Concept Drift Detection of Event Streams Using an Adaptive Window

Marwan Hassani
Department of Mathematics and Computer Science
Eindhoven University of Technology, The Netherlands
Email: m.hassani@tue.nl

KEYWORDS
Data Stream Mining; Process Mining; Concept Drift Detection; Event Streams; Evolving Models

ABSTRACT
Process mining is an emerging data mining task of gathering valuable knowledge out of the huge collections of business operation data. Despite its relatively young age, it has successfully provided many new insights into business workflows using established data mining techniques. Recently, with the huge improvements in the technologies of sensing, collection and storing of data, a big demand for both shorter mining times and adaptive models of streaming process events arose. This initiated the field of stream process mining very recently. Drifts in the underlying concepts of the business processes are of a great interest for decision makers. One important advantage of stream process mining techniques over static ones is the ability to detect such drifts and to adapt its models accordingly. In this paper, we introduce an efficient approach that uses the collected information of an event stream miner to detect concept drifts. We use a dynamic window, which grows in size for stationary process behavior and shrinks for diverting data and thus indicating a concept drift. This adaptive window is used to build a model by focusing only on up-to-date information and discarding outdated items. Extensive experimental evaluations over real and synthetic log files show the ability of our algorithm to detect sudden drifts. We additionally show the effectiveness of our concept detection method in setting the pruning period of a recent stream mining algorithm.

INTRODUCTION
In mid to large-sized business enterprises, non-conventional types of questions arose recently due to the increasing advances of business information tracking techniques. One emerging research area is process mining, which takes methods from fields like data mining and analyzes logs of process information to derive interesting correlations between departments or actions, for instance, within the company. The resources for these mining steps are supplied in form of process logs, which contain rich information about activities. These activities are connected with at least timestamps and form common cases with other activities. The tasks for process mining include the extraction of enough knowledge from these logs to create a model, improve existing models or monitor new cases by using conformance checks (van der Aalst, 2011).

Although process mining is relatively a young research field, several approaches (Weijters et al., 2006), (van der Aalst et al., 2004), (van Dongen and van der Aalst, 2004) and (Günther and Van Der Aalst, 2007) are already existing in its literature. These works presented interesting methods by examining different features of the event logs (cf. the Related Work Section). All of them have however assumed the existence of the complete event logs and the possibility to access it as much as needed to generate, in most of them, a single final process model. This is infeasible when considering the huge increases in the size of event logs generated from modern information systems supporting business processes (Burattin et al., 2014). The proposed approaches will face serious efficiency issues with the increase in both the size and the dimensionality of the collected events (Hassani, 2015). Decision makers will lose important insights over drifting process by having merely a single final model. An important additional evolving requirement in this context is the necessity to have instant knowledge about the process model in the real time of observing the event logs (cf. Figure 1).

With these new requirements, one started to speak about event streams, streams of process models and streaming process discovery (Burattin et al., 2014). An efficient approach for process discovery from event streams was presented in (Hassani et al., 2015) by using a sequential pattern mining method. Despite its fast and accurate performance, it had some hard coded parameter settings that are not flexible to changes of observed processes. An important recent research question in the field of process mining is the concept drift of the underlying business process (Bose et al., 2011).

In this paper, we present our novel event stream concept drift detection algorithm, called StrProMCDD, that uses intermediate results of the efficient event stream miner StrProM to manage an adaptive window. It allows to focus on fewer recent data in case of a concept drift, but enlarges the window size for uniform process (cf. Figure 2). After the detection, the contained information of the window can be used to build an HeuristicNet above the Heuristics Miner (Weijters et al., 2006). More precisely, our contributions are:
1. Fast detection of concept drifts by observing event streams instead of trace streams
2. Using intermediate results of an efficient and modern
event stream miner (Hassani et al., 2015) to improve results considering concept drifts
3. Using well-established ADWIN (Bifet and Gavalda, 2007) framework and modular ensemble of reasonable distance measures, and
4. Analysis of differently performing distance functions of frequency maps for detection.

The remainder of this paper is organized as follows: the following section lists some related work. In Preliminaries section we give a formulation of the problem of stream process mining and concept drift detection. The main section describes our novel method called StrProMCDD. In the following section we show our extensive experimental evaluation of StrProMCDD using synthetic datasets with known ground truth and compound of real-world datasets. The last section concludes this paper with some future directions.

RELATED WORK

Stream Process Mining

Dealing with data streams in any field of data mining is challenging and it is still studied extensively (Aggarwal, 2007). Adopting stream data mining methods to stream process mining is usually not trivial or straightforward. The static finite process log files are substituted with potentially unbounded event streams in streaming process discovery. Each event itself can be of very complex nature and is correlated to other events in cases. Few algorithms have been developed that are able to perform process discovery in a single pass over the data. Namely, Burattin et al. developed an online adaption of the Heuristics Miner to streaming event logs (Burattin et al., 2012)(Burattin et al., 2014) by using Lossy Counting to keep track of the frequencies of activities and direct-follows relations. (Hassani et al., 2015) used an indexed prefix-tree to collect frequencies of activities and relations to achieve a small process time per event. Another streaming process discovery algorithm was developed in (Redlich et al., 2014b), and is based on the process discovery algorithm Constructs Competition Miner (Redlich et al., 2014a). In (Hassani et al., 2019), a framework and lists of the challenges and opportunities on using sequential pattern mining for online process discovery, are presented.

Concept Drift Detection

Concept Drift is a very well established phenomenon in the field of typical data mining. It was first mentioned that way in (Schlimmer and Granger, 1986). Concept Drift refers to changes in the resulting output, which is caused by a change in the input data. Over the years, many different approaches have been developed to deal with concept drift. These can be categorized into the parts of detecting a drift, identifying the sort of drift and applying to the new conditions given by the changed input. Most process mining approaches deal with stationary logs and assume that at each point in time the underlying process model will be the same. In (Bose et al., 2014, 2011), the authors transferred the concept drift phenomenon to the process mining field. They performed statistical hypothesis testing over feature vectors to deal with change point detection and localization. In (Accorsi and Stocker, 2012) a method which clusters traces in a certain time window was developed. An abstract representation as polyhedrons is used in (Carmona and Gavalda, 2012), so for a batch of new traces, their affiliation to the polyhedron is checked. A drift is detected if a significant number of traces miss that membership test. (Burattin et al., 2012, 2014) also deal with concept drift. Using the Lossy Counting approach, the model evolves by forgetting rare relations and activities. This is the first approach to handle concept drift not for a static event log, but in an incremental way. (Maaradji et al., 2015) compresses traces into fewer runs to improve the process time before performing a statistical testing over sliding temporal windows. Although it implements the adaptive window concept similar to our approach, the method is aims by doing doing that at balancing the classification accuracy with the speed of detecting the drifts. Another shift of representation is used in (Manoj et al., 2015), by using a correlation function to divide events into event classes. The approach performs hypothesis testing with the compressed data.
but, different to our approach, does not handle streaming data or applies any adaptive window concept. In (Hassani et al., 2015), a decaying mechanism is applied to keep the mined result evolving. In (Spenrath and Hassani, 2019), an ensemble-based prediction method was presented to detect bottlenecks with recurrent concept drifts.

**PRELIMINARIES AND PROBLEM DEFINITION**

First we will give an overview about stream process mining by introducing the main concepts in this section.

Process Mining tries to augment the understanding of business processes and offers tools to analyze various aspects of underlying knowledge. In this work we concentrate on the mining of process log files to produce interpretable models of running processes. A business process in this context consists of different aspects of underlying knowledge. In this work we concentrate on the mining of process log files to produce interpretable models of running processes. A business process in this context consists of different actions, which are related to each other in a certain way. These elements comprise for example decisions, single production steps and communication sequences. We call these process actions activities and their textual representation a label. When we talk about a particular instance of an activity, we speak about an event. To model certain workflow instances in a process, sequences of events are aggregated into cases. Each case consists of exactly one temporal ordered sequence of events. We store an event \( e = (c, a, t) \) as a tuple of the case identifier \( c \), the activity label \( a \) and the time-stamp \( t \). The complete space \( E \) of all events is denoted by \( E \). It is possible to add further attributes to the event data to increase the payload of information, but we will not go into this direction here. The complete process can be stored as a multiset of events. We call this collection a process log.

For convenience, we define projection functions for all of the three event attributes \( c, a, t \), so for \( e = (c, a, t) \) we define \( c(e) = c \), \( a(e) = a \) and \( t(e) = t \). Sometimes we are only interested in sequences of activities corresponding to the same case. A trace \( \sigma \) of a certain case \( c \) is the temporal ordered sequence of all events in a process log sharing the same case identifier \( c \), only projected on its activities.

We move now from static log files to streams of events. As each event has a corresponding time-stamp, we can order the whole process log by time. Formally, we introduce a mapping \( S : \mathbb{N} \rightarrow E \) to describe an event stream. For any pair of events \( e_1 \) and \( e_2 \) with \( S(m) = e_1 \) and \( S(n) = e_2 \), the mapping is a valid stream if \( t(e_1) > t(e_2) \) for \( m > n \). To simplify the notation we refer to the following string representation for event streams:

\[
S = (e_1, e_2, e_3, \ldots)
\]

We already mentioned that activities can be related to each other. For further definitions we need to introduce the directly-follows relation \( \prec \). An activity \( a \) is directly-followed by another activity \( b \), if there are two events \( e_1 = (c, a, t_1) \) and \( e_2 = (c, b, t_2) \) corresponding to the same case \( c \), correctly ordered according to their timestamps \( t_1 < t_2 \) and there is no further event \( e_3 \) with \( c(e_3) = c \) and \( t(e_1) < t(e_3) < t(e_2) \). We denote with \( |a < b| \) the number of occurrences of \( a \) directly-followed by \( b \).

Collecting most of the frequency information of these relations is sufficient to infer a process model. This is shown in (Weijters et al., 2006). The presented Heuristics Miner uses data from a static log, but the heuristics used provide the baseline procedure for some other approaches. The Heuristics Miner mines a control-flow model, which establishes connections between pairs of activities if they show a high dependency. In this context, dependency for a particular pair of activities \( a \Rightarrow b \) is increased by the number of occurrences of \( a > b \) and decreased by the occurrences of \( b > a \). The ratio of the difference of both values and the total amount of both sorts of occurrences defines the dependency value then.

\[
a \Rightarrow b = \frac{|a > b| - |b > a|}{|a > b| + |b > a|} \in [-1, 1].
\]

(1)

The Heuristics Miner uses some constraints to keep the model of good quality regarding the ability to replay most traces in the log, while not allowing too much additional behavior and remaining human-interpretable. First of all, only pairs of activities are connected which show a minimum absolute number of occurrences. Any relations of activities with a support below a user-given threshold \( \tau_{PO} \) will be discarded.

\[
|a > b| \geq \tau_{PO}
\]

(2)
In addition, and using the dependency definition above, Heuristics Miner only establishes connections if both activities exceed the also user-defined dependency threshold $\tau_{dep}$.

$$a \Rightarrow b \geq \tau_{dep}$$

To keep the model simple and comprehensible, a third constraint is used to discard connections between $a$ and $b$ if there is a big difference to the top successor considering dependencies. We define

$$post_{best}(a) = \arg\max_{b'} a \Rightarrow b'$$
as the best successor of $a$. Heuristics Miner keeps, besides this edge, only connections from $a$ to $b$, such that the dependency of $a$ and $b$ differs at most by the user-defined $\tau_{best}$ from the best dependency value. Analogously, by defining

$$pre_{best}(b) = \arg\max_{a'} a' \Rightarrow b$$

This holds for the predecessors as well. The Heuristics Miner deals with mining a static log, but provides the baseline heuristics to deal with the mined information of an event stream.

In (Burattin et al., 2014) the introduced approaches collect frequency information from a stream and apply the heuristics of Heuristics Miner to build a model then. A very well performing algorithm in this work is the Heuristics Miner with Lossy Counting with Budget algorithm (LCB). It utilizes three data structures to collect information about the activities, relations and currently active cases. Lossy Counting is a method dealing with frequent item mining in data stream mining. This approach performed really well on streams.

A second approach dealing with the same task is the StrProM algorithm in (Hassani et al., 2015). Also using the above-mentioned heuristics, instead of using three structures as the Lossy Counting with Budget algorithm, it collects the observed information in an indexed prefix-tree. This method is inspired by the SS-BE method in (Mendes et al., 2008), which is also based on the Lossy Counting algorithm. After a certain period of time, the prefix-tree will be used to extract the frequency information and pruned to height 1. This approach works also very efficient considering the process time per event.

Both approaches deal with mining streams of event data, but only use some sort of decaying to evolve the model in case of changing behavior in the input stream data. To enhance the results and to consider concept drifts, we want to use the data collected by StrProM’s prefix-tree to transform the stream of event data to a stream of frequency maps.

Handling concept drift is a very well developed area in data stream mining. A good approach to deal with changing behavior is known as ADWIN (Bifet and Gavaldà, 2007). It uses a window to observe the recent part of a stream. But instead of a sliding window, which is fixed in its size, the window in ADWIN is adaptive in size. It will increase the length for new incoming objects. Then, it checks for each possible cut of this window $W$ into two subwindows $W_0W_1$, if both parts differ significantly by their means. If so and $|\mu_{W_0} - \mu_{W_1}| > \epsilon_{cut}$ for a suitable threshold $\epsilon_{cut}$, the oldest elements are discarded as long as there is still a significant diversity in any two-partitioning.

Using this mechanic leads to larger observation windows in periods of static stream behavior without any remarkable or negligible concept drift. When concept drift occurs, the window will decrease in size and focus on the recent observations after the concept drift to quickly forget the data before the drift.

### StrProMCDD: A Concept Drift Detection Method for StrProM

StrProM in its initial version uses static pruning periods to update the counts on relations and activities. Furthermore, it utilizes a decaying mechanism to down-weight older frequency information. This value was also set at the beginning and never changed during the stream observation. Using this forgetting strategy of evolving the data to improve the model works quite well for static or slowly changing processes. But we usually do not know in advance how fast and comprehensive a change in the underlying model will be. Therefore, we are interested in an adaptive way of choosing appropriate parameter values. To derive good and useful information about the consistency of the stream, we need a mechanism to detect changes quickly. In addition we do not want to spend much effort in observing stable data, but we want to concentrate our perception on interesting periods with higher fluctuation of the observed values.

The StrProM approach produces frequency maps at regular intervals. These intervals are called pruning periods (see Algorithm 1, Line 6), as they have their origin in a prefix-tree pruning step of the SS-BE algorithm. This algorithm deals with frequent item mining in data streams. We want to use these frequency maps to detect changes, which are collected in this tree pruning step. This information can be used by a change detector mechanism from the field of concept drift detection in stream data. The change detection mechanism we use in the following is based on ADWIN. It uses an adaptive window $W$, which focuses on short intervals for highly deviating periods or increase its width in case of uniform observations. As long as elements of same behavior are observed, the window size will be increased. If the algorithm detects a deviation within the observation window, the oldest elements will be discarded until the window is again consistent.

After a pruning period has finished, we want to use the information of the newly collected frequency lists for our detection method. This does not add much computational effort to the procedure, because the frequencies are already collected by the baseline algorithm to mine a process model. We store all frequency lists in a temporal ordered list. The oldest ones will be found at the beginning, while the most recent ones are located
at the end of this list. Following ADWIN’s procedure, we iterate progressively through all cuts of this window into two partitions $W = W_0W_1$. ADWIN in its native variant builds its observation window based on a stream of real numbers. It would compare averaged values of these subwindows to find a split. If ADWIN finds a partitioning with $\mu(W_0, W_1) > \epsilon_{cut}$ for a predefined threshold $\epsilon_{cut}$, it shrinks the main window. Due to the nature of our frequency lists, we are dealing with multidimensional data here. We could easily use the average of frequencies to construct a stream of real numbers, but due to the number of possible relations, this would blur a potential concept drift quite likely. Instead, we need alternatives for the mean value $\mu$ and use other distance functions. To make things easy, we only consider distances with a normalized range of $[0, 1]$. $\delta(W_0, W_1) = 0$ means that there is no difference, $\delta(W_0, W_1) = 1$ implies the highest costs to transform one element into the other. This allows an easy exchange of distance measures to focus on different aspects, which could indicate a drift. According to ADWIN, our method will discard items from the left of the observation window in case of exceeding the cutting threshold $\delta(W_0, W_1) \leq \epsilon_{cut}$ for an appropriate binary list-distance function $\delta$ (Algorithm 1, Line 15). Figure 3 gives an illustration of StrProM with ADWIN. For a pseudo-code implementation, take a look on Algorithm 1. Algorithm 2 shows the application of the presented concept drift detector to the Online Heuristics Miner.

To use this approach we have to find a suitable threshold value and distance functions $\delta$. The first

---

**Algorithm 1 StrProM CD Detector**

**Require**: $S$: event stream;

1: $W$: adaptive window
2: $\epsilon_{cut}$: cutting threshold
3: Initialize StrProM[T:indexed prefix-tree]
4: loop
5: $T \leftarrow$ observeStream()
6: if pruning step then
7: $FrequencyList \leftarrow collectTreeData(T)$
8: pruneTree()
9: $W \leftarrow W + FrequencyList$
10: if $W$.size $> maxWindow$.Size then
11: $w = W - W$.first
12: end if
13: while $W$ shrinks do
14: for all $W_0W_1 \leftarrow W$ do
15: if $\delta(W_0, W_1) > \epsilon_{cut}$ then
16: $W = W - W$.first
17: end if
18: end for
19: end while
20: if size of $W$ decreased significantly then
21: triggerCDAlarm()
22: end if
23: end if
24: end loop
naive approach to determine a distance between two frequency lists takes both the activity and relation frequency directly into account. The distances between corresponding pairs of activities and relations are determined and the average differences yield our first two distance values $\delta_{act}$ and $\delta_{rel}$. Related to the original StrProM approach, which uses the metrics of Heuristics Miner to derive a model, we utilize the dependency for the next distances. A change in the data behavior should be represented as changes in these metrics. To derive a distance function, we determine for every relation the two different dependency values based on both subwindows $W_0$ and $W_1$. We collect the differences between the same pairs of relations as in the previous distance function. The mean value of these differences is then noted as $\delta_{dep}$. $\delta_{dep}(W_0, W_1) = \frac{1}{|W_0 \cap W_1|} \sum_{(a,b) \in W_0 \cap W_1} |\text{dep}^{W_0}(a,b) - \text{dep}^{W_1}(a,b)|$. In fact, $\delta_{dep}$ is very data oriented, but in the case that we need a more model-aware distance, we use the edge distance. To achieve this goal, we build HeuristicNets by using the dependency values and calculate the costs for transforming edges by counting edge deletions and insertions to transform the first Heuristic Net into the second one. By normalizing it with the total number of potential edge manipulations, this leads to our fourth distance $\delta_{edge}$. For a better performance, we recommend using a simplified construction method by only using the dependency threshold and ignoring the other constraints.

$$E_{gim}(W_0, W_1) = \{(a,b) | \text{dep}^{W_0}(a,b) < \tau_{dep} < \text{dep}^{W_1}(a,b)\}$$

$$E_{loss}(W_0, W_1) = \{(a,b) | \text{dep}^{W_1}(a,b) < \tau_{dep} < \text{dep}^{W_0}(a,b)\}$$

$$\delta_{edge}(W_0, W_1) = \frac{1}{|W_0 \cap W_1|} (\sum_{(a,b) \in W_0 \cap W_1} |E_{gim}(W_0, W_1)| + |E_{loss}(W_0, W_1)|)$$

The last two distances are inspired by the Fuzzy Miner (Günther and Van Der Aalst, 2007). The idea to determine a routing significance for every activity is a suitable addition, as this aspect is not covered by the previous distance functions. To calculate the significance of a node resp. activity, the sum of incoming and outgoing edges is identified. This number is then normalized by the number of occurrences of this activity.

$$\text{sigrouting}(a,W) = \frac{|\{(a, \cdot) \in W\}| + |\{(\cdot, a) \in W\}|}{|\{a \in W\}|}$$

We define the routing distance as the maximum of all differences of routing significances activity-wise.

$$\text{diff}_{routing}(W_0, W_1) = \max_{a \in W_0 \cup W_1} \text{sigrouting}(a,W_0) - \text{sigrouting}(a,W_1)$$

The last distance function to be presented here is the relative importance distance. Also inspired by the Fuzzy Miner, this measure is a derivation of a existing significance measure.

$$\text{relImportance}(a,b) = \frac{1}{2} \frac{\text{sig}(a,b)}{\sum_{x \in A} \text{sig}(a,x)} + \frac{1}{2} \frac{\text{sig}(a,b)}{\sum_{x \in A} \text{sig}(x,b)}$$

The relative importance distance is, analogously to the routing distance, the maximum of differences of relative importance values for a certain significance value. As significance, we use the relation frequency here.

The usage of distance functions is rather modular, so one can think of other distance measures to focus on specific aspects. For example instead of considering the average dependency difference, it is possible to reduce it to the mean of the $k$ most diverging activity pairs. But we have to keep in mind that more complex distance functions greatly increase the processing time at every pruning step.

After one or a whole ensemble of distance functions and a suitable weighting is chosen, we need a reasonable cutting threshold. Typically, when the task is to detect concept drifts, we are dealing with streams showing mainly static behavior and only minor amounts of concept drifts. If the number of drifts is such high that we have a certain expectation of a drift in a small interval, we should preferably use a passive evolving method to decay the collected data. The computational effort of detecting drifts in a stream crowded by drifts is not meaningful. Considering this, we do not expect to find a concept drift at the beginning of a stream. We can use this assumption to use few pruning periods to determine the distances in the possible window cuts. Until the adaptive window reaches its capacity, we should
have enough distances to estimate a good threshold by using the maximum of all these values.

The adaptive window will grow and shrink during the stream observation. To build the HeuristicsNet model, we only use the contained items. All discarded information is ignored for the model construction. This leads to a model based on the most recent data and evolving over time while the window is following the stream.

The computational effort to find a cut in the window and deletion of elements until no other cut can be found is rather high by using this approach. For increasing window sizes, the number of possible cuts, which have to be checked, grows alike. In periods with no concept drift or at least very little variation in the relation counts, the adaptive window grows tremendously large, if no cut has to be done. So as a first step to decrease computational costs, we introduce an upper bound for the window size (Algorithm 1, Line 10). In our experiments, a maximum size of 10 pruning periods as a window size achieved good results by balancing computational effort with the amount of information to yield a sound model.

In the introduced algorithm, the frequency maps from the pruning steps are stored in the adaptive window. Then the algorithm looks for possible cuts. As a second step we recommend to compute the dependency values for each frequency map and storing this dependency list in the adaptive window. This reduces the number of dependency computations of the lists. For a window size of \( n \) lists, we have to apply \( n \) dependency computations instead of \( n^2 \) if we compute the two-partitions for every possible cut. For agglomerating sublists, we just use the mean of all contained dependency lists. This yields obviously not the same distances, but it can still be used to determine cuts in the adaptive window with another slightly modified cut threshold.

Complexity

The presented detection method for sudden drifts in event streams needs only constant computational time. To show this, let us consider a worst-case for the pruning step. In the worst case, we have to check every potential cut for a derivation. For a predefined observation window size \( n \), this yields \( n - 1 \) iterations. In each step, a distance function has to be computed. We do not want to discuss every function here, because their effort is quite similar. Instead, let us assume to only use the dependency distance. In this case, we have to determine all dependency values. For \( |A| \) activities, there are at most \( |A|^2 \) relations. Calculating the differences takes \( 2 \times |A|^2 \) steps of basic calculations, plus dividing by the size of the whole relation set. As the number of activities can be determined in advance, the complexity of the dependency distance calculation is in \( O(1) \), irrelevant of its parameters. Then, the effort is also constant for the whole window adjustment step.

EXPERIMENTAL EVALUATION

We evaluated our drift-sensitive StrProM approach on both synthetic and real-world datasets to measure its ability of detecting concept drifts. Only sudden drifts are considered here. To recognize more complex drifts, namely gradual, recurring and incremental drifts, a suitable way to store sufficient information about the drifts is needed. There has to be a comparison with previous drift detections to evaluate the character of a drift. This will be addressed in future works. To the best of our knowledge, there is no available similar approach, which also mines a stream of events in a single pass and detects drifts by analyzing single events instead of traces.

First we want to discuss the synthetic dataset experiments. We started with four artificially generated different processes. With the Process Log Generator (Burattin and Sperduti, 2010), we produced logs for all processes and generated logs. We merged them in a pseudo-chaotic order, yielding a log with over 100,000 events. We streamed the complete log and monitored the size of the adaptive window as shown in Figure 4. The thresholds were set by evaluating the first process and finding the maximum value, such that no drift was detected. The pruning period was 800 events with a case list size of 2000 cases.

The most obvious conclusion is the poor performance of the relation frequency distance. It does not detect anything here. The dependency and edge distances perform almost identical, recognizing all 15 artificial drifts. They also detect some drifts within process 3, which are false positives. The activity frequency distance misses two times a drift for process 2 and also detects the false drift within process 3. The routing distance works best on this dataset. It has a perfect detection rate and detects no false positives. The relative importance distance misses 7 drifts, but has no false alarms. On the contrary, some alarms are triggered with a delay. We compare the delay in Figure 5, as well as the detection accuracy.

As the relative importance distance had some problems to detect the drifts for this dataset accurately, it is difficult to compare the detection times with the other distance functions. The relation frequency distance is out of discussion as well. The routing distance performed really well on this dataset, regarding detection rate and delay time. Dependency distance and edge distance are closely related by definition, so their results are quite similar as well. Their performance is also capable of handling concept drifts in a larger stream scenario. The activity distance is mediocre, but can be used in an ensemble of distances with a lower weight probably.

For real-world datasets, it is very difficult to find a suitable publicly available dataset with a known ground truth. To overcome this issue we used the datasets from the BPI Challenge 2015. It is a collection of five datasets1, provided by five Dutch municipalities. It contains building permit applications about four years

---

1Dataset can be downloaded here: https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1
Fig. 4. Window size of the adaptive window during the stream of 4 merged synthetic logs. The whole stream consists of 100000 events. First row: dependency and edge distance; second row: activity and relation frequency distance; third row: routing and relative importance distance.

Fig. 5. Accuracy and mean delay for StrProMCDD on the artificial dataset. For accuracy, we determined the ratio of correctly detected drifts (precision) as well as the ratio of detected drifts, that are positive observations (recall). Both values are used to determine the F-score. The mean delay of the detection is counted in pruning periods. As every pruning period consists of 800 events, it can be easily transformed into event counts.

and is publicly available. We merged all five datasets, yielding a complete dataset with four reliable concept drifts. It contains about 260000 events. The observation window size is shown in Figure 6.

While the activity frequency distance performed mediocre, the relation frequency is even worse. In this case, it detected nothing for thresholds greater than 0.1. This is caused by the high number of activities and potential relations between them. The observed items in the stream pruning periods vary a lot. This leads to very low counts for each different element. Therefore, the differences between two windows is expectedly also small. The relative importance is very sensitive and reacts to small changes with a big impact on the resulting data. So it is fluctuating a lot here.

The dependency distance detected the first three transitions between the municipalities, but missed the last one. Instead it reacted to a deviation in the data of the third municipality. The edge distance detects all four shifts, but the first and last one only slightly. Instead it reveals some other drifts with higher peak. The routing distance also worked quite well by detecting ev-
ery process shift. The first drift is again very weak, but it showed similar observations as both distances before.

It is not meaningful to rank the given distance functions in general. All of them are focusing on different aspects of the data. Therefore, they have a certain importance for a particular dataset, but this can be different for another set of business data. Although the weights have to be redefined for other datasets, one can construct a derived weighted distance function based on these basic functions. For the given dataset, we estimated a weight vector $w = (0.3, 0.2, 0.15, 0.0, 0.3, 0.05)$. The resulting window size can be seen in the last measurement in Figure 6. Using this method yields a very good detection of the known drifts and probably unknown drifts. Of course, this hard-coded, non intuitive setting of these weights is not assumed to hold for other datasets. It serves merely as a hint for the potential of such method after observing the outputs of different functions. The effect of the evolving drifts on these weights must also be investigated. Due to the absence of a ground truth about the individual municipality datasets, we can not give any evaluation about false positives here.

CONCLUSION

In this paper, we introduce $StrPtoMCDD$, an efficient approach that uses the collected information of an event stream miner to detect concept drifts. We used a dynamic window, which grows in size for stationary process behavior and shrinks for diverting data and thus indicating a concept drift. This adaptive window is used to build a model by focusing only on up-to-date information and discarding outdated items. Extensive experimental evaluations over real and synthetic log files highlighted the ability of our algorithm to detect different kinds of drifts.

In the future, we would like to further benefit from the detection of drifts in the more complicated and realistic scenario where interval-based events are considered (Lu et al., 2017). In the case of overlapping, more relations are considered between these temporal events appear. This makes the task of concept drift even more challenging and interesting. Additionally, we would like to investigate the implementation of online detection of drifts using $StrPtoMCDD$ for observing deviations in streaming conformance checking applications (van Zelst et al., 2017) and anytime stream classification (Kraener et al., 2012). Additionally, we would like to get the most suitable threshold automatically that also adapts over time to the underlying process distribution.

Acknowledgments

The author has received funding within the BPR4GDPR project from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 787149. The author would like to thank Prof. Thomas Seidl for the useful discussions through various phases of this paper and Florian Richter for the implementation of some parts of $StrPtoMCDD$.

References


Fig. 6. The observation window sizes for all 6 distance functions and the combined distance function are shown for the complete stream. The window size has a limit of 10 frequency lists, so it cannot exceed this size. The dotted lines indicate the transitions from one dataset to the next one, so these are known concept drifts.

L. Mendes, B. Ding, and J. Han. Stream sequential pattern mining with precise error bounds. In ICDM ’08, pages 941–946, 2008.

Marwan Hassani Marwan Hassani is an assistant professor at the Process Analytics group in the department of computer science at Eindhoven University of Technology (TU/e), The Netherlands. Previous to that, he worked as a postdoc at the Data Management and Data Exploration Group at RWTH Aachen University, Germany. His research areas are: data mining, data science, process mining and big data analytics. Current interests are: stream data mining, real-time process mining, sequential pattern mining, privacy-aware process mining, subspace clustering and evolving graph mining. Marwan is using customer journey optimization as a use case in several running projects. He has co-authored more than 60 scientific publications and is acting as an organizer and a regular reviewer in several international conferences and journals.