Characterizing Drift from Event Streams of Business Processes

Alireza Ostovar, Abderrahmane Maaradji, Marcello La Rosa, and Arthur H.M. ter Hofstede

Queensland University of Technology, Australia
{alireza.ostovar, abderrahmane.maaradji, m.larosa, a.terhofstede}@qut.edu.au

Abstract. Early detection of business process drifts from event logs enables analysts to identify changes that may negatively affect process performance. However, detecting a process drift without characterizing its nature is not enough to support analysts in understanding and rectifying process performance issues. We propose a method to characterize process drifts from event streams, in terms of the behavioral relations that are modified by the drift. The method builds upon a technique for online drift detection, and relies on a statistical test to select the behavioral relations extracted from the stream that have the highest explanatory power. The selected relations are then mapped to typical change patterns to explain the detected drifts. An extensive evaluation on synthetic and real-life logs shows that our method is fast and accurate in characterizing process drifts, and performs significantly better than alternative techniques.

1 Introduction

Business processes evolve over time in response to different types of change, such as changes in regulations, competition, supply, demand, technological capabilities, as well as seasonal effects. Some process changes are intentional and planned ahead, while others may occur without being noticed or documented, such as changes resulting from ad-hoc workarounds initiated by individuals in emergency situations, or changes that are due to the replacement of human resources. Over time, these changes may affect process performance, and more generally hinder process improvement initiatives.

In this regard, there is a need for techniques and tools that can discover and characterize, as soon as possible, *process drifts* [13], i.e. statistically significant changes in the behavior of business processes. Accordingly, several techniques have been proposed to detect and localize process drifts from process execution logs (*event logs*) recorded by supporting IT systems [6,2,5,14,13,17]. However, detection and localization of a process drift does not provide, per se, enough insight to undertake a process improvement initiative, unless the drift is characterized, i.e. unless one can understand *what* has changed in the process behavior. To the best of our knowledge, there has not been any attempt to provide a systematic solution for characterizing process drifts.

In this paper, we propose a fully automated online method for characterizing process drifts from event streams. For each detected drift, we perform a statistical test to measure the statistical association between the drift and the distributions of the α^+ relations of process behavior extracted from the event stream before and after the drift. We then rank the relations based on their relative frequency change, and try to match them with a set of predefined change templates. The best-matching templates are then reported to the user as the changes underpinning the drift. We extensively evaluated the accuracy of our method by simulating event streams from artificial and real-life logs. The results show that the approach is fast and highly accurate in characterizing common change patterns, and performs significantly better than state-of-the-art techniques for log delta analysis and model-to-model comparison.

The paper is structured as follows. Section 2 discusses related work. Section 3 introduces the proposed method while Sections 4 and 5 present its evaluation on synthetic and real-life logs, respectively. Section 6 concludes the paper.

2 Related Work

The literature abounds of methods for detecting process drifts [6,2,5,14,13,17]. These methods are based on the idea of extracting features (e.g. patterns) from the process behavior recorded in event logs or in event streams. For example, Bose et al. [5] rely on a statistical test over feature vectors. The user is asked to specify which features to be used for drift detection, implying that they have a-priori knowledge of the possible nature of the drift. In our previous work we introduced two online drift detection methods based on streams of traces [13] or streams of events [17]. The basic idea is to monitor the distribution of a specific feature representing process behavior over two juxtaposed time windows sliding over the trace (event) stream in order to detect a process drift. However, as already remarked, all the above methods only focus on process drift detection, and while some can also localize with high accuracy the drift in the log, none can actually characterize the drift detected.

A possible approach to characterize process drifts is to compare the two process models automatically discovered from the sublogs (or substreams) before and after the drift point. In [3], Armas-Cervantes et al. identify behavioral differences between two process models using canonically reduced event structures. Despite the set of retrieved differences being complete, the accuracy of this approach for process drift characterization highly depends on the quality of the discovered process models. In fact, techniques for automated process discovery are not designed to create overfitting models, i.e. models that do not generalize the behavior of the log [1]. So these models may intentionally add behavior. In addition, these models may be underfitting, i.e. they may not be able to fully capture the process behavior recorded in the log, hence missing behavior [1], especially if the process behavior captured in the log is highly varied. To avoid the possible bias introduced by automated process discovery techniques, one can use log delta analysis techniques, i.e. perform the comparison directly at the level of the log, rather than at the level of the model extracted from the log. In this context, Van Beest et al. [4] propose a technique to detect behavioral differences between two event logs and explain them via natural language statements, by extracting event structures from logs. This technique may be applied for drift characterization by using the two event sublogs (substreams) extracted from before and after the drift point. In the evaluation of our method, we experiment both with the technique for model-to-model comparison in [3], in combination with state-of-the-art techniques for automated process discovery, as well as with the technique for log-to-log comparison in [4].

Drift detection has also been studied in the field of data mining [10], where a widely studied challenge is that of designing efficient learning algorithms that can adapt to data that evolves over time (a.k.a. concept drift). In this context, the term *drift characterization* is often used to refer to the identification of the drift nature, e.g. sudden or gradual [20], as well as the identification of features that explain the drift. For instance, in [18], brushed parallel histograms are used for visualizing concept drifts in multidimensional problem spaces. However, the methods developed in this context deal with simple structures (e.g. numerical or categorical variables and vectors thereof), while in business process drift characterization we seek to characterize changes in more complex structures, specifically behavioral relations between process tasks, such as concurrency, conflicts and loops. Thus, methods from the field of concept drift characterization in data mining cannot be readily transposed to business process drift characterization.

3 Drift Characterization Method

The purpose of process drift characterization is to identify the differences in the process behavior before and after the drift point that best explain the drift. In [17], the α^+ binary relations are shown to be suitable for capturing process behavior, in particular in the context of highly variable business processes. These behavioral relations and their frequencies are extracted from the time window containing the most recent events of the stream. As a preprocessing operation, each time this window slides, a snapshot of the process behavior is captured and stored as a *data point*. Each binary relation actually represents a dimension of the stored data point, while the frequency of this relation is the scalar in this dimension. Sliding the window along the event stream provides us with a set of data points representing snapshots of the pre-drift and post-drift process behaviors. These data points are used as input to our two-stage characterization method.

In Stage 1 we measure the statistical association of each of the α^+ relations with the drift using an information gain metric. Those relations that are significantly associated with the drift are then ordered based on their explanatory power with respect to the drift. In Stage 2, the resulting ordered list of relations is fed to a template matching algorithm, where we find the best-matching templates that characterize the drift. The identified templates are then reported to the user in natural language. An overview of our method is shown in Fig. 1. The rest of this section describes the method in detail.

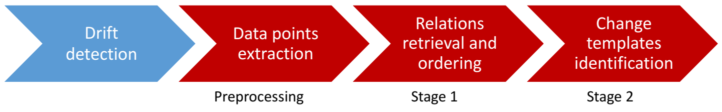

Fig. 1: Overview of our method for process drift characterization.

3.1 Preliminaries

Event logs are at the core of all process mining techniques. An event log is a set of traces, each capturing the sequence of events originated from a given process instance. Each event represents an occurrence of an activity. The configuration where these events are read individually from an online source is known as event streaming. An event stream is a potentially infinite sequence of events, where events are ordered by time and indexed. Events of the same trace do not need to be consecutive in the event stream, i.e. traces can be "overlapping". Formally:

Definition 1 (Event log, Trace, Event stream). Let L be an *event log* over the set of labels $\mathcal{L}$, i.e. $L \in \mathbb{P}(\mathcal{L}^*)$. Let $\mathcal{E}$ be the set of event occurrences and $\lambda : \mathcal{E} \to \mathcal{L}$ a labelling function. An *event trace* $\sigma \in L$ is defined in terms of an order $i \in [0, n-1]$ and a set of events $\mathcal{E}_\sigma \subseteq \mathcal{E}$ with $|\mathcal{E}_\sigma| = n$ such that $\sigma = \langle \lambda(e_0), \lambda(e_1), \ldots, \lambda(e_{n-1}) \rangle$. An *event stream* is a partial bijective function $S : \mathbb{N}^+ \to \mathcal{E}$ that maps every element from the index $\mathbb{N}^+$ to $\mathcal{E}$.

In this paper, we use the α^+ relations, as an extension of the α relations, to capture the behavior of a process. The α-algorithm defines three exclusive relations: *conflict*, *concurrency* and *causality*. The α^+-algorithm adds two more relations: *length-two loop* and *length-one loop*. The α^+ relations are formally defined as follows:

Definition 2 **(α^+ Relations from [15]).** Let L be an event log over $\mathcal{L}$. Let $a,b \in \mathcal{L}$:

- $a\triangle_L b$ if and only if there is a trace $\sigma = l_1l_2l_3...l_n$ and $i \in 1, ..., n-2$ such that $\sigma \in L$ and $l_i = l_{i+2} = a$ and $l_{i+1} = b$,
- $a \diamond_L b$ if and only if $a\triangle_L b$ and $b\triangle_L a$,
- $a >_L b$ if and only if there is a trace $\sigma = l_1l_2l_3...l_{n-1}$ and $i \in 1, ..., n-2$ such that $\sigma \in L$ and $l_i = a$ and $l_{i+1} = b$,
- $a \rightarrow_L b$ if and only if $a >_L b$ and ($b \not>_L a$ or $a \diamond_L b$),
- $a\#_L b$ if and only if $a \not>_L b$ and $b \not>_L a$, and
- $a \parallel_L b$ if and only if $a >_L b$ and $b >_L a$, and $a \not\diamond_L b$.

A length-two loop relation, including a and b, is denoted with $a\triangle_L b$. The frequency of this relation in a log is the number of occurrences of the substring aba. A causality relation from a to b is denoted with $a \rightarrow_L b$. The frequency of this relation in a log is the number of occurrences of the substring ab. A parallel relation between a and b is denoted with $a \parallel_L b$. The frequency of this relation in a log is the minimum of the frequencies of the two substrings, ab and ba. A conflict relation between a and b is denoted with $a\#_L b$, and indicates that there is no trace with the substring ab or ba. The frequency of this relation in a log is the sum of occurrences of a and b. The α^+-algorithm also discovers length-one loop relations (denoted as $\circlearrowleft$) as a pre-processing operation. For example, there is a length-one loop including the activity a in a log if there is a trace with the substring aa. The frequency of this relation in a log is the number of occurrences of the substring aa.

3.2 Preprocessing: Data Points Extraction

For drift detection, we use our technique in [17], which works in online settings with event streams of highly-variable business processes. This technique has been shown to be the state of the art in process drift detection, both in terms of detection accuracy and detection delay. This technique captures process behavior by extracting α^+ binary relations in two juxtaposed windows of the same size, namely *reference* and *detection* windows, sliding along the event stream. The most recent events are equally divided into these two windows, where the reference window contains the less recent events, and the detection window contains the more recent ones. The size of these windows is adjusted using a formula based on the maximum number of distinct activity labels within the two windows. This adaptive window sizing ensures that there are enough events in each window for accurately capturing the process behavior.

We use the detection window as a snapshot of the most recent process behavior. Each time this window slides with the stream on arrival of a new event, we extract α^+ relations and their frequencies and store them as a multidimensional data point in a buffer, namely *characterization buffer*. Each α^+ relation represents a dimension of this data point. By sliding the detection window the new data points are added to the head of the buffer. As a drift is detected, the $P\text{–}value$ of the statistical test drops below the detection threshold (drift point). At this point we stop inserting any new data point into the characterization buffer. We then remove the last w (window size at drift point) data points from the head of the characterization buffer, as these data points may include the post-drift process behavior. This results in a set of recent data points that only encode the process behavior from the pre-drift area. We retain these data points for characterizing the detected drift.

The $P\text{--}value$ remains below threshold until the process behaviors within the two reference and detection windows become statistically similar. In other words until the process behavior, reflected in the event stream, starts to stabilize. Therefore, we call the point where $P\text{--}value$ returns to above the detection threshold a *stabilization point*. This is where we start inserting new data points into the characterization buffer, as the detection window only includes the behavior from the post-drift process. We continue extracting data points from the event stream with the next n incoming events. We define n as the *characterization delay*, as it indicates the delay that is needed after the stabilization point to characterize the drift. Similarly, we consider only the n most recent pre-drift data points for drift characterization. In Section 4.2, we perform an experiment to determine the suitable characterization delay that leads to a hight accuracy of retrieving and ordering the relevant binary relations. The behavioral relations extraction, explained above, is illustrated in Fig. 2.

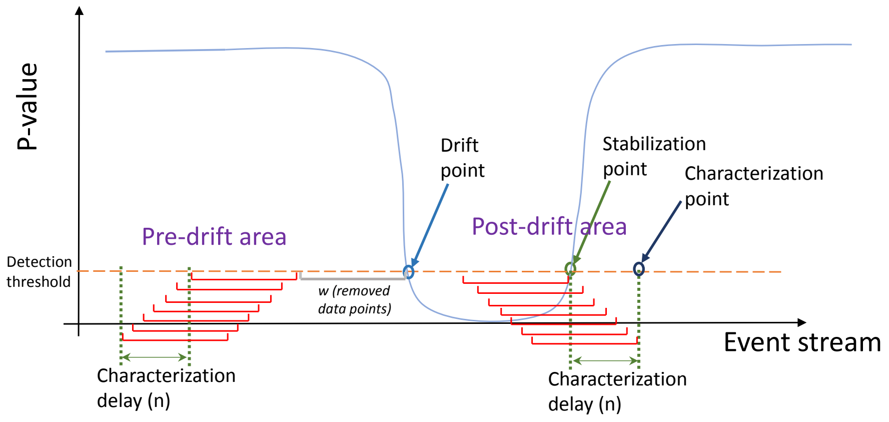

Fig. 2: From drift detection to drift characterization.

3.3 Stage 1: Relevant Binary Relations Retrieval and Ordering

The purpose of the first stage of our approach is to identify and order the α^+ binary relations that are statistically associated with the detected drift. In other words, we would like to measure the explanatory power of each relation with respect to the detected drift. We approach this issue as a classification problem with the α^+ binary relations, extracted from the event stream, as the explanatory variables, and the binary target variable defined with the labels *pre-drift* and *post-drift*. One might first opt for a logistic regression model because of its additive and interpretability properties. However, the logistic regression requires the least correlation between the independent variables (multicollinearity problem [16]). Such a requirement cannot be guaranteed, particularity in our case where the binary relations come from the same process (model). We opted for a less restrictive classification approach, namely decision tree, where we use K-sample permutation test (KSPT) in order to measure the statistical association between each individual explanatory variable (here a binary relation) and the target variable (the drift classification variable). Similarly to the *information gain*, the permutation test allows us to measure the mutual information between two variables. We opted for the permutation test since it is more suitable for small sample sizes [9]. We perform a pairwise permutation test to measure the significance of the statistical association of each binary relation with the target variable (drift). This latter is encoded with the value 0 (resp. 1) for the pre-drift (resp. post-drift) behavior. If the null hypothesis is rejected, we discard the relation as it is not significantly associated with the drift.

As suggested in [9], the KSPT can be applied to identify the relevant features, then an appropriate distance measure is used to order the selected features. Indeed, despite identifying the relations that are found to be statistically associated with our binary drift target variable, some relations may contribute more than others to the change that occurred. We use a measure that is similar to the chi-squared statistic to measure the contribution of each relation to the overall change. This metric measures the *relative frequency change (RFC)* of each relation, and is defines as $RFC = {}^{(O-E)^2}/_{max(O,E)}$, where O and E are the average frequencies of a relation before and after the drift point, respectively. In addition, *total relative frequency change (TRFC)* is defined as the sum of the $RFCs$ of all relations. With relations ordered based on their RFCs in descending order, we can filter out the relations with insignificant RFCs by retaining only the top relations, summing up to $x\%$ of the TRFC, where $x\% \cdot$ TRFC is defined as *cumulative relative frequency change (CRFC)*. In section 4.3, we perform an experiment to investigate the impact of varying CRFC on the characterization accuracy.

3.4 Stage 2: Change Templates Identification

The output of the Stage 1 is a list of relations ordered based on their explanatory power (RFC) with respect to the drift, where the first ordered relation and the last ordered relation have the highest and the lowest explanatory power, respectively. In the stage 2, we aim to match the relations with the typical change patterns that may characterize the drift the best. For that we define a set of templates based on the change patterns defined in [21]. These templates, summarized in Table 1, describe different generic change operations commonly occurring in business process models, such as adding/removing an activity, making an activity loopable, swapping two activities, or parallelizing two sequential activities. Each template is represented based on α^+ binary relations. We try to match the process relations, obtained from Stage 1, with the binary relations of the predefined templates. Using a matching confidence metric we find the best matching between templates and the process relations. In the rest, we explain our template matching algorithm in detail.

Code	Simple change template	Cat.
sre	Add/remove activity between two process fragments	I
pre	Add/remove activity to/from parallel branch	I
cre	Add/remove activity to/from conditional branch	I
cp	Duplicate activity	I
rp	Substitute activity	I
sw	Swap two activities	I
sm	Move activity to between two process fragments	I
pm	Move activity into/out of parallel branch	I
cm	Move activity into/out of conditional branch	I
cf	Make activities conditional/sequential	R
pl	Make activities parallel/sequential	R
cd	Synchronize two activities	R
lp	Make activity loopable/non-loopable	O
cb	Make activity skippable/non-skippable	O
fr	Change branching frequency	O

Table 1: Change templates from [21].

Example 1 *As a running example, let us assume the output of the stage 1 of our method is the ordered relation list of* $\langle\ e \rightarrow f$: $-$, $e \parallel f$: $+$, $e \rightarrow g$: $+$, $d \rightarrow f$: $+$, $a \rightarrow b$: $-$, $f \rightarrow g$: $\searrow$, $d \rightarrow e$: $\searrow$, $b \rightarrow c$: $-$, $a \rightarrow c$: $+\rangle$, *where* $+$ *(resp.* $-$*) indicates that the* *relation appeared (resp. disappeared) after the drift, and $\nearrow$ (resp. $\searrow$) indicates that the frequency of the relation increased (resp. decreased) after the drift.*

In the remainder of this paper, unless otherwise indicated, we use both "feature" and "relation" to refer to an α^+ binary relation between two activity labels. A *feature set* is used to represent the α^+ relations before or after the drift, and is defined as follows.

Definition 3 (Feature Set). Let $\mathcal{L}$ be a set of activity labels, and $\mathcal{T} := \{\rightarrow, \|, \#, \circlearrowleft, \triangle\}$ a set of binary α^+ relations symbols, denoting causality, concurrency, conflict, length one and two loops, respectively. A *feature set* $F\colon \mathcal{L} \times \mathcal{L} \rightarrowtail \mathcal{T}$ is a partial function that yields the type of α^+ relation between two labels.

Two feature sets, will be used to represent the sets of the discovered features before and after a given drift point, along with a classification of a feature frequency change before and after the drift point. The classification only considers the relations that existed both before and after the occurrence of the drift, in our example $\{f \rightarrow g,\ d \rightarrow e\}$. A relation is classified as increasing ($\nearrow$), decreasing ($\searrow$) or not applicable ($\perp$), depending on whether its frequency increased, decreased, or remained unchanged. A relation that disappeared (resp. appeared) after the drift does not need to be classified as it only belongs to the pre-drift (resp. post-drift) feature set. All the features existing before and after the drift are ordered in terms of their explanatory power. The two feature sets from before and after the drift, the classification and the ordering functions form a *drift feature set* which constitutes the output of the first stage of our method. Formally, a drift feature set is defined as follows:

Definition 4 (Drift Feature Set). Let $\mathcal{O} := \{\nearrow, \searrow, \perp\}$ be a set of feature frequency change types. A *drift feature set* is a tuple $D := \langle F_{pre}, F_{post}, \mathit{Diff}_D, \sqsubseteq, \mathcal{L}\rangle$, where F_{pre} (resp. F_{post}) is the feature set before (resp. after) a drift, Diff_D is a classification function defined as $\mathit{Diff}_D\colon F_{pre} \cap F_{post} \rightarrow \mathcal{O}$, and $\sqsubseteq$ is a total order on $F_{pre} \cup F_{post}$.

The following function returns the index of a feature in a given drift feature set.

Definition 5 (Rank). Let $\preceq$ be a total order on a finite set $\mathcal{B}$. For all $b \in \mathcal{B}$, $\mathit{Rank}(b, \preceq, \mathcal{B}) = |\{b' \in \mathcal{B} \mid b' \preceq b\}|$.

Example 2 *With the Definition 4, Example 1 is represented as a drift feature set* $D^1 = \langle F_{pre}^{D^1}, F_{post}^{D^1}, \mathit{Diff}_D, \searrow, \mathcal{L} \rangle$*, where* $\mathcal{L} = \{a, b, c, d, e, f, g\}$, $F_{pre}^{D^1} = \{e \rightarrow f, a \rightarrow b, f \rightarrow g, d \rightarrow e, b \rightarrow c\}$, $F_{post}^{D^1} = \{e \parallel f, e \rightarrow g, d \rightarrow f, f \rightarrow g, d \rightarrow e, a \rightarrow c\}$, $\sqsubseteq\ = \langle e \rightarrow f, e \parallel f, e \rightarrow g, d \rightarrow f, a \rightarrow b, f \rightarrow g, d \rightarrow e, b \rightarrow c, a \rightarrow c\rangle$*, and* $\mathit{Diff}_D = \{(f \rightarrow g, \searrow), (d \rightarrow e, \searrow)\}$.

Our drift characterization method aims at explaining a detected drift using predefined change templates. In this regard, we define a set of change templates representing the typical change patterns [21]. These templates are presented in Table 1. A change template is represented by a process model fragment before the change compared to another process model fragment after the change.

Consequently, a *template* is a generic way to describe a typical change pattern. It enumerates the expected sets of relations before and after the change based on a change pattern representation. The relations that are present in both process model fragments, before and after the change, need to be classified based on their expected frequency evolution in the change pattern. Besides, the importance of every relation in the change pattern is appended to the template. A template handles variables that can be instantiated with actual activity labels in a matching operation.

Definition 6 (Template). Let $\mathcal{V}$ be a set of variables, $\mathcal{T}$ a set of α^+ binary relations symbols, and $\mathcal{O}$ a set of relation frequency change types. A *template* is a tuple $T := \langle T_{pre}, T_{post}, \mathit{Diff}_T, \mathcal{S}, \mathcal{V}\rangle$ where $T_{pre} : \mathcal{V} \times \mathcal{V} \rightarrowtail \mathcal{T}$ represents the relations before the change, $T_{post} : \mathcal{V} \times \mathcal{V} \rightarrowtail \mathcal{T}$ represents the relations after the change, Diff_T is a classification function defined as $\mathit{Diff}_T : T_{pre} \cap T_{post} \rightarrow \mathcal{O}$, and $\mathcal{S}$ is a function specifying the importance of each relation to the template T defined as $\mathcal{S} : T_{pre} \cup T_{post} \rightarrow (0, 1]$.

Example 3 *Let us assume the two change templates, parallelize activities (T^{pl}) and remove activity (T^{sre}), for our example, illustrated in the Fig. 3 and Fig. 4, respectively. With the Definition 6* $T^{pl} = \langle\{X \rightarrow Y, W \rightarrow X, Y \rightarrow Z\}, \{X \parallel Y, W \rightarrow Y, X \rightarrow Z, W \rightarrow X, Y \rightarrow Z\}, \{(W \rightarrow X, \searrow), (Y \rightarrow Z, \searrow)\}, \{(X \rightarrow Y, 1), (W \rightarrow X, 1), (Y \rightarrow Z, 1), (X \parallel Y, 1), (W \rightarrow Y, 1), (X \rightarrow Z, 1)\}, \{W, X, Y, Z\}\rangle$, *and* $T^{sre} = \langle\{X \rightarrow Y, Y \rightarrow Z\}, \{X \rightarrow Z\}, \varnothing, \{(X \rightarrow Y, 1), (Y \rightarrow Z, 1), (X \rightarrow Z, 1)\}, \{X, Y, Z\}\rangle$.

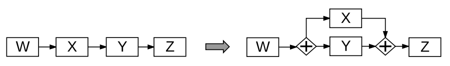

Fig. 3: Parallelize activities template (T^{pl})

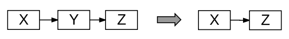

Fig. 4: Remove activity template (T^{sre})

In order to explain a drift, the discovered features represented with a drift feature set are matched to a predefined template. All the variables in the template need to be mapped to a label from the drift feature set. This operation is called a *valid instantiation*, and is defined as follows:

Definition 7 (Valid Instantiation). Given a drift feature set $D := \langle F_{pre}, F_{post}, \mathit{Diff}_D, \sqsubseteq, \mathcal{L}\rangle$, and a template $T := \langle T_{pre}, T_{post}, \mathit{Diff}_T, \mathcal{S}, \mathcal{V}\rangle$, a *valid instantiation of T through D* is a function $\mathcal{I}_{D,T} : \mathcal{V} \rightarrow \mathcal{L}$ such that

- $T_{pre}(v_1, v_2) = t_1$ *iff* $F_{pre}(\mathcal{I}_{D,T}(v_1), \mathcal{I}_{D,T}(v_2)) = t_1$,
- $T_{post}(v_3, v_4) = t_2$ *iff* $F_{post}(\mathcal{I}_{D,T}(v_3), \mathcal{I}_{D,T}(v_4)) = t_2$, and
- $\mathit{Diff}_T(v_5, v_6) = \vartheta$ *iff* $\mathit{Diff}_D(\mathcal{I}_{D,T}(v_5), \mathcal{I}_{D,T}(v_6)) = \vartheta$

Example 4 *In our example, we can have two valid instantiations, one per template. The first instantiation* $I_{D^1,T^{pl}} = \{W : d, X : e, Y : f, Z : g\}$ *, whereas the second instantiation* $I_{D^1,T^{sre}} = \{X : a, Y : b, Z : c\}$.

A confidence is calculated for each matching (valid instantiation) in order to assess the likelihood of such a matching. The *confidence of an instantiation* is based on the *Discounted Cumulative Gain* (DCG) measure [11], which indicates the quality of ranking relations in a drift feature set with regards to their predefined importance in a template. In our method, we consider the same importance of 1 for all the relations of a template. The confidence of an instantiation is defined as follows.

Definition 8 (Confidence in an Instantiation). Given a drift feature set $D := \langle F_{pre}, F_{post}, \mathit{Diff}_D, \sqsubseteq, \mathcal{L}\rangle$, a template $T := \langle T_{pre}, T_{post}, \mathit{Diff}_T, \mathcal{S}, \mathcal{V}\rangle$, and a valid instantiation $\mathcal{I}_{D,T} : \mathcal{V} \rightarrow \mathcal{L}$, the *confidence* $\mathcal{C}(\mathcal{I}_{D,T})$ of D matching T through $\mathcal{I}_{D,T}$ is:
$$\mathcal{C}(\mathcal{I}_{D,T}) = \sum\nolimits_{(x,y,t)\in T_{pre}\cup T_{post}} \frac{\mathcal{S}(x, y, t)}{\log_2(\mathit{Rank}((\mathcal{I}_{D,T}(x), \mathcal{I}_{D,T}(y), t),\ \sqsubseteq,\ F_{pre} \cup F_{post}) + 1)}$$

Example 5 *In our example, the confidence of $I_{D^1,T^{pl}}$ is calculated as follows:* $C(I_{D^1,T^{pl}}) = \frac{1}{\log_2(1+1)} + \frac{1}{\log_2(2+1)} + \frac{1}{\log_2(3+1)} + \frac{1}{\log_2(4+1)} + \frac{1}{\log_2(6+1)} + \frac{1}{\log_2(7+1)} \approx 2.25$. *The confidence of $I_{D^1,T^{sre}}$ is calculated in the same way and approximates to* 0.62.

As we want to find the best-matching template among all matching templates we need to rank them based on their confidences. However, as the number of relations in different templates may not be the same, we need to normalize the confidence of an instantiation with respect to the maximal confidence of its template. Similarly to the normalized DCG (nDCG) [11], we first define the notion of *ideal confidence* of a template T as the DCG obtained after ordering relations of T based on their importance defined by $\mathcal{S}$. The *normalized confidence (nC)* of an instantiation is calculated by dividing the confidence of the instantiation by the ideal confidence of its template.

Definition 8 (continued). The *Ideal confidence* $i\mathcal{C}(T)$ of T is computed as $i\mathcal{C}(T) = \sum_{(x,y,t)\in T_{pre}\cup T_{post}} \frac{\mathcal{S}(x,y,t)}{\log_2(Rank((x,y,t),\ \geq,\ \text{range}(\mathcal{S}))+1)}$, and the *normalized confidence* $n\mathcal{C}(\mathcal{I}_{D,T})$ of D matching T through $\mathcal{I}_{D,T}$ is computed as $n\mathcal{C}(\mathcal{I}_{D,T}) = \frac{\mathcal{C}(\mathcal{I}_{D,T})}{i\mathcal{C}(\mathcal{I}_{D,T})}$

Example 6 *In our example,* $i\mathcal{C}(T^{pl}) \approx 2.30$ *and* $n\mathcal{C}(I_{D^1,T^{pl}}) \approx 0.98$*, whereas* $i\mathcal{C}(T^{sre}) \approx 1.13$ *and* $n\mathcal{C}(I_{D^1,T^{sre}}) \approx 0.54$*. As* $n\mathcal{C}(I_{D^1,T^{pl}}) \geq n\mathcal{C}(I_{D^1,T^{sre}})$*,* T^{pl} *is identified as the best-matching template with the drift feature set.*

Simultaneous changes. Identifying one template is not enough as a process drift may involve more than one change. In order to characterize all the simultaneous changes, each time that a best-matching template with the drift feature set is identified, we remove the features that were used for this template instantiation from the drift feature set. The new resulting drift feature set is then reused for the identification of a new best-matching template. We repeat this cycle until we cannot find any more templates that match the remaining features within the drift feature set. It is worth mentioning that if there are two *overlapping* changes in the process, i.e. changes that share a non-empty set of features, only the one with higher nC can be matched with a template. This is because each time we find a best-matching template we remove the matched features from the drift feature set. This limits the ability of the proposed method to the identification of non-overlapping simultaneous changes.

Example 7 *In our example, as there is no feature shared between* $I_{D^1,T^{pl}}$ *and* $I_{D^1,T^{sre}}$*, both change templates can be identified. The identified templates,* T^{pl} *and* T^{sre}*, are then reported to the user using the two following statements, respectively:*

– *Before the drift, activity "e" preceded "f", while after the drift, they are in parallel.*
– *Activity "b" has been removed from between activities "a" and "c" after the detected drift.*

Time complexity. Given the number of data points $2n$, where n is the characterization delay, and the maximum possible number of α^+ relations $|\mathcal{L}|^2$, where $\mathcal{L}$ is the label set, the complexity of our drift characterization method is the maximum of the worst-case complexities of the following sequential operations: (i) performing KSPT between the α^+ relations and a binary target variable ($O(2n \cdot |\mathcal{L}|^2)$), (ii) computing the average frequencies and RFCs of the relations ($O(2n\ .\ |\mathcal{L}|^2)$), (iii) ordering the relations ($O(|\mathcal{L}|^2 \cdot \log(|\mathcal{L}|^2))$), and (iv) template identification $O(|\mathcal{L}|^2 \cdot m \cdot |\mathcal{L}|^2!)$.[1] Hence, the

[1] Matching a template of k relations to a drift feature set of $|\mathcal{L}|^2$ relations requires iterating over all possible permutations ($^nP_k = {}^{|\mathcal{L}|^2!}/_{(|\mathcal{L}|^2-k)!}$). The upper-bound complexity of this operation is $O(|\mathcal{L}|^2!)$. Next, to identify the best-matching template, we iterate over the number of predefined templates m. Finally, we need to match simultaneous changes which in the worse case are $|\mathcal{L}|^2$ (where each template has only one relation). The upper-bound time complexity of identifying multiple non-overlapping templates is $O(|\mathcal{L}|^2 \cdot m \cdot |\mathcal{L}|^2!)$.

time complexity of our method is $O(|\mathcal{L}|^2 \cdot m \cdot |\mathcal{L}|^2!)$. This time complexity is a theoretical upper-bound, however in practice the number of relations rarely approaches $|\mathcal{L}|^2$, and not all permutations are verified for the template identification operations (relations are first filtered based on their types, e.g. causality).

4 Evaluation on Synthetic Logs

We implemented the proposed method as an extension of the *ProDrift 2.0* plugin for the Apromore platform.[2] This tool is fed with an event stream replayed from an event log, and reports, for each detected drift, its characterization as a verbalization in natural language, based on the applicable templates. We used this tool to evaluate the effectiveness of our method with different parameters settings. In the rest of this section we discuss the setup of the experiments and a two-pronged evaluation to assess the effectiveness of the relevant relations retrieval and ranking with respect to each individual template, and the accuracy of template identification. Finally, we compare our method with model-to-model comparison in combination with automated process discovery, as well as log-to-log comparison.

4.1 Setup

We generated a synthetic dataset using the same approach and CPN[3] base model in [17] that represents a highly variable process. For each simple change template in Tab. 1, we generated a log featuring 9 drifts, each injected by alternatively activating and deactivating the template within the base model. For instance, for the template "sre" we alternatively added or removed an activity to or from the process model. For the particular change template "lp", three logs were generated with length-one, length-two and length-three loops, and the reported results for this template were averaged over these three logs. This resulted in 17 logs, each containing 10,000 traces with nine equidistant drifts of the same change template. To evaluate the characterization of drifts in the context of simultaneous changes, we organized our change templates in three categories: Insertion ("I"), Resequentialization ("R") and Optionalization ("O") (cf. Table 1). Limited to two and three simultaneous cross-category changes, these categories make four possible scenarios of simultaneous changes ("IR", "IO", "RO", "RIO"). For each such scenarios two logs were generated by randomly selecting single templates from different categories. For instance, a drift from the simultaneous changes scenario of "IR" could simultaneously add a new activity ("I") and a loop back ("R") in two different locations of the process. This resulted in eight logs for the simultaneous changes setting. All in all, the dataset contained 25 logs for both single and simultaneous changes.[4]

4.2 Impact of Characterization Delay on Relations Ordering

In Stage 1 of our method, the KSPT is used to retrieve the relations that are significantly associated with the drift, and discard the irrelevant ones. Then, the retrieved binary relations are ordered based on their RFCs with respect to the TRFC that occurred in the drift. For each detected drift, the ground truth (ideal case) is that the relations related to the injected drift template are correctly identified and placed in the top of the returned

[2] Available at http://apromore.org/platform/tools

[3] http://cpntools.org

[4] All the CPN models used for this simulation, the resulting synthetic logs, and the detailed evaluation results are available with the software distribution.

ordered list. However, some spurious relations may affect the relations ordering. We use the *normalized discounted cumulative gain (nDCG)* to evaluate the accuracy of the relations ordering. The nDCG is a relative measure where a value of 1.0 indicates that the ordered list corresponds to the ground truth, while 0.0 indicates that none of the relations related to the injected drift template have been retrieved. This measure is also used for computing the confidence of a template matching, as explained in Section 3.3.

In the first experiment, we study how the accuracy of the ordered binary relations list is impacted by changing the characterization delay. We vary the characterization delay from 200 to 1,000 events, and report the mean and the standard deviation of the nDCG over all the simple change templates, where each template was evaluated separately over nine injected drifts (cf. Fig. 5). In this experiment, we do not apply any filtering on the ordered binary relations list (CRFC = $100\% \cdot$ TRFC).

Not surprisingly, for a characterization delay of 200 events, the KSPT does not have enough data to identify the relevant binary relations causing the drift, which leads to a relatively low average nDCG of around 0.84 and a standard deviation of 0.19 over all templates. Consequently, spurious relations, most often resulting from a slight change in a branching probability, appear in the ordered relations list. However, we observe that the accuracy of the relations ordering increases when the characterization delay grows and eventually plateaus at an average of 0.98 with a standard deviation of 0.02. As expected, the more data points are fed to the KSPT, the more accurate is the statistical association between the explanatory variable (here an individual binary relation) and the target variable (the drift classification variable), and the better the estimation of the RFC for ordering the relations is. However, the characterization delay cannot grow indefinitely, hence, we select 500 events as a trade-off between a short characterization delay and a high characterization accuracy (fewer spurious relations). This value is used as the default delay in the remaining experiments.

We note that the characterization delay does not only indicate how many events our method needs to fetch from the event stream to obtain an accurate characterization, but it also allows us to infer the minimum inter-drift distance that our method can handle. In other terms, the next potential drift must occur at least after a number of events equal to this characterization delay (+ one detection window) after the stabilization point (cf. Fig. 2) in order to be accurately characterized.

4.3 Impact of Relation Filtering on Characterization Accuracy

As introduced in Section 3.3, the ordered relations list resulting from Stage 1 can be filtered based on the CRFC to discard the relations with insignificant RFCs. Thus, only the top relations that sum up their CRFC to a certain proportion of the TRFC are retained. The filtered list is then fed to the template identification stage to find the best-matching templates with the relations. In this experiment, we study how the filter affects the accuracy of template identification. We vary the CRFC threshold (x%) from 70% to 100% (no filtering), and report the *F-score* of the template identification averaged over the 25 synthetic logs. The F-score is measured as the harmonic mean of *recall* and *precision*, where recall measures the ratio of correctly identified change templates of a specific type over the total number of injected templates of the same type, and precision measures the ratio of correctly identified change templates of a specific type over the total number of identified templates of that same type. Figure 6 shows the average accuracy over all templates and per single change, double and triple simultaneous changes.

As expected, we observe that the F-score increases as the CRFC threshold increases. When the threshold is low, many relations are filtered out, and if only one relation corresponding to an injected template is discarded then its corresponding template will not be matched. On the other hand, when the threshold increases, more relations remain in the filtered list, thereby increasing the likelihood of matching the relevant template, leading to a higher recall. However, when no relations are filtered out (threshold = 100%), spurious relations will be matched with the frequency template "fr". This will impact the precision, explaining the drop in the average F-score at the threshold value of 100%. As an example, for the change template parallel move "pm" (with 8 relations), the output of the first stage of our method was an ordered list of 50 relations. A filter threshold of 70% retains only the top five relations out of 50, leading to a recall of 0 for this template. On the other hand, a threshold of 90% retains the top nine relations, leading to a recall of 1. In the remaining experiments we use a CRFC threshold of 95% that is suitable for both single and simultaneous changes.

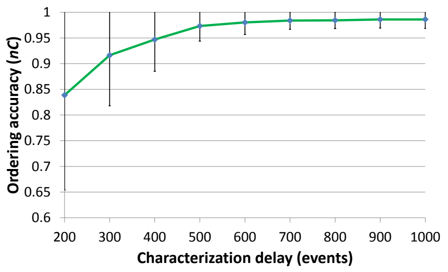

Fig. 5: Impact of characterization delay on relevant relations retrieval and ordering

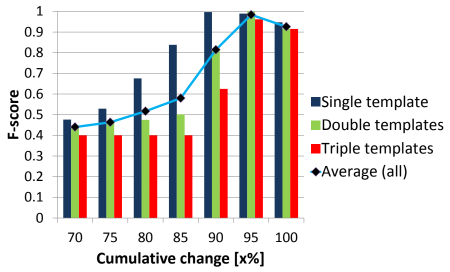

Fig. 6: Impact of relation filtering on characterization accuracy

4.4 Comparison with Baseline

As discussed in Section 2, a possible approach to process drift characterization is to apply automated process discovery before and after the drift point, and compare the resulting process models. We first conducted an exploratory experiment using a sample artificial log with a single injected drift. After drift detection, we extracted the pre-drift and post-drift sublogsand fed these to two state-of-the-art automated process discovery techniques: Inductive Miner [12] and BPMN Miner [7]. The resulting models, obtained from each technique, were then compared using the model-to-model technique in [3]. The comparison between the pair of models discovered by the Inductive Miner did not finish after six hours of execution. This is explained by the over-generalization introduced by the Inductive Miner in the discovered models. In the particular situation of a highly variable process, this miner tends to produce a model close to the so-called *flower model*. This causes the model-to-model comparison technique to explore the combination of all the possible execution paths from the two models. using BPMN Miner, the model comparison technique produced many incorrect differences. This false positives are due to the two models being underfitting. For instance, if the discovered pre-drift model misses to represent a particular process behavior, the comparison technique mistakenly reports this behavior as being added after the drift. Based on these results, we decided to discard this approach as a baseline to benchmark our method.

We then evaluated the possibility of using the log-to-log comparison technique in [4] as a baseline. This technique is designed to compare logs with complete traces, while in our setting the pre-drift and post-drift sublogs are extracted from an event stream, and hence contain many incomplete traces. As a first attempt, we fed the log comparison technique with the two sublogs before and after the drift as is, but as expected, the comparison led to a large number of misleading differences. We then decided to only use complete traces within the two sublogs. This was possible as we knew the start and end activities of the process. However, in an online setting such activities may not be known. For each change template, we evaluated the accuracy of the differences returned by the technique manually. We calculated recall by considering the missing differences for a given template as false negatives, so that a recall of 1 is obtained if a template is fully described by the differences. Similarly, precision was calculated by considering the statements that were not related to the template as false positives.

Figure 7 reports the F-score obtained for each change template for our method and for the baseline. Our method had almost a perfect F-score for every template as it could retain the (great majority of the) relations that were involved in the injected change template, without returning relations that did not fit the templates. On the other hand, the baseline produced a low F-score for all the change templates. Admittedly, this technique had a high average recall of around 0.85 over all logs. However, its precision was very low due to a high number of false positives (wrong differences returned). Indeed, the two sublogs capture partial process behavior, which, even if similar at the event level, is quite variable at the trace level. This was exacerbated by the high variability of the process. These results are in line with the findings in [17] on drift detection (the step preceding the drift characterization). In the latter study, we showed that techniques based on (abstraction of) complete traces such as [13] do not perform well when detecting drifts in highly variable logs and that finer-grained features such as the $\alpha+$ relations are more suitable to capture process behavior in high variability settings.

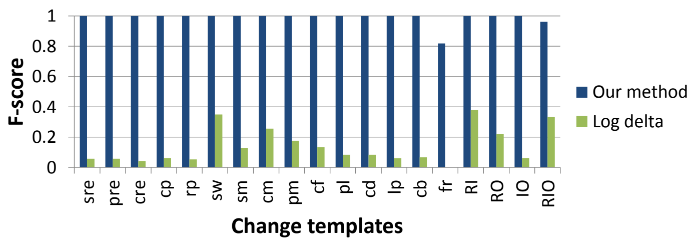

Fig. 7: F-score per change template, obtained with our method vs. [4].

We conducted all the experiments on an Intel i7 2.20GHz with 16GB RAM (64 bit), running Windows 7 and JVM 7 with standard heap space of 4GB. The time required to extract, order, and then match the α^+ relations to the predefined templates for each drift ranged from a minimum of 410ms to a maximum of 660ms with an average of 530ms. The baseline method took on average 15 seconds to report the differences between the pre-drift and post-drift sub-logs.

5 Evaluation on Real-life Log

We further evaluated our method on the BPI Challenge (BPIC) 2011.[5] We chose this log, which records patient treatments in a Dutch hospital, because of its high trace variability ($\sim 70\%$). We prepared the log by filtering out infrequent behavior using the noise filter in [8] with its default settings. This operation resulted in a log with 1,121 traces, of which 798 are distinct, and 42 activity labels. In [17], we had detected two

[5] http://dx.doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54

drifts from this filtered log, using our technique for drift detection. The two drifts were supported by the observation of a sudden increase, and a subsequent decrease in the number of events while the number of active cases was decreasing.

We applied our method for drift characterization in order to identify the change templates that explain these two drifts. Two frequency change templates were identified to characterize the first drift, while the second drift was explained by one frequency change template. This template was symmetric to the first frequency change template, identified for the first drift. After investigation, we found that the probability of the branch which was identified by the change template as increasing (resp. decreasing) after the first (resp. second) drift point included five activities in a loopback. The increase from 34% to 46% (resp. decrease from 46% to 34%) in the upper branch probability of the identified frequency change template is, in fact, the cause of the increased (resp. decreased) number of events after the first (resp. second) drift. Figure 8 depicts the identified template, with the activity labels in their original language.

As discussed in Section 4.4, the baseline technique for log-to-log comparison that we used [4], is designed to compare logs with complete traces. However, there was no complete trace within the pre-drift and post-drift sublogs. Thus, we ran the baseline technique using the sublogs containing only partial traces. However, we had to abort the experiment as it did not complete within six hours.

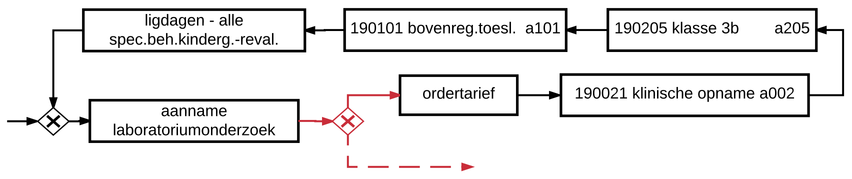
ligdagen - alle spec.beh.kinderg.-reval.
190101 bovenreg.toesl. a101
190205 klasse 3b a205
aanname laboratoriumonderzoek
ordertarief
190021 klinische opname a002

Fig. 8: Identified template for Drift 1 in BPIC 2011 log.

6 Conclusion and Future Work

We proposed a systematic online drift characterization method based on event streams. The method can characterize multiple simultaneous changes so long as they do not overlap in terms of process behavior. The strength of our method resides in the features used to encode the process behavior and its well-grounded statistical approach, that allow us to deal with highly variable processes. The collection of change templates that we use to describe a drift is based on a well-established categorization of typical change patterns. We do not claim this collection to be complete, but it can easily be extended.

We extensively evaluated our method using both highly variable synthetic logs as well as a real-life log. The results on the syntetic logs show high accuracy, low characterization delay and low time performance. And despite the lack of a ground truth to validate our findings on the real-life log, the results were supported by various observations from the log. In addition, the method outperforms state-of-the-art techniques for model-to-model comparison, in combination with automated discovery techniques, as well as techniques for log-to-log comparison.

A first avenue for future work is to provide a visual description of the identified templates as a simple and effective way to communicate the nature of the drift, as in [3]. Another avenue is to support process fragments, so as to detect more sophisticated changes that include subprocesses, overlapping changes or even nested changes. An idea is to combine colocated relations to discover process fragments, such as single-entry-single-exit fragments or local process fragments [19]. Finally, the characterization may be extended to other process aspects, such as process data and resources.

Acknowledgments. This research is partly funded by the Australian Research Council (grant DP150103356).

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Accorsi, R., Stocker, T.: Discovering workflow changes with time-based trace clustering. In: Data-Driven Process Discovery and Analysis. Springer (2012)
3. Armas-Cervantes, A., Baldan, P., Dumas, M., García-Bañuelos, L.: Behavioral comparison of process models based on canonically reduced event structures. In: BPM. Springer (2014)
4. van Beest, N.R., Dumas, M., García-Bañuelos, L., La Rosa, M.: Log delta analysis: Interpretable differencing of business process event logs. In: Proc. of BPM. Springer (2015)
5. Bose, R.P.J.C., van der Aalst, W.M.P., Zliobaite, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE Transactions on NNLS (2014)
6. Carmona, J., Gavalda, R.: Online techniques for dealing with concept drift in process mining. In: International Symposium on Intelligent Data Analysis. Springer (2012)
7. Conforti, R., Dumas, M., García-Bañuelos, L., Rosa, M.L.: BPMN miner: Automated discovery of BPMN process models with hierarchical structure. Information Systems (2016)
8. Conforti, R., La Rosa, M., ter Hofstede, A.H.: Filtering out infrequent behavior from business process event logs. IEEE Transactions on Knowledge and Data Engineering (2016)
9. Frank, E., Witten, I.H.: Using a permutation test for attribute selection in decision trees. In: International Conference on Machine Learning. Morgan Kaufmann (1998)
10. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Computing Surveys (CSUR) (2014)
11. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of ir techniques. ACM Transactions on Information Systems (TOIS) (2002)
12. Leemans, S.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs-a constructive approach. In: International Conference on Applications and Theory of Petri Nets and Concurrency. Springer (2013)
13. Maaradji, A., Dumas, M., La Rosa, M., Ostovar, A.: Fast and Accurate Business Process Drift Detection. In: Proc. of BPM (2015)
14. Martjushev, J., Bose, R.P.J.C., van der Aalst, W.M.P.: Change Point Detection and Dealing with Gradual and Multi-order Dynamics in Process Mining. In: BIR (2015)
15. de Medeiros, A.A., van Dongen, B.F., Van der Aalst, W.M.P., Weijters, A.: Process mining: Extending the α-algorithm to mine short loops. Tech. rep. (2004)
16. Menard, S.: Applied logistic regression analysis. Sage (2002)
17. Ostovar, A., Maaradji, A., Rosa, M.L., ter Hofstede, A.H.M., van Dongen, B.F.: Detecting drift from event streams of unpredictable business processes. In: ER (2016)
18. Pratt, K.B., Tschapek, G.: Visualizing concept drift. In: Proc. of the ninth ACM SIGKDD international conference on knowledge discovery and data mining. ACM (2003)
19. Tax, N., Sidorova, N., van der Aalst, W.M., Haakma, R.: Heuristic approaches for generating local process models through log projections. In: IEEE Symposium on Computational Intelligence and Data Mining (CIDM) (2016)
20. Webb, G.I., Hyde, R., Cao, H., Nguyen, H.L., Petitjean, F.: Characterizing concept drift. Data Mining and Knowledge Discovery (2016)
21. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features–enhancing flexibility in process-aware information systems. DKE (2008)