ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/304480367

Discovering Decision Models from Event Logs

Conference Paper · July 2016

DOI: 10.1007/978-3-319-39426-8_19

CITATIONS
28

READS
882

3 authors, including:

Ekaterina Bazhenova
Hasso Plattner Institute
14 PUBLICATIONS **167** CITATIONS
SEE PROFILE

Mathias Weske
Hasso Plattner Institute
369 PUBLICATIONS **12,086** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Ubiquitous Business Processes View project

Declarative process discovery View project

All content following this page was uploaded by Ekaterina Bazhenova on 17 May 2018.

The user has requested enhancement of the downloaded file.

Discovering Decision Models from Event Logs

Ekaterina Bazhenova, Susanne Buelow, and Mathias Weske

Hasso Plattner Institute at the University of Potsdam,
ekaterina.bazhenova,mathias.weske@hpi.de,
susanne.buelow@student.hpi.de

Abstract. Enterprise business process management is directly affected by how effectively it designs and coordinates decision making. To ensure optimal process executions, decision management should incorporate decision logic documentation and implementation. To achieve the separation of concerns principle, the OMG group proposes to use Decision Model and Notation (DMN) in combination with Business Process Model and Notation (BPMN). However, often in practice, decision logic is either explicitly encoded in process models through control flow structures, or it is implicitly contained in process execution logs. Our work proposes an approach of semi-automatic derivation of DMN decision models from process event logs with the help of decision tree classification. The approach is demonstrated by an example of a loan application in a bank.

Key words: business process, decision management, decision mining

1 Introduction

Business process management is widely used by many companies to run their businesses efficiently. Business process performance essentially depends on how efficiently operational decisions are managed. An interest from academia and industry in the development of decision management has lead to the recently emerged DMN standard [13] aimed to be complementary to the BPMN standard [12].

To assist companies with successful automated decision management, knowledge about ”as-is” decision making needs to be retrieved. This can be done by analysing process event logs and discovering decision rules from this information. Existing approaches to decision mining concentrate on the retrieval of control flow decisions but neglect data decisions and dependencies that are contained within the logged data. To overcome this gap, we extended an existing approach to derive control flow decisions from event logs [14] with additional identification of data decisions and dependencies between them. Furthermore, we proposed an algorithm for detecting dependencies between discovered control flow and data decisions. The output of this approach is a complete DMN decision model which explains the executed decisions, which can serve as a blueprint for further decision management.

The remainder of the paper is structured as follows. In Section 2, we introduce the foundations and a running example for the paper. Section 3 presents the discovery of control flow and data decisions from the event logs. In Section 4 we propose an algorithm for identifying dependencies between discovered decisions. We then apply the presented concepts on the use case and introduce a prototypical implementation of

the DMN model extraction in Section 5. Related work is then discussed, followed by the conclusion in Section 7.

2 Preliminaries

2.1 Definitions

For our work, we rely on notions of process model and execution as follows.

Definition 1 (Process Model). A *process model* is a tuple $m = (N, C, \alpha)$, where $N = T \cup G$ is a finite non-empty set of control flow nodes, which comprises sets of activities T, and gateways G. $C \subseteq N \times N$ is the control flow relation, and function $\alpha : G \rightarrow \{xor, and\}$ assigns to each gateway a type in terms of a control flow construct. $\diamond$

Definition 2 (Process Execution, Process Instance, Activity Instance). Let m be a process model. A process execution is a sequence of activity instances $t_1 \ldots t_n$, with $n \in \mathbb{N}$ and each t_i is an instance of an activity in the set of activities T of m. $\diamond$

Definition 3 (Event Instance, Event Attributes, Trace, Event Log). Let E be the set of event instances and A a finite set of attributes. Each attribute $a \in A$ is associated with the corresponding domain $V(a)$, which represents a set of either numeric or nominal values. Each event instance $e \in E$ has tuples $(a, v),\ a \in A,\ v \in V(a)$ assigned to it. A trace is a finite sequence of event instances $e \in E$ such that each event instance appears in the trace only once. An event log L is a multi-set of traces over E. $\diamond$

We assume that an activity name in the process model corresponds to related event instance name in an event log. For simplification purpose, we also assume that the business processes do not contain loops, which we plan to consider in future work.

To represent the knowledge about decisions taken in business processes, we use the DMN standard, which distinguishes between two semantic levels: the *decision requirements* and the *decision logic*. The first one represents how *decisions* depend on each other and what *input data* is available for the decisions; these nodes are connected with each other through *information requirement edges*.

Definition 4 (Decision Requirement Diagram). A decision requirement diagram *DRD* is a tuple (D_{dm}, *ID*, *IR*) consisting of a finite non-empty set of decision nodes D_{dm}, a finite non-empty set of input data nodes *ID*, and a finite non-empty set of directed edges *IR* representing the information requirements such that $IR \subseteq D_{dm} \cup ID \times D_{dm}$, and $(D_{dm} \cup ID, IR)$ is a directed acyclic graph (*DAG*). $\diamond$

A decision may additionally reference the decision logic level where its output is determined through an undirected association. One of the most widely used representation for decision logic is a decision table, which we utilize for the rest of the paper.

Definition 5 (Decision Table). Decision table $DT = (I;\ O;\ R)$ consists of a finite non-empty set I of inputs, a finite non-empty set O of inputs, and a list of rules R, where each rule is composed of the specific input and output entries of the table row. $\diamond$

An example of the decision model is presented in Fig. 5b: *decisions* are rectangles, *input data* are ellipsis, *information requirement edges* are directed arrows. The decision logic is not presented visually in the decision requirements diagram.

2.2 Running Example

Our example process represents a loan application in a bank, as shown in Fig. 1. Although we used a Petri net for the model representation, our approach can be applied to a wider class of notations, e.g., BPMN. The process starts with the registration of the user's claim, depicted in the model by the transition *Register claim*. The claim details are recorded in the bank system as the attributes of a token produced by this transition: the *Amount* the person claims (in EUR), the desired payback *Rate* (in EUR) per month, the payback *Duration* (in months), and if the customer has a *Premium* status. Afterwards, an expert decides if a *Full check*, a *Standard check*, or a *No check* activity should be executed. The process proceeds by executing an *Evaluation* activity, deciding if the client's claim is accepted or rejected followed by sending corresponding letters to the client. This transition is followed by recording the *Risk* attribute of the token produced.

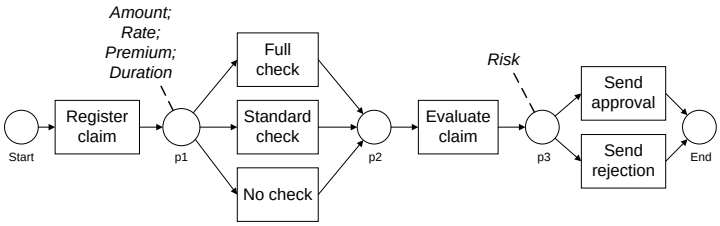


Fig. 1: Process model of the loan application in a bank

For analysing the example process, we created an event log with the help of the simulation system CPN Tools[1]. This tool uses coloured Petri nets for models' representation which allow tokens to have data values attached to them, as in our example process. We used the simulation parameters as presented in Table 1.

Task / Attribute Name	**Simulation Parameters**
Trace ID	1 to 200 (incrementing)
Amount	discrete(2,99)
Premium	random boolean
Duration	discrete(2,30)
Rate	*Amount / Duration*
Risk	if *Amount* $\geq$ 50 and *Duration* $>$ 15 : *Risk* $=$ 4 if *Amount* $\geq$ 50 and *Duration* $<$ 15 and *Duration* $>$ 5 : *Risk* $=$ 3 if *Amount* $\geq$ 50 and *Duration* $<$ 5: *Risk* $=$ 2 if *Amount* $<$ 50 and *Duration* $>$ 20: *Risk* $=$ 3 if *Amount* $<$ 50 and *Duration* $<$ 20 and *Duration* $>$ 10 : *Risk* $=$ 2 if *Amount* $<$ 50 and *Duration* $<$ 10 : *Risk* $=$ 1
$p1$	if *Amount* $\geq$ 50 and *Premium* $=$ *false* : *Full check* if *Amount* $<$ 50 and *Premium* $=$ *false*: *Standard check* if *Premium* $=$ *true*: *No check*
$p3$	if *Risk* $\leq$ 2: *Send approval* if *Risk* $>$ 2: *Send rejection*

Table 1: Simulation parameters for generating the event log of the process from Fig. 1

Table 2 shows a fragment of the simulated event log for the process depicted in Fig. 1. For each event instance e, the event log records the event ID, the trace ID referring it

[1] http://cpntools.org/

to the corresponding process instance, the name of the executed task, and the set of other event attributes $a \in A$ logged when a token is produced by the corresponding transition. For example, the event instance 1 has the following attributes: *Amount* $a_1 = 84$ [EUR], *Rate* $a_2 = 2.8$ [%], *Duration* $a_3 = 30$ [Mths], *Premium* $a_4 = false$. All other information, e.g., the timestamps of event instances is discarded.

Event ID	Trace ID	Name	Other attributes
1	1	Register claim	Amount = 84 [EUR], Rate = 2.8 [%], Duration = 30 [Mths], Premium = false
2	1	Full check	-
3	2	Register claim	Amount = 80 [EUR], Rate = 4.4 [%], Duration = 18 [Mths], Premium = true
4	2	No check	-
5	1	Evaluate	Risk = 3
6	1	Send rejection	-

Table 2: An excerpt of the event log for the process depicted in Fig. 1

Whereas the knowledge about the process decisions can be empirically derived from the logged expert decisions depicted in Table 2 in the form of credit evaluation rules, the corresponding process model depicted in Fig. 1 does not allow for decision knowledge to be obtained. Moreover, simply applying these rules for the development of credit scoring systems can lead to the unjust treatment of an individual applicant; e.g. judging the applicant's creditability by the first letter of a person's last name [6]. Thus, tt seems reasonable to use automated credit scoring systems [5] complemented with an explanatory model, and therefore, we propose further in this paper to derive the DMN decision model from the process event log. An advantage of using such a model is that the separation of process and decision logic maximizes agility and reuse of decisions [15]. We demonstrate the DMN model derivation for the presented use case in Section 5.

3 Discovering Decisions from Event Logs

In this section we introduce different types of decisions in process models and propose ways of detecting them in process models and event logs.

3.1 Discovering Control Flow Decisions

In our previous work [3] we analysed around 1000 industry process models from multiple domains and discovered that the exclusive gateway was the most common decision pattern used in 59% of the models. To indicate such of type of decisions, we will use the notion of *control flow decisions*, which are represented in process models by decision structures of a single split gateway with at least two outgoing control flow edges. When the control flow decision occurs, a token placed at the split gateway, fires the needed transition, to which we will refer as to a *decision outcome*. For example, the control flow decision occurs when a token, placed at the split gateway $p1$ (see Fig. 1), fires the needed transition, thus, it is decided which of the activities *Full check*, *Standard check*, or *No check* should be executed.

For deriving the control flow decisions and the decision logic behind it, we propose the following approach. Firstly, the control flow decisions are identified in the input process model. Given a process model *m*, we determine constructs of directly succeeding control flow nodes, that represent a gateway succeeded by a task on each of the

outgoing paths. The step output is a set of M control flow decisions $P = \{p1, \ldots, pM\}$ corresponding to a process model m.

Next, we want to derive the decision logic from the event log in the form of a decision table. A decision table is a tabular representation of a set of related input and output expressions, organized into rules indicating which output entry applies to a specific set of input entries [13]. Below we introduce the notion of the decision rule.

Definition 6 (Decision Rule). Given is an event log L and a corresponding set of attributes $A = (A_1, \ldots, A_v), v \in \mathbb{N}^*$. The *decision rule* is a mapping

$$A_1 \; op \; q_1, \ldots, A_w \; op \; q_w \longrightarrow c_l, \; 1 \leq w \leq v \tag{1}$$

where the attributes A are decision inputs, op is a comparison predicate, $q_1, \ldots, q_w$ are constants, and $c_l \in C_{p*} = (c_1, \ldots, c_s)$, $s, l \in \mathbb{N}^+, 1 \leq l \leq s$ is a decision output of the control decision $p* \in P$. ◇

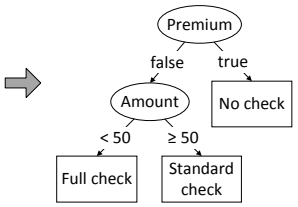
For example, the decision rule for the control flow decision $p1$ (see Fig. 1) is:

$$Premium = false, \; Amount < 50 \longrightarrow Full\ check \tag{2}$$

The control flow decision rules can be derived using the approach introduced in [14]. It is based on the idea that a control flow decision can be turned into a classification problem, where the classes are process decisions that can be made, and the training examples are the process instances recorded in the event log. All the attributes recorded in the event log *before* the considered choice construct are assumed as relevant for the case routing. In accordance with [14], we propose to use decision trees for solving the presented problem. Among other popular classification algorithms are neural networks [1] and support vector machines [16]. Pursuing the goal of deriving an *explanatory* decision model for a business process, we stick to the decision trees, as it delivers a computationally inexpensive classification based on few comprehensible business rules with a small need for customization [2].

Case ID	Amount	Premium	Rate	Duration	Class
1	97	false	6.9	14	Full check
2	68	true	6.8	10	No check
3	30	false	3	10	Standard check

(a) Training examples



(b) Decision tree

Fig. 2: Control flow decision $p1$ represented as classification problem

The possibly influencing attributes for the control flow decision $p1$ are: *Amount, Premium, Rate, Duration*. The learning instances are created using the information from the event log as presented in the table in Fig. 2a, where lines represent process instances, and columns represent possibly influencing attributes. The *Class* column contains the decision outcome for a process instance. An example of the classification of process instances by a decision tree is presented in Fig. 2.

3.2 Discovering Data Decisions

Besides the explicit control flow decisions, process models can contain implicit data decisions. Our idea for detecting such type of decisions assumes that the values of

the atoms in the decision rules can be determined for a certain process instance by knowledge of the variable assignments of other attributes. For example, in Fig. 1, it is not exhibited how the value of an attribute *Risk* is assigned, however, in practice, it depends on the values of attributes *Amount, Rate, Premium* and *Duration* recorded in the system while registering the clients claim. We distinguish the data decisions into functional and rule-based data decisions.

Definition 7 (Rule-Based Data Decision). Given is a set of attributes $A = (A_1, \ldots, A_v)$ An attribute A_j is a *rule-based decision* if there exists a non-empty finite set of rules R which relate a subset of the given set of attributes to the aforementioned attribute A_j:

$$R = \bigcup_i A_1 \; op \; q_1, \ldots, A_l \; op \; q_l \longrightarrow V^i(A_j) \tag{3}$$

where the attributes $A_1, \ldots, A_l$ are decision inputs, *op* is a comparison predicate, $q_1, \ldots, q_l$ are constants; hereby $v, l, i \in \mathbb{N}^+,\ 1 \leq l \leq v$. ◇

A rule from the rule-based data decision for *Risk* of the process model in Fig. 1 is:

$$Premium = false,\ Amount < 50,\ Duration < 10 \longrightarrow Risk = 4 \tag{4}$$

Definition 8 (Functional Data Decision). Given is a set of attributes $A = (A_1, \ldots, A_v)$ An attribute A_j is a *functional data decision* if there exists a function $f : (A_1, \ldots, A_k) \longrightarrow A_j$ which relates a subset of the given set of attributes to the aforementioned attribute A_j; hereby $v, k \in \mathbb{N}^+,\ 1 \leq k \leq v$. ◇

A functional data decision for our running example for attribute *duration* is:

$$Duration = Amount/Rate \tag{5}$$

In Algorithm 1, we propose a way to retrieve data decisions using a process model m, an event log L, and a corresponding set of attributes of the event instances $A = (A_1, \ldots, A_v), v \in \mathbb{N}^*$ as inputs. As any of the attributes from the set A can potentially be the ouput of a data decision, the procedure runs for each attribute $a \in A$ (line 2).

Algorithm 1 Retrieving Data Decisions

```
1:  procedure FINDDATADECISIONS(processModel m, eventLog L, attributes A)
2:      for all a ∈ A do
3:          A_inf ← possibly influencing attributes for a
4:          dt ← decision tree for a using A_inf as features
5:          if dt correctly classifies all instances then
6:              return rule-based data decision for a
7:          else
8:              if a has a numeric domain then
9:                  A_infnum ← attributes from A_inf with numeric domain
10:                 operators ← {+, -, *, /}
11:                 funcs ← {function "a o b" | a, b ∈ A_infnum ∧ o ∈ operators}
12:                 for all func ∈ funcs do
13:                     if func correctly determines value of a for all instances then
14:                         return functional data decision for a
15:                         break
```

Firstly, in line 3, a set A_{inf} of attributes possibly influencing a is determined using the assumption that all the attributes are recorded in the event log before or equal to the transition to which the attribute a is referred. Afterwards, the procedure detects whether

an attribute a represents a *rule-based data decision* (lines 4 - 6). For this, we build a decision tree dt classifying the values of the attribute a using the set of possibly influencing attributes A_{inf} as features. If the built decision tree classifies all training instances correctly, a rule-based data decision for the attribute is yielded (here and further in the paper we assume that the classification correctness can be adapted for business needs by using a user-defined correctness threshold in percent). If no set of rules was found, the algorithm searches for a *functional data decision* for attribute a (lines 8 - 15). For this, we consider only such attributes which have a numeric domain. We determine the function form by a template representing combinations of two different attributes from A_{inf} connected by an arithmetic operator (10). The functions representing all possible combinations of such kind, are tested for producing the correct output for all known instances (lines 11 - 12). If that is the case, a functional data decision for a is returned (line 14). If it is determined that an attribute a is neither a rule-based, nor a functional data decision, a is discarded as a possible data decision and is treated as a normal attribute.

For our running example, the algorithm finds a rule-based data decision for attribute *Risk* depending on the attributes *Amount, Premium*, and *Duration* as presented in Fig. 3a. An instance of a functional decision is Equation 5.

3.3 Mapping of the discovered decisions with DMN model

The detected decisions represent decision nodes in the DMN decision requirement diagram which conforms to the original process model. The algorithm for constructing this diagram is straightforward. Firstly, for each discovered control flow decision $p \in P = \{p1, \ldots, pM\}$, we create a new decision node which is added to the set of decision nodes D_{dm} of the decision requirements diagram. Further, for each attribute of the event log $A_j \in A = (A_1, \ldots, A_v), v \in \mathbb{N}^*$ it is checked whether it is a rule-based or functional decision over other attributes from the set of attributes A. If this is the case, then a new decision node corresponding to this attribute A_j is added to the output decision requirements diagram *DRD*. Otherwise, we create a data node corresponding to this attribute A_j which is added to to the set of input data nodes *ID* of the decision requirements diagram. The decision nodes reference the corresponding decision tables containing the extracted rules. An example mapping is presented in Fig. 5a.

4 Discovering Decision Dependencies from Event Logs

The decisions discovered in process models as presented above, are used for creating a set of the decision nodes D_{dm} in the output decision requirement diagram *DRD*. For now, these decisions represent isolated nodes in the decision requirement diagram, and to "connect" them by the information requirements *IR*, in this section we propose an algorithm of discovering the decision dependencies from the event log.

4.1 Discovering Decision Dependencies

We propose to distinguish between two following types of decision dependencies.

Definition 9 (Trivial decision dependency). If a decision d in the set of decision nodes detected in the event log ($d \in D_{dm}$) depends on the attribute $a_k \in A$ from the event log and this attribute is detected to be a data decision ($dd \in D_{dm}$), then there is a *trivial dependency* between this decision d and the decision dd; here $k \in \mathbb{N}^+$. ◇

Definition 10 (Non-trivial decision dependency). *Non-trivial dependencies* are dependencies between control flow decisions and other decisions. ◇

To find the dependencies between either control flow, or data decisions detected in the event log of a process model, we propose Algorithm 2. The inputs to the algorithm are process model m, event log L, and a corresponding set D_{dm} of the detected decisions. Each decision d is tested for being influenced by other decisions (line 2). Firstly, the algorithm searches for *trivial decision dependencies* (lines 3 - 6). In line 3, we identify all attributes A_{inf} influencing the decision d (those are the attributes appearing the set of rules or in the function of the decision). For each attribute a_{inf} in the set of influencing attributes A_{inf}, we check if there is a data decision deciding a_{inf}. If this is the case, the algorithm yields a trivial decision dependency between this data decision on a_{inf} and the decision d (line 6).

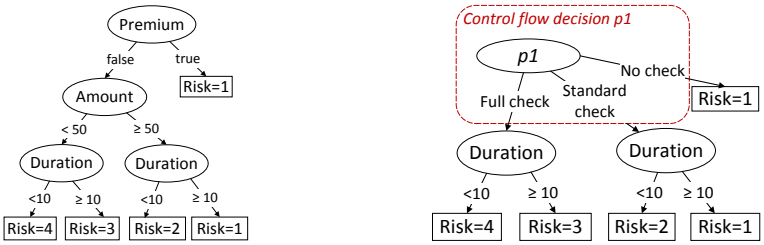
Algorithm 2 Retrieving Decision Dependencies

```
1:  procedure FINDDEPENDENCIES(processModel m, eventLog L, decisions D_dm)
2:      for all d ∈ D_dm do
3:          A_inf ← attributes influencing d
4:          for all a_inf ∈ A_inf do
5:              if D_dm contains a data decision dd for a_inf then
6:                  return decision dependency dd → d                ▷ trivial dependency
7:          D_inf ← possibly influencing control flow decisions for d
8:          for all d_inf ∈ D_inf do
9:              Feat ← A_inf - attributes influencing d_inf ∪ {d_inf}
10:             dt ← decision tree for d using Feat as features
11:             if dt correctly classifies all instances then
12:                 return decision dependency d_inf → d             ▷ non-trivial dependency
```

Afterwards, the algorithm searches for *non-trivial decision dependencies* (lines 7 - 12). We firstly identify a set of control flow decisions D_{inf} possibly influencing the decision d in line 7. Data decisions are not considered, because a data decision influencing another decision always results in a trivial decision dependency. Any decision $d_{inf} \in D_{inf}$ should satisfy two conditions: (1) d_{inf} is bound to a transition in the model that lies before or is equal to the transition of d; and (2) the set of influencing attributes of d_{inf} is a subset of the set of influencing attributes of d. Next, the algorithm detects whether there is a non-trivial decision dependency between each found possibly influencing decision d_{inf} and d. For this sake, we determine a new set of features *Feat* for the decision d (line 9). This set consists of the attributes influencing d without the attributes influencing d_{inf}, but with the output of decision d_{inf}. Using the features from *Feat*, we build a new decision tree deciding on d. If the tree is able to correctly classify all training instances, we have found a non-trivial decision dependency between d_{inf} and d.

In our example, the decision on attribute *Risk* depends on the attribute *Duration* (Equation 4). As we identified *Duration* as being a data decision (Equation 5), there is a trivial decision dependency between the decisions *Duration* and *Risk*. For an example of a non-trivial dependency, have again a look at the *Risk* decision in Fig. 3b. The found decision tree for *Risk* depends on the attributes *Amount, Premium* and *Duration*. We identify the control flow decision $p1$ as a possibly influencing decision, as (1) the decision $p1$ happens before the decision *Risk*; and (2) its influencing attributes (*Amount, Premium*) are a subset of the influencing attributes of the decision *Risk*. Further, we can

build a decision tree that correctly classifies all instances by using the output of decision $p1$ instead of the attributes *Amount* and *Premium*. Note that the attribute *Duration* is in the output decision tree in Fig. 3b, as it is not part of the decision $p1$.



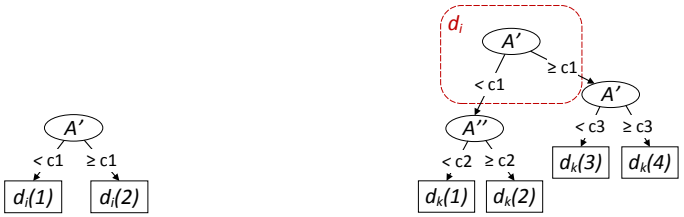
(a) Rule-based data decision *Risk* (b) Decision dependency between $p1$ and *Risk*

Fig. 3: Decision trees for attribute *Risk*

It might be the case that circular dependencies between attributes in the same transition in process model are discovered. For example, in Equation 5, the discovered functional data decision *Duration* depends on *Amount* and *Rate*. However, as *Duration, Amount* and *Rate* appear in the same transition, Algorithm 2 finds the data decisions for *Amount* depending on *Duration* and *Rate*, as well as for *Rate* depending on *Duration* and *Amount*. However, in the output decision model two of these three data decisions should be discarded to avoid cyclic dependencies. We leave it to the process expert to determine which decision out of a set of cyclic data decisions is the most relevant for the output decision model.

4.2 An Improved Approach to Finding Non-Trivial Decision Dependencies

The decision dependencies in Algorithm 2 are retrieved under the assumption that if an arbitrary decision d_k depends on another decision d_i $(k, i \in \mathbb{N}^+)$, then this decision d_k can *not* additionally depend on attributes that d_i depends on. This problem is illustrated in Fig. 4: here, d_i depends on attribute A' and d_k depends on decision d_i and additionally on the attributes A' and A''. The Algorithm 2 tries to rebuild the decision tree for d_k without considering the attributes of d_i, and it only utilizes attribute A''. Thus, this algorithm finds no dependency between d_i and d_k, which is not correct.



(a) The decision tree for decision d_i (b) The decision tree for decision d_k

Fig. 4: Decision d_k depends on another decision d_i, *and* on the attributes influencing d_i

To overcome this problem, we propose an alternative Algorithm 3 for finding non-trivial decision dependencies in the process event log (finding of trivial decision dependencies is equivalent to lines 3 to 6 in Algorithm 2). Firstly, the Algorithm 3 identifies a set of possibly influencing control flow decisions D_{inf} and for each $d_{inf} \in D_{inf}$ it builds

a decision tree dt_{inf} containing (1) one root node, that splits according to d_{inf}; (2) as many leaf nodes as d_{inf} has decision outcomes, thereby, each of them containing a set of learning instances. Then, for each leaf node a subtree is built that decides on d for all

Algorithm 3 Alternative Retrieving of Non-Trivial Decision Dependencies

```
1:  procedure FINDDEPENDENCIESNEW(processModel m, eventLog L, decisions D_dm)
2:      for all d ∈ D_dm do
3:          D_inf ← possibly influencing control flow decisions for d
4:          for all d_inf ∈ D_inf do
5:              dt_inf ← decision tree where the root node splits according to d_inf
6:              dt_new ← dt_inf
7:              for all leaf ∈ dt_inf do
8:                  dt_leaf ← decision tree for d classifying instances from leaf
9:                  dt_new ← add dt_leaf to the leaf of dt_new
10:             levels_inf ← number of levels of dt_new
11:             levels_orig ← number of levels of original decision tree for d
12:             if (levels_inf <= levels_orig then
13:                 return decision dependency d_inf → d
```

learning instances of this leaf, thereby the features are all attributes that were used in the original found decision tree for d. In lines 7, all subtrees are attached to dt_{inf} resulting in dt_{new} which is a decision tree for d. Next, it is checked that the complexity of the newly constructed tree dt_{new} has not increased in comparison to the original decision tree for d by measuring and comparing the corresponding maximum number of nodes from the root to the leafs (lines 11 - 14). If this is the case, then the algorithm outputs a decision dependency between d_{inf} and d.

4.3 Mapping of Discovered Decision Dependencies with DMN model

The trivial and non-trivial decision dependencies detected in the process event log are directly mapped to the set of information requirements *IR* represented by directed arrows between the discovered decision nodes D_{dm} and input data nodes *ID* in the output decision requirements diagram *DRD*. An example mapping is presented in Fig. 5b.

5 Application of the Approach on an Example Log

For evaluating our approach of the decision model discovery from the event log of a process model, we implemented it as a plug-in for the ProM framework 5.2[2] by extending the existing plug-in "Decision Point Analysis" for the discovery of control flow decision points [14] with our concepts presented in Sections 3 and 4. The ability of the tool to derive decision models from event logs is shown in a screencast[3] using the running example from Section 2.2. The input for the approach is an event log of a process model simulated as discussed in Section 2.2, from which we mine the process model using one of the ProM process mining algorithms. As we have now both process model in the form of Petri net (Fig. 1), and corresponding event log, we can start the discovery of decisions. The screencast reflects our step-by-step approach proposed for the discovery of decisions from event logs, which is described below.

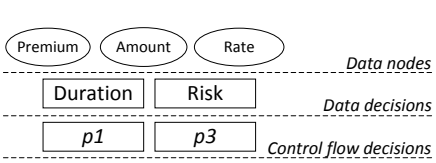
1. Discovery of control flow decisions. According to approach from Section 3.1, the program identifies two control flow decisions: (1) $p1$ with decision alternatives *Full*

[2] http://www.promtools.org/

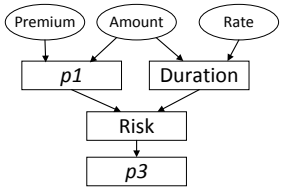
[3] https://bpt.hpi.uni-potsdam.de/foswiki/pub/Public/WebHome/DMNanalysis.mp4

check, Standard check, and *No check*; and (2) $p3$ with decision alternatives *Send approval* and *Send rejection*. A decision tree constructed for $p1$ is depicted in Fig. 2.

2. Discovery of data decisions. Executing Algorithm 1, the program finds: (1) A rule-based decision *Risk* (Fig. 3a); (2) A functional decision *Duration* (Equation 5).



(a) Discovered decisions



(b) Discovered decision requirements diagram

Fig. 5: The discovered decisions and the DMN model for the example process

The aggregate of the decisions discovered by the program is presented schematically in Fig. 5a. Those are the elements which are used further for the construction of the decision requirements diagram: (1) Data nodes (*Premium, Amount, Rate*); (2) Data decisions (*Duration, Risk*); and (3) Control flow decisions ($p1$, $p3$). Additionally, the plug-in creates the decision table for each decision found (see screencast for details).

3. Discovery of decision dependencies. Further, the program executes Algorithm 2 to mine the dependencies between the discovered decisions from Fig. 5a, and it outputs the fully specified DMN decision model as depicted in Fig. 5b. Thus, the program finds the trivial dependencies between the decisions *Duration* and *Risk*, as well as between *Risk* and $p3$, also, the non-trivial dependency between the decisions $p1$ and *Risk*. In case of circular dependencies, a random decision is kept. The decision dependencies discovery, whereby the influenced decision can reuse attributes from the influencing decision as described by Algorithm 3, was not implemented yet, but it is planned for future work.

The extracted decision model (Fig. 5b) shows explicitly the decisions corresponding to the process from Fig. 1, and thus, could serve for compliance checks by explaining the taken decisions. Also, the derived decision model can be executed complementary to the process model, thereby supporting the principle of separation of concerns [15].

6 Related Work

An interest from academia and industry towards exploring the advantages of separating of process and decision logic is demonstrated a number of works [10, 17, 15, 8]. Improving business process decision making based on past experience is described in [9], but the specifics of the DMN standard is not considered. [11] explore the possibilities of improving decisions within a DMN model, but it is not concerned with the integration of process and decision models problem. This paper extends our work in [4] and is closely related to [14], which describes the extract of decision rules for control flow decisions from event logs. However, we additionally identify data decisions, and decision dependencies. [7] extends [14], but in contrast to our paper, the authors seek to improve the performance of the rule extraction algorithm for control flow decisions, while we seek to complete the decision knowledge derived from event logs beyond control flow decisions. [3] also deals with the semi-automatic extraction of process decision logic but only on the modeling level.

7 Conclusion

In this paper we provided a formal framework enabling the extraction of complete decision models from event logs on the examples of Petri nets and DMN decision models. In particular, we extended an existing approach to deriving control flow decisions from event logs with additional identification of data decisions and dependencies between them. Furthermore, we proposed a modified approach to rebuilding decision trees to identify the dependencies between discovered decisions and overcame the problem of reusing attributes in a dependent decision. An assumption of our approach was that the decisions do not appear within loops, which we plan to investigate in future work.

The extracted DMN decision model reflects the decisions detected in the event log of a process model, which could be served as an explanatory model used for compliance checks. Additionally, executing this model complementary to the process model supports the principle of separation of concerns by providing increased flexibility, as changes in the decision model can be executed without changing the process model.

References

1. B. Baesens, R. Setiono, C. Mues, and J. Vanthienen. Using neural network rule extraction and decision tables for credit-risk evaluation. *Management science*, 49(3):312–329, 2003.
2. B. Baesens, T. Van Gestel, S. Viaene, M. Stepanova, J. Suykens, and J. Vanthienen. Benchmarking state-of-the-art classification algorithms for credit scoring. *Journal of the Operational Research Society*, 54(6):627–635, 2003.
3. K. Batoulis, A. Meyer, E. Bazhenova, G. Decker, and M. Weske. Extracting decision logic from process models. In *CAiSE*, pages 349–366. Springer, 2015.
4. E. Bazhenova and M. Weske. Deriving decision models from process models through enhanced decision mining. *Proceedings of 3th International Workshop on Decision Mining and Modeling for Business Processes*, Springer, 2015 (Accepted for publication).
5. Board. Report to the Congress on Credit Scoring and Its Effects on the Availability and Affordability of Credit. Technical report, 2007.
6. N. Capon. Credit scoring systems: A critical analysis. *Journal of Marketing*, 46(2), 1982.
7. M. De Leoni, M. Dumas, and L. García-Bañuelos. Discovering branching conditions from business process execution logs. In *Fundamental Approaches to Software Engineering*, pages 114–129. Springer, 2013.
8. T. Debevoise and J. Taylor. *The MicroGuide to Process Modeling and Decision in BPMN/DMN*. CreateSpace Independent Publishing Platform, 2014.
9. J. Ghattas, P. Soffer, and M. Peleg. Improving business process decision making based on past experience. *Decision Support Systems*, 59(0):93 – 107, 2014.
10. E. Kornyshova and R. Deneckère. Decision-making ontology for information system engineering. In *ER*. Springer-Verlag, 2010.
11. S. Mertens, F. Gailly, and G. Poels. Enhancing declarative process models with DMN decision logic. In *Business Information Processing*, volume 214:151-165. Springer, 2015.
12. OMG. Business Process Model and Notation (BPMN), v. 2.0.2, 2013.
13. OMG. Decision Model And Notation (DMN), v. 1.0 - Beta 2, 2015.
14. A. Rozinat and W.M.P. van der Aalst. Decision mining in ProM. In *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, pages 420–425, 2006.
15. B. Von Halle and L. Goldberg. *The Decision Model: A Business Logic Framework Linking Business and Technology*. Taylor and Francis Group, 2010.
16. C.J. Walder. Support vector machines for business applications. *Business Applications and Computational Intelligence*, page 267, 2006.
17. A. Zarghami, B. Sapkota, Mohammad Z. Eslami, and M. van Sinderen. Decision as a service: Separating decision-making from application process logic. In *EDOC*. IEEE, 2012.

View publication stats