# An End-to-End Approach for Online Decision Mining and Decision Drift Analysis in Process-Aware Information Systems: Extended Version

Beate Scheibel[1] and Stefanie Rinderle-Ma[2]

[1]Research Group Workflow Systems and Technology, Faculty of Computer Science, University of Vienna,
beate.scheibel@univie.ac.at
[2] Technical University of Munich, Germany; TUM School of Computation, Information and Technology
stefanie.rinderle-ma@tum.de

**Abstract**

Decision mining enables the discovery of decision rules from event logs or streams, and constitutes an important part of in-depth analysis and optimisation of business processes. So far, decision mining has been merely applied in an ex-post way resulting in a snapshot of decision rules for the given chunk of log data. Online decision mining, by contrast, enables continuous monitoring of decision rule evolution and decision drift. Hence this paper presents an end-to-end approach for the discovery as well as monitoring of decision points and the corresponding decision rules during runtime, bridging the gap between online control flow discovery and decision mining. The approach provides automatic decision support for process-aware information systems with efficient decision drift discovery and monitoring. For monitoring, not only the performance, in terms of accuracy, of decision rules is taken into account, but also the occurrence of data elements and changes in branching frequency. The paper provides two algorithms, which are evaluated on four synthetic and one real-life data set, showing feasibility and applicability of the approach. Overall, the approach fosters the understanding of decisions in business processes and hence contributes to an improved human-process interaction.

## 1 Introduction

Process mining and specifically decision mining allows for increased transparency of processes, which is crucial across all domains [11]. Decision mining is a part

of process discovery, allowing for the discovery of decision points in a process model and the corresponding decision rules guarding that decision based on data elements [7, 19]. A decision rule can consist of multiple conditions, which are usually of the form $v(ariable)$ $op(erator)$ $c(onstant)$, e.g., *temperature below 50°*. Conditions can be concatenated to form decision rules. Decision mining can be seen as a classification problem. Therefore the potential branches that can be chosen and executed are regarded as decision classes.

Existing decision mining methods [7] are applied in an ex-post manner. However, especially when aiming at increased transparency, runtime analysis is particularly interesting, as information about decisions can be communicated to the user in almost real-time. In addition, runtime analysis allows for the prompt detection of decision drift, i.e., the manifestation of changing decision rules and decision points in event logs and streams. Decision drift can occur due to errors or changes in the environment. Detecting drifts is important to ensure correctness and compliance of a process, i.e., assuring that the drift occurred intentionally and not due to errors. This is crucial across domains such as manufacturing to ensure quality of products and health care to ensure quality of patient care. This is especially relevant as *"[e]ffective decision making – that is connected, contextual and continuous – results in a host of business benefits, including greater transparency, accuracy, scalability and speed* [18]. Our previous work [21] introduced an approach for detecting decision rule changes during runtime. However, the approach has several limitations. It is assumed that decision points are already known. Therefore the approach cannot be used as end-to-end approach, i.e. decision discovery methods have to be applied complicating analysis during runtime. In addition, the definition of decision drift has been limited to changes in decision conditions, neglecting decision point changes.

As running example consider a simplified loan application process depicted in Fig. 1. A customer applies for a loan, the application data is checked for completeness. Then either a normal or extensive check is performed. The results of the check contribute to the overall assessment that results in either rejection or acceptance of the loan. Lastly, the assessment is communicated to the customer. The exemplary process includes one decision point, i.e., whether a normal or extensive check is necessary. Multiple data elements are part of the process and serve as basis for the decision, e.g., the requested amount.

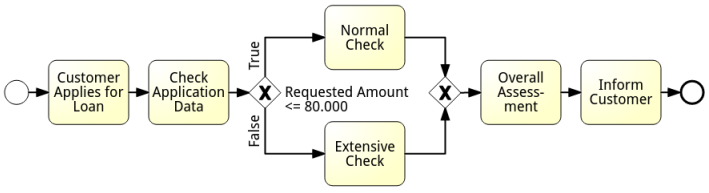



Figure 1: Running Example, Loan Application

Standard decision mining techniques result in the following decision rule:

*IF amount_loan* <= 80.000 *THEN Normal ELSE Extensive*. The decision depends on data element *amount_loan*. Assume that during process execution changes can occur, e.g., a change in regulation leads to stricter checks and therefore an extensive check is already required for any amount greater than 50.000. Other changes might include additional data that becomes available during process execution and can be used to more accurately mine decision rules, e.g., *income*, the addition of an additional branch (class) at the existing decision point, e.g., a branch *Simple Check* is added, or the addition of a new decision point, e.g., a customer is handled differently depending on whether the assessment resulted in rejection or acceptance of the application.

A comprehensive, end-to-end decision mining and monitoring approach should be able to mine and monitor decision rules and decision drifts during runtime as soon and as accurately as possible. End-to-end requires minimum involvement of users for setting up the algorithms and providing meaningful results. Online requires continuous mining with limited storage and appropriate handling of outdated data. These requirements can be addressed based on the following questions: RQ1: What exactly is decision drift and when/why does it happen? RQ2: How to mine decision points, rules, and drifts in a connected and continuous way without prior knowledge of the process model? RQ3: How to deal with limited storage? and RQ4: How to deal with outdated data which might become useful or even detrimental for mining current decision rules?

To address RQ1–4, this paper derives and discusses a definition of decision drift and its triggers based on literature ($\mapsto$ RQ1). In order to meet RQ2, the approach is designed to only require an event stream as input to provide users with information about current decision points, the corresponding rules, and potential drifts. The presented approach is comprehensive as it mines decision points, rules, and drifts in a connected way ($\mapsto$ RQ2). For this, data values as well as the frequency of branching conditions are taken into consideration as indicators for decision drift. If drift is detected, users are notified about the drift and changed decisions, and can check if these changes are intentional ($\mapsto$ RQ2). To account for limited storage and outdated data, a window based approach is taken, where only the recent data is taken into account ($\mapsto$ RQ3 and RQ4). Overall, used in conjunction with a process aware information system (PAIS), that continually provides new event data in the form of an event stream, the approach provides continuous and increased transparency for users.

Section 2 discusses and defines decision drift concepts. Section 3 describes the algorithms, which are evaluated in Sect. 4. Related work is discussed in Sect. 5 and a conclusion is provided in Sect. 6.

## 2 Decision Drift: Definition and Analysis

Decision drift refers to different kinds of changes affecting decisions in a process. A decision is defined by a corresponding decision point in a process model (control flow) and the associated decision rule defining which branch is chosen based on process data (data flow). Decision drift, consequently, can occur due

to control flow change, data flow change, and changes to the decision rule itself. Hence, in the following, we analyze state-of-the art approaches for process change (patterns), changes of data and decisions, as well as concept drift, aiming at achieving an understanding and definition of decision drift.

[24] provide a framework of process change patterns referred to as adaptation patterns (AP). The following APs are relevant for decision drift: AP1 describes the insertion of process fragments, including a conditional insert, i.e., an activity is inserted into a process model together with a condition, i.e., decision point. AP2 refers to the deletion of a process fragment, which can also entail the deletion of a decision point. AP8 describes the addition of a loop, which also involves the addition of a decision point. AP10 refers to the addition of a decision point in the process. Lastly, AP13 refers to modifications of decision rules.

[8] define change patterns in Decision Model and Notation (DMN) models. DMN models consist of a decision requirement diagram (DRD), depicting input of decisions and the dependencies between elements. Elements can either be decision nodes or input nodes. Each decision node can be represented by a table, including multiple decision rules. A decision rule consists of combinations of input and output variables, i.e., decision classes. [8] propose four change categories. First, change within decision rules, i.e., the decision table changes by including or deleting input or output, or a change in the decision logic. Second, change on decision rules in their entirety, i.e., the inclusion or exclusion of a decision rule from a decision table. Third, change of the decision nodes in the DRD, i.e., deleting/adding a decision node (consisting of multiple decision rules). And lastly, change of the input data nodes in the DRD, i.e., including or excluding data as input.

Summarizing the literature analysis results, the following decision changes can potentially occur in a process: changes of data values in a condition, addition/deletion of a condition a decision rule, addition/deletion of data elements in a decision rule, addition/deletion of decision classes in a decision rule, and addition/deletion of decision points in a process model.

Concept drift [2] describes changes in processes with regards to their manifestations in the process execution (logs), i.e., sudden, recurring, gradual, and incremental drift. Accordingly, decision drift can be understood as manifestation of decision changes in process execution logs (ex post) or event streams (online). With respect to their detection and monitoring, decision drifts can be further classified in changes of decision rules (incorporating changes in conditions), decision classes, and decision points. Changes on higher levels, i.e., decision points, classes or rules, can also entail changes on lower levels, i.e., conditions, rules or classes. In [21], a significant drop in accuracy when predicting newly incoming instances is used as sign that a decision drift occurred. [12] suggests detecting changes in decision rules by monitoring the branching frequency, i.e., how often a specific branch is chosen. [23] looks at changes in data values to determine if a concept drift occurred. Overall, decision drift can be detected based on (1) decreased performance, (2) changing branching frequency and (3) changes in data elements or data ranges. A comprehensive decision drift

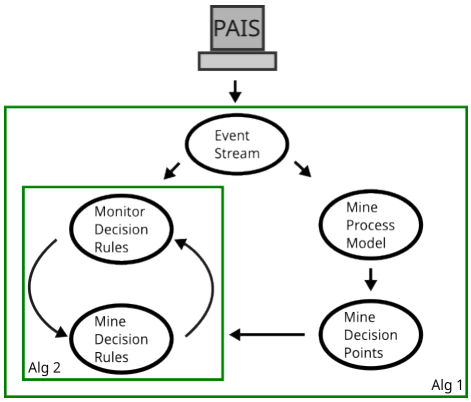



Figure 2: End-to-End Approach – Overview

analysis approach should be able to monitor occurrence of (1)–(3) in order to detect decision drift.

# 3 End-to-End Runtime Decision Mining, Monitoring, and Decision Drift Detection Approach

The overall approach is depicted in Fig. 2. The input is an event stream, emitted by, e.g., a PAIS, which is used to mine a process model using online process discovery methods (for an overview of online methods see [3]). The process models are the basis for determining the decision points and the corresponding decision classes, which, in turn, are the basis to mine decision rules for each of the decision points. The decision rules are continuously monitored, using newly incoming events. If either the performance, the frequency of taken branches, or the data value ranges change, we assume that decision drift occurred and remining is performed. Therefore the approach consists of two continuous processes: first, the process discovery part continues considering newly occurring events and remining the process model if necessary. If this leads to new or changed decision points, decision rules are remined as well (cf. Alg. 1). Second, the existing decision rules are continually checked for compliance with newly incoming events and remined if necessary (cf. Alg. 2).

Algorithm 1 reflects the overall framework, combining existing methods for process discovery and decision point analysis with new techniques for decision rule mining to achieve a fully automatic end-to-end approach. Algorithm 1 calls Alg. 2 for decision rule and remining in case of decision drifts.

Algorithm 1 works as follows: as soon as a new event occurs it is stored in a queue, which continues to store newly incoming events, while the next phases of the approach are executed in parallel. Therefore, even while process models and decision rules are discovered, new incoming elements are not lost. Each event is stored in the queue and will be processed. In addition to the event stream,

**Algorithm 1** Runtime Decision Mining, End-to-End Approach

**Input: Queue of Elements From Event Stream, Grace Period**
**Output: Decision Points, Decision Rules for Each Decision Point**

```
Trace_Dict = {}, Window Size WS = Grace Period
while Element in Queue do
    Adapt Directly-Follows-Graph(Element), using Lossy Counting
    HN = Make Heuristics Net(Directly-Follows-Graph)
    if New HN then                                   ▷ Find Decision Points
        Find Places with Multiple Outgoing Arcs as PS
        Discover Respective Classes (i.e. next activities)
        DPS_Data = Dictionary with Decision Point as Key and Empty Values
    end if
    Get Data from Element and Store in Trace_Dict
    if Current Event in DPS_Data.Keys then
        Store Data Up to This Point From Trace_Dict in DPS_Data[Current Event]
        if DPS_Data[Current Event] > WS then
            Remove Oldest Instance
        end if
        if Current Event is the Last Decision Point of Instance then
            Remove Instance From Trace_Dict
        end if
    end if
    if Grace Period Finished then                    ▷ Initial Rule Mining
        for DP in DPS_Data.Keys do
            Data = From DPS_Data[DP]
            Build Decision Tree
            Build ADWIN Models for Data, Decision Classes, Accuracy
            Store Decision Tree and ADWIN Models in DMS
        end for
    end if
    if Current Event in DPS_Data.Keys AND Initial Mining Finished then
        DMS, WS = Monitoring(Current Event, DPS_Data, DMS) see Alg. 2
    end if
end while
```

the initial grace period is needed as input, either set manually, or by default, a grace period of 200 instances will be used. The appropriate grace period is oriented towards how frequently new instances arrive and how complex the contained data and underlying decision rules are expected to become. However, when testing the approach, we did not observe a significant impact by different settings of the grace period.

Each new event is instantly stored as part of the directly-follows-graph, which contains two events and the count of how often this combination occurred. *Lossy counting* and the *S-BAR method* [26] are used, which continually drop less frequent combinations, thereby accounting for finite storage and concept drift. The next step, the **process model discovery**, is realized using the Heuristics Miner (HM) [25]. The HM is used as discovery technique, as only a directly-follows-graph is needed as input, whereas other algorithms often need defined start and end events. This is not trivial when dealing with runtime discovery, as it is not know when an instance is finished.

As output, a petri net is generated, which is used as input for the **decision point discovery**. Using the algorithm proposed in [19], decision points, i.e., the places with multiple outgoing arcs, as well as the decision classes, i.e., the next occurring events, are discovered. A discovered decision point could look like this: "$Checkapplicationdata$" : ["$NormalCheck$","$ExtensiveCheck$"], i.e. the event before the decision as well as the decision classes are specified.

Up to this point, the only used data structure is the directly-follows-graph which is stored in a hash table, more specifically a dictionary. As soon, as decision points are mined, two additional hash tables are created. First, `Trace_Dict` is a dictionary, where each new instance is stored, using the instance identifier as key and all events and corresponding data elements of the respective instance as values. Second, `DPS_Data` uses decision points as key and every time an event occurs that was beforehand identified as a decision point, all available information in *Trace_Dict* up to this point for this instance, is stored as the value.

After the grace period finishes and decision points have been found, **decision rule mining** is performed, using the data stored in `DPS_Data` for the decision points. Any decision mining method can be applied. Here, a CART decision tree is used. The mined rules for each decision point are stored in a dictionary `DMS` using the decision point as key and the current decision tree models as well as some statistics as values, for example the accuracy, that will be added in the monitoring phase. In addition, a new ADWIN instance is generated for the average accuracy, each decision class, and each data element that is part of the associated decision point and stored in `DMS`. ADWIN is a well-known approach for concept drift detection [1], where the window size is adapted according to the change of data in the window. ADWIN is used in the monitoring phase to detect changes.

For each newly occurring event that reflects a decision point, Alg. 2 for **Monitoring and Remining** is called.

With regards to limited storage, the directly-follows-graph in combination with the S-Bar method, is inherently storage efficient. As for the `Trace_Dict`, instances are removed as soon as all decision points have occurred and therefore all data for this instance is already stored in `DPS_Data`. As for `DPS_data`, for each decision point, the most recent instances are kept, the exact number depends on the window size *WS*. The window size is by default set as the grace period and therefore the same for all decision points. As soon, as a drift is detected at a decision point, the window size is set to the ADWIN window size for that decision point. See Alg. 2 for details.

**Algorithm 2** Monitoring and Remining

**Input: Decision Point DP, DPS_Data, DMS**
**Output: DMS with Updated Decision Rules**

```
1: Compare Current Data with DPS_Data
2: if New Data Element then
3:     DMS[DP][Drift] = True
4: end if
5: Predict Class for Current Event using DMS
6: Calculate Overall Average of Accuracy, Data, Frequency
7: Add Averages to Respective ADWIN Models, Calculate Drift
8: if ADWIN Drift Detected then
9:     DMS[DP][Drift] = True
10:    WS = ADWIN.window
11: end if
12: if DMS[DP][Drift] == True then
13:    Remine Decision Model, Store Decision Model in DMS
14:    Reset Stored Data (Averages, ADWIN Models,...)
15:    DMS[DP][Drift] = False
16: end if
17: Return DMS, WS
```

Algorithm 2 builds on our previous work presented in [21] and has been significantly extended and adapted. Instead of relying on changes in the performance of decision rules, the monitoring also includes, data elements, i.e. new data elements occurred at a decision point or ranges of data values changed and changes in the branching frequency. Branching frequency refers to the frequency that decision classes, i.e. the respective branch, is chosen. The function is called with an event that has been identified as a decision point, e.g., for the running example, a new event *Normal Check* following an event *Check Application Data* occurred, which is part of a decision point. First, all data that occurred up to this point for this instance, and is stored in `Trace_Dict`, is gathered. The names of the data elements are compared to the data elements that have occurred before at this decision point and have been stored in `DPS_Data`. E.g., up until the event *Normal Check* the data elements *requested amount* and *age* were logged for this instance. In `DPS_Data` for the current decision point also only these two data elements are stored. Therefore no new elements have been detected. If unseen data elements, e.g., a data element *income*, are discovered, the variable `DMS[DP][Drift]` is set to True and stored in `DMS`. Otherwise, the class for the current decision point is predicted and compared to the actual class, to calculate the current accuracy, which is used in the next step to calculate the average decision rule accuracy.

The drift detection method ADWIN [1] is employed in order to detect whether a drift has occurred, either in the performance, i.e., the accuracy of the decision rules, the data values, or the branching frequency. ADWIN is

the basis for window-based concept-drift detection methods such as [9, 13] and compares statistics between windows to check if these are significantly different, i.e., a drift happened. As setting the window sizes manually is not trivial, the sizes are chosen according to the amount of changes in the data. If the data is stationary, the window is increased to improve accuracy. If drift occurs, the window is decreased. [12] propose a method for identifying changes in process models based on changes in the branching frequency ex-post. This change detection method is not able to work with event streams. Therefore, we opted for the ADWIN approach that is specifically optimized for runtime analysis.

For the end-to-end approach presented in this paper, the average decision rule accuracy is calculated each time the monitoring function is called, using the overall accuracy and the overall number the function was called, which are both stored in `DMS`. The same is done to calculate the average branching frequency, i.e., the average percentage which branches are taken, i.e. which classes are chosen. For the running example, *Extensive Check* is on average performed for 30% and *Normal Check* for 70% of instances, but then the averages change to 40% and 60% respectively, which could be a sign that decision drift occurred. In addition, the average value for all data elements are calculated. Here, the average *requested amount* could be 45.000, whereas the average age is 39.

The calculated averages are used as input to the ADWIN models. In Alg. 1, ADWIN models for the accuracy, the decision classes and data values are built at each decision point and stored in `DMS`. The calculated averages are added to the respective models, which then calculate whether a drift occurred. If a drift is detected, `DMS[DP][Drift]` is set to True. The ADWIN window size from the model where drift was detected is set as window size. The window size is used to control the maximum size of `DPS_Data`, i.e., if the ADWIN window size is, for example, 500 after a drift was detected, no more than 500 instances are stored for that decision point in `DPS_Data`. The First In - First Out principle is applied, i.e., the oldest instance is removed as soon as the maximum size is reached. This allows to dynamically increase and decrease the amount of instances stored, which is necessary as storage is limited and outdated data should not be used for remining decision rules. As the window decreases when drift is detected, only the more recent instances are used to remine.

If `DMS[DP][Drift]` is set to True, a new decision model is mined and stored in `DMS`. Lastly, all corresponding data, e.g., the calculated averages and ADWIN models are reset.

# 4 Evaluation

The approach was implemented using python and as available online [1].

As we propose, to the best of our knowledge, the first end-to-end runtime decision mining approach, the evaluation does not contain a comparison to other approaches. Instead, the general feasibility and applicability of the approach are evaluated. The requirements for data sets to be used for evaluation are:

[1] `https://github.com/bscheibel/dmma-e`

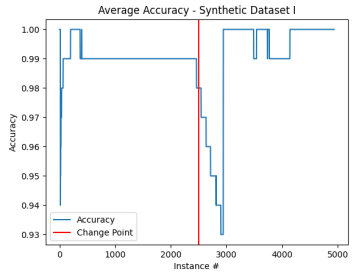
1. Process-based data set, e.g., an event log or stream
2. Underlying process model contains one or more decisions
3. Decisions are based on numeric data attributes
4. Decision drifts occur
5. Available ground truth: decision rules and drifts are known

Requirements 1-3 are necessary to be able to use the approach at hand. Requirement 4 is prerequisite to show the ability of the approach to detect different kinds of drift. Requirement 5 enables the validation of the results, i.e., to check whether the detected decision rules and drifts correspond to reality. We analyzed publicly available real-life data sets from the BPI Challenge[2] and from[3] along Requirements 1–5. The BPIC17 log fulfills Requirements 1–3 and the data elements are named to allow intuitive interpretation. Hence, the BPIC17 log is chosen for evaluating the applicability of the approach. In order to show its feasibility, we start with four synthetic data sets (SD) for which we know the ground truth. The evaluation results contain the average accuracy and for the synthetic data sets SD I–IV the number of instances from a decision drift until new decision rules are remined.

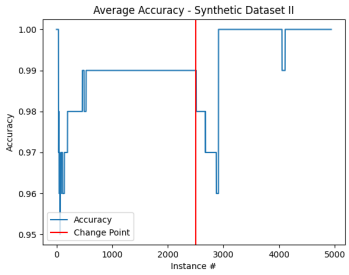
## 4.1 Feasability: Synthetic Datasets

SD I–IV are based on the running example depicted in Fig. 1 and contain the following decision drifts: I) value changes in a condition, II) additional data elements in a decision rule, III) additional branch for a decision point and IV) an additional decision point. The corresponding datasets have been created using random variations. Each data set consists of 5000 instances and is stored in a CSV file. In a pre-processing step, the contents are stored in a queue (event by event) in order to simulate an event stream. SD I–IV together with the complete evaluation results, including all decision rules, are available online[4].
**SD I** reflects a decision rule change. The initial rule:
$IF\ amount\_loan <= 80.000\ THEN\ Normal\ ELSE\ Extensive$
is changed to:
$IF\ amount\_loan <= 50.000\ THEN\ Normal\ ELSE\ Extensive$
at instance 2500. The overall average accuracy is 0.99. The drift was detected 505 instances after it occurred, at instance number 3005, and the decision rule was remined. The result can be seen in Fig. 3a. The accuracy continually decreases after the drift occurred, immediately after remining at instance number 3005, a sharp increase in accuracy can be seen.
**SD II** simulates an additional data element that is added to the decision rule. The initial decision rule is:

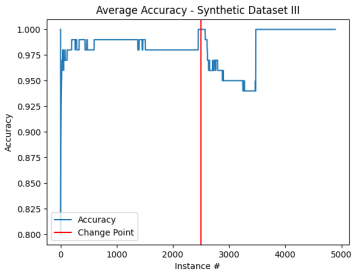
[2] https://www.tf-pm.org/competitions-awards/bpi-challenge
[3] https://data.4tu.nl/
[4] https://github.com/bscheibel/dmma-e



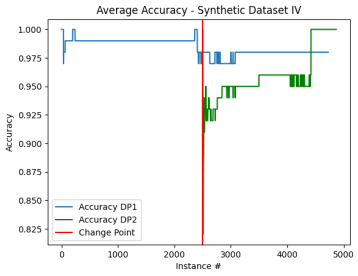

(a) SD I: Decision Rule Change.




(b) SD II: Decision Rule Change.




(c) SD III: Decision Class Change.




(d) SD IV: Decision Point Change.

Figure 3: Evaluation of SD I-IV.

$IF\ amount\_loan <= 80.000\ THEN\ Normal\ ELSE\ Extensive$
The data element *income* is added to the decision rule at instance number 2500 and the rule is therefore changed to:
$IF\ amount\_loan <= 80.000\ AND\ income\ > 3000\ THEN\ Normal\ ELSE\ Extensive$
To overall average accuracy is 0.99. The drift was detected 473 instances after it occurred, at instance number 2973, and the decision rule was remined. The result can be seen in Fig. 3b. The accuracy decreases after the drift occurred. After remining the accuracy increases.
**SD III** includes a decision class change, i.e., at first, only two decision classes are part of the decision point:
$IF\ amount\_loan\ <= 70.000\ THEN\ Normal$
$IF\ amount\_loan > 70.000\ THEN\ Extensive$
then the additional class *Simple Check* is added at instance number 2500:
$IF\ amount\_loan <= 30.000\ THEN\ Simple$
$IF\ amount\_loan > 30.000\ AND\ amount\_loan <= 70.000\ THEN\ Normal$
$IF\ amount\_loan > 70.000\ THEN\ Extensive$

The average accuracy is 0.98. The drift was detected 62 instances after it occurred, at instance number 2562, and the decision rule was remined. However, the mined rules did not reflect the new rule accurately, therefore a second

remining occurred at instance number 3585. The result can be seen in Fig. 3c. The accuracy decreases after the drift occurred. After the first remining the accuracy still decreases, whereas the second remining leads to an increase.
**SD IV** represents a decision drift in the form of an additional decision point. After *Overall Assessment*, two alternative branches are inserted: *Write Acceptance Letter* and *Write Rejection Letter*, before the branches are joined and the event *Inform Customer* occurs.
DP1: $IF\ amount\_loan <= 80.000\ THEN\ Normal\ ELSE\ Extensive$
After instance number 2500, a second decision point is added:
DP2: $IF\ risk\_level <\ 4\ AND\ amount\_loan < 80.000\ THEN\ Write\ Acceptance\ Letter$ $ELSE\ IF\ risk\_level <=\ 1\ AND\ amount\_loan => 80.000\ THEN\ Write\ Acceptance\ Letter$ $ELSE\ Write\ Rejection\ Letter$

The overall average accuracy for DP1 is 0.98 and for DP2 0.96. The drift was detected 133 instances after it occurred, at instance number 2622 and the decision rules were remined, including the new decision point. The result can be seen in Fig. 3d. The accuracy for the first decision point remains relatively constant. However, after remining, the accuracy for the second decision point is added in the plot. The accuracy for the second decision point is relatively low at first. However, at instance number 4552 another remining for the second decision point occurs, resulting in increased accuracy.

## 4.2 Applicability: BPIC17


Figure 4: BPIC17: Mined Process Model.

The BPIC17 data set[5] consists of a loan application process from a financial institute, including loan applications and offers, where each application can contain multiple offers. For the evaluation, the offer data set was used. Before applying the approach, pre-processing was done to simulate an event stream.

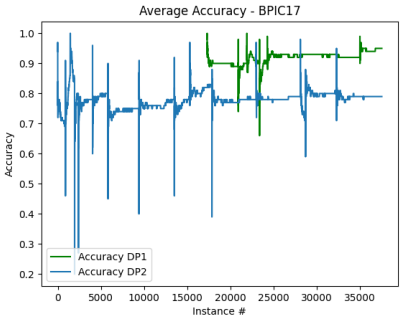



Figure 5: BPIC17 Results.

Fig. 4 shows the final process model with two decision points. The process model was remined throughout the process: in the beginning it contained only one decision point. Figure 5 shows the accuracies of the decision

[5] `https://data.4tu.nl/articles/dataset/BPI_Challenge_2017/12696884`

points. The average accuracy for the first decision point is 0.78 and for the second decision point 0.91. In total, remining occurred 19 times, 14 times for the first decision point and 5 times for the second decision point. Both decision rules include the data elements *NumberOfTerms*, i.e. the number of payback terms agreed to, and *CreditScore*. For example, the last mined decision rule for decision point 2, i.e., if the offer was refused or accepted, contains the condition that the offer is accepted if the CreditScore is above 324. The exact values in the decisions change with each remining.

### 4.3 Discussion

The evaluation of the synthetic datasets shows that the approach is feasible and able to discover different kinds of decision drift during runtime. The full results also show that the mined decision rules are equal to the underlying rules. For the real-life dataset, the evaluation shows that the approach is able to work with real-life data. However, the BPIC17 dataset probably did not encompass any decision drifts, and the frequent remining was executed rather due to insufficient data than to mine accurate decision rules. Additional evaluations in real-life settings will be part of future work to show the generalisability of the approach. In terms of interpretability, the output consists of textual decision rules, which enables manual interpretation and analysis. The approach can also be easily adapted to work with different kinds of decision mining approach, see Sect. 5, enabling the inclusion of e.g. categorical or time-series data as input.
**Limitations and threats to validity:** The approach might not work for cases where drifts are happening very frequently. If lots of unfinished traces occur, this could lead to storage build-up. In addition, only sudden drifts have been tested in the evaluation. Theoretically the approach should also work for incremental and gradual drifts, however this would probably include frequent remining until a stable decision rule is discovered. Furthermore, the current definition of data change is a change in the average, this is of course a very restricted definition.

## 5 Related Work

The first decision mining approach [19] includes an algorithm for detecting decision points and the corresponding decision classes from a Petri Net as well as classification techniques to mine the decision rules. Subsequent approaches focus on specific aspects of decision mining, e.g., including overlapping rules [14], incorporating decision rules based on linear relationships between variables [6], or mining decision rules based on time series data [20] (for an overview see [7]). Existing approaches employ ex-post algorithms. Recently, online or runtime analysis is gaining traction for online process discovery [4, 15], conformance checking [10], drift detection [5, 22], and predictive process monitoring [17]. Especially, drift detection, partly overlaps with decision drift analysis as changes in decision points are part of control flow drift. For process discovery, [15] introduce an approach to mine data-aware declarative process models, i.e., con-

straints, from event streams. The approach is similar as constraints could also be seen as decision rules, but still quite different as constraints do not translate straightforwardly to decision points and rules. [16] present an approach for predictive decision mining for operational support. None of these approaches include decision drift analysis, the remining of decision points and rules, and the textual generation of decision rules. [12] propose a method for identifying decision rule changes based on changes in the branching frequency. This is done ex-post and neither decision point discovery nor remining are part of the approach. However, part of the approach is included in this approach for detecting change. Our previous work [21] assumes that decision points are already known and no changes of decision points occur. Hence, this work constitutes a significant extension of [21].

# 6 Conclusion

This paper presents an end-to-end approach for mining decision rules during runtime, as well as monitoring of decision drift, and updating decision points and the associated decision rules if necessary. Decision drift encompasses different changes with regards to decisions in processes, i.e., changes in decision rules, decision classes, and decision points. The change detection is based on the drift detection method ADWIN and monitors the performance, the branching frequency as well as data values for changes. The approach is optimized for runtime use, i.e., limited storage as well as forgetting outdated data is taken into account. An event stream generated by, e.g., a PAIS is used as input. The output comprises textual decision rules for each discovered decision point, that are updated as soon as decision drift is detected, as support for users to evaluate if these changes are intentional. This enables increased transparency and can be used as basis for process enhancement. The evaluation shows that the approach is able to detect different kinds of decision drift with high accuracy and to work with real-life data. However, further testing is planned for future work as well as the analysis of drift patterns, root-cause analysis, and drift prediction.

**Acknowledgements**
This work has been partly supported and funded by the Austrian Research Promotion Agency (FFG) via the Austrian Competence Center for Digital Production (CDP) under the contract number 881843 and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 277991500.

# References

[1] Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In: SIAM Int'l Conf. on Data Mining. pp. 443–448 (2007)

[2] Bose, R.P.J.C., van der Aalst, W.M.P., Zliobaite, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE Trans. Neural Networks Learn. Syst. **25**(1), 154–171 (2014). https://doi.org/10.1109/TNNLS.2013.2278313

[3] Burattin, A.: Streaming Process Mining. In: Process Mining Handbook, pp. 349–372. Springer International Publishing (2022)

[4] Burattin, A., Cimitile, M., Maggi, F.M., Sperduti, A.: Online Discovery of Declarative Process Models from Event Streams. IEEE Transactions on Services Computing **8**(6), 833–846 (2015)

[5] Carmona, J., Gavaldà, R.: Online Techniques for Dealing with Concept Drift in Process Mining. In: Advances in Intelligent Data Analysis XI, vol. 7619 (2012)

[6] de Leoni, M., Dumas, M., García-Bañuelos, L.: Discovering Branching Conditions from Business Process Execution Logs. In: Fundamental Approaches to Software Engineering. pp. 114–129 (2013)

[7] de Leoni, M., Mannhardt, F.: Decision Discovery in Business Processes. In: Encyclopedia of Big Data Technologies, pp. 1–12 (2018)

[8] Hasić, F., Corea, C., Blatt, J., Delfmann, P., Serral, E.: Decision model change patterns for dynamic system evolution. Knowledge and Information Systems **62**(9), 3665–3696 (2020)

[9] Hassani, M.: Concept Drift Detection Of Event Streams Using An Adaptive Window. In: Modelling and Simulation. pp. 230–239 (2019)

[10] Koenig, P., Mangler, J., Rinderle-Ma, S.: Compliance Monitoring on Process Event Streams from Multiple Sources. In: Process Mining. pp. 113–120 (2019)

[11] Leewis, S., Berkhout, M., Smit, K.: Future Challenges in Decision Mining at Governmental Institutions. In: Americas Conf. on Information Systems. p. 12 (2020)

[12] Lu, Y., Chen, Q., Poon, S.K.: Detecting and understanding branching frequency changes in process models (2021), `https://arxiv.org/abs/2103.10742`

[13] Maaradji, A., Dumas, M., La Rosa, M., Ostovar, A.: Fast and Accurate Business Process Drift Detection. In: Business Process Management. pp. 406–422 (2015)

[14] Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Decision Mining Revisited - Discovering Overlapping Rules. In: Advanced Information Systems Engineering. pp. 377–392 (2016)

[15] Navarin, N., Cambiaso, M., Burattin, A., Maggi, F.M., Oneto, L., Sperduti, A.: Towards Online Discovery of Data-Aware Declarative Process Models from Event Streams. In: Neural Networks. pp. 1–8 (2020)

[16] Park, G., Küsters, A., Tews, M., Pitsch, C., Schneider, J., van der Aalst, W.M.P.: Explainable Predictive Decision Mining for Operational Support (2022), https://arxiv.org/abs/2210.16786

[17] Pauwels, S., Calders, T.: Incremental Predictive Process Monitoring: The Next Activity Case. In: Business Process Management. pp. 123–140. Cham (2021)

[18] Rollings, M.: How to make better business decisions (2021), https://www.gartner.com/smarterwithgartner/how-to-make-better-business-decisions

[19] Rozinat, A., van der Aalst, W.M.P.: Decision Mining in ProM. In: Business Process Management. pp. 420–425 (2006)

[20] Scheibel, B., Rinderle-Ma, S.: Decision Mining with Time Series Data Based on Automatic Feature Generation. In: Advanced Information Systems Engineering. pp. 3–18. Springer (2022)

[21] Scheibel, B., Rinderle-Ma, S.: Online Decision Mining and Monitoring in Process-Aware Information Systems. In: Conceptual Modeling. pp. 271–280 (2022)

[22] Stertz, F., Rinderle-Ma, S.: Process Histories - Detecting and Representing Concept Drifts Based on Event Streams. In: OTM Conferences. pp. 318–335 (2018)

[23] Stertz, F., Rinderle-Ma, S.: Detecting and Identifying Data Drifts in Process Event Streams Based on Process Histories. In: CAiSE Forum, pp. 240–252 (2019)

[24] Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features – Enhancing flexibility in process-aware information systems. Data & Knowledge Engineering **66**(3), 438–466 (2008)

[25] Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: Computational Intelligence and Data Mining. pp. 310–317 (2011)

[26] van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Event stream-based process discovery using abstract representations. Knowledge and Information Systems **54**(2), 407–435 (2018)