Interactive Process Drift Detection Framework

Denise Maria Vecino Sato[1,2(✉)], Jean Paul Barddal[1], and Edson Emilio Scalabrin[1]

[1] Pontifical Catholic University of Paraná (PUCPR), Imac. Conceição. 1155, 80215-901 Curitiba, Brazil
{denise.sato,jean.barddal,scalabrin}@ppgia.pucpr.br
[2] Federal Institute of Paraná (IFPR), João Negrão. 1285, 80230-150 Curitiba, Brazil
denise.sato@ifpr.edu.br

Abstract. This paper presents a novel tool for detecting drifts in process models. The tool targets the challenge of defining the better parameter configuration for detecting drifts by providing an interactive user interface. Using this interface, the user can quickly change the parameters and verify how the process evolved. The process evolution is presented in a timeline of process models, simulating a "replay" of models over time. One instantiation of the framework was implemented using a fixed-size sliding window, discovering process maps using directly-follows graphs (DFGs), and calculating nodes and edges similarities. This instantiation was evaluated using a benchmarking dataset of simple and complex drift patterns. The tool correctly detected 17 from the 18 change patterns, thus confirming its potential when an adequate window size is set. The user interface shows that replaying the process models provides a visual understanding of the changing process. The concept drift is explained by the similarity metrics' differences, thus allowing drift localization.

Keywords: Process drift · Concept drift · Drift detection · Evolving environment

1 Process Mining in Evolving Environments

Process Mining (PM) is gathering more enthusiasts in recent years. The growing interest can be explained by the current availability of process data recorded by the informatics systems (big data) and the increasing development of tools to provide easy access to different PM techniques. The primary input of any PM technique is event data, which contains information about business process executions, and can be accessed in the form of event logs (historical information about the process) or event streams (continuous flow of events associated with processing instances). The event data must include at least an identifier of the process instance (case), the event (indicating the occurrence of activity), and the timestamp in which the event occurred.

Supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brasil (CAPES)-Finance Code 001, Grant No.: 88887.321450/2019-00.

L. Rutkowski et al. (Eds.): ICAISC 2021, LNAI 12855, pp. 192–204, 2021.
https://doi.org/10.1007/978-3-030-87897-9_18

There are three types of PM: discovery, conformance checking, and enhancement [1]. Discovery aims at learning what is happening with the processes by automatically discovering process models from the event logs without any *a priori* knowledge. Conformance checking compares a discovered or designed process model to the event log to pinpoint possible deviations. The goal is to help business analysts to understand problems or even unexpected behavior to support improvements. In the enhancement, an existent process model is extended (including new perspectives, e.g., performance) or improved using information from the event logs. Business analysts can then change the process model to reflect better reality based on the information provided by discovering or conformance. Discovery, conformance checking, and enhancement are usually applied offline by analyzing an event log, but PM techniques can also perform online analysis.

Regardless of the type of analysis, the business processes are not static. Companies and their analysts are always trying to improve the processes to minimize costs or maximize customer satisfaction. The processes also change when there is a new regulation or in case of unexpected events. Recently, the whole world had to adapt roughly every process to address the COVID-19 pandemic situation. So, it is naïve to assume that models do not change when business processes are placed in evolving environments. Besides, most state-of-the-art PM techniques consider the processes to be in a "steady-state", i.e., thus assuming that an event log contains information about a single version of the process. This assumption does not consider the existence of process drifts.

A process drift, or concept drift in processes, occurs when the business process changes while being analyzed [7]. The correct identification of the process drifts is critical when conducting process analysis in evolving environments. By detecting process drifts, the analysts should better understand the actual process model without facing a mixed model from different versions of the process. Process drift detection can improve reporting and diagnosis analysis, predictions, recommendations, and operational support by assuring that a more adherent version of the process model is considered. In online analysis, the process drift detection can help discovery or conformance checking techniques to maintain an up-to-dated process model, without the need to continually rediscover it.

Process drift was one of the challenges cited in the Process Mining Manifesto [7]. Since 2011, researchers have developed different process drift detection tools, such as ProM ConceptDrift plugin [5,6,10], Apromore ProDrift [8,9,12–14], and other experimental tools [3,11,15,16,18–21]. However, specific issues make it difficult to compare the tools affecting the adoption in real scenarios.

We identified the following experimental tools for process drift detection: Process Drift Detector Plugin (PDD) for ProM [5,6,10], Tsinghua Process Concept Drift Detection (TPCDD) [21], Concept-Drift in Event Stream Framework (CDESF) [3,11], Visual Drift Detection (VDD) [18,19], Dynamic Outlier Aggregation (DOA) [20], Online Trace Ordering for Structural Overviews (OTOSO) [15]. These tools share the drawback of ProM and Apromore, i.e., drift detection accuracy is highly affected by the hyperparameter configuration.

Some of them are essentially affected by the window size: TPCDD, CDESF (the window size is defined by the time-horizon parameter, controlling model updates), and VDD. The approaches based on clustering, i.e., TPCDD, CDESF, VDD, DOA, and OTOSO, are also sensitive to the clustering hyperparameters. The PDD applies an adaptive window approach aiming to solve the window size choice, but the adaptation is limited to user-given upper and lower bounds.

This paper proposes the Interactive Process Drift Detection (IPDD) Framework to provide a practical tool for process drift detection that can be easily applied in real scenarios. Section 2 describes process drifts and explores the available tools. In Sect. 3, the new framework is described. Section 4 presents results obtained with an instantiation of the proposed framework. Finally, Sect. 5 concludes the paper and indicates next steps for IPDD enhancement.

2 Process Drift

A process drift indicates a point in time where the process changes for a reason. The changes can be planned and documented or unexpected. A change in the process reflects a change in its process model, meaning a new one replaces the existing process model's version. The new process model can affect the ongoing process instances in different ways: sudden or gradual [5,6].

In a sudden drift, all the ongoing process instances start to emanate from the new process model when the drift occurs. It can occur when a new regulation should be followed or even within an epidemic situation. In a gradual drift, the new process starts emanating process instances, but both versions coexist for some time, indicating a gradual replacement of the model. The authors in [5,6] also describe two different dynamics for the changes: recurring and incremental. In the recurring drift, the current model is replaced by a new one, but the new model is replaced by the previous one after some time. The incremental drift represents minor incremental changes implemented during some time. The recurring drift can indicate seasonal changes, for instance. An incremental drift can occur in companies to minimize risks for significant changes. In each process model transition, we can have a sudden or a gradual drift.

Process drifts can occur at different time granularities. For instance, we can have a recurring drift that occurs every season, e.g., summer, winter, fall, and spring; and another drift occurring at the last week of the month, e.g., for specific accounting tasks. Both drifts coexist, yet they occur at different time granularities. In [10], the authors named this situation as multi-order dynamics, reinforcing that any drift detection mechanism should deal with different time granularities when identifying process drifts.

A process drift can affect one or more perspectives of the model, which are partially overlapping. The most common perspectives described in [1] are:

1. **Control-flow.** It represents the process model's behavior based on the structure of activities (sequential, parallel, choice, loops).
2. **Organizational.** Resources related to the process activities, which can be people, systems, departments, or others.

3. **Data.** It is related to the information relevant to the process associated with the case, e.g., supplier, or to a specific activity, e.g., a machine.
4. **Time.** Time and frequency of activities.

Tools and methods form handling process drifts should consider the types and perspectives of change and can also address different problems [5,6]:

1. **Change point detection.** Identify the point in time (timestamp) that the drift occurred. The change point can be reported by the case index, the event index, or the date/time where the change starts. Change point detection is the most common problem addressed by the available tools.
2. **Change localization.** Report the process model region which has changed, e.g., between activities A and B. This problem partially overlaps with "unravel process evolution", however, ProM and Apromore addressed this challenge without providing the process evolution. This problem is more about local changes and not the global picture of the drift.
3. **Change characterization.** Specify the type of change and in which perspective it occurred. This problem is less explored because few methods detect different process drifts or consider different perspectives.
4. **Unravel process evolution.** Relate and explore the former discoveries, putting everything together to understand the process's evolution over time. We did not identify any tool that explored this problem.

We focus on problems 1, 2, and 4. We propose IPDD, a framework for detecting change points and visually presenting its localization and process model evolution. IPDD deals with sudden drifts in the control-flow perspective. Incremental, recurring drifts, or multi-order dynamics are addressed by the user interface, allowing to check the detected drifts with different parameters.

2.1 Process Drift Detection Tools

Process drift detection tools can be divided into two categories: academic and experimental tools. We did not find any commercial tool with a process drift detection mechanism. We only report the tools that provide the source code or an executable interface due to space limitations.

The first identified tool for process drift detection is the Concept Drift plugin in ProM[1]. Different approaches have been implemented in this plugin [5,6,10] to address change point detection for sudden and gradual drifts and multi-order dynamics. The user can also search for change localization by selecting a pair of activities; in this case, the tool checks if there is a change between the selected activities. The user has to select many options for using the plugin: log configuration (join logs, split the log or not), feature to be compared (global or local), parameters of the feature, window strategy (fixed or adaptive, sliding over traces or time periods) with parameters, type of drift to detect (gradual or sudden),

[1] ProM is an open source framework that provides a big set of tools for the discovery and analysis of process models from event logs: http://www.processmining.org.

and the statistical test applied along its parameters. The user is required to set many configurations before even know if there is potential drift in the event log. Furthermore, the two types of drifts detected (sudden and gradual) must be separately checked. The plugin is designed for offline analysis, and it is not working if any global feature is selected, thus raising an exception. The results are highly sensitive to the parameters chosen, and there is no user-friendly interface to compare the results from different configurations. It is not possible to obtain the accuracy of the method, e.g., *F-score*, for synthetic logs in the plugin's interface.

Apromore[2] is another academic tool that provides a plugin for concept drift detection reporting change points, change characterization, and localization (ProDrift). ProDrift has different approaches implemented [8,9,12–14] and can detect drifts from a stream of traces (based on runs, which is an abstraction for the traces) or event streams. The tool can detect sudden and gradual drifts from event streams in an integrated way, i.e., the user does not have to specify the type of drift to be detected. ProDrift also has different types of parameters: approach (runs or events), windowing strategy (fixed or adaptive), and window size. The event-based approach includes a noise filter threshold, drift detection sensitivity, characterization (on/off), characterization method (activity-based or fragment-based), and characterization noise filter threshold. The accuracy of the detection method is highly sensitive to the chosen parameter values. An advantage of Apromore is that it has default values for each parameter. Yet, a drawback is that it is not possible to quickly check the differences in the process model before and after a detected change point. The adaptive window approach uses an initial window size value as input, but the plugin applies an algorithm to initialize it if the user does not specify. However, this initial size definition is not explained in the papers supporting the implementation [8,9,12–14].

The identified experimental tools for process drift detection are: Process Drift Detector Plugin for ProM [16], Tsinghua Process Concept Drift Detection (TPCDD) [21], Concept-Drift in Event Stream Framework (CDESF) [3,11], Visual Drift Detection (VDD) [18,19], Dynamic Outlier Aggregation (DOA) [20], Online Trace Ordering for Structural Overviews (OTOSO) [15]. These tools share the drawback of ProM and Apromore, i.e., drift detection accuracy is highly affected by the hyperparameter configuration. Some of them are essentially affected by the window size: TPCDD, CDESF (the window size is defined by the time-horizon parameter, controlling model updates), and VDD. The approaches based on clustering, i.e., TPCDD, CDESF, VDD, DOA, and OTOSO, are also sensitive to the clustering hyperparameters. The Process Drift Detector Plugin for ProM applies an adaptive window approach aiming to solve the window size choice, but the adaptation is limited to user-given upper and lower bounds.

The remaining issue for the available tools is hyperparameter setup. The choice of the window size is critical in any drift detection approach because a small window size may lead to false positives, and a large one may lead to false

[2] Apromore is a collaborative business process analytics platform with distinct editions. The ProDrift is an experimental plugin: https://apromore.org/platform/tools/.

negatives making it challenging to pinpoint the exact location of the drift [9]. Hyperparameter tuning is essential for providing a tool for detecting drifts in real-world scenarios. The current tools do not provide accuracy metrics neither reveal process evolution, turning the hyperparameter tuning an arduous task. By default, the tools focus on reporting the change points and leaving the user to understand the process model's change by splitting the log and applying discovery techniques in all the resulted sub-logs. Our proposal (IPDD) fills this gap by providing an interactive process drift detection tool, easy to use, and a reduced number of parameters. The interface allows the user to quickly verify the detection results by visualizing the process models over time.

3 Interactive Process Drift Detection Framework

The proposed framework (IPDD) aims at overcoming the reported issues of the available tools: a not so user-friendly interface, the difficulty in comparing results obtained by different parameter configurations (not allowing hyperparameter tuning), complex configuration, and not reporting the accuracy metrics in a common manner. Figure 1 shows an overview of the proposed framework.

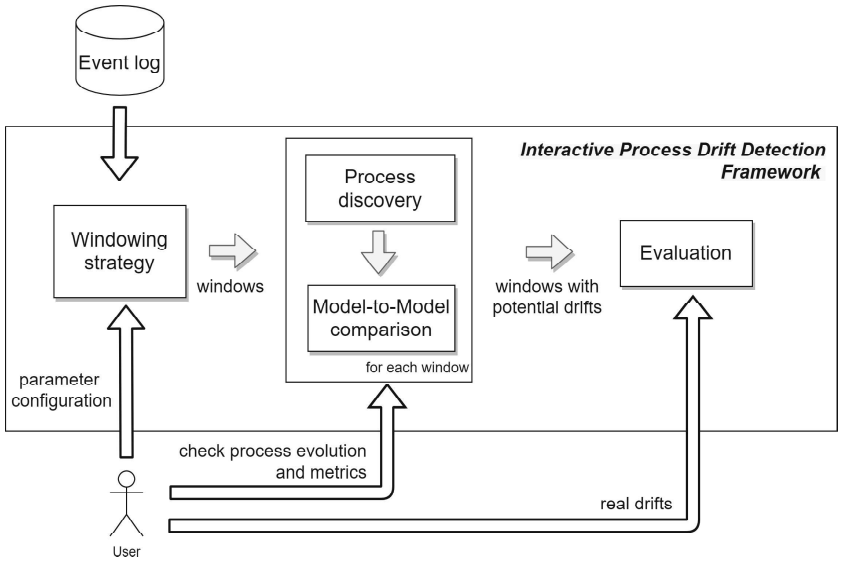

Fig. 1. Overview of interactive process drift detection framework.

The definition of the parameters' values is challenging when starting a process drift analysis because the user has to determine *a priori* what will be a "good" parameter for each situation. We propose to solve this issue by providing an interactive and easy user interface, leaving the user free to test different parameters quickly. The commercial tools for process discovery (e.g., Disco) inspired

this format, as they usually present a simplified version of the process model in which the user can navigate by zooming in or out to understand the process. The goal is to allow the user to navigate between different granularities of change and inspect the process evolution. The hyperparameter tuning is allowed by the user interface, which provides the process evolution for each tested parameter configuration. The user can also verify the chosen parameters' accuracy by checking the evaluation module.

3.1 Windowing Strategy

An event log in XES format[3] can be uploaded into the IPDD. Next, by applying a windowing strategy, the events are split into separated windows. The window strategy can consider the event log as time series of traces, i.e., a stream of traces, by ordering the traces by the first event's timestamp. The windows can also be defined based on events, sorted by their timestamp, i.e., time series of events. The size of the window can be fixed or adaptive. Using fixed-size windows, the user must specify the size as the number of traces or event or as a time window, e.g., hours, days. The windows can be overlapping or non-overlapping, continuous, or non-continuous. All of the identified options can be implemented in the windowing strategy step of the framework. The window slots containing a set of traces/events are forward to the next step, named *process discovery*.

3.2 Process Discovery

For each window slot provided in the first step, IPDD applies a process discovery algorithm to derive a process model. Several algorithms have been proposed for process discovery, and any discovery technique has its own representational bias, i.e., the process model that can be discovered [1]. The resulted process models can be declarative or imperative, and several notations are available [1]. Another option is to use a DFG, also named process map, for simplicity and scalability. This option simplifies the mined process models but still shows relevant insights about the process paths with metrics.

The discovery algorithm's choice is related to the resulted process model, its ability to filter noise, its performance, etc. In the *process discovery* step, any implemented discovery algorithm can be called, like a black-box. The derived models are forward to the next step. An advantage of using the process model is that the framework can quickly show the process changes over time to the user.

3.3 Model-to-Model Comparison

In the previous step, IPDD derives a process model for each window slot. Comparing models from adjacent windows allow identifying differences. Any difference can be a potential drift, and the type of change that can be detected is related to the metric chosen for this comparison. There are several metrics for

[3] See www.xes-standard.org for detailed information about the standard.

comparing process models [4], and they are related to the notation of the process model mined. In this step, IPDD calculates the implemented similarity metrics between adjacent models.

The metric should be adherent to the process model derived by the chosen discovery algorithm. There is no limitation about the number of metrics that can be included, and the user can select one or more of the available metrics. IPDD implements a timeout mechanism for avoiding freezing the user interface when calculating the metrics. If the timeout is reached, only the finished metric results are shown. The result of this stage is reporting each metric's value. If a metric has a value bounded in the [0, 1] interval, with one indicating that the two models are similar, a potential drift can be considered when this metric value is below one. Any metric with a value indicating dissimilarity triggers an update in the user interface, marking the window as a potential drift.

We choose to use a model-to-model comparison instead of trace comparison scheme, e.g., applying statistical or clustering approaches, because we can identify drifts that affect the process model. The chosen process model notation can provide the detection of different drifts. For instance, if we choose to generate a model with the frequencies annotated in the path between activities, IPDD can implement a metric for comparing these frequencies. In this example, IPDD can handle control-flow drifts that do not affect the structure of the process model but affects the routing of cases. It is also possible to localize the drift in the process model by checking the similarity metrics' differences.

3.4 Evaluation

IPDD receives the real drift position as trace/event indexes or date/time values. It can then calculate an accuracy metric to measure if the windows reported as potential drifts include the real drifts. We identified two metrics for measuring the accuracy of concept drift in process models: *F-score* and *mean delay*, reported in [8–10,12–14,16,21]. IPDD reports a drift by indicating the window that renders a model dissimilar to the one obtained in the previous window, so the *F-score* can be applied to evaluate the accuracy of the detected drifts.

The *F-score* represents the harmonic mean of recall and precision, calculated based on the true positives (TP), false positives (FP), and false negatives (FN). It is critical to define that a TP should consider an interval of indexes because the detection mechanism cannot detect the drift by the time it has occurred. In other words, if the real drift initiates in the i-th trace, a TP occurs when the detection method reports a drift in the interval $[i, i + \text{et}]$, where et indicates an error tolerance and should be configured. The FPs and FNs should also be consistent with the definition of the TP. The *mean delay* represents the distance between the occurrence of the real drift and the drift flagged. This distance relates to the windowing strategy adopted. For instance, if the instantiation used a window over traces, this distance is the difference between the trace index of the real drift and the detected one.

4 Results

We implemented a prototype to validate the IPDD and its user interface[4]. This prototype is an instantiation of the framework that encompasses the following:

1. **Windowing strategy.** Non-overlapping and continuous windows of traces (ordered as time series, based on the first activity's timestamp). The windows have a fixed size of traces defined by the user.
2. **Process discovery.** We applied the Pm4Py[5] framework to discover the DFG, with the frequencies of activities and paths.
3. **Model-to-model comparison.** We calculated the node similarity (NS) and edge similarity (ES) scores between two consecutive process maps (P and Q). NS is calculated using Eq. 1 [2], where n_p and n_q are the number of activities in process maps P and Q, respectively, and n_{cs} indicates the number of common activities between P and Q. ES is calculated using Eq. 2, which is similar to NS, however using: e_p is the number of edges in P, e_q is the number of edges in Q, and e_{cs} indicates the number of common edges in both P and Q.

$$NS = 2 * n_{\text{cs}}/(n_p + n_q) \tag{1}$$

$$ES = 2 * e_{\text{cs}}/(e_p + e_q) \tag{2}$$

   The prototype calculates both metrics, and if one or both is less than zero, IPDD marks the current window as a drift.
4. **Evaluation.** There is no evaluation metric already implemented, but we have the *F-score* metric defined to measure the detected drifts' accuracy. Because of the window strategy choice, a TP represents a window reported as a drift containing a trace inputted as a real drift. An FP should be counted when a window reporting a drift does not contain any of the traces inputted as real drifts. Finally, an FN should be incremented when a window that does not report a drift contains any traces inputted as real drifts.

Figure 2 shows a snapshot of the prototype. After setting the window size, the prototype mine the models and calculate the similarity metrics between adjacent models (NS and ES). The window with a similarity metric value below one is marked in red, indicating a drift. The user can check the process mined for this window and verify the metric value and the differences. In Fig. 2 (displayed in the lower-left corner), the edges from activity "Assess eligibility" to "Send acceptance pack" and to "Send home insurance" are included, indicating what has changed in the model. In this example, the activities "Prepare acceptance pack" and "Check if home insurance quote is requested" are optional after the drift; the new edges allow skipping both activities after performing "Assess eligibility".

To validate the tool's usage, we apply the drift detection in some of the logs publicized in [9]. The dataset contains 72 logs with different change patterns

[4] Available at https://github.com/denisesato/InteractiveProcessDriftDetectionFW.
[5] PM4Py is a python open source PM platform: https://pm4py.fit.fraunhofer.de/.

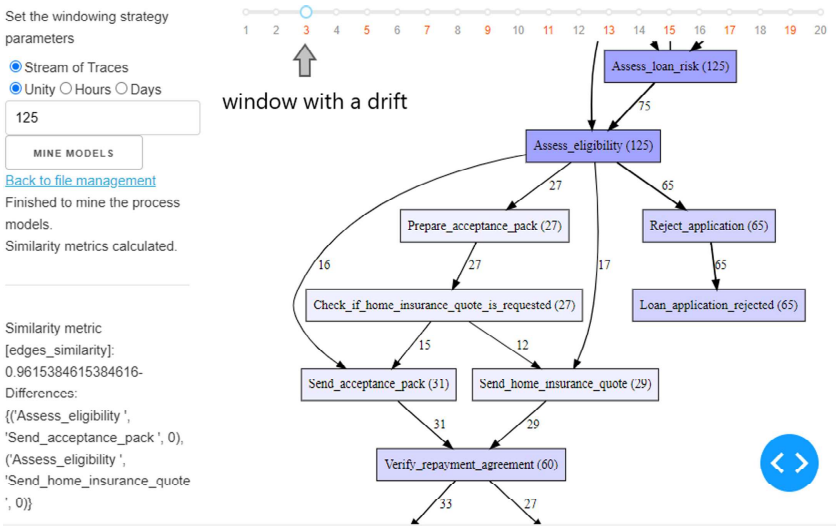

Fig. 2. Snapshot of the prototype implementation.

Table 1. Experiments description.

Change pattern	Category	Drifts detected?	Metric that detects the drifts
cb: make fragment skippable/non-skippable	O	Yes	ES
cd: synchronize two fragments	R	Yes	ES
cf: make two fragments conditional/sequential	R	Yes	ES
cm: move fragment into/out of the conditional branch	I	Yes	ES
cp: duplicate fragment	I	Yes	ES
fr: change branching frequency	O	No	-
lp: make two fragments loopable/not-loopable	O	Yes	ES
pl: make two fragments parallel/sequential	R	Yes	ES
pm: move fragment into/out of parallel branch	I	Yes	ES
re: add/remove fragment	I	Yes	NS and ES
rp: substitute fragment	I	Yes	NS
sw: swap two fragments	I	Yes	ES
IOR	IOR	Yes	NS and ES
IRO	IRO	Yes	NS and ES
OIR	OIR	Yes	NS and ES
ORI	ORI	Yes	NS and ES
RIO	RIO	Yes	ES
ROI	ROI	Yes	NS

and inter-drift distances (distance between each injected drift). The authors use a business process for assessing loan applications containing 15 activities and different control-flow structures as the base model. Next, they injected different types of control-flow changes, simulating sudden recurring drifts (9 drifts in each log). The base model was changed using 12 simple change patterns (described in [17]) and 6 complex patterns (the composition of 3 simple patterns). Each change pattern is injected using 4 inter-drift distances (250, 500, 750, and 1,000). We select all change patterns with the inter-distance of 500 to validate IPDD (Table 1). For the change patterns *lp* and *re*, we used the files named 2.5k as these files contain inter-drift distance of 500. Each pattern is categorized as Insertion (I), Resequentialization (R), and Optionalization (O). For the complex pattern, the authors randomly apply one pattern from each category in a nested way.

Table 1 shows that the implemented instantiation for IPDD correctly detects drifts for 17 out of the 18 patterns using NS and ES. The detection is possible when configuring a window size equal to the inter-drift distance (500). The *fr* pattern is not detectable because there is no structural difference between the models; the drift only changes the frequencies in one branch of the process. The choice of the window size is still an issue. If the window size is configured with a value higher than the inter-drift distance, the drift will not be detected.

5 Conclusion

IPDD has been validated by a prototype implementation that demonstrates its use for detecting sudden drifts in the control-flow perspective. The main contribution of the novel approach is the interactive user interface, which provides the user with a tool for quickly checking different values for window sizes. The drift can be visually checked by analyzing the process models from adjacent windows and the metrics' value describing the detected differences. The implemented metrics (NS and ES) can detect 17 (from 18) change patterns in the public dataset. The window size choice is still a challenge, but the interactive user interface provides an easy way of testing different values. We plan to extend IPDD by including new instantiations implementing the defined evaluation metric (*F-score*) and a similarity metric related to the frequencies between activities.

References

1. van der Aalst, W.M.: Process Mining: Data Science in Action. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. Akkiraju, R., Ivan, A.: Discovering business process similarities: an empirical study with SAP best practice business processes. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 515–526. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17358-5_35
3. Barbon Junior, S., Tavares, G.M., da Costa, V.G.T., Ceravolo, P., Damiani, E.: A framework for human-in-the-loop monitoring of concept-drift detection in event log stream. In: WWW 2018: Companion Proceedings of the The Web Conference 2018, vol. 2, pp. 319–326. Association for Computing Machinery (ACM) (2018)

4. Becker, M., Laue, R.: A comparative survey of business process similarity measures. Comput. Ind. **63**(2), 148–167 (2012)
5. Bose, R.P.J.C., van der Aalst, W.M., Žliobaite, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE Trans. Neural Netw. Learn. Syst. **25**(1), 154–171 (2014)
6. Bose, R.P.J.C., van der Aalst, W.M.P., Žliobaitė, I., Pechenizkiy, M.: Handling concept drift in process mining. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 391–405. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21640-4_30
7. van der Aalst, W., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28108-2_19
8. Maaradji, A., Dumas, M., Rosa, M.L., Ostovar, A.: Detecting sudden and gradual drifts in business processes from execution traces. IEEE Trans. Knowl. Data Eng. **29**(10), 2140–2154 (2017)
9. Maaradji, A., Dumas, M., La Rosa, M., Ostovar, A.: Fast and accurate business process drift detection. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 406–422. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_27
10. Martjushev, J., Bose, R.P.J.C., van der Aalst, W.M.P.: Change point detection and dealing with gradual and multi-order dynamics in process mining. In: International Conference on Business Informatics Research, pp. 1–15 (2015)
11. Mora, D., Ceravolo, P., Damiani, E., Tavares, G.M.: The CDESF toolkit: an introduction. In: ICPM Doctoral Consortium and Tool Demonstration Track 2020, vol. 2703, pp. 47–50 (2020). CEUR-WS.org
12. Ostovar, A., Leemans, S.J.J., Rosa, M.L.: Robust drift characterization from event streams of business processes. ACM Trans. Knowl. Discovery from Data **14**(3), 1–57 (2020)
13. Ostovar, A., Maaradji, A., La Rosa, M., ter Hofstede, A.H.M.: Characterizing drift from event streams of business processes. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 210–228. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_14
14. Ostovar, A., Maaradji, A., La Rosa, M., ter Hofstede, A.H.M., van Dongen, B.F.V.: Detecting drift from event streams of unpredictable business processes. In: Comyn-Wattiau, I., Tanaka, K., Song, I.-Y., Yamamoto, S., Saeki, M. (eds.) ER 2016. LNCS, vol. 9974, pp. 330–346. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46397-1_26
15. Richter, F., Maldonado, A., Zellner, L., Seidl, T.: OTOSO: online trace ordering for structural overviews. In: Leemans, S., Leopold, H. (eds.) ICPM 2020. LNBIP, vol. 406, pp. 218–229. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72693-5_17
16. Seeliger, A., Nolle, T., Mühlhäuser, M.: Detecting concept drift in processes using graph metrics on process graphs. In: Proceedings of the 9th Conference on Subject-oriented Business Process Management, S-BPM ONE 2017, vol. Part F1271 (2017)
17. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data Knowl. Eng. **66**(3), 438–466 (2008). ISSN 0169023X
18. Yeshchenko, A., Di Ciccio, C., Mendling, J., Polyvyanyy, A.: Comprehensive process drift detection with visual analytics. In: Laender, A.H.F., Pernici, B., Lim, E.-P., de Oliveira, J.P.M. (eds.) ER 2019. LNCS, vol. 11788, pp. 119–135. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33223-5_11

19. Yeshchenko, A., Mendling, J., Ciccio, C.D., Polyvyanyy, A.: VDD: a visual drift detection system for process mining. In: ICPM Doctoral Consortium and Tool Demonstration Track 2020 (2020). CEUR-WS.org
20. Zellner, L., Richter, F., Sontheim, J., Maldonado, A., Seidl, T.: Concept drift detection on streaming data with dynamic outlier aggregation. In: Leemans, S., Leopold, H. (eds.) ICPM 2020. LNBIP, vol. 406, pp. 206–217. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72693-5_16
21. Zheng, C., Wen, L., Wang, J.: Detecting process concept drifts from event logs. In: Panetto, H., et al. (eds.) OTM 2017. LNCS, vol. 10573, pp. 524–542. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69462-7_33