A Novel Decision Mining Method Considering Multiple Model Paths

Pietro Portolani[1,2(✉)], Diego Savoia[1], Andrea Ballarino[2], and Matteo Matteucci[1]

[1] Politecnico di Milano - DEIB, Milan, Italy
{pietro.portolani,diego.savoia,matteo.matteucci}@polimi.it
[2] Consiglio Nazionale delle Ricerche - STIIMA, Milan, Italy
andrea.ballarino@stiima.cnr.it

Abstract. The automatic extraction of a process model from data is one of the main focuses of a Process Mining pipeline. Decision Mining aims at discovering conditions influencing the execution of a given process instance to enhance the original extracted model. In particular, a Petri Net with data is a Petri Net enhanced with guards controlling the transitions firing in correspondence of places with two or more output arcs, called decision points. To automatically extract guards, Decision Mining algorithms fit a classifier for each decision point, indicating what path the case will follow based on event attributes. Retrieving the path followed by the case inside the model is crucial to create each decision point's training dataset. Indeed, due to the presence of invisible activities, having multiple paths coherent with the same trace in the event log is possible. State-of-the-art method consider only the optimal path discarding the other possible ones. Consequently, training sets of related decision points will not contain information on the considered case. This work proposes a depth-first-based method that considers multiple paths possibly followed by a case inside the Petri Net to avoid information loss. We applied the proposed method to a real-life dataset showing its effectiveness and comparing it to the current state of the art.

Keywords: Decision Mining · Process Mining · Decision Trees · Machine Learning

1 Introduction

Process Discovery is one of the fundamental tasks of Process Mining. It aims at extracting a process model automatically from logs recorded by an information system. The discovered model focuses only on the activities' control flow, representing their order relations with frameworks such as Petri Nets or Business Process Model and Notation[1] models. The extracted models can have various purposes, from simulation to conformance checking and process optimisation.

[1] https://www.omg.org/spec/BPMN/2.0.2/About-BPMN/.

H. van der Aa et al. (Eds.): BPMDS 2023/EMMSAD 2023, LNBIP 479, pp. 79–87, 2023.
https://doi.org/10.1007/978-3-031-34241-7_6

Decision Mining, another Process Mining subfield, takes a further step and enhances the process model with information about the decisions influencing its execution. In recent years, the focus on decisions has gained importance as they are among an organisation's most crucial assets [1]. The topic is so relevant that the Object Management Group developed the Decision Model and Notation standard[2], an industrial standard to model decisions.

Decision Points Analysis is a particular type of Decision Mining based on Petri Net models and it uses data to generate information and knowledge to support decision-making processes in the form of annotated decisions [2]. Specifically, it retrieves the decision rules solving a classification problem in every place with two or more output arcs, i.e. a decision point.

Indeed, to create a valid training dataset, it is necessary to know the path the process instance follows inside the process model to locate the crossed decision points correctly. Retrieving such a path is not straightforward, mainly due to invisible activities and loops. Previous approaches developed developed different strategies to select the path, however, current techniques lead to a loss of information considering only one route inside the process model.

In this work, we propose a novel method to consider multiple allowed model paths the case could have taken to go from one activity to the following. The idea is that considering multiple paths will help better characterise the invisible branches of the decision points, resulting in less information loss and, thus, better rules[3].

2 Related Works

Authors in [3] and [4] made major contributions to the Decision Points Analysis field. In [3] the authors propose for the first time to transform every decision point into a classification problem and derive the decision rules from a decision tree used to fit it.

To create the training dataset needed, the authors track the path followed by the case from an activity in the sequence to the following one and assign the related event attributes to the encountered decision points. In case the following activity is an invisible one, the tracking continue until it founds the first next visible activity. It can terminate its search prematurely in correspondence of an ambiguous parts of the net, such as a joint, and consequently it is not able to identify the encountered decision points for that specific trace.

In [4], the authors solve the issue by considering a complete realisation of the net, exploiting the optimal alignment between the Petri Net and the event log introduced in [5]. A downside of this approach, thus, is that the algorithm discards all the suboptimal sequences.

Authors in [6] propose a different but very interesting approach to discover an holistic decision model linking decision to changes in variable.

[2] https://www.omg.org/spec/DMN.

[3] code available at https://github.com/piepor/multiple-paths-decision-mining.

In this work, based on [3] and [4], we propose to use multiple possible paths in the model connecting two successive activities in a variant to identify as many decision points affected by the case as possible. Since multiple paths are mainly a consequence of invisible activities, considering more than one path helps to add information about the invisible branches of decision points in their training set. More information allows for better classification and more precise rules.

3 Preliminaries

We briefly describe the problem we want to solve and two entities useful to understand it better.

A *Petri Net* [7] is a bipartite graph composed of a set of places and transitions connected by a set of arcs. Tokens distributed inside the places of the net represent the *state* of a Petri Net, called *marking*. They allow a transition to fire if all its input places contain at least one token. When the transition fires, it removes one token for each input place and gives one to every output place, changing the *state* of the net.

A *Petri Net with data* [8] is a *Petri Net* in which transitions (modeling activities) can *read* and *write* variables [4]. Write operations let firing transitions modify the value of a set of attributes, while read operations enable the use of guards. The latter are additional conditions on the firing of transitions; to be allowed to fire, the guard of a transition must evaluate to true.

The problem a decision mining technique aims to solve is to discover the conditions controlling the behaviour of an examined process. Given a process model in the form of a Petri Net and the related event log, the overall output of the method should be the original model augmented with guards on places with more than one output arc alongside relations between transitions and write operations on attributes, i.e. a Petri Net with data. In particular, we focus on the extraction of the guards' rules.

4 Methods

As already mentioned, our work aims to create the training sets for the decision points classification with the least amount of information loss. Given two subsequent events in a trace, multiple paths in the Petri Net model of the process could be eligible to go from the activity executed in the first event to the next one.

Our method tries to incorporate information from all the allowed paths as much as possible. We use a backward depth-first search to extract the encountered decision points and related targets between two subsequent activities in the trace to achieve this result.

The complete method has two main phases: the decision points extraction and the dataset creation. The first step searches and stores the places with more than one outgoing arc crossed by the allowed paths for every pair of subsequent activities in a variant. When a place is stored, the method also adds the related

output transition, i.e. the choice made. Then, for every trace belonging to the variant, the algorithm creates the dataset using the previously extracted decision points.

We used Decision Trees classifiers as in [3] and trained them with the C4.5 algorithm [9] which also describes methods to extract the decision rules.

Algorithm 1. Algorithm to extract the decision points between two activities. Starting from one activity running the backward search on the previous in the trace until a reachable one is found.

```
function extractDPs(prevSequence, currTrans)
    mapDPs ← ∅
    prevSequenceReversed ← reverseSequence(prevSequence)
    for prevAct ∈ prevSequenceReversed do
        mapDPs, found ← bwdDFS(prevAct, currTrans, ∅)
        if found == True then
            break
        end if
    end for
    return mapDPs
end function
```

4.1 Decision Points Extraction

As already introduced, our method relies on a backward depth-first search to identify the decision points crossed going from the activity contained in an event to the following one in a trace. Considering a sequence of two activities, $\langle A, B \rangle$, the method starts from the transition B in the Petri Net and searches backwards for the previous activity, A.

In the context of a Petri Net, backwards indicates that the depth-first search follows only the input arcs of every place or transition. In order to consider valid paths, the search is allowed to continue only if the transition encountered is invisible. If the transition considered is visible, whether it is the desired one or not, the search shall stop following that particular path and return the result of the search. If the visible transition is the desired one, the algorithm will add the saved decision points to the ones found on the allowed paths.

Two main issues arise designing the algorithm, namely loops and non-reachable activities. Loops are a problem when considering all the possible paths between two activities: if a loop is composed of only invisible transitions, an infinite number of allowed paths exist. To address this problem, every time an invisible transition is visited, it is added to a list of visited transitions. If the algorithm finds an already visited transition, it will stop the search along that path.

The other issue is related to non-reachable activities. An activity can be non-reachable from another for two main reasons: concurrent activities and not perfectly fitting models. If two transitions, A and B, are on two parallel branches

Algorithm 2. Recursive algorithm to extract decision points and related targets by performing a backward depth-first search of the Petri Net through invisible transitions starting from $currAct$ up to $prevAct$.

```
function BWDDFS(prevAct, currAct, passedActs)
    prevActFound = False
    for inputArc ∈ getInputArcs(currAct) do
        inPlace ← getSource(inputArc)
        backTrans ← ∅
        for innInputArc ∈ getInputArcs(inPlace) do
            transition ← getSource(innInputArc)
            backTrans ← addTrans(transition, backTrans)
        end for
        if prevAct ∈ backTrans then
            prevActFound = True
            if isDP(inPlace) then
                mapDPs ← updateDPs(inPlace, currAct, mapDPs)
            end if
            continue
        end if
        invActs ← getInvisibleTransitionsFromInputs(backTrans)
        for invAct ∈ invActs do
            if invAct ∉ passedActs then
                passedArcs ← addArc(invAct, passedArcs)
                mapDPs, found ← BWDDFS(prevAct, invAct, passedActs)
                passedArcs ← removeArc(invAct, passedActs)
                if found == True ∧ isDP(inPlace) then
                    mapDPs ← updateDPs(inPlace, invAct, mapDPs)
                end if
                prevActFound = prevActFound ∨ found
            end if
        end for
    end for
    return mapDPs, prevActFound
end function
```

of the model, it will be impossible for the backward search to find A starting from B. In the same way, if the model does not perfectly represent a variant, two activities may be non-reachable from one another. To solve the non-reachability issue, if the method cannot find the previous activity, it will search for the next previous activity in the sequence. The search will stop when it finds a reachable activity or all the previous activities are non-reachable.

Algorithm 1 reports the overall search algorithm. The search is done for every activity in the sequence, starting from the last and going backwards. The cycle breaks if the backward depth-first search finds the previous activity. Algorithm 2 reports the backward depth-first search algorithm, which has a recursive structure. We remind the reader that every transition in a Petri Net has only input places, and conversely, places have only input transitions.

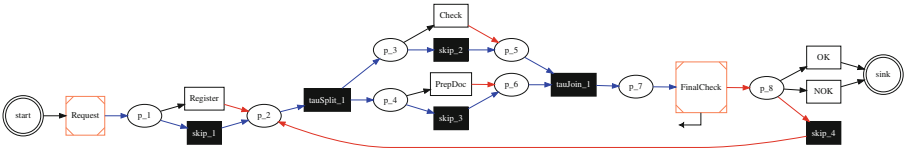

Fig. 1. Petri Net of the running model. The highlighted transitions are the activities used in Sect. 4 to explain the decision points selection. The black arrow indicates the direction and initial transition of the depth-first search. Blue arcs represent valid paths where the recursion will go on. Red ones, instead, track the recursive function until it stops.

Example. To clarify how the algorithm works, consider the variant $\langle$*Request*, *FinalCheck*, *OK*$\rangle$ and the running model reported in Fig. 1. We want to find decision points crossed by the variant going from activity "*Request*" to "*FinalCheck*". Blue arcs are the valid paths that lead the algorithm to the desired transition while red ones represent paths finding other visible transitions than the target one and, consequently, stopping the search.

The algorithm finds the two valid paths using two recursion lines, one passing through "*p*_5" and the other through"*p*_6". When the first recursion finds "*Request*", the function returns a boolean variable indicating the output to the upper levels until "*tauJoint*_1", where the other recursion line will start. Every recursion level has to remove the visited arcs; otherwise, the second recursion line will encounter the input arc of "*tauSplit*_1", marked as already visited, and will stop without reaching the desired transition.

4.2 Training Dataset Creation

The overall algorithm to create the training datasets for the decision trees has two parts. Firstly, for every variant, the method retrieves the data structure relating decision point and targets to every activity contained in the variant, as already explained.

Then, it considers the traces belonging to the variant. For every event in the trace, the algorithm creates a row containing the attributes seen until the previous event and adds it to the decision points mapped in the data structure. If an attribute is repeated multiple times in the sequence of events, the method considers only the most recent one. If an attribute in the row is not present in the dataset, the algorithm adds a column with missing values in previous entries and then adds a new row.

To study the influence of attributes locality on the quality of the classification, we also consider a variation of the method with datasets created considering only the last event in the sequence.

Since every search finds at maximum one visible transition and possibly many invisible ones, the resulting datasets may be unbalanced towards invisible targets. To overcome the problem and balance the data, we use an under-sampling technique, training multiple classifiers with different datasets, each discarding part of the over-represented targets.

5 Results

In this Section we report the results of our method. We use the Road Traffic Fine Management Process [10] to compare our algorithm to the implementation of [4] in the ProM software [11]. We use the F1 score to evaluate and compare the method, which considers the classification's accuracy and precision. Since the classifier predicts the output based on the same splits composing the final rules, better predictions, i.e. higher F1 scores, lead to more precise rules.

We know that more precise rules do not imply better or more meaningful ones, and we use this score for a straightforward comparison with the ProM implementation of the state-of-the-art method that reports this metric.

It is crucial to note that our method does not aim to outperform the state-of-the-art on the classification task but to avoid its information loss. We use the classification performance as a first assessment of our work's validity and leave the analysis of the meaningfulness of the rules and the quality of the resulting model to future studies.

Moreover, since the method aims at discovering decisions inside a process assumed to be static, we are not interested in the classifier's generalisation power and compute the F1 score on the training set without cross-validation.

Table 1. F1 score for the Road Traffic Fine Management Process dataset.

Decision Point	Optimal Alignment	Multiple Paths - Last Event	Multiple Paths
p_1	0.755	0.428	**0.788**
p_2	0.458	0.909	**0.909**
p_3	0.567	0.588	**0.818**
p_5	-	0.259	**0.83**
p_7	-	0.572	**0.893**
p_{12}	0.808	0.727	**0.898**
p_{13}	-	0.11	**0.895**
p_{14}	-	0.766	**0.948**
p_{15}	-	0.649	**0.686**
p_{19}	-	0.872	**0.957**
p_{26}	0.933	0.756	**0.952**

Table 1 reports the F1 score for two different versions of our method compared to the state-of-the-art method based on optimal alignment. To extract the model we used the Inductive Miner [12] with the noise threshold set to 0. As mentioned in Sec. 4.2, we also report the results of our method considering only attributes from the last event in the sequence. The results of our methods are the average across ten different classifiers due to the under-sampling explained in Sec. 4.2.

As expected, the results considering only attributes from the last event in the trace are noticeably worst, with few exceptions like "p_2" and "p_19", meaning that decisions depend on variables written in different parts of the model.

The state-of-the-art method based on optimal alignment cannot discover some of the decision points in the model, while our complete method can find all of them. Considering the decision points discovered by both methods, our has consistently higher F1 scores.

6 Conclusions and Future Works

In this work, we presented a novel method for decision mining on Petri Net that considers multiple possible paths taken by the variant. It can find all the decision points with higher classification quality than the state-of-the-art method.

In future works, we plan to study the quality of the extracted rules and the related model as well as differences with the rules and model retrieved by the state-of-the-art method. We will also test the approach on different datasets, hence different Petri Net sizes, and focus on the influence of the initial Petri Net fitness and complexity on the method's output.

Lastly, we plan to change the method implementation to guarantee that the model's arcs and nodes are visited at maximum once for search.

References

1. Blenko, M.W., Mankins, M.C., Rogers, P.: The decision-driven organization. Harv. Bus. Rev. **88**(6), 54–62 (2010)
2. Leewis, S., Smit, K., Zoet, M.: Putting decision mining into context: a literature study. In: Agrifoglio, R., Lamboglia, R., Mancini, D., Ricciardi, F. (eds.) Digital Business Transformation. LNISO, vol. 38, pp. 31–46. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-47355-6_3
3. 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). https://doi.org/10.1007/11841760_33
4. de Leoni, M., van der Aalst, W.: Data-Aware Process Mining: Discovering Decisions in Processes Using Alignments, pp. 1454–1461 (2013)
5. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance checking using cost-based fitness analysis. In: IEEE 15th International Enterprise Distributed Object Computing Conference (2011)
6. De Smedt, J., Hasić, F., vanden Broucke, S.K., Vanthienen, J.: Holistic discovery of decision models from process execution data. Knowl. Based Syst. **183**, 104866 (2019)
7. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice Hall. (1981)
8. Sidorova, N., Stahl, C., Trčka, N.: Soundness verification for conceptual workflow nets with data: early detection of errors with the most precision possible. Inf. Syst. **36**, 1026–1043 (2010)
9. Salzberg, S.L.: C4.5: Programs for Machine Learning" by J. Ross Quinlan. Morgan Kaufmann Publishers Inc (1993). (Mach Learn 16, 235–240 (1994))
10. Mannhardt, F., de Leoni, M., Reijers, H.A., et al.: Balanced multi-perspective checking of process conformance. Computing **98**, 407–437 (2016)

11. Eric Verbeek, J.B., van der Aalst, W.M.P.: ProM 6: The Process Mining Toolkit. Eur. J. Oper. Res. (2010)
12. Bogarín, A., Cerezo, R., Romero, C.: Discovering learning processes using inductive miner: a case study with learning management systems (LMSs). Psicothema **30**, 322–329 (2018)