# Online Decision Mining and Monitoring in Process-Aware Information Systems

Beate Scheibel[1][0000−0002−1513−199X] and Stefanie Rinderle-Ma[2][0000−0001−5656−6108]

[1] Research Group Workflow Systems and Technology, Faculty of Computer Science, University of Vienna, Austria
beate.scheibel@univie.ac.at
[2] Chair of Information Systems and Business Process Management, Technical University of Munich, Germany
stefanie.rinderle-ma@tum.de

**Abstract.** Decision mining enables discovery of decision rules guiding the control flow in processes. Existing decision mining techniques deal with different kinds of decision rules, e.g., overlapping rules, or including data elements, for example, time series data. Though online process mining and monitoring are gaining traction, online decision mining algorithms are still missing. Decision rules can be, similarly to process models, subject to change during runtime due to, for example, changing regulations or customer requirements. In order to address these runtime challenges, this paper proposes an approach that i) discovers decision rules during runtime and ii) continuously monitors and adapts discovered rules to reflect changes. Furthermore, the concept of a decision rule history is proposed, enabling (manual) identification of change patterns. The feasibility and the applicability of the approach is evaluated based on three synthetic datasets, BPIC12, BPIC20 and sepsis data set.

**Keywords:** Online Decision Mining · Decision Rule Evolution · Decision Rule Monitoring · Process-Aware Information Systems

## 1 Introduction

Process mining enables process discovery, conformance checking and process enhancement [1]. An important part of process discovery is decision mining, providing techniques to discover decision points in processes as well as the underlying decision rules guarding that decision based on event logs [6]. Process and decision mining are increasingly gaining traction as transparency and standardization become more and more important across different domains [5]. Existing decision mining approaches [7, 8, 11, 14] operate in an ex-post way, i.e., by analyzing event logs after the process instances have been completed. Recent process mining research focuses on online process mining, for example, online process discovery [2], online sensor stream analysis [3] or online concept drift detection, i.e. detecting control flow changes during runtime, see for example [17]. However, to

the best of our knowledge, neither runtime decision mining algorithms nor algorithms to detect and monitor decision rule evolution, independent of control flow changes, exist. Hence, this work focuses on discovering decision rules during runtime and updating decision rules if changes occur in the process or the process environment. Such process or process environment changes include:

- *Available data:* Internal process data i.e., data attributes available in the event log such as patient age in a medical scenario, as well as external data, that can be mapped to process events, e.g. sensor data, has changed or additional data has become available. Examples comprise newly installed sensors and varying ranges of measured values in a workpiece within the specified tolerance range. Being able to adapt existing decision rules to additional information can lead to more comprehensive decision rules.
- *Seasonal or cyclical changes:* For example seasonal price adaptations in the tourism domain that are not specified as a data element.
- *Environmental changes:* Decision rules reflect business or objective rules based on, e.g., regulatory documents. The underlying rules might change over time, e.g., new regulations occur or customers change their requirements, resulting in an evolution of the corresponding decision rule.

If changes are not taken into account, outdated rules might drive decisions in running process instances, no longer reflecting, for example, state of the art parameters or current regulations.

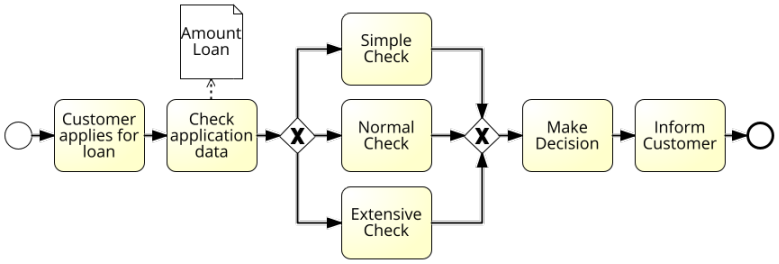


**Fig. 1.** Running example: loan application (modeled using Signavio©).

Consider the loan application process as depicted in Fig. 1. A customer applies for a loan. There are three levels of scrutiny: simple, normal, and extensive. Assume that the levels depend solely on the amount that is applied for as reflected by the following decision rule:
$IF\ amount\_loan <= 20.000\ THEN\ Simple$
$IF\ amount\_loan > 20.000\ AND\ amount\_loan <= 80.000\ THEN\ Normal$
$IF\ amount\_loan > 80.000\ THEN\ Extensive$
During execution, regulations might change, resulting in a lower limit for an extensive check, i.e, an amount greater than 50.000.

A comprehensive decision mining and monitoring approach would mean to i) discover the first version of the decision rule when the loan process starts to be instantiated and executed based on the event stream emitted during process execution ($\mapsto$ **RQ1**), ii) monitor and update the rule when the rule evolution comes into effect, again based on the event stream ($\mapsto$ **RQ2**). In order to tackle **RQ1** and **RQ2**, this paper proposes a novel algorithm to discover decision rules during runtime. It takes an event stream as input and determines the current decision rule as output. The algorithm uses windows that adapt according to the current performance measured by the F1 score. The different versions of a decision rule discovered during runtime are stored in a *decision rule history*. The decision rule history can be exploited for further analysis, e.g., to discover patterns in the decision rules such as seasonal variations. Three scenarios for decision rule evolution during runtime are described and the corresponding synthetic data sets are used for evaluating the feasibility of the approach. In addition, three real world data sets are used to evaluate the applicability of the approach.

Section 2 describes the decision mining and monitoring algorithm, which is evaluated in Sect. 3 and discussed in Sect. 4. An overview of related work is given in Sect. 5 and a conclusion is provided in Sect. 6.

## 2 Decision Rule Mining and Monitoring

During runtime, process instances are started and executed based on a process model that consists of control and data flow. The execution history is stored in the associated process event trace. Each trace is signified by a unique id and constitutes a sequence of events that reflect the execution of process tasks. The order of the activities is reflected by timestamps in the process event log. Events can hold additional information, i.e. data elements. An event stream contains the same elements as an event log. However, whereas an event log is finite (i.e., no more events are incoming) and complete (each trace has a start and end event), an event stream constitutes a continuous flow of events produced by process instances, with no specific start or end [17].

The *decision mining and monitoring algorithm (DMMA)* proposed in this work uses an event stream as input, with new instances being executed continually. We assume a predefined decision point $DP$ in the underlying process model. DP could be discovered in a previous step using (online) process mining. At the start of the DMMA, no data and no decision rule are available. Hence, a grace period is implemented, where data is gathered, but not analyzed yet. As soon as the grace period has finished, a decision rule for $DP$ is mined and added to the *decision rule history* which holds all discovered versions of a decision rule (cmp. process model history as proposed in [17]), formally:

**Definition 1 (Decision rule history).** *Let $P$ be a process model with a decision point $DP$. The decision rule history $RH_{DP} :=< R_0, R_1, \dots R_k >$ for $DP$ constitutes a list of discovered versions $R_i := (n_i, p_i, l_i)$ for the decision rule $R$ connected to $DP$ ($i \in \mathbb{N}$). $n_i$ denotes the number of instances active at the point*

*in time when $R_i$ was discovered, $p_i$ the performance of the algorithm for $R_i$, and $l_i$ a link to the textual description of $R_i$. $R_k$ denotes the current version of the decision rule for DP.*

The decision rule history $RH_{DP}$ is updated continually, i.e., as soon as a new decision rule version is mined. In addition to the rules itself, the history includes the number of instances a specific rule was in place as well as the mean performance. We opt to store the decision rules textually to enable manual analysis. Note that in a process with multiple decisions, multiple decision rule histories might exist, each one covering decision rule versions for a specific decision point.

In the DMMA, a decision rule version $R_k$ is mined and continually checked if it still aligns with new process instances. The window size determines the point in time to check the current rule. The alignment is assessed based on the performance, measured using the F1 Score, i.e., the ability of the rule to accurately classify new instances. If the rule is able to achieve a high performance, the window size is increased. If the performance drops below a certain threshold, i.e. the *remining criterion* is met, we assume that either the underlying data or the underlying rule has changed, and therefore the rule has to be remined which is then added to the decision rule history. If the remined rule is still not able to accurately classify new instances, the window size is decreased and increased in parallel for the next iterations to check which leads to more performant rule versions. The DMAA continues until no new process instances occur.

The pseudo code for DMAA is provided in Alg. 1. Input parameters are upper limit, lower limit, increase size, threshold, and step size (all parameter values in [0,1]). The upper limit *up* marks the point at which the performance of a rule version is seen as robust. On the contrary, the lower limit *low* is the threshold where the current version is not assumed to be robust anymore. The threshold $t$ is used when deciding if the difference in F1 Score of the current ($R_k$) and the previous version ($R_{k-1}$) is significant. The increase $i$ defines the percentage the window size is increased by if the rule version is robust, whereas the step size $s$ defines the percentage by which the window is increased/decreased to find a new appropriate window size. In addition, a maximum window size can be set to avoid increasing the window to a size larger than the amount of incoming data, leading to no further performance checks. Similarly, a minimum size can be set. These mainly depend on the expected amount of data and the need to minimize computing resources, as smaller windows lead to more computational effort.

For rule mining a CART decision tree, a standard decision mining tool [6], is used. Any other decision mining implementation can also be used at this point, e.g. Incremental Decision Trees [10], which are optimized for runtime classification. Similarly, multiple metrics can be used to evaluate the performance of a decision rule, i.e., how accurately a rule classifies instances, such as accuracy, F1 Score, precision, recall, logloss, or AUROC. For DMAA, we choose the F1 Score, which is commonly used as performance metric for concept drift detection [15]. The F1 Score combines recall and precision. Precision is defined as all instances that are correctly labeled as class_x, divided by all instances of class_x, and recall as all instances that are correctly labeled as class_x, divided by all

**Algorithm 1** DMAA

**Input: start window size** $w$**, increase** $i$**, threshold** $t$**, upper limit** $up$**, lower limit** $low$**, step size** $s$

**Output: current rule version, decision rule history**

```
1: initial = True, robust = True, w_smaller = w, w_bigger = w
2: create trace dictionary
3: while new event do
4:     add event to trace dictionary, counter + = 1
5:     if initial mining AND grace period reached then    ▷ grace period default 200
6:         initial = false
7:         mine decision rule, store in model
8:         calculate F1 Score
9:         output decision rule, add rule to decision rule history
10:    end if
11:    if (robust AND counter >= w) OR not robust AND
12:        (counter >= w_bigger OR counter >= w_smaller) then
13:        set w_test to appropriate size                 ▷ depending on if condition
14:        F1 Score_old = F1 Score
15:        counter = 0                                    ▷ not for w_smaller
16:        calculate F1 Score for w_test and model
17:        if F1 Score < F1 Score_old*t then              ▷ t default 0.9
18:            remine decision rule, store in model
19:            calculate F1 Score
20:            output decision rule, add rule to decision rule history
21:        end if
22:        if F1 Score > up then                          ▷ up default 0.98
23:            w = w_test
24:            if robust then
25:                increase w by i                        ▷ i default 1%
26:            end if
27:            w_smaller = w_test, w_bigger = w_test, robust = True
28:        else if F1 Score < low then                    ▷ low default 0.90
29:            robust = False
30:            w_smaller = w_smaller * (1 − s)            ▷ s default 20%
31:            w_bigger = increase_bigger * (1 + s)
32:        end if
33:    end if
34:    if length dictionary > w_bigger then               ▷ set to biggest window size
35:        remove first dictionary entry                  ▷ first in, first out
36:    end if
37: end while
```

instances labeled class_x. Overall, an F1 Score close to 1 indicates high recall and precision values. The F1 Score is calculated for each class separately and then weighted according to the occurrence of the specific class in the data. For DMMA, we calculate the F1 Score each time the window size is reached and when a new rule version is mined. We compare the current F1 Score to the

last F1 Score. If the performance has significantly decreased, defined by a user-defined threshold $t$, the *remining criterion* is met, a new rule version is mined. If a rule achieves an F1 Score of above the threshold $up$, we assume that it is robust, i.e. able to adequately classify the current instances.

Checking the remining criterion for each instance is computationally inefficient, in particular in connection with a large number of instances. Therefore, we use a window based technique, where only the most recent instances are kept in memory. The input, in form of a process stream, where new events arrive continually, is kept in an ordered dictionary and as soon as the window size is reached, the first instance with all respective events and data elements, is removed, i.e. the first in, first out principle is applied. This window also defines when the performance is checked, i.e., as soon as a complete turnover of stored instances has taken place, the performance is checked.

The optimal window size depends on the data, the underlying rules, as well as if and in what way the rule evolves and can impact the results significantly. Setting it too small might result in rules not being discovered, as the available data does not appropriately cover the rules. On the contrary, a too large size, might lead to the inability to discover a robust rule as multiple versions are present in the current window [15]. Therefore the goal is to find a balance discovering new versions as fast as possible and keeping the ability to mine robust and comprehensive rules, while minimizing computational complexity. The window size can be set statically or in an adaptive manner, i.e. the window size changes during the process. In related approaches it was shown that adaptive setting can lead to optimized results, reducing the impact of the initial size [9]. DMMA uses adaptive windows with a manually set starting size. If the last calculated F1 Score is above a fixed limit $up$, the window size is increased by $i$ as we see the rule as robust. If the score is below lower limit $low$, the window size should be either increased, because the algorithm is not able to discover complex rules or the window size should be decreased to be able to discover rule changes. As we cannot know which is the case, two windows, $w\_smaller$ and $w\_bigger$, are used in parallel, which are set by subtracting and adding a fixed step $s$ to the current window size. Further incoming instances are checked and the rule remined if necessary for each of the different window sizes. If no robust rule is achieved, the windows are further decreased and increased. As soon as a robust rule is found, the current window size is set as $w$, and the process continues as described before until no more instances arrive.

## 3 Evaluation

The approach is implemented in Python, using the 'scikit-learn' module [13] as decision tree implementation. All data sets are preprocessed, simulating an event stream by reading in each event in the order of timestamps. In addition to the loan example, presented in Sect. 1, we provide a manufacturing and tourism use case with synthetic data sets, reflecting different decision rule evolutions.Data set characteristics are described together with the results in the following sections.

The source code, the data sets as well as the full results are available online[3]. For the synthetic data sets, the exact point of change is known, therefore the results contain the mean number of instances from the point where the underlying rule change happened until a new robust rule version is mined (not including the grace period), as well as the number of *transition rules*, being defined as rules mined in between changes, where the resulting rule does not fully represent either the old or the new rule version. This is currently detected manually.

**Use Case 1 - Loan Application:** A synthetic data set according to the process shown in Fig. 1 is generated, containing 5000 instances in total. The rule change happens at instance number 2500. After the initial grace period, a rule was discovered that stayed consistent until the underlying rule changed at instance 2500. 241 instances later, a new rule version was successfully mined and stayed consistent until the process finished. The overall F1 Score is 97%.

**Use Case 2 - Manufacturing:** This scenario consists of a simplified manufacturing scenario where a workpiece is produced. The observed decision is whether the workpiece is OK or not (NOK). In the beginning the only available data are the diameter measurements. During production, a second data element, the temperature is made available. The synthetic data set contains about 5000 instances, the additional sensor is added from the $2500th$ instance on. Before the additional sensor was added, 16 rule versions, i.e. transition rules were mined indicating that no robust rule was found due to lack of data. As soon as the sensor was added it took 23 instances until a robust version, appropriately reflecting the underlying rule, was discovered. The mean F1 Score is at 0.93.

**Use Case 3 - Tourism:** This use case describes the process of calculating a room offer for hotel guests and can be seen in Fig. 2. The synthetic data set contains four seasons in total, each containing 2500 instances. Depending on the season and the occupancy rate, either a standard or a premium price is offered. Assuming that there are more guests in winter than in summer, the premium price is already offered at 60% capacity in winter, the summer rule version states that a premium is offered at 90% capacity. The seasons are not logged, therefore no respective data element exists. In total 6 rule versions are discovered during runtime, meaning that the four seasonal rules are discovered as well as two transition rules after changes occurred ($70 - 100$ instances later), containing old as well as new versions. However, if these rules are not taken into account the F1 Score is high with 98%, with those rules the mean F1 Score is at 0.95.

**Real Life Data:** The applicability is tested on three real life data sets, BPIC12[4], BPIC20[5], and a sepsis data set[6]. The BPIC12 contains a loan application process similar to the running example. The observed decision is whether a loan application was accepted or rejected. The only available data element is the requested amount. The decision rule history discovered by the DMMA contains 64 rule versions, i.e., after each window remining was necessary due to the low F1

[3] `https://github.com/bscheibel/dmma`
[4] `https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204`
[5] `https://doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51`
[6] `https://data.4tu.nl/articles/dataset/Sepsis_Cases_-_Event_Log/12707639`

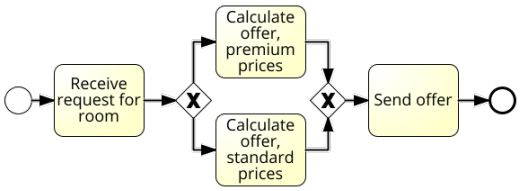


**Fig. 2.** Use Case 3: Tourism.

scores between 0.28 and 0.59. However, the discovered rule versions are similar to the one described in [7], i.e. mostly loans are accepted when their amount is neither too high nor too low, e.g., one of the version states that the amount has to be between 5750 an 42000 to be accepted. The BPIC20 log contains permit applications and declarations from a travel expense reimbursement system. The analyzed decision determines whether an instance is marked as 'Overspent'. The discovered rule stays consistent over the entire runtime, with a mean F1 Score of 0.99, leading to the assumption that the underlying rule is robust and does not evolve. The sepsis data set contains event logs from a hospital. The decision point whether a patient is admitted to either "Normal Care" or "Intensive Care" is analyzed. Available data elements include lab values as well as other information such as patient age. In total, 7 decision rule versions are discovered by the DMMA with F1 Scores varying between 0.49 and 0.98 for the last discovered version. The last rule version states the lactic acid values as the decisive factor.

## 4 Discussion

The evaluation shows that the approach is feasible and able to discover decision rules and their evolution during runtime, enabling greater transparency of a process and its decision points. For the BPIC20 and the sepis data set we assume that not all necessary information is available to mine robust rules, leading to frequent remining. The extracted rule versions are stored in a decision rule history and can be used to manually analyze rule evolution, e.g., seasonal or alternating rules.

**Limitations and threats to validity:** Several parameters have to be set manually. In future work, guidelines on how to set these parameters based on process characteristics, e.g., amount of expected data or frequency of expected changes, will be developed. One window size is used to determine the amount of stored instances, the number of instances that are used to check a rule version as well as to the number of instances that are used to remine a rule version. Differing window sizes could be used for these values. However, this would increase the complexity of the approach without a clear benefit to it.

**Computational complexity and runtime** of DMAA depend on the underlying data, the window size (also depending on the underlying data) as well as

the number of decision points in a given process. The computationally most expensive part is the rule (re)mining. If no underlying changes occur, rule mining happens only once. In contrast, if changes happen frequently and therefore the rule evolves continually, the computational complexity increases. The extreme case is that remining occurs for each window. An optimization with regards to computational complexity is part of future work.

## 5 Related Work

Decision mining was introduced as a way to discover data conditions that impact the routing of a specific instance at a decision point [14]. Multiple different decision mining algorithms exist, focusing on different aspects [6], e.g. aligning control flow and data flow to discover decision rules [7], discovering overlapping rules [11], incorporating linear relationships between variables [8], or mining decision rule based on time series data [16]. These algorithms are applied ex-post, i.e., after the process instances have finished. Recent work in online process mining focuses on process discovery [2], conformance checking [4], drift detection [17] and predictive process monitoring [12] during runtime. However, none of these works focus on online decision mining, especially with regards to the generation of textual decision rules, decision rule evolution and decision rule histories.

## 6 Conclusion

This paper presents DMMA, an algorithm for discovery and monitoring decision rule evolution during runtime. Decision rule evolution might become necessary due to newly available data and changes in environmental conditions. The DMMA algorithm takes as input an event stream and processes it as events occur. The output of the DMAA algorithm is the decision rule history for a decision point containing all versions of the decision rule connected with this decision point. The decision rule history enables the analysis of how decision rules evolve over time, i.e., lays the foundation for decision rule pattern analysis. In future work, we will investigate the potential of decision rule history analysis, together with root cause analysis why decision rules evolved. This provides insights into the evolution of the process and the decision structures behind. The decision pattern analysis will require the comparison of decision rule (versions) and similarity measures based on decision rules.

**Acknowledgements** This work has been partially supported and funded by the Austrian Research Promotion Agency (FFG) via the Austrian Competence Center for Digital Production (CDP) under the contract number 881843.

## References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer-Verlag (2016). https://doi.org/10.1007/978-3-662-49851-4

2. 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). https://doi.org/10.1109/TSC.2015.2459703
3. Ehrendorfer, M., Mangler, J., Rinderle-Ma, S.: Assessing the Impact of Context Data on Process Outcomes During Runtime. In: Service-Oriented Computing. pp. 3–18 (2021). https://doi.org/10.1007/978-3-030-91431-8˙1
4. Koenig, P., Mangler, J., Rinderle-Ma, S.: Compliance Monitoring on Process Event Streams from Multiple Sources. In: Process Mining. pp. 113–120 (Jun 2019). https://doi.org/10.1109/ICPM.2019.00026
5. Leewis, S., Berkhout, M., Smit, K.: Future Challenges in Decision Mining at Governmental Institutions. p. 12 (2020)
6. de Leoni, M., Mannhardt, F.: Decision Discovery in Business Processes. In: Encyclopedia of Big Data Technologies, pp. 1–12 (2018). https://doi.org/10.1007/978-3-319-63962-8˙96-1
7. de Leoni, M., van der Aalst, W.M.P.: Data-aware process mining: discovering decisions in processes using alignments. In: Symposium on Applied Computing. p. 1454 (2013). https://doi.org/10.1145/2480362.2480633
8. 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). https://doi.org/10.1007/978-3-642-37057-1˙9
9. Maaradji, A., Dumas, M., Rosa, M.L., Ostovar, A.: Detecting Sudden and Gradual Drifts in Business Processes from Execution Traces. IEEE Transactions on Knowledge and Data Engineering (2017). https://doi.org/10.1109/TKDE.2017.2720601
10. Manapragada, C., Webb, G.I., Salehi, M.: Extremely Fast Decision Tree. In: PKnowledge Discovery & Data Mining. pp. 1953–1962 (2018). https://doi.org/10.1145/3219819.3220005
11. Mannhardt, F., Leoni, M.d., Reijers, H.A., van der Aalst, W.M.P.: Decision Mining Revisited - Discovering Overlapping Rules. In: Advanced Information Systems Engineering. pp. 377–392. Springer, Cham (Jun 2016). https://doi.org/10.1007/978-3-319-39696-5˙23
12. Pauwels, S., Calders, T.: Incremental Predictive Process Monitoring: The Next Activity Case. In: Business Process Management. pp. 123–140 (2021). https://doi.org/10.1007/978-3-030-85469-0˙10
13. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D.: Scikit-learn: Machine Learning in Python. Machine Learning In Python pp. 1–6
14. Rozinat, A., van der Aalst, W.M.P.: Decision Mining in ProM. In: Business Process Management. pp. 420–425 (2006). https://doi.org/10.1007/11841760˙33
15. Sato, D.M.V., de Freitas, S.C., Barddal, J.P., Scalabrin, E.E.: A Survey on Concept Drift in Process Mining. ACM Comput. Surv. **54**(9), 1–38 (2022). https://doi.org/10.1145/3472752
16. Scheibel, B., Rinderle-Ma, S.: Decision Mining with Time Series Data Based on Automatic Feature Generation. In: Conference on Advanced Information Systems Engineering (2022). https://doi.org/10.1007/978-3-031-07472-1˙1
17. Stertz, F., Rinderle-Ma, S.: Process Histories - Detecting and Representing Concept Drifts Based on Event Streams. In: On the Move to Meaningful Internet Systems. pp. 318–335 (2018). https://doi.org/10.1007/978-3-030-02610-3˙18