Predicting Business Process Bottlenecks In Online Events Streams Under Concept Drifts

Citation for published version (APA):

Spennath, Y., & Hassani, M. (2020). Predicting Business Process Bottlenecks In Online Events Streams Under Concept Drifts. In M. Steglich, C. Muller, G. Neumann, & M. Walther (Eds.), *Proceedings of European Council for Modelling and Simulation (ECMS) 2020* (pp. 190-196). (Proceedings European Council for Modelling and Simulation; Vol. 34, No. 1). European Council for Modeling and Simulation. <https://doi.org/10.7148/2020-0190>

DOI:

[10.7148/2020-0190](https://doi.org/10.7148/2020-0190)

Document status and date:

Published: 01/06/2020

Document Version:

Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Predicting Business Process Bottlenecks in Online Events Streams under Concept Drifts

Yorick Spenrath

Marwan Hassani

Department of Mathematics and Computer Science
Eindhoven University of Technology, The Netherlands
y.spenrath@tue.nl m.hassani@tue.nl

KEYWORDS

Gradual Concept Drift; Recurrent Concept Drift; Concept Drift Detection; Business Process Bottlenecks; Event Streams

ABSTRACT

Process performance analysis is an important sub-task of process mining that aims at optimizing the discovered process models. In this paper we focus on improving process throughput by predicting congestions in the process execution (bottlenecks). We discuss an ongoing work on incorporating gradual and seasonal concept drift in this bottleneck prediction. In the field of process mining, we develop a method of predicting whether and which bottleneck will likely appear based on data known before a case starts. We introduce GRAHOF, a Gradual and Recurrent Adaptive Hoeffding Option Forest approach, which adapts to gradual and seasonal concept drifts when predicting bottlenecks of business processes in an online setting. We evaluate the parameters involved in GRAHOF using a synthetic event stream and a real-world event log.

INTRODUCTION

Concept drift expresses the occurrence of a shift in relations between input and output data over time. The challenge with concept drift is that it is difficult to assess, detect, and adapt to such evolutions. One method of solving concept drift is to retrain models periodically. This method poses two major disadvantages. First, this reduces the quantity of available data, which is a big issue when the number of data points per time unit is low, or when dealing with unbalanced datasets. The combination of these two causes over-fitting of the model on larger classes. Second, retraining a model with new data inherently discards former data, which could remove valuable information on seasonality.

In this paper, we present an ongoing work on predicting bottlenecks in business processes under (gradual and seasonal) concept drift. Making predictions about process execution is one form of predictive process mining. In predictive process mining two type of concept drift exist. First there can be a drift in predictions (how the relations between process execution information and bottlenecks evolve) or a drift in the

process itself (how the process execution model evolves over time), see for example [11, 16]. In this paper we are concerned with concept drift of the former, assuming the process execution model to be constant.

The advantage of being able to predict bottlenecks, whether in general or under concept drift, is that it allows process managers to make informed decisions about the process execution, as well as early identification of possible delays. Figure 1 shows the example process used in this paper. Mechanics in an installations services company do repair work. The financial administration of the company is then responsible for invoicing the repairs. This process consists of four activities *Check Repairs*, *Collection Information*, *Make Invoice*, and *Send Invoice*. To increase the throughput of the example process (thus minimising the time between the repair and sending the invoice), we aim to predict if and where in the process bottlenecks occur. We make these predictions based on information available after repairs. This is data on the size of the company the repair was at, the distance the mechanic travelled, and the type of repair executed.

In this work, the goal is to make a prediction of the bottleneck *before* the case starts. In particular, we do not incorporate process execution information. This paper contributes as an extension to the previous work in [18]. The extended method, GRAHOF, is capable of handling *online* analysis of event streams (rather than post-mortem event logs). The extension uses incremental learners to adapt to gradual concept drift. The specific model, the Adaptive Hoeffding Option Tree [2], has the advantage of being very fast by design, which allows the online analysis of large event streams. The full implementation of GRAHOF, the simulation, and the experiments, including a visualisation, is available under:

<https://github.com/yorick-spenrath/GRAHOF>.

The remainder of this paper is organised as follows. The following section introduces related work. The third section sets out some necessary preliminaries. The fourth section provides the algorithm used to assess bottlenecks. The fifth Section presents GRAHOF, the streaming algorithm used to analyse event streams while adapting to gradual and recurrent drift. The sixth section describes a simulation of a synthetic event stream followed by an experimental evaluation of

GRAHOF using that dataset and then using a real-world dataset. Finally, the last section concludes the paper and outlines future work.

RELATED WORK

This section presents related work on Stream Process Mining and on Concept Drift Adaption.

Predictive and Stream Process Mining

Predictive process analytics is becoming increasingly popular in literature. Opposed to classical process mining, predictive process mining aims to assist by making predictions about future behaviour. In a study on handling concept drifts in business processes, [15] investigates the different solutions to adapt to concept drift in predicting process outcomes. The authors use information on both process executing (i.e. past events in a case), and on involved data (case data, and data that is learned during process execution). In particular, they show the effectiveness of incremental machine learning models in handling different types of concept drift. In [5, 6], the authors consider drift detection in processes. Contrary to the drift looked at in this paper, their focus is on drift detection in the process itself, rather than in the relations between process data and for example bottlenecks. They do so by extracting information about process execution from event logs, and defining features that allow them to detect drift in the underlying process. They argue that their technique can be used successfully on data streams as well. Finally, [8] also develops a drift detection method on event streams. The idea behind their method is to abstractly estimate the concept, by spanning a subspace containing the frequency of activities in traces of the currently looked at event log. Part of the traces will be part of this abstract space, and as such a measure of how well the space represents the event log can be derived. After stability of this measure, new traces are evaluated, if as such the measure considerably changes, drift is assumed to have taken place.

In recent years, analysing events on-line instead of post-mortem has become increasingly important [13]. The authors of [21] discuss an online conformance checking technique. While receiving events from an event stream, they verify whether the cases follow the intended business process model to observe deviations the moment they occur. The works in [12] and [7] discuss a different part of process mining on event streams: discovery. They consider analysing event streams, but refrain from analysing each event. Instead, they keep a maximum to their memory size, and prune old events and results when needed, periodically using all information in memory to discover process models at that time. The authors of the latter show that this allows concept drift detection as well. While the concept drift detection effect of the former has been discussed in [11].

This paper contributes to the existing literature by analysing and predicting bottlenecks in the processes under concept drift. It can help process managers iden-

tify potential process delays early, even under seasonal fluctuations.

Concept Drift Detection and Adaptation

Concept drift is the phenomenon where relations between input (features) and output (labels in the case of the experiments in this paper) change over time, because the underlying models and/or distributions change, see for example [9]. The authors note that this applies to a variety of research areas, including process mining. In our use case, the predictions on bottlenecks will become less accurate if no adaptations to concept drift are made. As such, the effectiveness in preventing bottlenecks is reduced.

Most existing literature focuses on *detecting* concept drift. Detection is a first step, adapting to it is the next step. Literature has a wide variety of detection techniques. The authors of [1] discuss a technique based on changes in classification accuracy to detect abrupt and gradual concept drift. They check how the difference in classification error progresses over time. If the difference is far smaller than a previously measured distance in the same concept, they suppose the occurrence of concept drift. A different technique is to consider the importance of features over time. In [4], the authors present a technique which can adapt to concept drift. They train ‘experts’, each expert takes a specific combination of feature values and recommends the label based on the latest labels for that specific combination. All experts are then weighted, where incorrect experts lose their weight over time, whereas correct experts increase in weight. One way to adapt to concept drift is by simply retraining the model whenever a drift in concept is detected. A disadvantage of this technique occurs if the drift is periodic, i.e. a drift of concept is detected, but the previous concept may still be of use at a later point in time, just not right after the detected drift. In [22], the authors do so by training a new model for every specified number of data points, creating an ensemble of different models. Each model is then weighted by the accuracy score on recent datasets. Challenges in seasonal concept drifts have been extensively addressed in literature as well. The method in [14] keeps an ensemble of models, and selects the model that is expected to be best adapted to the current effects concept drift. Finally, [17] also considers an ensemble of classifiers. Batches of data are tested against this ensemble. If no classifier can explain the new batch, the ensemble is weighted based on how each model explains the batch. If this weighted ensemble is again unable to adequately explain the new batch, the batch is considered as a new concept, and a new model is trained.

PRELIMINARIES

We define an *event* e as a tuple $(e.cid, e.act, e.time)$, stating that the activity $e.act \in \mathcal{A}$ was completed for the case with id $e.cid$ at time $e.time \in \mathbb{R}_+$. An *event log* $\mathcal{L}$ is a set of such events. A case c is a sequence of events $\langle c(0), c(1), \dots, c(|c| - 1) \rangle$. Cases further have

Fig. 1. An installation services company has four activities that are part of the invoicing process, starting after repair activities complete. We use this process as a running example.

a unique id $c.cid$. The trace or variant of c , is the sequence of activities in c , i.e.

$$variant(c) = \langle c(0).act, c(1).act, \dots, c(|c| - 1).act \rangle$$

The total duration of a case is defined as

$$duration(c) = c(|c| - 1).time - c(0).time$$

Finally, $c.start$ denotes the start of the case, i.e. $c(0).time$. We will write $c \in \mathcal{L}$ if there is at least an event $e \in \mathcal{L}$ for which $e.cid = c.cid$. In other words, we will represent the elements of $\mathcal{L}$ as cases or events. From a case c we can derive the duration of each activity in c from the events in c . In particular, for $i = 1 \dots |c| - 1$, we have that activity $c(i).act$ took $c(i).time - c(i - 1).time$ time. As such we cannot compute the duration of the first activity in a case; this is generally not a problem since this activity only indicates the start of the case.

Cases further have feature data $c.x$. This feature data is any case information that is available at the start of the case, which will be used to predict bottlenecks. An event stream $\mathcal{S}$ is a (possibly infinite) set of events. We assume that events are chronological, i.e. for any event e that we receive from $\mathcal{S}$ and for any event e' we received from $\mathcal{S}$ before e , we have that $e'.time \leq e.time$. We define $\mathcal{S}.next()$ to retrieve the next event in $\mathcal{S}$. We further assume that, given an event e , the predicate $\mathcal{S}.end(e)$ tells whether an event e was the last event of the case. In our example of Figure 1, we could have $\mathcal{S}.end(e) \Leftrightarrow e.act = Send\ Invoice$. Finally, we assume that the first event we receive for a case contains the feature data of that case. As such we refer to such events as a *start event*. A start event hence contains all required information to start a case (feature data, id, and start time).

BOTTLENECK COMPUTATION

In this Section we present how we decide on the bottlenecks in a case. The way in which we define these is that we point out a *single* activity to be *the* bottleneck. The idea is that this activity was responsible for a considerable delay compared to other cases that started around the same time. Addressing this specific (predicted) bottleneck should be seen as the best opportunity on improving the throughput of a case.

First, we split the event log in $\mathcal{L}_{short}$ and $\mathcal{L}_{long}$. $\mathcal{L}_{short}$ contains all cases that are completed within a given time duration, set by domain experts. Let this duration be d , i.e. $\mathcal{L}_{short} = \{c \in \mathcal{L} | duration(c) \leq d\}$, and $\mathcal{L}_{long} = \{c \in \mathcal{L} | duration(c) > d\}$. We next compare cases in $\mathcal{L}_{short}$ with cases in $\mathcal{L}_{long}$. We do so per

variant. Let there be N variants in $\mathcal{L}$, and let V_i be these variants, $i \in 1 \dots N$. We partition $\mathcal{L}_{short}$ into sets $\mathcal{L}_{short}^i = \{c \in \mathcal{L}_{short} | variant(c) = V_i\}$. We similarly split $\mathcal{L}_{long}$ into sets $\mathcal{L}_{long}^i$.

For each variant V_i , we use $\mathcal{L}_{short}^i$ as a ‘benchmark’ for how long events should take, and then assess the duration of events in $\mathcal{L}_{long}^i$. We require that bottlenecks are both resolvable and relevant. We formalise ‘Resolvability’ by requiring that for an activity to be a bottleneck, the duration should be longer than the average for that activity in $\mathcal{L}_{short}^i$. We formalise ‘Relevance’ by requiring that the event takes at least α part of the total duration of the case. $\alpha \in [0, 1]$ is a parameter, also set by domain experts. Setting it too low will remove the requirement, setting it too high make no event meet it. Of all events in a case meeting both requirements, we select the one that deviates the most from the benchmark derived from $\mathcal{L}_{short}^i$.

Handling the cases per variant has multiple advantages. First, the technique is not limited by more complex process structures, such as loops and choices. Second, the algorithm is robust against cases that do not conform the (expected) process. Finally, cases that deviate from the process (they do not conform) are compared to cases that deviate from the process in the same way. We refer to our previous work [18] for a detailed formalisation of the above.

OUR APPROACH: GRAHOF

In this Section we introduce **GRAHOF**. **GRAHOF** is an acronym for Gradual and Recurrent Adaptive Hoeffding Option Forest. Similar to the work presented in [18], **GRAHOF** uses an ensemble to adapt to gradual and recurrent concept drift. Different to it, **GRAHOF** requires incremental models, and uses them to adapt to gradual drift. Our approach does not require a specific incremental learner, we implemented it using the Adaptive Hoeffding Option Tree (AHOT) [2] as incremental learner. The choice of AHOT is inspired by [15], where they use the learner in predicting process outcomes. In this paper we focus on how **GRAHOF** uses the AHOT as incremental learner. We hence refrain from the details of AHOT and we use the default settings of AHOT in [3].

In the following, we formalise the execution algorithm, provide the model updating and creation, and discuss the making of predictions.

Main GRAHOF execution

The full method is presented in Algorithm 1. In the following, the line numbers refer to that Algorithm.

We define a *batch* as follows:

Algorithm 1 GRAHOF on an Event Stream $\mathcal{S}$

```
1:  $M \leftarrow \emptyset$ 
2:  $B^* \leftarrow \text{Batch}(0, S) // S \text{ is the batch size}$ 
3:  $\mathcal{B} \leftarrow \{B^*\}$ 
4: while True do
5:    $e \leftarrow \mathcal{S}.\text{next}()$ 
6:   if  $e$  is start event then
7:     while  $B^*.t_e \leq e.time$  do
8:        $B^* \leftarrow \text{Batch}(B^*.t_0 + S, B^*.t_e + S)$ 
9:        $\mathcal{B} \leftarrow \mathcal{B} \cup \{B^*\}$ 
10:    end while
11:     $c \leftarrow \text{new case with id } e.cid$ 
12:    Add  $c$  to  $B^*$ 
13:    Predict  $c.y_{pred}$  using  $c.x, M$  and Equation 1
14:  else
15:    Find batch  $B$  and case  $c$  with  $c \in B \wedge c.cid = e.cid$ 
16:    Add  $e$  to  $B$ 
17:    if  $\mathcal{S}.\text{end}(e)$  then
18:       $c.closed = \text{True}$ 
19:      if  $\forall c.cases \in B : c.closed \wedge B \neq B^*$  then
20:         $\text{close}(B)$ 
21:      end if
22:    end if
23:  end if
24: end while
```

Definition 1: A batch B is an extension to an event log, with cases that start in a specific interval. A batch has the following properties:

- A batch is initiated as $\text{Batch}(t_0, t_e)$, with $t_0, t_e \in \mathbb{R}_+$ and $t_e - t_0 = S$. The batch size $S \in \mathbb{R}_+$ is a parameter for GRAHOF.
- A batch contains a set of cases $B.cases$ that start on or after $B.t_0$ and before $B.t_e$: $\forall c \in B.cases : c.start \in [B.t_0, B.t_e)$.

We create the first batch at the start of the event stream. This first batch has interval $[0, S)$ (Line 2). Since we assume the stream to produce events chronologically, we create a new batch whenever a case starts outside of the interval of the current batch. In the remainder, we use B^* to refer to the most recently created batch. (Lines 2, 8).

We receive the events from the event stream one-by-one. For any received event e , we first decide if that event is a start event. If so, we create a new case (Line 11), and add the case to the current batch (or create a new batch if the case start is after the current batch end). We then make the prediction (see the subsection on making predictions) (Line 13). If the event is not a start event, then the case the event belongs is already added to a batch (as per Line 12). We as such add it (Line 16) to the corresponding batch (Line 15). If e is the final activity of a case, we mark the case as complete (Line 18).

Whenever we complete a case we check if 1) all cases in B are closed, and 2) whether $B \neq B^*$ (Line 19). If both predicates hold, then we neither expect more events for existing cases, nor new cases for B . The

batch is therefore complete and we can close it. Closing the batch ($\text{close}(B)$) consists of computing the true labels (See the section on bottleneck computation), updating or existing an AHOT (See the section on updating models), and removing B from $\mathcal{B}$. The batch B is therefore no longer relevant after it is used for training.

Algorithm 2 Finding the best model in GRAHOF

Ensure: For a dataset $\mathbb{ID}$ we update an existing model or create a new one

```
1: Get a balanced, stratified 80%  $\mathbb{ID}_{\text{train}}$  and 20%  $\mathbb{ID}_{\text{test}}$  from  $\mathbb{ID}$ 
2:  $M' \leftarrow M$ 
3: while  $M' \neq \emptyset$  do
4:    $m^* = \arg \max_{m \in M'} f(\mathbb{ID}_{\text{test}}, m)$ 
5:   if  $f(\mathbb{ID}_{\text{test}}, m^*) \geq \phi f_{m^*}$  then
6:      $f_{m^*} \leftarrow \phi f_{m^*} + (1 - \phi) f(\mathbb{ID}_{\text{test}}, m^*)$ 
7:     Update  $m^*$  with  $\mathbb{ID}_{\text{train}}$ 
8:     Add  $\mu_B$  to  $m^*.\mu_{\text{train}}$ 
9:   return
10:  else
11:     $M' = M' \setminus \{m^*\}$ 
12:  end if
13: end while
14:  $m^* \leftarrow \text{new AHOT}$ 
15: Update  $m^*$  with  $\mathbb{ID}_{\text{train}}$ 
16:  $f_{m^*} = f(\mathbb{ID}_{\text{test}}, m^*)$ 
17: Add  $\mu_B$  to  $m^*.\mu_{\text{train}}$ 
18:  $M \leftarrow M \cup \{m^*\}$ 
```

Updating Models

In the following, $F_1(\mathbb{ID}, m)$ denotes the F_1 score of a model m on a dataset $\mathbb{ID}$. Whenever all cases that belong to a batch are complete, we compute the bottleneck labels as per the section on bottleneck computation for the cases in the batch. This creates a dataset $\mathbb{ID}$. There is a possibility that one of the activities is labelled as a bottleneck more often than others, i.e. that we are dealing with an unbalanced dataset. We therefore apply a k -medoid reduction on $\mathbb{ID}$, to reduce the size of overrepresented to at most twice the size of the smallest class. We split the resulting balanced dataset into a stratified $\mathbb{ID}_{\text{train}}$ and $\mathbb{ID}_{\text{test}}$. Initially, we have no models (the ensemble, M , is empty), and we create a new AHOT m , which we proceed to train with $\mathbb{ID}_{\text{train}}$. We store $f_m = F_1(\mathbb{ID}_{\text{test}}, m)$, and add m to M . After the first model creation M is no longer empty, so we first verify if there is a model m^* that can be improved with $\mathbb{ID}$. We therefore first evaluate all existing models starting with the model $m^* \in M$ that has the highest F_1 on $\mathbb{ID}_{\text{test}}$. If $F_1(\mathbb{ID}_{\text{test}}, m^*)$ is at least $\phi \cdot f_{m^*}$, then we update f_{m^*} to $\phi \cdot f_{m^*} + (1 - \phi) \cdot F_1(\mathbb{ID}_{\text{test}}, m)$ and train m^* with $\mathbb{ID}_{\text{train}}$. $\phi \in [0, 1]$ is a parameter for GRAHOF. If m^* does not meet the requirement, we try the model with the next-best score, and so on. If none of the models meet the requirement, we create a new AHOT as above. The procedure of updating a model is presented separately in Algorithm 2.

Apart from setting or updating f_m , we also update

a set $m.periods$. This set tracks all of the batches that have been used to train m . More specifically, $m.periods$ is initialised as $\emptyset$, and $[B.t_0, B.t_e)$ is added to it for every batch B that has been used to update m . This set is used when computing the weights of the model for a new batch.

Making predictions

The weight a model receives for a batch depends on how many seasonally-similar training periods there are in the model. Formally, if we make a prediction at time t , the weight of model m , will be¹

$$|\{t' | t' \in m.periods \wedge t - t' \equiv_{\rho} 0\}| \quad (1)$$

$\rho \in \mathbb{R}_+$ is a parameter for **GRAHOF**. In particular, we expect the period of the recurrent drift to be ρ , which would equal year for seasonal drift.

When making a prediction, each model computes its predicted probability for each bottleneck label; the predicted label is then the label with the highest weighted probability. This label can be used during operations by process managers, and is later evaluated against the true label computed by the bottleneck algorithm (Line 13).

EXPERIMENTAL EVALUATION

We simulate an event log based on the running example. We define three features, the repair *Type* (Electric, Water, Gas, Mechanical), the travelled *Distance*, and the client company *Size*. We simulate 250 cases per *Type* per month for 57 years.

The *Dist* and *Size* values are drawn from a discrete uniform distribution from 1 to 100. The values of *Dist*, *Size*, and *Type* determine if and where a bottleneck will occur in a case.

Gradual drift The idea behind the gradual concept drift is as follows: we take about $\frac{1}{5}$ of the combinations of values for *Dist* and *Size*. If a given case has one of these combinations, the case will have no bottleneck, otherwise it will. Formally, let $A(t) \subseteq S_{Dist} \times S_{Size}$, with $S_{Dist} = S_{Size} = 1 \dots 100$. A case c with *Dist* and *Size* has a bottleneck if and only if $(Dist, Size) \in A(t)$. $A(t)$ is a time-dependent subset of $S_{Dist} \times S_{Size}$, and is visualised in Figure 2.

Seasonal drift We simulate recurrent concept drift by defining four concepts, each of which occurs during a different quarter. In the first quarter each year, the repair type *Electric* causes a bottleneck in activity *Check Repair*, *Water* in *Collection Information*, *Gas* in *Make Invoice*, and *Mechanical* in *Send Invoice*. We then rotate this mapping each quarter, as depicted in Table I. We make this switch over a period of two weeks, using the sigmoid function presented in [2].

Results on the Simulated Dataset

In this Section we evaluate **GRAHOF**, as a verification of the method. We do so using the simulation previously described. We analyse the effect of ϕ and S

Qt.	E	W	G	M
1	CR	CI	MI	SI
2	SI	CR	CI	MI
3	MI	SI	CR	CI
4	CI	MI	SI	CR

TABLE I: Bottlenecks as determined by topic and quarter of the start of the case. For example, we have that a *(W)ater* repair causes a bottleneck in the activity *(M)ake (I)nvoice* in the fourth quarter.

separately. When analysing one parameter, we set the other parameters to $\phi = 0.75$ or $S = 3$ months (see out main algorithm section for details.). The results presented are the averages over 10 simulations.

The effect of ϕ Figure 3 shows that there is a trade-off in ϕ . For lower values, models can be updated with data that the model can not properly explain. In fact, only a single model is used for $\phi \leq 0.2$. For $\phi > 0.75$, F_1 decreases as a result of the increase in the number of models. Because models become harder to update, more models are created. As such, older models (which due to gradual concept drift are no longer updated) have a negative influence on the prediction, while newer models have seen fewer datapoints.

The effect of S Figure 4 shows the results for different values of S . As expected, $S = 3$ has a higher F_1 . For larger batch sizes, models are created and updated on multiple concepts. As such, **GRAHOF** cannot properly differentiate between the concepts, and as such unable to properly predict the correct bottleneck labels. For $S = 2$, there are two models that are trained on multiple (2) seasonal concepts, whereas other models correctly get a single concept. This slightly decreases the F_1 . $S = 1$ also has a slightly lower F_1 than $S = 3$. This is likely caused by the increase in the number of models.

Results on a Real-World Dataset

We applied the described method in an initial case study on a real-life event log used in the BPI Challenge 2017 [20]. The original event log contains many different variants and events, we limit ourselves to work-flow events only (removing application and offer events, see the original dataset), and limiting to the most common variant. As a result, we have a total of 6664 cases, each having three potential bottlenecks. We applied **GRAHOF** for different values of S and ϕ . The results are shown in Figure 5.

The results indicate the potential for **GRAHOF** to work on real-life event logs. Though there seem to be small differences between the values for both parameters, future work on more extensive event logs should make more conclusions about the stability of the approach with respect to the required parameters.

CONCLUSION AND FUTURE WORK

In this paper, we presented **GRAHOF**, an extension from earlier work. The analysis in previous sec-

¹ $\equiv_{\rho}$ is the module operator with divisor ρ .

Fig. 2. Visualisation of $A(t)$

Fig. 3. F_1 , simulation time, and number of created models for different values of ϕ

Fig. 4. F_1 , simulation time, and number of created models for different values of S

Fig. 5. Results for the case study. The different values of ψ use $S = 14$, the different values of S use $\phi = 0.6$.

tion show the potential of **GRAHOF** on event stream. We have presented a first evaluation of two hyperparameters of **GRAHOF** in Sections and . The results present the effect of each parameter (ψ and S) on its own, future work should elaborate on the dependence between their values.

Several improvements for **GRAHOF** are targeted in the future. The bottleneck algorithm has the robustness that is required to deal with event logs that do not conform with the process model. On one hand, this removes restrictions on the quality of the event log. On the other hand, the algorithm does not allow to account

for more complex process patterns such as choices and loops. This paper has specifically targetted seasonal drift with a known recurrence frequency, keeping ρ constant. Including dedicated recurrent drift detectors or finding ways to remove the need for ρ will be the next step in improving **GRAEC**. As is clear from the simulation results, picking the right value for S is important. Such a value can be estimated from an initial event log. An extension to **GRAHOF** is to make S dynamic. Future work on **GRAHOF** could be evaluating whether and how the performance decreases when the number of events or models is limited. The way in which **GRAHOF** is defined causes cases that span multiple batches to be included in a single batch. A further direction for future work could be to not only add cases to batches they start in, but also to batches they run in. Finally, we would like to test **GRAHOF** in concrete process mining applications focusing on customer journey optimization and changes of consumer behavior ([19] and [10]).

REFERENCES

- [1] Manuel Baena-García, José Campo-Ávila, Raúl Fidalgo-Merino, Albert Bifet, Ricard Gavaldà, and Rafael Morales-Bueno. Early Drift Detection Methods. *Intl. Workshop on Knowledge Discovery from Data Streams*, pages 77–86, 2006.
- [2] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New Ensemble Methods for Evolving Data Streams. In *KDD'09*, pages 139–148, 2009.
- [3] Albert Bifet, Geoffrey Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: massive online analysis. *JMLR*, 11, 2010.
- [4] Avrim Blum. Empirical Support for Winnow and Weighted-Majority Algorithms: Results on a Calendar Scheduling Domain. *Mach. Learning*, 26(1):5–23, 1997.
- [5] R P Jagadeesh Chandra Bose, Wil M P van der Aalst, Indrè Žliobaitė, and Mykola Pechenizkiy. Handling Concept Drift in Process Mining. In *AISE*, pages 391–405, 2011.
- [6] R P Jagadeesh Chandra Bose, Wil M P van der Aalst, Indrè Žliobaitė, and Mykola Pechenizkiy. Dealing With Concept Drifts in Process Mining. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):154–171, jan 2014.
- [7] Andrea Burattin, Alessandro Sperduti, and Wil M P van der Aalst. Control-flow discovery from event streams. In *CEC'14*, pages 2420–2427. IEEE, jul 2014.
- [8] Josep Carmona and Ricard Gavaldà. Online Techniques for Dealing with Concept Drift in Process Mining. In *IDA*, pages 90–102, 2012.
- [9] João Gama, Indrè Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A Survey on Concept Drift Adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37, 2014.
- [10] Joël Goossens, Tiblets Demewez, and Marwan Hassani. Effective steering of customer journey via order-aware recommendation. In *ICDM Workshops*, pages 828–837, 2018.
- [11] Marwan Hassani. Concept Drift Detection of Event Streams Using an Adaptive Window. In *33rd International ECMS Conference On Modelling And Simulation*, 2019.
- [12] Marwan Hassani, Sergio Siccha, Florian Richter, and Thomas Seidl. Efficient process discovery from event streams using sequential pattern mining. In *SSCI 2015*, pages 1366–1373, 2015.
- [13] Marwan Hassani, Sebastiaan J. van Zelst, and Wil M. P. van der Aalst. On the application of sequential pattern mining primitives to process discovery: Overview, outlook and opportunity identification. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, 9(6), 2019.
- [14] Paulo Mauricio Gonçalves Jr and Roberto Souto Maior de Barros. RCD: A recurring concept drift framework. *Pattern Recognition Letters*, 34(9):1018–1025, 2013.
- [15] Marco Malsenbacher and Matthias Weidlich. Handling Concept Drift in Predictive Process Monitoring. In *SCC'17*, pages 1–8, jun 2017.
- [16] J. Martijushev, R P Jagadeesh Chandra Bose, and Wil M P van der Aalst. Change Point Detection and Dealing with Gradual and Multi-order Dynamics in Process Mining. In *Change*, volume 229 of *Lecture Notes in Business Information Processing*, pages 161–178, Cham, 2015.
- [17] Sasthakumar Ramamurthy and Raj Bhatnagar. Tracking recurrent concept drift in streaming data using ensemble classifiers. In *ICMLA 2007*, pages 404–409, dec 2007.
- [18] Yorick Spenrath and Marwan Hassani. Ensemble-Based Prediction of Business Process Bottlenecks With Recurrent Concept Drifts. *Workshop proceedings of the EDBT/ICDT 2019 Joint Conference*, 2019.
- [19] Alessandro Terragni and Marwan Hassani. Optimizing customer journey using process mining and sequence-aware recommendation. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019*, pages 57–65, 2019.
- [20] Van Dongen, B.F. (Boudewijn). BPI Challenge 2017, 2017.
- [21] Sebastiaan J. van Zelst, Alfredo Bolt, Marwan Hassani, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Online conformance checking: relating event streams to process models using prefix-alignments. *Int. J. Data Sci. Anal.*, 8(3):269–284, 2019.
- [22] Haixun Wang, Wei Fan, Philip S Yu, and Jiawei Han. Mining Concept-drifting Data Streams Using Ensemble Classifiers. In *KDD '03*, pages 226–235. ACM, 2003.

linecks in stream process mining under concept drift.

Marwan Hassani is 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 65 scientific publications and is acting as an organizer and a regular reviewer in several international conferences and journals.