Process Histories - Detecting and Representing Concept Drifts Based on Event Streams

Florian Stertz, Stefanie Rinderle-Ma

University of Vienna, Faculty of Computer Science, Vienna, Austria
{firstname.lastname}@univie.ac.at

Abstract. Business processes have to constantly adapt in order to react to changes induced by, e.g., new regulations or customer needs resulting in so called concept drifts. By now techniques to detect concept drifts are applied on process execution logs ex post, i.e., after the process is finished. However, detecting concept drifts during run-time bears many benefits such as instant reaction to the concept drift. Introducing *process histories* as a novel way to detect and represent incremental, sudden, recurring, and gradual concept drifts through mining the evolution of a process model based on an event stream will face this challenge. Therefore, a formal definition of process histories is given, the concept of process histories is prototypically implemented and compared with existing approaches based on a synthetic event log.

Keywords: Process Mining, Event Streams, Runtime, Concept Drift, Process Histories

1 Introduction

Business processes have to constantly adapt in order to react to changes [15] induced by, for example, new regulations or customer needs resulting in so called concept drifts [5]. A recent example for new regulations possibly forcing business process changes is the General Data Protection Regulation as non-compliance with these regulations can cause fines up to €20 million [8]. Process changes are explicitly defined and stored in so called change logs [16] and hence are known to the company. Contrary, concept drifts are happening as the process evolves and hence are to be detected from process execution logs, i.e., logs that store events of executing process instances such as starting or completing process tasks. By now techniques to detect concept drifts in business processes work on process execution logs and are hence applied ex post, i.e., after the process is finished. However, detecting concept drifts during run-time bears many benefits such as being able to instantly react to the concept drift.

Run-time detection of concepts drifts works on event streams rather than on process execution logs. As event streams are infinite, online concept drift detection faces the following challenges: start and end event of the stream are unknown; it is not known how many events belong to a trace; it is not known which future events will occur and when they will occur. Moreover, different

kinds of concept drift are to be distinguished, i.e., incremental, sudden, recurring, and gradual drifts [5]. So far, the focus has been put on incremental and sudden drifts only, however, detecting recurring and gradual drifts can be important for many application domains as well. These challenges will be tackled along the following research questions:

RQ1 How to detect and reflect process model evolution based on event streams?
RQ2 How to detect incremental, sudden, recurring, and gradual concept drifts based on event streams?

For addressing RQ1, *process histories* are introduced. A process history reflects viable models that are discovered for a process based on an event stream. Process histories provide a novel way to detect and represent concept drifts through mining the evolution of a process model based on an event stream. The challenging question is when a new model is created, i.e., which event or sequence of incoming events triggers the creation of a new model in the history. We present two new algorithms. The first algorithm creates the process history and discovers new viable models. The detection of a viable model, is based on conformance [17] and the "age" of the event information using the sliding window approach, i.e., older process instances do have no impact on the current business process logic. The second algorithm determines concept drifts based on the synthesised process histories (RQ2) and enables the detection of incremental, sudden, recurring, and gradual drifts. The evaluation comprises a prototypical implementation as well as a comparison with existing approaches on detecting concept drifts based on synthetic and real-life logs. In summary, this work provides means to detect incremental, sudden, recurring, and gradual concept drifts based on event streams and the concept of process histories during run-time.

The paper is structured as follows: In Sec. 2, the required definitions and techniques for this work are described. Sec. 3 features the main contribution of this work, followed by an evaluation, Sec. 4, based on a synthetic log created using the process models of [4]. The related work is presented in Sec. 5 and an outlook and summary is provided in the last section, Sec. 6.

2 Fundamentals

This section introduces fundamentals on business processes and process mining, defines process histories and discusses event streams in comparison with process execution logs.

Process Execution Log. Every time a business process is executed, the information on the execution is usually recorded using the XES[1] [1] format. In short, a business process corresponds to a **log** node in a XES file. One log can have zero to many **trace** elements. A trace corresponds to an executed process instance of the business process and it contains information about the process instance, like an id,Runtime and has zero to many **event** elements. An event

[1] Extensible Event Stream

symbolises an executed activity in the process instance and contains information related to this event and the id of the log as well. In a stream based architecture, the last part is important to relate specific events to their corresponding log.

These log files, allow for the three main types of process mining [18].

- *Process Model Discovery.* This technique is used for finding a fitting process model for the log file.
- *Process Conformance Checking.* This technique takes an event log and an already discovered process model, and checks if the log fits the process model.
- *Process Enhancement.* This technique allows to change and improve the already discovered process model with a new log file.

Process History. A process history contains every process model for one business process and is defined as follows:

Definition 1 (Process History). *Let P be a business process. A process history H_P is a list of viable process models $M_n, n \in \mathbb{N}$ that have been discovered for P with M_n being the current model for P, formally:*

$$H_P :=< M_0, M_1, ..., M_{n-1}, M_n, .. > \tag{1}$$

For synthesising a process history, process model discovery and process conformance checking techniques are applied in Alg. 1 in Sec. 3 to find, check and adapt the current process model.

Since log files are static and created after the execution of process instances, process mining approaches are often applied offline, i.e., ex-post. Our contribution aims at synthesising a process history for a business process online, i.e., at run-time. To achieve this goal, an event stream, instead of log files is used [23].

Event Stream. An event stream represents a continuous flow of events produced by process instances of a business process. To help identify which events belong to which trace or event log, a unique identifier is embedded in the event itself. There are at least two main differences between a log file and an event stream. First, a log file is finite meaning that the information on the number of events per trace and which events appear is available. Second, a log file is also complete, so the specific end and start events of a business process are known. In an event stream it is not guaranteed, that there are no more events for a trace coming in. It is unlikely as well, that we listen to the stream from the time when the first event of the first process instance has occurred, until the last event of the last process instance, because of main memory issues and of course usability. It should be possible to start listening to an event stream at any time. For the implementation of Alg. 1, special data structures are required, to discover a process model.

Data structures: A process history covers all viable process models that have been executed for a specific process. Any time an event is sent to the process execution engine, it will be processed using the three types of process mining. For process model discovery, there is a plethora of algorithms available to mine process models, for example, the α-miner [19] or the inductive miner [10].

For process model discovery, we are using an adapted version of the stream-based abstract representation (S-BAR) [23]. S-BAR introduces an abstract representation of the directly follows set of events. This set of events consists of every observed pair of subsequently executed events. This is achieved by creating two maps. In this case, a map relates to the well-known data structure of a *hash table* [7], consisting of keys and their corresponding values.

The first map, **trace_map**, is built using the trace id of a trace as key. The corresponding value to a trace id, is the whole trace. In an event stream, one event at a time is processed. After processing the event, is put into the trace_map. To cope with memory issues and to help determine active traces, the point in time when the first and currently last event of a trace is being processed is also stored. The second map, **directly_follows_map** represents the directly follow relations of all events w.r.t. the trace_map. As key, the preceding event is being stored, with the following event as the corresponding value.

The usage of these maps is explained in detail in Sec 3. Figure 1, shows the trace_map and directy_follows_map for the traces [A,B,C] and [A,C,B]

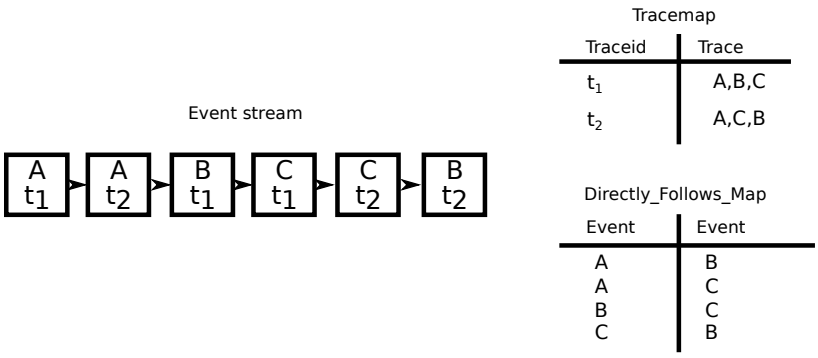

Fig. 1. Event stream containing two traces with different order of events.

For using the inductive miner, an assumption of specific start and end events is required. Since we look at an event stream, it cannot be guaranteed to identify the correct end or start events. E.g., a set of collected start events of each trace can be taken as input. The example in Fig. 1, shows two traces that both have the same starting event "A", so this event would be the only start event for the inductive miner. For the end events, we can only consider the last known events of already known traces. The traces in Fig. 1 provide two possible end events. The first trace has as an end event "C", while the second trace has as an end event "B". This results in a set containing two end events marked for the inductive miner, namely "B" and "C".

Conformance Checking: Every time a new event is processed, we check if this newly extended trace is fitting an already mined process model using conformance checking [17]. Conformance checking replays a trace on a given process model and tries to align it as good as possible. The fitness of a trace in

its most basic way, is calculated using costs for inserting events into the model if there are too many events in the log and inserting events in the log, if there are not enough events in the log to fit the model.

Sliding Window: The two maps, trace_map and directly_follows_map, contain the necessary information about every processed event. Since business processes change, already finished or older traces may be part of a preceding version of the business process. Another important factor is, that not every trace can be saved in the main memory, because of capacity issues. To resolve that, the sliding window approach is used. The sliding window only stores k data entries, here keys in the trace_map. If there are already k entries stored, the oldest one is removed before storing the new entry. If the trace_map would be exceeding k, the key value pair with the oldest currently known end event is removed. This method ensures, that only currently active and newer traces are taken into account while discovering a new process model and checking its fitness.

Concept Drift: The last important concept for process histories are concept drifts [21]. A concept drift reflects a shift in the business process logic, meaning that the execution of a business process changed over time. There are several reasons for a change in the process model, like a new business policy or adaptions in the business process logic to meet customer needs. Every time the business process logic changes, a new process model is discovered.

[4] defines 4 kinds of types of concept drifts.

- *Sudden Drift*. It shows a complete new workflow for the business process, for example caused by a new legislation, like GDPR.
- *Recurring Drift*. There could be a process model that is used for a specific time in the year, for example, Christmas season, in which workflows are executed differently to meet customer needs. These drifts appear periodically and replace the process model with another already known process model.
- *Incremental Drift*. Describes small changes, that are natural in the evolution of a business process. Especially in the beginning, the process history will be often extended, because a new process model is discovered after each new event in the event stream. This results in many sub process models.
- *Gradual Drift*. The process got changed and all process instances since the change point have a different process model. M_n and M_{n-1} coexist, as long as already started process instances of M_{n-1} are still running.

It is to be noted, that recurring drifts and incremental drifts, can also be gradual drifts, since process instances of the PM_{n-1} could still be executed.

A process history enables the detection of each of these drifts and is defined in the next section.

3 Contribution

This section describes the contribution of this paper, synthesising process histories and detecting concept drifts.

3.1 Synthesising a Process History

For tackling RQ1, a process history, H_P, is synthesised. The process history contains a list of already known process models, M_i, for a process P, where M_0 is the first known process model for P and M_n is the last known and currently used model for P. With the list of process models, all historical changes of P's logic are represented in H_P, and show the evolution of P.

These models are discovered using an event stream.

The developed Algorithm 1 synthesises a process history and is described in the remainder of this subsection. As input an event stream, ES, a window limit k and the thresholds ϕ and σ are required. The thresholds are described in detail in the following paragraphs.

At the beginning the process history, H_P, is an empty list and does not contain any process models. The trace_map, explained in Sec. 2, contains no items in the beginning. The directly_follows_map is created after an unfit trace is detected.

For usage of the sliding windows approach, the window size k must be defined. Only k items are possible in the trace_map. Every time a new event is processed, it is checked, if its trace id is already existing in the trace_map. If it does not exist and the size of the map is smaller than k, the trace id is used as key and as a value, the event is used as the starting event of the corresponding trace. If the map has already k items, the oldest trace is removed from the trace_map. If the trace id is found in the trace_map, this event will be appended to the trace.

Afterwards, if there is already at least one model in the process history, the fitness of the active trace is checked. For this purpose, we use common conformance checking techniques. Conformance checking returns the fitness value for a trace for a process model by replaying and aligning the trace to the model [2]. For the alignment costs of the trace, two different costs are calculated. The costs for a move in a log, describe if an event is found in the log but not in the model at this position. On the other hand, costs for a move in a model, describe if an event is found in the model but not in the log. For our purposes, only moves in a log are considered, because in an online environment, it is not known, if a process instance reached its end event yet, which means, that the trace can still fit the model. The fitness value of a trace for a model ranges between 0, does not match at all, and 1, matches the model perfectly.

The model in Fig. 2 for example, is our last known model in the process history. The two traces that would match completely would be [A, B, C, D] and [A, C, B, D]. Since we only take the moves in the log into account, the two traces [A, B, C] and [A, C, B] receive a perfect score, and it is assumed, that those process instances are still being executed and the end event "D" has not been processed at the moment.

The last trace [A, D, B], received a lower fitness score, based only on moves in the log. The second event "D" is not expected this early in the process model and cannot be aligned in a perfect way, so it is moved in the log.

To define if a trace fits the model, a threshold, σ is introduced, ranging from 0 to 1. While 0, would result in any trace fitting any model, 1 would only consider

Input: Event Stream ES (a series of events)
k (Limit for number of trace_map items)
σ (Threshold for the fitness of a trace for a model, [0,1])
ϕ (Threshold for distinction of a new viable model [0,1])
Result: Process History H_P (contains all viable process models in chronological order.)

```
H_P = [ ]
M_duration = 0
trace_map<trace_id,trace> = 0
for e in ES do
    if trace_map contains_key e.trace_id then
        trace_map['e.trace_id'].append(e)
    else
        if trace_map.size ≥ k then
            trace_map.delete_oldest
        trace_map.insert(e.trace_id,e)
    if H.size ≠ 0 and conformance_checking(traces[e.trace_id],H_P.last) < σ
     then
        directly_follows_map<event,event> = 0
        for t in trace_map.values do
            if conformance_checking(t,H_P.last) < σ then
                for i in t.size do
                    if i != 0 then
                        directly_follows_map.insert(t[i-1],t[i])
        Model = inductive_miner(directly_follows_map)
        fitting_traces_counter = 0
        durations = [ ]
        for t in trace_map.values do
            if conformance_checking(t,Model)≥ σ then
                fitting_traces_counter+= 1
                if t.end_event in Mode.end_events then
                    durations.append(t.end_event.time-t.start_event.time)
        Score_{Model,trace_map.values} s= fitting_traces_new / trace_map.values.size
        if s ≥ φ then
            H_P.append(Model)
            M_duration = durations.average + durations.std_deviation
            unfinished_traces =
              trace_map.get_unfinished(H_P[H_P.size-1],trace_map, M_duration)
            detect_concept_drift(trace_map.values,unfinished_traces,H_P,φ,δ)
    if |H_P| = 0 then
        H_P.append(inductive_miner(e))
```

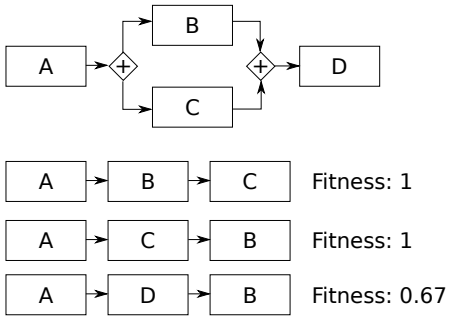

Fig. 2. A process model with one parallel gateway and 3 related traces. While the Move-Log fitness is perfect for the first two traces, the last trace contains an additional event and receives a lower score

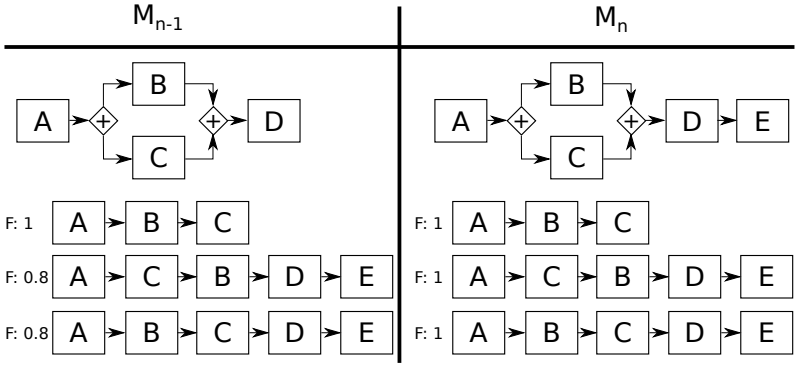

Fig. 3. Model M_{n-1} is only fitting one the first trace perfectly. Model M_n is fitting all traces. M_n is now the new model.

perfectly matching traces as fitting. For the purpose of detecting viable process models, a high threshold like 1 is suggested. This guarantees to only consider perfectly matching traces for the distinction.

If the trace of the currently processed event, does not fit the last known process model of the process history, a new model is mined, using the inductive miner. As input for the inductive miner, the abstract representation of the directly follows relation is sufficient. Only unfitting traces in the current window are used for discovering the new process model. The inductive miner always produces sound workflow nets and suffers from less instabilities like the α-miner, which does not detect short loops for example.

To distinguish between viable new process models and anomalous process instances, a score for a process model for a set of traces is defined as:

Definition 2 (Model score). *Let T be a given set of traces, M a process model, and $\phi \in (0,1]$ be a threshold. Moreover let $A \subseteq T$ be the set of all traces having a fitness score greater or equal than ϕ and let $\chi_A(t)$ be the indicator function, returning 1 if $t \in T$ is in A, 0 otherwise.*

Then $\mathcal{S}_{M,T}$, the model score of process model M w.r.t. T, is defined as

$$\mathcal{S}_{M,T} = \frac{\sum_{t \in T} \chi_A(t)}{|T|}.$$

In the algorithm, for calculating the score for the new model using the current trace_map, the values of the whole map, are checked for conformance with the newly discovered process model. If a trace is fitting the new model, a counter is increased by 1, starting at 0. In addition, if the trace's end event is one of the end events of the new model, the complete execution time is calculated and stored in a list of execution times for this model. The variable *M_duration* describes the average execution time of M plus the standard deviation. The number of fitting traces is then divided by the number of all possible traces from the trace_map, which results in the score for the new model and the trace_map. The score for this model, ranges from 0 to 1 as well. To determine if the new model is viable, the score must be greater or equal than ϕ. ϕ is introduced as a threshold between 0 and 1, where 0 considers any model as viable and 1 only models that fit every trace from the current window to the new model. For the history in the evaluation, only models fitting at least 90% of the traces from the current window have been considered to get a strict list of viable models, with results discussed in Sec. 4.

In Fig. 3, the detection and creation of a new process model in the process history is shown. On the left, the two longer traces do not fit the last known process model in the process history. The newly discovered model, visible in Fig. 3 on the right, is able to fit all current traces into the model. Since the new model fits all current traces, the new model is appended to the process history. The old model is now M_{n-1} in the process history and the newly created model is now the last and current model in our process history, M_n.

Every time a new model is discovered and appended to H_P, a concept drift is detected. To determine the type of the concept drift, unfinished traces for M_{n-1} from the trace_map need to be collected. A trace is likely to be unfinished if its end event is not part of the end events of M_{n-1} and its current execution time is lower than the execution time stored in *M_duration*. If its execution time is larger, the process instance is likely to be cancelled.

3.2 Concept Drift Distinction

Algorithm 1 synthesises a process history for a specific process. Every time a new process model is appended to the process history, a concept drift is detected. The 4 types of concept drifts, in relation to a process history, can be defined formally as follows:

Definition 3 (Concept Drift Types). *Let T be a given set of traces and U be a given set of unfinished traces. Moreover let H be a process history for a process P and $\delta \in [0,1]$, $\epsilon \in [0,1]$ be thresholds and the function fitness, defined for one trace and a model, ranging from 0 to 1. The following drift types are defined as follows:*

- *Incremental Drift if* $|H| \geq 2 \wedge \exists(t \in T,\ fitness(t, M_{n-1}) \geq \delta \wedge\ fitness(t, M_n) \geq \delta)$
- *Recurring Drift if* $|H| \geq 3 \wedge \neg\ IncrementalDrift \wedge \exists\ m \in \mathbb{N}, 2 \leq m \leq n,\ |S_{M_n,T} - S_{M_{n-m,T}}| \leq \epsilon$
- *Gradual Drift if* $U \neq \emptyset$
- *Sudden Drift if* $\neg IncrementalDrift \wedge \neg RecurringDrift \wedge \neg GradualDrift$

As a fitness function, this work is using again conformance checking with only considering moves in the log [2].

It is to be noted, that an incremental drift and a recurring drift can be a gradual drift as well. This approach allows to detect concept drifts and identify the type of the concept drift with the use of Alg. 2 and tackle RQ2.

As input parameters a list of traces T, the traces from the trace_map, a list of unfinished traces U for M_{n-1}, collected by Alg. 1, for detecting gradual drifts, a process history H_P, δ for determining fit traces and ϵ are required. ϵ describes the maximum error that is allowed between two model scores to be equally viable for T and ranges from 0 to 1, where 0 only determines equal scores to be similar viable and 1 determines any scores to be similar viable.

If there are less than two process models in the process history, it can be concluded that there is no concept drift, since a drift appears when the business process logic changes and a new model is discovered.

For every process model of H the model score is calculated using traces from T, like described in Alg. 1. The variable `Incremental` is calculated during the calculation of the scores to save execution time. If "Incremental" equals 1 an incremental drift is detected, otherwise not.

If there are traces out of T that fit the preceding model and the current model, an incremental drift is detected. As as long as U is not empty, the incremental drift is a gradual drift as well. Otherwise it is a sudden incremental drift.

For recurring drifts, the score of any model from M_{n-2} to M_0 is calculated. If there is at least one model M_m, where the difference between $S_{M_n,T}$ and $S_{M_m,T}$ is less or equal ϵ, a recurring drift is detected. Then it is again distinguished between a gradual recurring drift and a sudden recurring drift, using the same approach as before.

If it is not a recurring drift or an incremental drift, it number of elements in U is checked. If there is at least one trace, a gradual drift is detected. Otherwise it is not a gradual drift and a sudden drift is detected, since it is already concluded that it is not an incremental or recurring drift either.

The return value is a vector with 4 items corresponding to `Incremental Drift`, `Recurring Drift`, `Gradual Drift` and `Sudden Drift`. E.g., a gradual recurring drift return [0,1,1,0], while a sudden drift returns [0,0,0,1].

In Fig 4, a complete process history is shown with $\epsilon = 0.05$ and $\delta = 1$. The first concept drift from M_0 to M_1 is detected with T containing t_1 [A,B,C,D,E] and t_2 [A,C,B,D,E]. An incremental drift can be detected between M_0 and M_1, since there is only a new event, E, added to the end of the process. The same traces that fit M_0, fit M_1 as well.

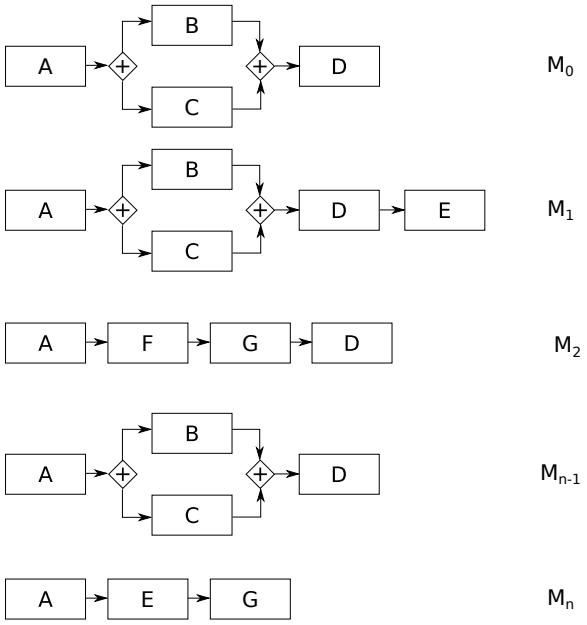

Fig. 4. Complete process history of a single business process

Let $t_{2,3}$ [A,B,C,D], $t_{4,5}$ [A,C,B,D] and $t_{6,\ldots,9}$ [A,F,G,D] be new traces in the event stream. With a small window size k, e.g, 5, M_2 is mined. Only $t_{6,\ldots,9}$ are considered for the model, since they are not fitting M_1. The difference between $S_{M_2,t_{5-9}}$ and $S_{M_1,t_{5-9}}$ or $S_{M_0,t_{5,\ldots,9}}$ is greater than ϵ, so it is not a recurring drift. There are no traces fitting M_2 and M_1 as well, so an incremental drift is not possible. Since t_5 is likely to be not finished for M_1, a gradual drift is detected.

Assume the next traces in the event stream are t_{10-12} [A,B,C,D] and t_{13} [A,C,B,D]. This results in M_{n-1}. The difference between $S_{M_0,t_{9,\ldots,13}}$ and $S_{M_{n-1},t_{9,\ldots,13}}$ is 0. A recurring drift is detected with the recurrence of M_0. Since t_9 s already finished, it is not a gradual drift as well.

Let the next traces be, $t_{14,\ldots,17}$ [A,E,G], which result in M_n. There is no equally similar score to $S_{M_n,t_{13,\ldots,17}}$ and there are no unfinished traces.

In the next section, the two algorithms are evaluated on a synthesised log, following the insurance example used in [4].

4 Evaluation

This section describes a tested example of process histories. Process Execution log files have been synthesised, transformed into an event stream and the process history discovered. The business process depicts an insurance process, first described in [4]. Since not all concept drifts have been integrated in the source, additional concept drifts have been added. With this process history, the type of concept drifts has been detected. The first part of this section covers the imple-

Input: Traces traces (list of traces), Traces u (list of unfinished traces), H (Process History),
δ (Threshold for fitting models $\in$ [0,1]
ϵ (maximum error between similar process models)
Result: type_vector[0,0,0,0] (Positions represent Drifts [Inc,Rec,Grad,Sudden], 1 represents this type of drift occurred.)

```
if H.size <= 1 then
    return "Error: No drift"
Scores = [ ]
Incremental = 0
for M in H do
    model_score = 0
    for t in traces do
        if conformance_checking(t,PM)>= δ then
            model_score += 1
            if M == M_{n-1} and conformance_checking(t,M_n)>= δ then
                Incremental = 1
    Scores.append(model_score/traces.size)
Scores = Scores.reverse // Reverse order so Scores[0] == M_n
if (Incremental == 1) then
    if u.size ≠ 0 then
        return [1,0,1,0] //Incremental Gradual Drift
    else
        return [1,0,0,0] // Incremental Drift
for i in Scores.size do
    //Start with 0 if i ≤ 1 then
        next
    if (|Scores[0]-Score|≤ ϵ) then
        if u.size ≠ 0 then
            return [0,1,1,0] //Recurring Gradual Drift
        else
            return [0,1,0,0] // Recurring Drift
if u.size ≠ 0 then
    return [0,0,1,0] // Gradual Drift
return [0,0,0,1] // Sudden Drift
```

mentation of the algorithms and the framework for the evaluation. The second part shows the execution and results.

4.1 Implementation

A tool to synthesise process execution log files, has been implemented in Ruby [12]. A web service has been implemented as well, to transform static log files into an event stream. The generated log files have time stamps in every event as information. The web service extracts the list of events from the log files and orders them, based on time stamps. This results in a chronological correct event stream. Every event is then sent to the main web service. This web service, written in Ruby as well, processes each event and runs both algorithms described in Sec. 3. The process history is constantly adapted and provided through a REST interface. For the mining algorithm we are using the inductive miner, implemented in ProM called from extension RapidProm [3] of Rapidminer. The output is then retrieved using the REST interface.

4.2 Evaluation

For the evaluation, we synthesised process execution log files, based on the process models used in [5]. Small modifications have been applied, because only one path of some decisions showed concept drifts. For every process model 100 process instances were created. Since not all types of drifts are detectable in these models, we added new process instances to find every type of concept drift. For the creation of the process history, k was set to 50, δ to 1 and σ to 0.9. For the distinction of a concept drift, ϵ was set to 0.05 and σ to 1.

The first 100 process instances consisted of "Register", "Decide High/Low", , "High Insurance Check", "High Med.History Check", "Contact Hospital", "Prepare Notification", "By Phone", "By Email","By Post" and "Notification Sent". The order of "High Insurance Check", "High Med. History Check" and the order and existence of "By Phone", "By Email", "By Post" have been randomised, so that the inductive miner is able to detect the parallel paths and decisions. The first models produced can vary a lot, depending on the order of events in the event stream. Fig. 5 shows the first discovered viable process models in the process history. M_0 consists of only one event. During the first 100 instances, the process model evolves and, depending on the order of the execution of the process instances, the first part of the first parallel gateway can be seen in M_4. Algorithm 2 detects for the first process models in the history only incremental drifts, as expected. This can be reasoned because, every time a new event is found at the end of a trace or a new parallel order instead of sequence is mined, all other traces from the previous model are fitting the new model, e.g, the trace ["Register", "Decide High/Low", "High Insurance Check","Contact Hospital"] and the trace ["Register", "Decide High/Low", "Contact Hospital", "High Insurance Check"] are both fitting M_5.

After each possible combination is executed, M_{n-4} (Fig. 7), is discovered. To create a gradual drift, the next 50 instances are fitting M_{n-4}, but did not finish

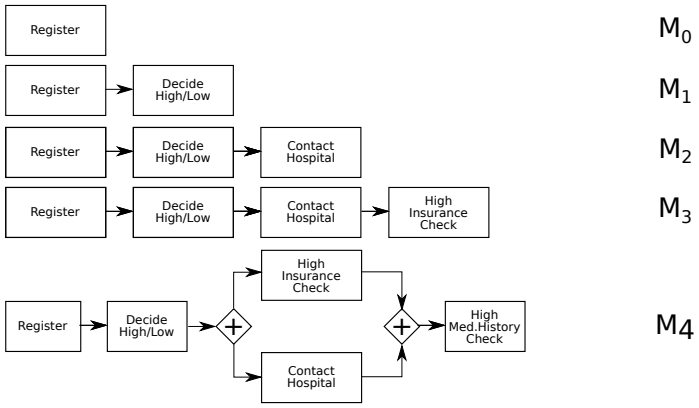

Fig. 5. Process Models containing concept drifts.

before the next 100 instances started in the stream, containing small adaptations. Instead of a parallel gateway for the medical checks, cheaper checks, like "High Insurance Check", are done at the beginning. If this check fails, the other checks are automatically skipped. Unfit traces now contain only a subset of the 3 events. After 45 instances, σ and the score of the new process model M_{n-3} for the traces of the trace_map are equal, so the model is appended to the process history. Since there are 5 instances for M_{n-4} not finished as well, we detected a gradual drift. An incremental drift has been detected as well, since some traces fit M_{n-4} and M_{n-3} perfectly.

All the concept drifts from [4] cannot be detected with our approach. As can be seen in Fig. 6, the first process model includes the events "By Phone", "By Email", "By Post" in optional parallel paths. The first concept drift described in [4] changes the parallel gateway to a decision, where only one event is chosen (Fig. 6(b)). Since all paths are optional anyway, all traces fit, even if only one event is present. To negate this, a periodical model could be mined, using all traces in the trace_map to detect a stricter model fitting all traces. The other model containing again a subset of choices already possible in the parallel optional model, suffers from the same problem. The concept drift from Fig. 6 (b) to (c) could be detected, but only if b is discovered. The drift from (a) to (c) cannot be detected.

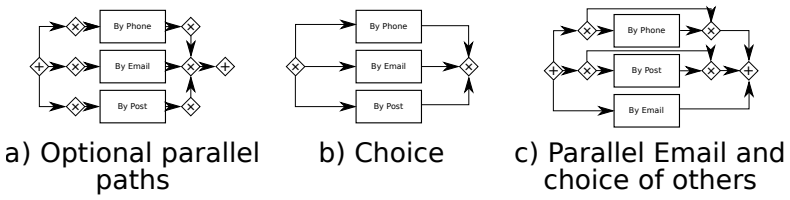

Fig. 6. The concept drift from a to b is not detectable as well as the drift from a to c, since traces from b and c fit a. The drift from b to c is detectable.

We injected another 100 instances into the event stream, representing a new legislation. The split of high and low insurance claims has been removed, i.e., every claim is treated the same way. The notifications are only allowed to be sent per post as well. After 45 instances, the model M_{n-2} is discovered. This model varies vastly from M_{n-3}, since the score from M_{n-2} is 0.9 and the score of M_{n-3} is 0.1. No traces from M_{n-3} match M_{n-2}. Also M_{n-2} does not conform to any other known model in the process history. A sudden drift is detected.

In the next 100 instances, a new event at the end has been discovered. "Receive delivery confirmation" is appended to the end of the new process instances. Again after 45 instances, M_{n-1} is discovered. Since the 5 oldest traces still fit M_{n-2} and M_{n-1} an incremental drift is detected.

For the last 100 instances, the first 100 instances have been injected into the stream again with modified time stamps. As expected, after 45 instances, M_n is discovered, which is identical to M_{n-4} and both have the same score of 0.9. A recurring drift has been detected.

All four types of concept drifts can be detected. A problem occurs, if the process model after the concept drift is just a stricter model. This means if new traces fit the current model perfectly, no new model will be discovered and no concept drift will be detected. This can be negated by discovering a new model periodically instead of only if an unfit trace has been found, but this could lead to big mixed process models, if not only the unfit traces are used for discovering a new model. E.g., the sudden drift in Fig. 7 from M_{n-3} to M_{n-2}, could also be interpreted with a decision after the "Register" event, which leads to the path from M_{n-3} or the path from M_{n-2}.

Section 5 covers the related work in this field.

5 Related Work

A plethora of algorithms use XES files for discovering process model. The most prominent mining algorithm is the α-miner [13]. This mining technique transforms a directly follow abstraction [23] into a Petri Net [14]. These XES files do not change while they are used for process mining.

Online process mining: An online setting using abstract methods is described in [23] working with directly follows relation, [6] using the heuristics net miner [20] or [11] detecting concept drifts in ltl declared models. The requirements for an online setting, are (a) finite memory. Process execution logs tend to get larger and larger. The size of these files gets so big, that mining the entire XES file at once is not possible, because there is not enough main memory available. Since the files get larger, there is also more data to process. For an online setting, the calculations need to be finished at run-time (b), therefore there are run-time constraints. To cope with these requirements in an offline setting, many process mining approaches, create an abstract representation of an event log to retrieve a process model. The S-BAR approach complies with the following principles. It reuses existing approaches for finding the process model. For the usage of existing techniques, an abstract representation is built.

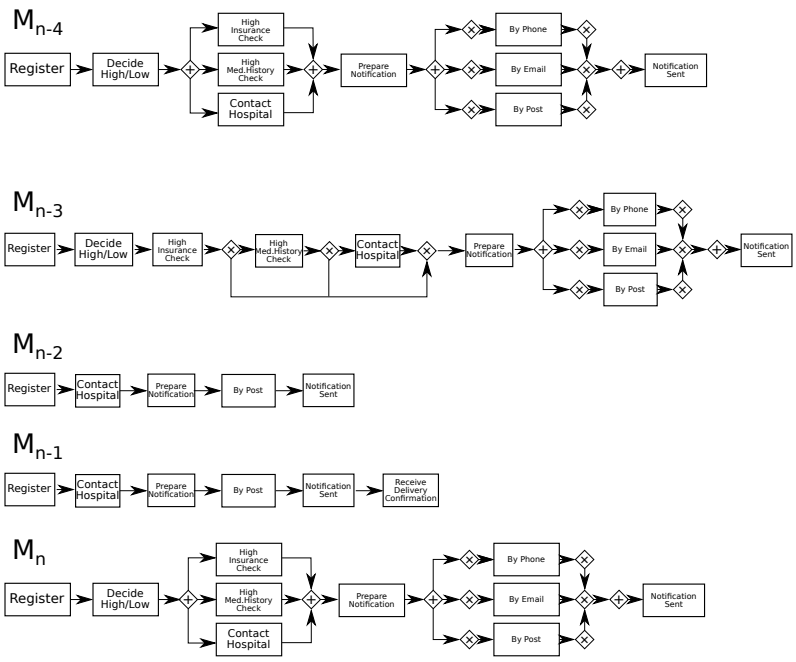

Fig. 7. Process Models containing concept drifts.

Concept Drift in Process Mining: A sudden shift, induced by a new legislation for example, can cause a change in a business process. This is followed by a change in the process execution and the process execution logs. The already mined process model might be not suitable any more for conformance checking, a concept drift [21]. In [4], an approach to find concept drifts in process execution logs is discussed. Features to interpret the relationships of events are introduced.

- `Relation Type Count`. Defines a vector for every event, containing the number of events that always, sometimes and never follow a specific event.
- `Relation Entropy`. The average rate at which a specific relation is being created.
- `Window Count`. The count is defined for a specific relation, like the follows relation, on a give length.
- `J-measure`. Originally proposed by Smyth and Goodman[9], to calculate the goodness of a rule, like b follows a. This is done with cross-entropy of two events and a specific windows size.

The first two features are calculated using the whole log, while the other two features are calculated on each trace. Concept drifts, are detected by splitting the log in smaller sub-logs and finding the point of change through statistical tests, like the **Kolmogorov-Smirnov test** and the **Mann-Whitney U** test.

The related work, offers ways to retrieve process models from process execution logs and detect concept drifts in an offline environment. While it is possible

to detect the exact point in time when the drift is happening, there is no differentiating of types of concept drifts. Also since it is done offline, the results are ex-post. The online mining approaches use a similar strategy to discover process models at run-time. Concept drifts can be detected, but are not differentiated and not all types of concept drifts can be detected.

6 Summary and Outlook

This work introduces process histories to reflect the evolution of a process based on an event stream during run-time. The histories consist of a sequence of viable models of this process. Based on this model sequence, incremental, sudden, recurring, and gradual concept drifts can be detected. For synthesizing the process histories, an algorithm utilizing conformance checking and the "age" of event information has been presented. All concepts are evaluated through a proof-of-concept implementation and a comparison with existing approaches. With static log files [4], the exact point of time of a concept drift can be detected, but is not using an online environment and does not differentiate the types of concept drifts. In an online environment [11] [22], concept drifts can be detected, but not all types of concept drifts have been covered and the drift is detected relatively late. The advantage of the other approaches is the detection of stricter process models, since they are not focused on detecting drifts, but discovering new process models. Future work will focus on the refinement of synthesizing process histories, i.e., parallel events at the end of a process, other techniques to calculate the fitness of a specific trace, detecting concept drifts in a stricter model as well as testing other mining algorithms including the frequency of events and other approaches like lossy counting instead of sliding window for the determination of impactful traces.

Acknowledgment

This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-072.

References

1. IEEE standard for extensible event stream (XES) for achieving interoperability in event logs and event streams. IEEE Std 1849-2016 pp. 1–50 (Nov 2016)
2. Van der Aalst, W., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
3. van der Aalst, W.M., Bolt, A., van Zelst, S.J.: RapidProM: mine your processes and not just your data. arXiv preprint arXiv:1703.03740 (2017)
4. Bose, R.J.C., van der Aalst, W.M., Žliobaitė, I., Pechenizkiy, M.: Handling concept drift in process mining. In: International Conference on Advanced Information Systems Engineering. pp. 391–405. Springer (2011)

5. Bose, R.J.C., Van Der Aalst, W.M., Zliobaite, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE transactions on neural networks and learning systems 25(1), 154–171 (2014)
6. Burattin, A., Sperduti, A., van der Aalst, W.M.: Heuristics miners for streaming event data. arXiv preprint arXiv:1212.6383 (2012)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT press (2009)
8. Drolet, M.: How much will non-compliance with gdpr cost you? CSO (Oct 2017), https://www.csoonline.com/article/3234685/data-protection/how-much-will-non-compliance-with-gdpr-cost-you.html
9. Goodman, R.M., Smyth, P.: Rule induction using information theory. G. Piatetsky (1991)
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.M., Desel, J. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 311–329. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
11. Maggi, F.M., Burattin, A., Cimitile, M., Sperduti, A.: Online process discovery to detect concept drifts in ltl-based declarative process models. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". pp. 94–111. Springer (2013)
12. Matsumoto, Y., Ishituka, K.: Ruby programming language (2002)
13. Alves de Medeiros, A., Van Dongen, B., Van Der Aalst, W., Weijters, A.: Process mining: Extending the alpha-algorithm to mine short loops. Tech. rep., BETA Working Paper Series (2004)
14. Peterson, J.L.: Petri net theory and the modeling of systems (1981)
15. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer (2012), https://doi.org/10.1007/978-3-642-30409-5
16. Rinderle, S., Reichert, M., Jurisch, M., Kreher, U.: On representing, purging, and utilizing change logs in process management systems. In: Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings. pp. 241–256 (2006)
17. Rozinat, A., Van der Aalst, W.M.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64–95 (2008)
18. Van Der Aalst, W., Adriansyah, A., De Medeiros, A.K.A., Arcieri, F., Baier, T., Blickle, T., Bose, J.C., van den Brand, P., Brandtjen, R., Buijs, J., et al.: Process mining manifesto. In: International Conference on Business Process Management. pp. 169–194. Springer (2011)
19. Van Der Aalst, W., Van Hee, K.M., van Hee, K.: Workflow management: models, methods, and systems. MIT press (2004)
20. Weijters, A., van Der Aalst, W.M., De Medeiros, A.A.: Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP 166, 1–34 (2006)
21. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine learning 23(1), 69–101 (1996)
22. van Zelst, S.J., Bolt, A., Hassani, M., van Dongen, B.F., van der Aalst, W.M.: Online conformance checking: relating event streams to process models using prefix-alignments. International Journal of Data Science and Analytics pp. 1–16 (2017)
23. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.: Event stream-based process discovery using abstract representations. Knowledge and Information Systems 54(2), 407–435 (2018)