to have them pay right on the spot. Existing process mining techniques support these types of drifts partially.

The following are four cases from the Italian road ticket fines log[1]:

1. 10 Jan. 2011: ⟨Lodging ticket, Appeal, Appeal, Payment, Close ticket⟩
2. 15 Jan. 2011: ⟨Lodging ticket, Appeal, Appeal, No payment, Close ticket⟩
3. 04 Feb. 2011: ⟨Lodging ticket, Appeal, Payment, Close ticket⟩
4. 06 Feb. 2011: ⟨Lodging ticket, Appeal, No payment, Close ticket⟩

We observe a sudden drift here due to the introduction of a new law. After 4 Feb. 2011, it is not possible to lodge a second appeal. Therefore, in formal terms, from case 3 onwards, the rule that multiple appeals occur before the ticket closes abruptly decreases in confidence. In DECLARE, we denote this rule as ALTERNATERESPONSE(Appeal,Close ticket).

2.2 Requirements

Based on the analysis of process change scenarios from the literature, like the road ticket fines discussed previously, we identified five requirements for process drift analysis:

R1. **Identify drifts:** The points at which a business process undergoes drifts should be identified based on precise criteria;
R2. **Categorize drifts:** Process drifts should be according to their types;
R3. **Drill down and roll up analysis:** Process drifts should be characterized at different levels of granularity, e.g., drifts that concern the entire process or only its parts;
R4. **Quantitative analysis:** Process drifts should be associated with a degree of change, a measure that quantifies to which extent the drift entails a change in the process;
R5. **Qualitative analysis:** Process drifts should convey changes in a business process to process analysts effectively.

Table 1 provides an overview of the state-of-the-art methods to process drift analysis with the reference to the requirements. Note that collectively these methods implement all the requirements, whereas each individual methods addresses only a subset thereof.

Approaches like ProDrift [14] and Graph Metrics on Process Graphs [22] put an emphasis on requirement R1. The evaluation of ProDrift in [14] shows that two types of

[1] https://doi.org/10.1007/s00607-015-0441-1

drifts are found with high accuracy (sudden and gradual drifts), hence partly addressing requirement R2; note that the authors report high sensitivity of the technique to the choice of the method parameters. The approach relies on the automated detection of changes in business process executions, which are analyzed based on causal dependency relations studied in process mining [24]. The Tsinghua Process Concept Drift Detection approach (TPCDD) [26] uses two kinds of behavioral relationships (direct succession and weak order). The approach computes those relations on every trace, so as to later identify the change points through their merge and clustering. The sole type of drift that TPCDD detects is the sudden drift.

The other approaches emphasize requirement R5. The approach based on Process Trees uses ProDrift for drift detection, and aims at explaining how sudden drifts influence behavior of the process [17]. To this end, process trees for pre-drift and post-drift sections of the log are built and used to explain the change. The Performance Spectra approach [7] focuses on drifts that show seasonality. The technique filters the control-flow and visualizes identified flow patterns. It is evaluated against a real-world log, in which recorded business processes show year-to-year seasonality. A strength of the Comparative Trace Clustering approach [12] is its ability to include non-control-flow characteristics in the analysis. Based on these characteristics, it partitions and clusters the log. The differences between the clusters, then, indicate the quantitative change in the business processes, refer to requirement R4. The Graph Metrics on Process Graphs approach [22] discovers a first model, called a reference, using the Heuristic Miner on a section of the log [1]. Then, it discovers models for other sections of the log and uses graph metrics to compare them with the reference model. The technique interprets significant differences in the metrics as drifts. The reference model and detection windows get updated, once a drift is detected.

Table 1: Process drift detection in process mining.

Approach	R1	R2	R3	R4	R5
ProDrift [14,18]	+	+/-	-	-	-
TPCDD [26]	+	-	-	-	-
Process Trees [17]	+	-	-	-	+
Performance Spectra [7]	-	-	+	-	+
Comparative Trc. Clustering [12]	-	-	-	+	+
Graph Metrics On Proc.Graphs [22]	+	-	-	+	+
VDD approach (this paper)	**+**	**+**	**+**	**+**	**+**

This discussion, summarized in Table 1, witnesses that none of the state-of-the-art methods addresses all the five requirements. Thus, the work at hand, to address the gap.

3 Preliminaries

In this section, formal preliminaries of the approach are given. Section 3.1 discusses DECLARE specification as the main body of process mining research we build upon. Section 3.2 describes clustering and change point detection methods, which are the main instruments of our approach.

An event log L (*log* for short) is a collection of recorded traces that correspond to process executions. In this paper, we abstract the set of activities of a process as a finite non-empty alphabet $\Sigma = \{a,b,c,...\}$, and we define a trace as a finite sequence of activities $a_i \in \sigma, 1 \leqslant i \leqslant n$. Case 1 of the road ticket process from Section 2.1 is an example of a trace. Cases 1-4 are an example of an event log. In the following examples, we shall also resort on the string-representation of traces (i.e., $\sigma = a_1 a_2 \cdots a_n$) defined over Σ.

Table 2: Example DECLARE constraints.

Constraint	Explanation	Examples			
ATMOSTONE(a)	If a occurs, then it occurs at most *once*	✓bcc	✓bcac	×bcaac	×bcacaa
RESPONSE(a,b)	If a occurs, then b occurs eventually after a	✓baabc	✓bcc	×caac	×bacc
ALTERNATERESPONSE(a,b)	If a occurs, then b occurs eventually afterwards, and no other a recurs in between	✓cacb	✓abcacb	×caacb	×bacacb
CHAINRESPONSE(a,b)	If a occurs, then b occurs immediately afterwards	✓cabb	✓abcab	×cacb	×bca
PRECEDENCE(a,b)	If b occurs, then a must have occurred before	✓cacbb	✓acc	×ccbb	×bacc
ALTERNATEPRECEDENCE(a,b)	If b occurs, then a must have occurred before and no other b recurs in between	✓cacba	✓abcaacb	×cacbba	×abbabcb
CHAINPRECEDENCE(a,b)	If b occurs, then a occurs immediately beforehand	✓abca	✓abaabc	×bca	×baacb
NOTSUCCESSION(a,b)	a occurs if and only if b does not occur afterwards	✓bbcaa	✓cbbca	×aacbb	×abb

Event log L is a multiset of traces, as the same trace can be repeated multiple times in the same log: denoting the multiplicity $m \geqslant 0$ as an exponent of the trace, we have that $L = \{\sigma_1^{m_1}, \sigma_2^{m_2}, ..., \sigma_N^{m_N}\}$ (if $m_i = 0$ for some $1 \leqslant i \leqslant N$ we shall simply omit σ_i). The size of the log is defined as $|L| = \sum_{i=1}^{N} m_i$, i.e., the sum of its traces' multiplicities. For example, the size of the Italian help desk log is 150370. A sub-log $L' \subseteq L$ of L is a log $L' = \{\sigma_1^{m'_1}, \sigma_2^{m'_2}, ..., \sigma_N^{m'_N}\}$ such that $m'_i \leqslant m_i$ for all $1 \leqslant i \leqslant N$. A log consisting of cases 1-3 from the example log L in Section 2.1 is a sub-log of L.

3.1 DECLARE modeling and mining

A declarative process specification represents the behavior of a process by means of *constraints*, i.e., temporal rules that specify the conditions under which activities may, must, or cannot be executed. In this paper we focus on DECLARE, one of the most well-established declarative process modeling languages to date [2].

DECLARE provides a standard library of templates (*repertoire* [20,9]), i.e., constraints parametrized over activities. Examples of DECLARE constraints are RESPONSE(a,b) and CHAINPRECEDENCE(b,c). The former constraint applies the RESPONSE template on tasks a and b, and states that if a occurs then b must occur later on within the same trace. In this case, a is named *activation*, because it is mentioned in the "if" clause, thus triggering the constraint, whereas b is named *target*, as it is in the consequent clause [9]. CHAINPRECEDENCE(b,c) asserts that if c (the activation) occurs, then b (the target) must have occurred immediately before. Given an alphabet of activities Σ, we denote the number of all possible constraints that derive from the application of DECLARE templates to all activities in Σ as $\#_{\text{cns}} \subseteq O(\Sigma^2)$ [9]. For the Italian road ticket fine log, $\#_{\text{cns}} = 1584$. Table 2 shows some of the templates of the DECLARE repertoire, together with the examples of traces that satisfy (✓) or violate (×) them.

Declarative process mining tools can measure to what degree constraints hold true in a given event log [15]. To that end, diverse measures have been introduced. Among them, we consider here *support* and *confidence* [10]. Their values range from 0 to 1. In [10], the support of a constraint is measured as the ratio of times that the event is triggered and satisfied over the number of activations. Let us consider the following example event log: $L = \{\sigma_1^4, \sigma_2^1, \sigma_3^2\}$, having $\sigma_1 = \text{baabc}$, $\sigma_2 = \text{bcc}$, and $\sigma_3 = \text{bcba}$. The size of the log is $4+1+2=7$. The activations of RESPONSE(a,b) that satisfy the constraint amount to 8

because two a's occur in σ_1 that are eventually followed by an occurrence of b, and σ_1 has multiplicity 4 in the event log. The total amount of the constraint's activations in L is 10 (see the violating occurrence of a in σ_3). The support thus is 0.8. By the same line of reasoning, the support of ChainPrecedence(b,c) is $\frac{7}{8} = 0.875$ (notice that in σ_2 only one of the two occurrences of c satisfies the constraint). To take into account the frequency with which constraints are triggered, confidence scales support by the ratio of traces in which the activation occurs at least once. Therefore, the confidence of Response(a,b) is $0.8 \times \frac{6}{7} \approx 0.69$ because a does not occur in σ_2. As b occurs in all traces, the confidence of ChainPrecedence(b,c) is 0.875.

3.2 Clustering and change point detection algorithms

In this paper, we focus on the analysis of time-series data. A *time series* is a sequence of ordered data points $\langle t_1, t_2, \cdots, t_d \rangle = T \in \mathbb{R}^d$ consisting of $d \in \mathbb{N}^+$ real values. Figure 3(f) illustrates an example of time series. A *multivariate time series* is a set of $n \in \mathbb{N}^+$ time series $D = \{T_1, T_2, ..., T_n\}$. We assume a multivariate time series to be piece-wise stationary except for its *change points*.

In our approach, we take advantage of the following techniques.

Time series clustering is an unsupervised data mining technique for organizing data points into groups based on their similarity [4]. The objective is to maximize data similarity within clusters and minimize it across clusters. More specifically, the *time-series clustering* is the process of partitioning D into non-overlapping clusters of multivariate time series, $C = \{C_1, C_2, ..., C_m\} \subseteq 2^D$, with $C_i \subseteq D$ and $1 \leqslant m \leqslant n$, for each i such that $1 \leqslant i \leqslant m$, such that homogeneous time series are grouped together based on a *similarity measure*. A *similarity measure* $\mathrm{sim}(T, T')$ represents the distance between two time series T and T' as a non-negative number. Time-series clustering is often used as a subroutine of other more complex algorithms and is employed as a standard tool in data science for anomaly detection, character recognition, pattern discovery, visualization of time series [4].

Change point detection is a technique to detect the points in which multivariate time series exhibit changes in their values [6]. Let D^j denote all elements of D at position j, i.e., $D^j = \{T_1^j, T_2^j, ..., T_n^j\}$, where T^j is a j-th element of time series T. The objective of change point detection algorithms is to find $k \in \mathbb{N}^+$ changes in D, where k is previously unknown. Every element D^j for $0 < j \leqslant k$ is a point at which the values of the time series undergo significant changes. In Fig. 3(f), e.g., each vertical black dashed line is one of the $k = 9$ change points. To detect change points, the search algorithms require a *cost function* and a *penalty* parameter as inputs. The former describes how homogeneous the time series is. It is chosen in a way that its value is high if the time series contains many change points and low otherwise. The latter is needed to constrain the search depth. The supplied penalty should strike a good balance between finding too many change points and not finding any significant ones. Change point detection is a technique commonly used in signal processing and, more in general, for the analysis of dynamic systems that are subject to changes [6] .

4 Technique

In this section, we introduce the VDD approach. First, we derive a multivariate time series from an event log, where each time series represents how the confidence values

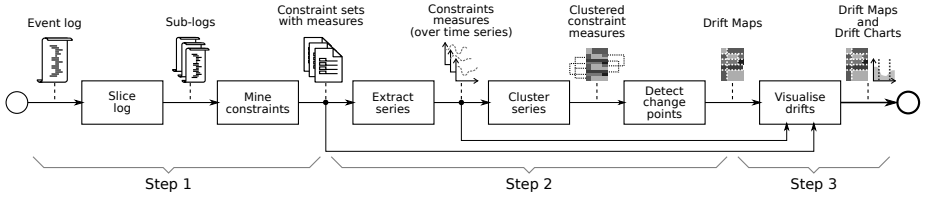


Fig. 2: The VDD approach.

of some DECLARE constraint evolve over time. We prefer confidence over support to prevent that sporadically occurring activities bias our detection algorithms. Then, we cluster sub-sets of time series to group together the constraints that expose a similar trend in their confidence value. Next, using change point detection techniques, we identify the process drifts, i.e., the points in which significant changes in the confidence of behavioral rules occur. Finally, we assess and explain behavioral changes through visual diagrams and numerical reports on drift metrics. Figure 2 illustrates the multi-step VDD approach.

Step 1: Mining DECLARE windows. In this step, we split the log into sub-logs. From each sub-log, we mine the set of DECLARE constraints and compute their confidence.

Step 2: Slicing the DECLARE constraints space into time and behavior sub-spaces. This step begins with the extraction of multi-variate time series that represent the trends of the constraints' confidence. Thereupon, we cluster those time series to find groups of constraints that exhibit similar confidence trends (henceforth, *behavior clusters*). The step ends by returning the detected change points both in the entire multi-variate time series and in each cluster, so as to find overall and behavior-specific drifts, respectively.

Step 3: Explaining drifts. In the last step, we plot Drift Maps and Drift Charts to visually identify and characterize the detected drift. In the following, we detail those steps.

4.1 Mining DECLARE windows

The first step takes as input a log L, and two additional parameters (win_{size} and win_{step}). It returns a multivariate time series D based on the confidence of mined DECLARE constraints.

First, we sort the traces in the event log L by the timestamp of their respective first events. Thereupon, we extract a sub-log from L as a window of size $\text{win}_{\text{size}} \in \mathbb{N}^+$, with $1 \leqslant \text{win}_{\text{size}} \leqslant |L|$. We subsequently shift the sub-log window by a given step ($\text{win}_{\text{step}} \in \mathbb{N}^+$, with $1 \leqslant \text{win}_{\text{step}} \leqslant \text{win}_{\text{size}}$). Notice that we have sliding windows if $\text{win}_{\text{step}} < \text{win}_{\text{size}}$ and tumbling windows if $\text{win}_{\text{step}} = \text{win}_{\text{size}}$. Thus, the number of produced sub-logs is equal to: $\#_{\text{win}} = \left\lfloor \frac{|L| - \text{win}_{\text{size}} - \text{win}_{\text{step}}}{\text{win}_{\text{step}}} \right\rfloor$. Having win_{size} set to 5000 and win_{step} set to 2500, $\#_{\text{win}}$ amounts to 57 on the Italian road fine ticket log.

For every sub-log $L_j \subseteq L$ thus formed ($1 \leqslant j \leqslant \#_{\text{win}}$), we check all possible DECLARE constraints that stem from the activities alphabet of the log, amounting to $\#_{\text{cns}}$ (see Section 3.1). For each constraint $i \in 1..\#_{\text{cns}}$, we compute its confidence over the sub-log L_j, namely $\text{Conf}_{i,j} \in [0,1]$. This generates a time series $T_i = (\text{Conf}_{i,1},...,\text{Conf}_{i,\#_{\text{win}}}) \in [0,1]^{\#_{\text{win}}}$ for every constraint i. In other words, every time series T_i describes the confidence of all the DECLARE constraints discovered in the i-th window of the event log. The multivariate time series $D = \{T_1, T_2, ..., T_{\#_{\text{cns}}}\}$ encompasses the full spectrum of all constraints. Next, we detail the steps of slicing the DECLARE constraints and explaining the drifts.

4.2 Slicing the DECLARE constraints space into time and behavior sub-spaces

The second step processes the previously generated multivariate time series D to derive (i) a set C of clusters exhibiting similar confidence trends, and (ii) a set of $k \in \mathbb{N}^+$ change points representing the process drifts.

Change point detection. To detect change points, we use the *Pruned Exact Linear Time (PELT)* algorithm [13]. This algorithm performs an exact search, but requires the input dataset to be of limited size. Our setup is appropriate as by design the length of the multivariate time-series is limited by the choice of parameters $\mathrm{win}_{\mathrm{size}}$ and $\mathrm{win}_{\mathrm{step}}$. Also, this algorithm is suitable for cases in which the number of change points is unknown a priori [6, p. 24], as in our case. We use the *Kernel cost function*, detailed in [6], which is optimal for our technique, and adopt the procedures described in [13] to identify the optimal *penalty* value.

Clustering time series of DECLARE constraints. By applying a change point detection algorithm on the entire multivariate time-series, we are able to pinpoint the window (i.e., the sub-log) where overall behavior changes occur. However, the level of granularity may be inappropriate as we could not single out the phenomena that are local to certain behavioral rules. That would interfere with the accuracy of results. Therefore, we use time-series clustering techniques [4] to group together similarly changing pockets of behavior of the process. One time series describes how one constraint changes its confidence over time. By clustering, we find all the time series that share similar trends of values, hence, we find all similarly changing constraints. We use *hierarchical clustering*, as it is reportedly one of the most suitable algorithms when the number of clusters is unknown [4]. As a result, we obtain a partition of the multivariate time series of DECLARE constraint confidence values into behavior clusters.

4.3 Explaining drifts

After clustering the behavior of the log and finding the change points, we expand the classification of the types of drifts found in the literature by being able to identify, pinpoint, and categorize the drifts within behavior clusters. We also allow for an assessment of how erratic the clusters are by means of the novel measure described next.

Finding erratic behavior clusters. The behavioral changes in one cluster can be visually depicted by a plot like that in Fig. 3(f). Thus, in order to find and pinpoint the most interesting (erratic) behavior clusters, we define a measure inspired by the idea of finding the length of a poly-line in a plot. The rationale is, straight lines denote a regular trend and have the shortest length, whilst more irregular, wavy curves evidence more behavior changes and their length is higher. We are, therefore, mostly interested in longer lines.

We compute our measure as follows. We calculate for all constraints i such that $1 \leqslant i \leqslant \#_{\mathrm{cns}}$ the Euclidean distance $\delta : [0,1] \times [0,1] \to \mathbb{R}_+$ between consecutive values in the time series $T_i = (T_{i,1},\ldots,T_{i,\mathrm{win}_{\mathrm{size}}})$, i.e., $\delta(T_{i,j},T_{i,j+1})$ for every j s.t. $1 \leqslant j \leqslant \mathrm{win}_{\mathrm{size}}$. For every time series T_i, we thus derive the overall measure $\Delta(T_i) = \sum_{j=1}^{\mathrm{win}_{\mathrm{size}}-1} \delta(T_{i,j},T_{i,j+1})$. Thereupon, to measure how erratic a behavior cluster is, we devise the following measure:

$$\mathrm{Ertc}(C) = \sum_{j=1}^{|C|} \sqrt{1 + (\Delta(T_i) \times \#_{\mathrm{win}})^2} \tag{1}$$

The most erratic behavior cluster has the highest Ertc value.

Visual drift classification. We enable the visual identification of the patterns illustrated in Fig. 1 with a graphical representation that we name Drift Maps: they depict clusters and their constraints' confidence measure evolution along the time series, together with the drift points. We allow the user to inspect every single cluster and its drifts in dedicated diagrams that we name Drift Charts.

Drift Maps, such as those illustrated in Fig. 3(a) or Fig. 4(b), plot all drifts data on a two-dimensional plane. The visual representation we adopt is inspired by [25]. The x-axis is the time axis, while every constraint corresponds to a point along the y-axis. We add vertical lines to mark the identified change points, i.e., drift points, and horizontal lines to demark clusters. Constraints are sorted by the similarity of the confidence trends. The values of the time series are represented through the plasma color-blind friendly color map [25], from blue (low peak) to yellow (high peak).

To analyze the time-dependent trend of specific clusters, we build Drift Charts, such as those depicted in Fig. 3(f) or Fig. 4(c). They have time on the x-axis and average confidence of the constraints in a cluster on the y-axis. We add vertical lines as in Drift Maps.

Drift Maps permit the users to have a global picture of the clusters and of the process drifts. Drift Charts allow for a visual categorization of the drifts according to the classification introduced in [11] (Fig. 1). The following section demonstrates applications of this visual-aided approach on synthetic and real-world logs.

5 Evaluation

This section presents our evaluation setup, its results on detecting and explaining drifts, and a discussion of the results.

5.1 Evaluation setup

We evaluate our approach both on synthetic and real-world event logs.[2,3,4] We also compare the obtained results with the state-of-the-art methods. Table 3 summarizes the event logs used in the evaluation and indicates related work which used these logs. To discover DECLARE constraints, we used MINERful[5] because of its high performance [10]. We opted for the *ruptures* python library[6] for change point identification. We used the *scipy* library[7] for the clustering of time-series, including the hierarchical clustering. By experimenting with the clustering algorithm, we tuned the parameters to attain the best outcome, such as the weighted method for linking clusters (distance between clusters defined as the

Table 3: Event logs used in the evaluation.

Origin	Event log	Related work
Synthetic	ConditionalMove	ProDrift 2.0 [18]
Synthetic	ConditionalRemoval	ProDrift 2.0 [18]
Synthetic	ConditionalToSequence	ProDrift 2.0 [18]
Synthetic	Loop	ProDrift 2.0 [18]
Real-world	Italian help desk[1]	Process Trees [17]
Real-world	BPI2011[3]	ProDrift 2.0 [18]

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

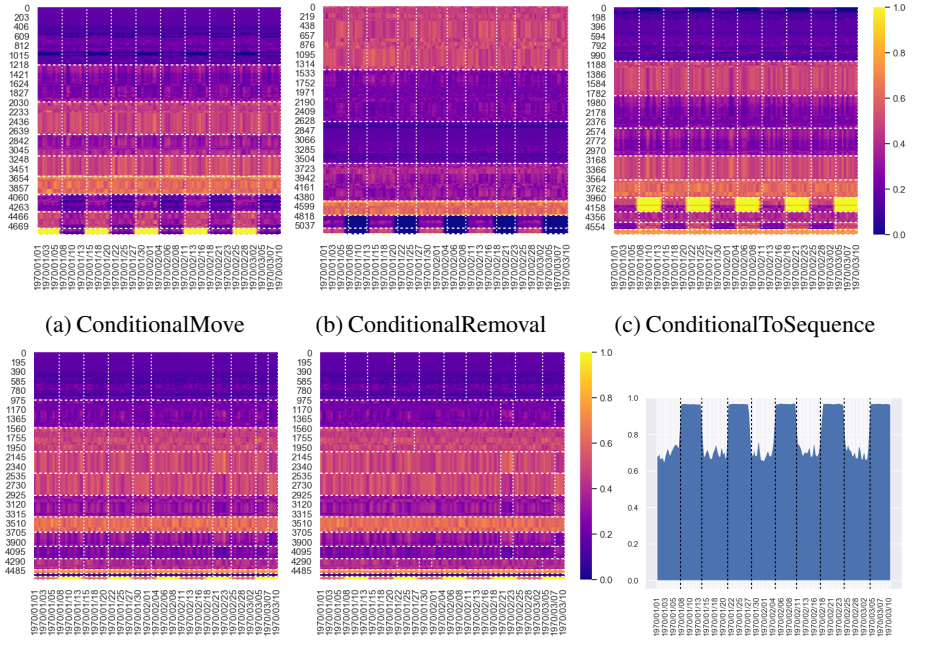
[3] https://doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07

[4] https://doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54 (preprocessed as in [18])

[5] https://github.com/cdc08x/MINERful

[6] https://github.com/deepcharles/ruptures

[7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html



(a) ConditionalMove (b) ConditionalRemoval (c) ConditionalToSequence

(d) Loop (e) Loop, drifts by cluster (f) Loop, most erratic cluster

Fig. 3: Evaluation results on synthetic logs.

average between individual points), and the correlation metric (to find individual distances between two time-series). To enhance Drift Map visualizations, we sort the time-series of each cluster with the mean squared error distance metric. We implemented our approach in Python 3. Its source code is publicly available.[8]

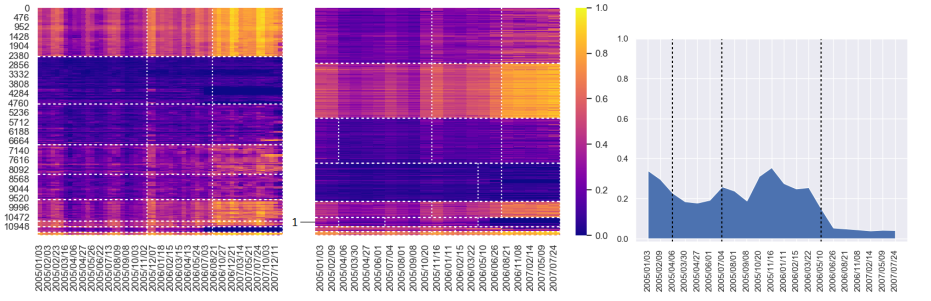
5.2 Detecting drifts

To demonstrate the accuracy with which our technique detects drifts, we first test it on synthetic data in which drifts were manually inserted, to show that we detect drifts at the points in which they occur. Thereafter, we compare our results with the state-of-the-art algorithm ProDrift [18] on real-world event logs.

Synthetic data. Ostovar et al. [18] published a set of synthetic logs that they altered to artificially include drifting behavior: ConditionalMove, ConditionalRemoval, ConditionalToSequence, and Loop.[9] Figure 3 illustrates the results of the application of the VDD technique on these logs. By measuring *precision* as the fraction of correctly identified drifts over all the ones retrieved by VDD and *recall* as the fraction of correctly identified drifts over the actual ones, we computed the F-score (harmonic mean of precision and recall) of our results for each log. Using the default settings and no constraint set clustering, we achieve the F-score of 1.0 for logs ConditionalMove, ConditionalRemoval, ConditionalToSequence, and 0.89 for the Loop log. When applying the cluster-based change detection for the Loop log, we achieve the F-score of 1.0. The Drift Map for the *Loop* log

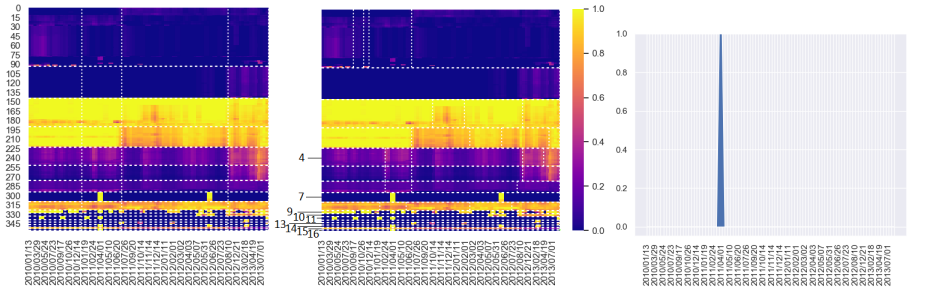
[8] https://github.com/yesanton/Process-Drift-Visualization-With-Declare

[9] http://apromore.org/platform/tools



(a) Overall change points (b) Drifts by cluster (c) Most erratic cluster

Fig. 4: BPIC2011 hospital log VDD visualizations.



(a) Overall change points (b) Drifts by cluster (c) Most erratic cluster

Fig. 5: Italian help desk log VDD visualizations.

is depicted in Fig. 3(e). In contrast to [18] we can see which behavior in which cluster contributes to the drift. The Drift Chart in Fig. 3(f) illustrates the trend of confidence for the most erratic cluster for the *Loop* log.

Real-world data. Figure 4(a) illustrates the Drift Map constructed for the BPIC2011 event log.[3] As in [18], two drifts are detected towards the second half of the time span of the log. However, in addition, our technique identifies drifting behavior at a finer-granular level. Figure 4(b) shows the drifts pertaining to clusters of constraints. The trend of the confidence measure for the most erratic cluster is depicted in Fig. 4(c).

Our technique correctly detects drifts in the Italian help desk log, by identifying the same two drifts that were found by ProDrift [17], approximately in the first half and towards the end of the time span. As illusrated by the VDD visualization in Fig. 5(a), in addition we detected another sudden change in the first quarter. Following on that, we analyzed the within-cluster changes (Fig. 5(b)) and noticed that the most erratic cluster contains an outlier, as is shown by the spike in Fig. 5(c).

5.3 Explaining drifts

To better understand a particular drift, we further examine the constraints that participate in the drift. Using the example of the Italian help desk log presented above, we examine the most erratic behavior clusters' drifts (calculated using Eq. (1)), shown in Table 4. In Fig. 6, we present the most erratic examples of behavior, and in Table 5 we present the constraints that describe that specific behavior after applying the constraint minimization algorithm.

Figure 6(a) shows an erratic behavior, which visually corresponds to the *reoccurring concept* classification from Fig. 1. Examining the constraints that constitute this behavior, the analyst could conclude that in the dates of the peak in Fig. 6(a) the activity Create SW anomaly always had Take in charge ticket executed immediately beforehand, and otherwise in the other parts of the plot. Also, she could conclude that before Create SW anomaly the Assign seriousness activity was executed, and no other Create SW anomaly occurred in between.

Figure 6(b) has four spikes, where Schedule intervention occurred. Immediately before Schedule intervention, activity Take in charge ticket occurred. Also, Assign seriousness occurred had to occur before Schedule intervention recurred. We notice, however, that this cluster shows *outlier* behavior, due to its rare changes.

Table 4: Italian help desk log erratic clusters.

Drift number	Ertc **measure**
without drift	89.000
9	780.041
11	328.881
14	293.887
10	292.712
13	289.103
7	232.401
4	196.012
15	171.012
16	166.111

Finally, Fig. 6(c) depicts a *gradual* drift until June 2012, and the *incremental* drift afterward. We notice that all constraints in the cluster have Wait either as an activation (e.g., with ALTERNATERESPONSE(Wait, closed)) or as a target (e.g., with CHAINRESPONSE(Take in charge ticket,Wait)).

5.4 Discussion

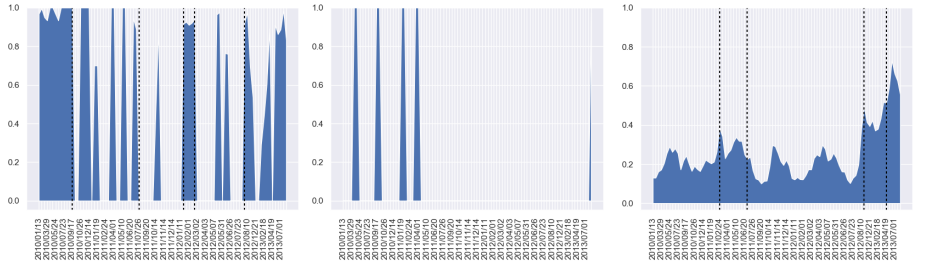
Our method addresses all the five requirements for process drift detection presented in Section 2.2 as follows:

R1 We evaluated our method with the synthetic logs showing its ability to identify drifts precisely;

R2 We developed a visualization approach based on Drift Maps and Drift Charts for classification of process drifts and have shown its effectiveness for real-world logs. Our enhanced approach based on change point detection has yielded effective to automatically discover the exact points at which *sudden* and *reoccurring concept*

Table 5: Italian ticket log constraints; including min, max, and mean confidence.

Cluster	**Constraint**	**Activity 1**	**Activity 2**	**Min**	**Max**	**Mean**
9	CHAINPRECEDENCE	Take in charge ticket	Create SW anomaly	0.0	100.0	42.8
	ALTERNATEPRECEDENCE	Assign seriousness	Create SW anomaly	0.0	100.0	49.0
11	CHAINPRECEDENCE	Take in charge ticket	Schedule intervention	0.0	100.0	9.9
	ALTERNATEPRECEDENCE	Assign seriousness	Schedule intervention	0.0	100.0	9.9
4	CHAINRESPONSE	Take in charge ticket	Wait	9.4	69.6	23.2
	NOTSUCCESSION	Resolve ticket	Wait	10.0	77.2	26.0
	NOTSUCCESSION	Wait	Assign seriousness	10.0	78.0	26.6
	NOTSUCCESSION	Wait	Take in charge ticket	9.8	73.3	22.1
	ALTERNATERESPONSE	Assign seriousness	Wait	9.0	72.3	23.8
	ALTERNATERESPONSE	Wait	Closed	8.3	61.4	22.5
	ALTERNATERESPONSE	Wait	Resolve ticket	8.3	61.4	22.8
	ATMOSTONE	Wait		9.8	68.6	25.1



(a) Cluster 9, Ertc: 780.04 (b) Cluster 11, Ertc: 328.88 (c) Cluster 4, Ertc: 196.01

Fig. 6: Italian help desk log detailed clusters.

drifts occur. The indicative approximation of long-running progress of *incremental* and *gradual* drifts was also found. *Outliers* were detected via time series clustering;

R3 Using clustering, Drift Maps, and Drift Charts, the method enables the drilling down into (rolling up out) sections with a specific behavior (general vs. cluster-specific groups of constraints);

R4 We introduced, and incorporated into our technique, a drift measure called Ertc that quantifies the extent of the drift change;

R5 To further qualitatively analyze the detected drifts, VDD shows how the process specification looks before and after the drift (as a list of DECLARE constraints, refer to Table 5).

We found that the size of the window does not introduce significant difference in results for the automatic evaluation of the VDD, so we recommend using the number of windows that will guide the visual search best, that is around 60 windows should be produced for one graph. That means the recommended parameters are: $\text{win}_{\text{step}} = \frac{|L|}{60+1}$ and $\text{win}_{\text{size}} = 2 \cdot \text{win}_{\text{step}}$ for smooth visual representation.

6 Conclusions

In this paper, we presented a visual technique for detecting and analyzing process drifts in logs of executed business processes. First, the technique uses the MINERful technique to discover declarative process constraints from logs. Second, it applies clustering and change point detection methods over time series of characteristics of the discovered constraints to detect process drifts (in parts of) business processes. The technique then devises visualizations of the detected clusters and change points for the visual classification of drifts. Finally, we presented a technique for evaluating and explaining process drifts.

We evaluated our technique both on synthetic and real-world data. On synthetic logs, the technique achieved an average F-score of 0.96 and outperformed all the state-of-the-art methods. On real-world logs, the technique managed to describe all types of process drifts in a comprehensive manner. Also, the evaluation reported that our technique can identify outliers of process behavior.

Limitations of the work at hand naturally give rise to future research. First, one can study the problem of automatic classification of process drifts; we plan to use shapelets [3] to solve this problem. Second, one can study how the use of other declarative process constraints, e.g., the 4C spectrum [21] or branched DECLARE [8], impacts the effectiveness of the technique. Third, an empirical evaluation with the potential users of the technique

can provide further insights for improving the usability of the approach. Finally, we argue that, based on the identified past process drifts, and using time-series prediction algorithms, one can predict future drifts to prepare for forecasted changes [19].

Acknowledgements. This work is partially funded by the EU H2020 program under MSCA-RISE agreement 645751 (RISE_BPM). Artem Polyvyanyy was partly supported by the Australian Research Council Discovery Project DP180102839.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action. Springer (2016)
2. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. CS - R&D **23**(2), 99–113 (2009)
3. Abanda, A., Mori, U., Lozano, J.A.: A review on distance based time series classification. DMKD **33**(2), 378–412 (2019)
4. Aghabozorgi, S., Seyed Shirkhorshidi, A., Ying Wah, T.: Time-series clustering - a decade review. IS **53**(C), 16–38 (Oct 2015)
5. Bauer, M., Senderovich, A., Gal, A., Grunske, L., Weidlich, M.: How much event data is enough? A statistical framework for process discovery. In: CAISE. pp. 239–256 (2018)
6. Charles Truonga, Laurent Oudre, N.V.: Selective review of offline change point detection methods (2019), https://arxiv.org/abs/1801.00718
7. Denisov, V., Belkina, E., Fahland, D.: BPIC'2018: Mining concept drift in performance spectra of processes (2018)
8. Di Ciccio, C., Maggi, F.M., Mendling, J.: Efficient discovery of target-branched Declare constraints. Inf. Syst. **56**, 258–283 (2016)
9. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. IS **64**, 425–446 (Mar 2017)
10. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM TMIS **5**(4), 24:1–24:37 (2015)
11. Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Comput. Surv. **46**(4), 44:1–44:37 (2014)
12. Hompes, B., Buijs, J.C.A.M., van der Aalst, W.M.P., Dixit, P., Buurman, H.: Detecting change in processes using comparative trace clustering. In: SIMPDA 2015. pp. 95–108 (2015)
13. Killick, R., Fearnhead, P., Eckley, I.A.: Optimal detection of changepoints with a linear computational cost. Journal of the American Statistical Association **107**(500), 1590–1598 (2012)
14. Maaradji, A., Dumas, M., La Rosa, M., Ostovar, A.: Detecting sudden and gradual drifts in business processes from execution traces. IEEE TKDE **29**(10), 2140–2154 (2017)
15. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: CIDM. pp. 192–199. IEEE (2011)
16. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. Computing **98**(4), 407–437 (2016)
17. Ostovar, A., Leemans, S.J., La Rosa, M.: Robust drift characterization from event streams of business processes (2018), https://eprints.qut.edu.au/121158/
18. Ostovar, A., Maaradji, A., La Rosa, M., ter Hofstede, A.H.M., van Dongen, B.F.: Detecting drift from event streams of unpredictable business processes. In: ER. pp. 330–346 (2016)
19. Poll, R., Polyvyanyy, A., Rosemann, M., Röglinger, M., Rupprecht, L.: Process forecasting: Towards proactive business process management. In: BPM. pp. 496–512. Springer (2018)
20. Polyvyanyy, A., Armas-Cervantes, A., Dumas, M., García-Bañuelos, L.: On the expressive power of behavioral profiles. Formal Asp. Comput. **28**(4), 597–613 (2016)

21. Polyvyanyy, A., Weidlich, M., Conforti, R., La Rosa, M., ter Hofstede, A.H.M.: The 4C spectrum of fundamental behavioral relations for concurrent systems. In: Petri nets. pp. 210–232. Springer (2014)
22. Seeliger, A., Nolle, T., Mühlhäuser, M.: Detecting concept drift in processes using graph metrics on process graphs. In: S-BPM. p. 6 (2017)
23. Tsymbal, A.: The problem of concept drift: definitions and related work. Computer Science Department, Trinity College Dublin **106**(2), 58 (2004)
24. van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. TKDE **16**(9), 1128–1142 (2004)
25. Ware, C.: Information visualization: perception for design. Elsevier (2012)
26. Zheng, C., Wen, L., Wang, J.: Detecting process concept drifts from event logs. In: OTM CoopIS. pp. 524–542 (2017)