Detecting Branching Condition Changes in Process Models

Yang Lu[0000−0002−9002−8650], Qifan Chen[0000−0003−1068−6408], and Simon K. Poon[0000−0003−2726−9109]

School of Computer Science, The University of Sydney, Sydney, NSW 2006, Australia
{yalu8986, qche8411}@uni.sydney.edu.au
simon.poon@sydney.edu.au

Abstract. Business processes are continuously evolving in order to adapt to changes due to various factors. One important process drift perspective yet to be investigated is the detection of branching condition changes in the process model. None of the existing process drift detection methods focus on detecting changes of branching conditions in process models. Existing branching condition detection methods do not take changes within the process into account, hence results are inadequate to represent the changes of decision criteria of the process. In this paper, we present a method which can detect branching condition changes in process models. The method takes both process models and event logs as input, and translates event logs into decision sequences for change points detection. The proposed method is evaluated by simulated event logs.

Keywords: Process science · Data science · Concept drift detection · Branching condition changes.

1 Introduction and Motivation

Business processes are continuously evolving in order to adapt to changes. In process science, such changes are referred as process drifts. Changes of processes can occur from many perspectives. However, most of existing research papers only focus on detecting and understanding process control-flow structure changes (eg. the order of two activities swap after a certain time).

There are typically many decision points in process models. Studies such as [21, 22, 23, 25] propose methods to discover branching conditions for these decision points. However, those methods assume the conditions do not change. Without taking process drifts into consideration, the results can be less accurate.

Fig.1 shows a simple online payment process. Initially, when the amount is larger than $1000, a second authentication is required, and users need to type in the SMS codes sent from their banks. A payment will only be processed after the SMS code has been verified. However, to ensure payment security, the amount is changed into $500 after a certain time point. It is clear to see that we cannot accurately discover the branching conditions without the knowledge of the change point.

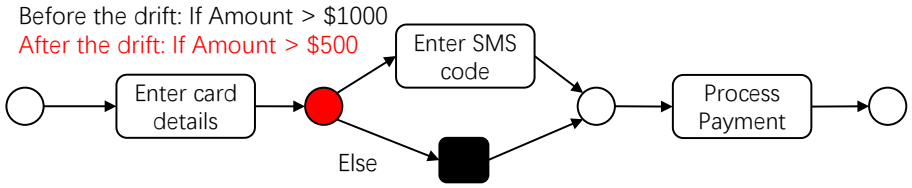

Fig. 1. An example branching condition change

A previous study in [2] proposes an approach to discover branching frequency changes in process models (i.e. changes in frequencies between different options when there is an exclusive choice). In this paper, based on [2], we propose a method to detect branching condition changes. The method takes both event logs and process models as input and translates event logs into decision sequences. External change points detection methods can then be applied to discover branching condition changes. To the best of our knowledge, this is the first paper proposing a method to discover branching condition changes in process models. The rest of the paper is structured as follows: Section 2 is a literature review of related work, Section 3 introduces the previous research to discover branching frequency changes in process models. Our proposed method is introduced in Section 4, and the evaluation results are reported in Section 5. We finally conclude our paper in Section 6.

2 Background

Traditionally, studies on process drift focus on the control-flow structures. Studies such as [1, 3, 4, 5] treat each process drift as a time point when there is a significant change among process behaviours. [1, 3, 4, 5] use a sliding window to obtain consecutive samples among event or trace streams converted from event logs. Then statistical tests such as the chi-square tests are applied to see if there are significant differences. A process drift point is reported if a significant change is found. The main goal of those methods are to locate the process control-flow change points as accurately as possible and to reduce the impact of noises.

Instead of discovering the time points of process drifts, the aims of studies such as [8, 9, 10, 13, 14] are to provide comprehensive results to users (i.e. to show user what have been changed in the process). Based on the process drift detection method in [5]. Ostovar et al. [10] applies the inductive miner to get a process tree between each pair of process drift points and uses nature languages to describe the differences between each pair of consecutive process trees. Yeshchenko et al. [8, 9] use Declare miners to represent the process and find if there is a change to each declare constraint. In addition, a comprehensive visualisation is also provided to users. Although these methods are of benefits to provide comprehensive information about process drifts to users, they often only focus on the control-flow perspective.

Some studies focus on process drifts other than the control-flow perspective. For example, [2] explores the changes of frequencies between different options when there is an exclusive choice in process models. Stertz et al. [17] detects the change of event attribute values in business processes in an online setting. Brockho et al. [20] applies the earth mover's distance to detect process drifts from both the control-flow and time perspective.

Other studies focus on the cause-effect relationship between data and control-flow perspective. For example, Stertz et al. [18] and Adams et al. [19] find change points from multiple perspectives and try to determine if changes from one perspective can potentially cause changes from another perspective (eg. if the change of room temperature can cause the change of the process control flow structure).

Most process discovery algorithms have the ability to discover decision points, however unable to determine their branching conditions based on the data flow [15]. Several studies about detecting branching conditions in process models have been proposed to address the gap such as [21, 22, 23, 25]. The core idea of those methods is to find related data attributes at the time when a decision is made. Other approaches (eg. decision tree algorithms) are then applied to discover branching conditions. Those methods could lead to inaccurate results when the detected branching conditions are changing.

In summary, current process mining techniques could produce inaccurate results without the knowledge of concept drift detection. There are no existing approaches which can detect branching condition changes in business processes. A method which can detect changes of branching conditions is needed in this field.

3 Discovering Branching Frequency Changes in Process Models

The method we propose in this paper is an extension to a previous research in [2]. Lu et al. [2] proposes a method to discover branching frequency changes in process models. In this section, we briefly describe the previous approach. Firstly, some formal definitions are introduced.

Definition 1 (Perti net). *A Petri net N is a triple (P, T, F) where P is a set of places, T is a set of transitions and* $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs between transitions and places. For place* $p \in P$, $\bullet p \subset T$ *are the incoming transitions of p, and* $p\bullet \subset T$ *are the outgoing transitions of p.*

Definition 2 (Decision point and Branching frequency). *A decision point D in a Petri net N = (P, T, F) is a tuple (p, B) where* $p \in P, B \equiv p\bullet$. *Label is a function which assigns each branch a unique label (i.e.* $Label(b) = L_b$, $b \in B$*). The branching frequency* $P(b)$ *for* $b \in B$ *from time* t_0 *to* t_1 *is the probability of choosing b when the decision point D is executed.* $\sum_{b \in B} P(b) = 1$.

Definition 3 (Branching frequency change).

A branching frequency change point for a decision point $D = (p, B)$ *is a time point t, where there are significant changes among* $P(b)$, *where* $b \in B$.

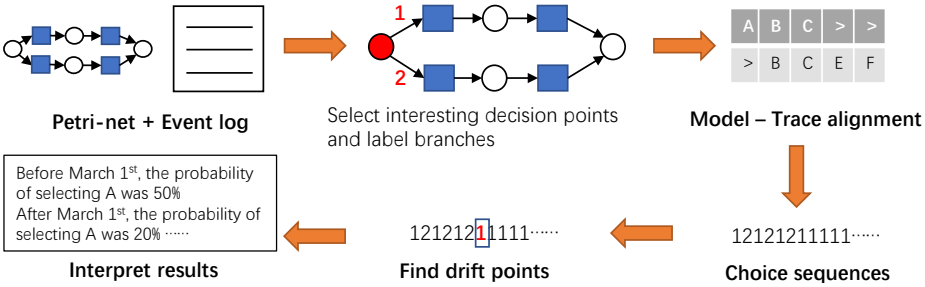

Fig. 2. An overview of the branching frequency detection method in [2]

Fig.2 shows an overview of the method in [2]. The method takes both Petri nets and event logs as input. Users are asked to select the decision points they are interested in. For each decision point D selected, a label L_b is assigned for each $b \in B$. In [2], an integer is used to label the branches. Then model-trace alignment is used to identify the time when a decision point is executed and also the branch selected for the corresponding execution. Each decision point has a choice sequence, and each time it is executed, a new element will be added into it. The formal definition of choice sequence is given below.

Definition 4 (Choice sequence). *A choice sequence for a decision point D is a sequence of tuples* $\{t, L_b\}$ *where t is the execution time of the decision point, and* L_b *is the label of the branch selected. The choice sequence is sorted by the timestamps.*

An example of discovering choice sequences is presented in Fig.3. For each synchronize move (or silent move) in the alignment, if its corresponding transition $t \in B$, and there is an interesting decision point $D = (p, B)$, a new element can be added into the choice sequence.

Finally, we can use existing change point detection methods on the choice sequence to discover branching frequency changes. In [2], the external library called Ruptures [7] is used to detect change points, which is a python package containing a collection of change point detection algorithms for various dynamic systems.

4 The Proposed Method

In this section, we introduce the methods we propose to detect branching condition changes in process models.

Definition 5 (Branching condition). *A branching condition* c_b *for a branch b in decision point D (i.e.* $D = (p, B)$, $b \in B$*) from time* t_0 *to* t_1 *is a set of rules* R_b*. Each rule is associated with a set of data attributes and values.*

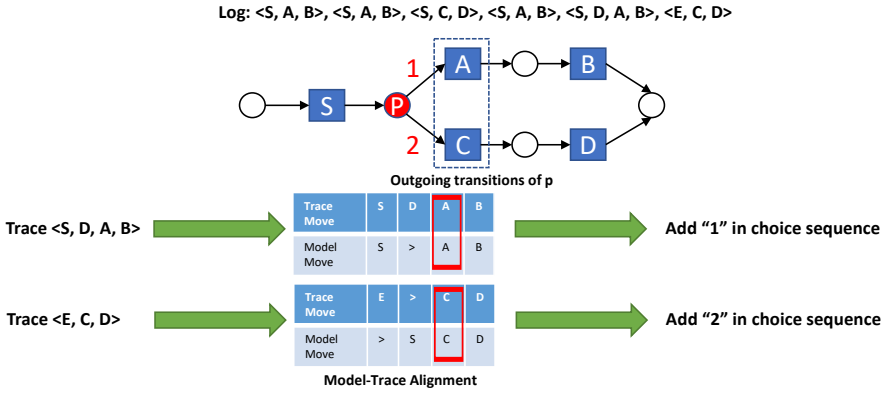

Fig. 3. Discovering Choice Sequences

Definition 6 (Branching condition change). *A branching condition change for a decision point $D = (p, B)$ is a time point t, where there are significant changes among c_b, $b \in B$.*

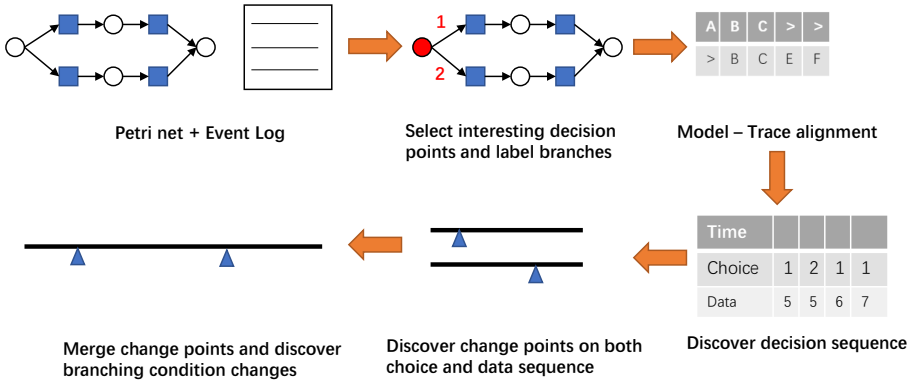

Fig. 4. An overview of the proposed method

Fig.4 shows an overview of our proposed method. Similar to [2], the method takes both Petri nets and event logs as input. Based on the inputs, the method will convert the event log into a decision sequence which contains both a choice sequence and data flow. Change point detection methods are finally applied to discover branching condition change points.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 10 September 2021 doi:10.20944/preprints202109.0191.v1

4.1 Select Interesting Decision Points and Label Branches

Firstly, users need to choose the decision points they are interested in to investigate the change of branching conditions. For each selected decision point $D = (p, B)$, an integer label L_b is given to each branch (i.e. $b \in B$, $L_b = Label(b)$).

4.2 Model-Trace Alignment and Discover Decision Sequence

In Section 3, we introduce the concept of choice sequence, which is the sequence of decisions made each time the decision point is executed. In this section, we extend the concept to decision sequence, which contains not only the sequence of decisions, but also the sequence of data attributes and values which can impact the decision made. The formal definition of decision sequence is given in Definition 7.

Definition 7 (Decision sequence). *A decision sequence for a decision point D is a sequence of tuples $\{t, L_b, Data\}$ where t is the execution time of the decision point, L_b is the label of the branch selected. Data is the set of data attributes and values which can impact the decisions made. The decision sequence is sorted by the timestamps. A decision sequence for a decision point D can be seen as a $m \times n$ matrix, where $m - 2$ is the number of data attributes which can affect the decisions made, and n is the number of times the decision point D has been executed.*

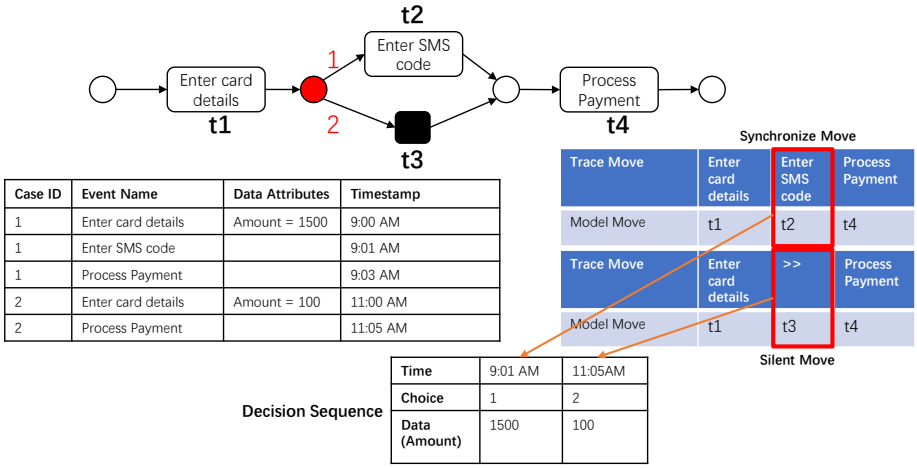

Fig. 5. Discovering decision sequences

The method to discover the decision sequence is very similar to the one to discover choice sequence in Section 3. Firstly, model-trace alignment is performed

to align each trace with the model. Whenever there is a synchronize move (or silent move), if $t \in B$ is its corresponding transition, and there is a decision point $D = (p, B)$, a new element can be added into the choice sequence. An example of the process is shown in Fig.5, and the interesting decision point is marked in red colour. The two possible choices are labeled as "1" and "2" respectively. For the first case, we find a synchronize move on transition t_2. For the second case, we find a silent move on transition t_3. As a result, we can conclude that for the first case, a SMS code is required, and it is not required for the second case. The decision point is executed two times in total for the two cases. Thus, two elements are added into the choice sequence. For the first case, as there is a synchronize move on t_2, we take the time of "Enter SMS code" directly as the decision time. For the second case, since t_3 is a hidden transition, we can take either the time of "Enter card details" or "Process payment" as the decision time. In this example, we choose the later one.

It has to be noted that in this paper, we assume the data attributes which can affect the decision is known. Discovering which attributes can affect the decision is beyond the scope of this paper.

4.3 Discover change points on both choice and data sequences

The core idea of the proposed method is that changes of branching frequencies and data attribute values are very likely to indicate branching condition changes. When a change point is found on the decision sequence, there can be three possible scenarios:

1. There is a change on the sequences of data values, but there is no branching frequency change (Fig.6).
2. There is a change on the branching frequencies, but there is no change on the sequences of data values (Fig.7).
3. There are changes on both data values and branching frequencies.

The first two scenarios are very likely to cause branching condition changes. For example, in the branching condition change shown in Fig.6, the only change we detect is the change of amount values, and the probability of receiving SMS code is not changed. Before the change point, 50% of the amount values are larger than 1000, and 50% of the amount values are smaller than 1000. After the change point, 30% of the amount values are larger than 1000, and 70% of the amount values are smaller than 1000. Clearly, if the branching condition is not changed, the probabilities between decisions must be changed. As a result, the original condition for "Enter SMS code" ($if Amount > 1000$) can no longer hold.

For the second example shown in Fig.7, the only change we detect is the change of probabilities between the twe decisions. Before the change point, 50% of the transactions require SMS code for verification, which is the same as the proportion of amount values larger than 1000. After the change point, the percentage increases to 70%, but the proportion of amount values smaller than

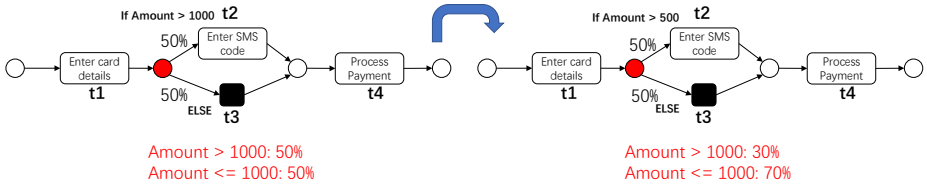

Fig. 6. Changes only on data values

1000 remains unchanged. As a result, the original condition for "Enter SMS code" ($if Amount > 1000$) can no longer hold.

When there are changes on both the branching frequencies and data value sequences with similar timestamps, there could still be incidences of branching condition changes. Further operations are needed to validate these change points.

All three scenarios stated above could be considered as incidences of changes of branching conditions. In this step, we discover all change points on the decision sequences. We can then apply existing change point detection algorithms. The pipeline of this step is presented in Algorithm 1.

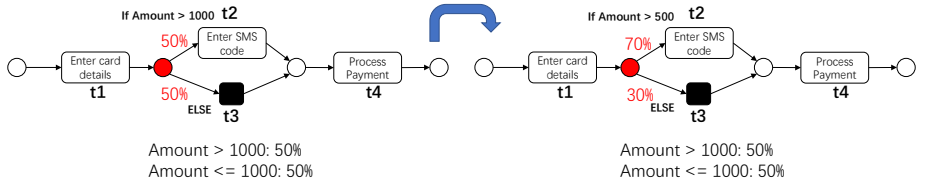

Fig. 7. Changes only on probabilities between decisions

Algorithm 1: Discovering Change Points

Input: A DecisionSequence for decision point D
Output: A list of change points C

```
ChoiceSquence ← DecisionSequence.getChoiceSequence()
C.push(ChoiceSequence.findChangePoints())
Attributes ← DecisionSequence.getDataAttributes()
for attr in Attributes do
    DataSequence ← DecisionSequence.getDataSequence(attr)
    C.push(DataSequence.findChangePoints())
end
```

4.4 Merge change points and discover branching condition changes

By applying Algorithm 1 on the decision sequence, we get a list of possible change points. It has to be noted that it is possible that we get multiple change points with similar timestamps. For example, if the decision sequence contains multiple data attributes (i.e. the decision relies on values of multiple data attributes), and there are changes on values of multiple data attributes, we could find change points with similar timestamps. In this circumstance, we use clustering algorithms to combine the change points (using DBSCAN as the clustering algorithm). DBSCAN needs two parameters: a minimal number of points in each cluster (minPts) and a maximum radius of a neighborhood (eps). In this method, minPts is always set to 1 since a single change point on the data/choice sequence can indicate a branching frequency change. All change points within distances of eps are merged into a single change point. The process is illustrated in Fig.8.

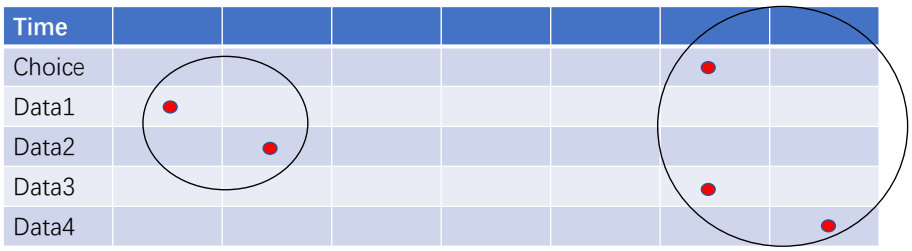

Fig. 8. Merging change points, each red dot represents a detected change point

Finally, we can apply existing methods to discover branching conditions between each pair of merged change points. It is important to note that the detected change points may not necessarily reflect all true change points of branching conditions (eg. there are changes on both data values and branching frequencies, and the original branching conditions still hold after the changes). If the same branching conditions are discovered before/after a certain change point, it can be ignored as a post-processing step.

5 Evaluation

We reference the artificial process model from [24] to design our experiment and build our simulated event logs. The process describes a simple procedure a sales representative needs to follow when an order is received. After entering product and quantity, if the total amount is larger than a certain amount, a department manager approval will be required. In our experiment, when the order date is within 2020, the amount is 200k. In 2021, the amount is changed into 400k. Two artificial logs are created to evaluate the algorithm. Each log contains 1000 traces in 2020 and 1000 traces in 2021.

- Log 1: In both 2020 and 2021, 50% of traces require department manager approval. In 2020, 50% of the traces have an amount value lager than 200k (25% of the traces have an amount value larger than 400k). In 2021, 50% of the traces have an amount value lager than 400k.
- Log 2: In both 2020 and 2021, 50% of the traces have an amount value larger than 200k (There are no changes among the distribution of amount values). In 2020, 50% of traces require department manager approval. In 2021, the percentage drops to 25%).

The method we propose can work with any change point detection algorithms. In this experiment, we use the Pelt algorithm within the Ruptures [7] library as the change point detection algorithm.

Since there is only one decision point in the illustration model, each log is translated into one decision sequence. Change point detection algorithms are then applied on the decision sequences. In both logs, our method successfully identifies a change point at the end of 2020. In the first log, a change point is found on the sequence of amount values. In the second log, a branching frequency change is detected. These results show that the proposed method has potential to detect branching condition changes in process models.

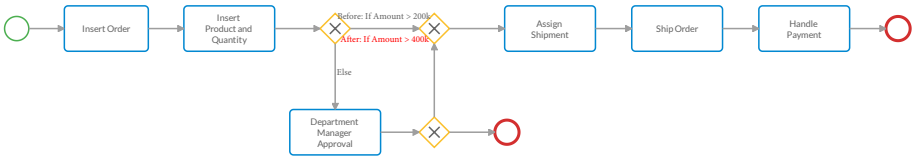

Fig. 9. Changes only on probabilities between decisions

6 Conclusion

In this paper, we present an early-stage work of developing a new method to detect the changes of branching conditions in process models. Our method translates each decision point to be investigated as a decision sequence and then discovers change points on both sequences of data values and branching frequencies. Since changes of branching frequencies and data values are very likely to cause branching condition changes, by merging change points identified on the decision sequence, we can validate if there are branching condition changes before/after each change point.

The limitation of this paper is that due to the lack of real-life data, only simple experimentation on simulated event logs are performed. This preliminary evaluation shows that the proposed method has potential to detect branching condition changes in process models. For future work, we aim at evaluating the method on real-life data sets. In addition, we also aim at exploring the impact of

different change point detection algorithms and data complexity to our proposed approach. Finally, we also aim at implementing the method in online settings.

References

1. Lu, Y., Chen, Q., Poon, S.: A Robust and Accurate Approach to Detect Process Drifts from Event Streams. In: Polyvyanyy A., Wynn M.T., Van Looy A., Reichert M. (eds) Business Process Management. BPM 2021. Lecture Notes in Computer Science, vol 12875. Springer, Cham (2021)
2. Lu, Y., Chen, Q., Poon, S.: Detecting and Understanding Branching Frequency Changes in Process Models. In: Augusto A., Gill A., Nurcan S., Reinhartz-Berger I., Schmidt R., Zdravkovic J. (eds.) Enterprise, Business-Process and Information Systems Modeling. BPMDS 2021, EMMSAD 2021. Lecture Notes in Business Information Processing, vol 421. Springer, Cham (2021)
3. Maaradji, A., Dumas, M., 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, Heidelberg (2015)
4. Maaradji, A., Dumas, M., La Rosa, M., Ostovar, A.: Detecting sudden and gradual drifts in business processes from execution traces. IEEE TKDE 29(10), 2140–2154 (2017)
5. 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)
6. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 2(2), 182–192 (2012)
7. Truong, C., Oudre, L., Vayatis, N.: Selective review of offline change point detection methods. Signal Process. 167 (2020)
8. Yeshchenko, A., Di Ciccio, C., Mendling, J., Polyvyanyy, A.: Visual Drift Detection for Event Sequence Data of Business Processes. arXiv preprint (2020)
9. 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)
10. Ostovar, A., Leemans, S.J. and Rosa, M.L.: Robust drift characterization from event streams of business processes. ACM Transactions on Knowledge Discovery from Data (TKDD), 14(3), pp.1-57 (2020)
11. 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.) PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013)
12. Zheng, C., Wen, L., Wang, J.: Detecting process concept drifts from event logs. In: OTM CoopIS, pp. 524–542 (2017)
13. 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, p. 6:1 (2017)
14. Stertz, F., Rinderle-Ma, S.: Process histories - detecting and representing concept drifts based on event streams. In: Panetto, H., Debruyne, C., Proper, H.A., Ardagna,

C.A., Roman, D., Meersman, R. (eds.) OTM 2018. LNCS, vol. 11229, pp. 318–335. Springer, Cham (2018)
15. van der Aalst, W.M.P.: Process Mining - Data Science in Action. Springer, Heidelberg (2016)
16. Berti, A., van Zelst, S.J., van der Aalst, W.M.P.: Process mining for python (PM4PY): bridging the gap between process - and data science. CoRR abs/1905.06169 (2019)
17. Stertz, F., Rinderle-Ma, S.: Detecting and identifying data drifts in process event streams based on process histories. In: Cappiello, C., Ruiz, M. (eds.) CAiSE 2019. LNBIP, vol. 350, pp. 240–252. Springer, Cham (2019)
18. Stertz F., Rinderle-Ma S., Mangler J.: Analyzing Process Concept Drifts Based on Sensor Event Streams During Runtime. In: Fahland D., Ghidini C., Becker J., Dumas M. (eds) Business Process Management. BPM 2020. Lecture Notes in Computer Science, vol 12168. Springer, Cham (2020).
19. Adams J.N., van Zelst S.J., Quack L., Hausmann K., van der Aalst W.M.P., Rose T.: A Framework for Explainable Concept Drift Detection in Process Mining. In: Polyvyanyy A., Wynn M.T., Van Looy A., Reichert M. (eds) Business Process Management. BPM 2021. Lecture Notes in Computer Science, vol 12875. Springer, Cham (2021)
20. Brockhoff, T., Uysal, M.S. and van der Aalst, W.M.: Time-aware Concept Drift Detection Using the Earth Mover's Distance. In 2020 2nd International Conference on Process Mining (ICPM) pp. 33-40. IEEE (2020)
21. de Leoni, M., Dumas, M., García-Bañuelos, L.: Discovering branching conditions from business process execution logs. In: Cortellessa, V., Varró, D. (eds.) FASE 2013 (ETAPS 2013). LNCS, vol. 7793, pp. 114–129. Springer, Heidelberg (2013)
22. Bazhenova, E., Buelow, S., Weske, M.: Discovering decision models from event logs. In: Abramowicz, W., Alt, R., Franczyk, B. (eds.) BIS 2016. LNBIP, vol. 255, pp. 237–251. Springer, Cham (2016)
23. Rozinat, A., van der Aalst, W.M.P.: Decision mining in ProM. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 420–425. Springer, Heidelberg (2006)
24. Dubinsky, Y., Soffer, P.: Detecting the "Split-Cases" Workaround in Event Logs. In: Augusto A., Gill A., Nurcan S., Reinhartz-Berger I., Schmidt R., Zdravkovic J. (eds) Enterprise, Business-Process and Information Systems Modeling. BPMDS 2021, EMMSAD 2021. Lecture Notes in Business Information Processing, vol 421. Springer, Cham (2021)
25. de Leoni, M., van der Aalst, W.M.P.: Data-Aware Process Mining: Discovering Decisions in Processes Using Alignments. In: Proc. of the 28th ACM Symposium on Applied Computing (SAC 2013). ACM (2013)