Optimal Acquisition of Input Data for Decision Taking in Business Processes

Ekaterina Bazhenova
Hasso Plattner Institute
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam, Germany
ekaterina.bazhenova@hpi.de

Mathias Weske
Hasso Plattner Institute
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam, Germany
mathias.weske@hpi.de

ABSTRACT

Business process management is directly affected by how effectively decision making is designed and coordinated. The recent Decision Model and Notation (DMN) standard prescribes that decision processes should be modeled and executed complementary to business processes. One of the challenges that companies face is that many business situations require decision making dealing with huge amounts of input data, but there exist no guidelines on how to prioritize inputs acquisition so that the associated costs are minimal. In this paper, we propose an approach for executing process-level decisions based on prioritization of input data through conversion of decision logic into optimal decision trees. As optimization criteria, our algorithm uses excluding of pre-existing data about a process instance from inputs to be acquired, costs associated with inputs acquisition, and predictions about decision outcomes. We demonstrate the approach on a real-world decision process by showing how to minimize and prioritize questions asked by a bank to a loan applicant so that associated costs are minimal.

CCS Concepts

•Information systems → Enterprise applications; •Applied computing → Business process management;

Keywords

Business Process, Decision Execution, Optimal Inputs Acquisition

1. INTRODUCTION

Decision making plays a crucial role in business process management, which is acknowledged as an essential asset to run a company [6]. Once process and decision models are designed, they need to be executed. During process execution, a point can be reached at which a decision needs to be made. If the principle of separation of process and decision logic [5, 15] is supported, the process activity that invokes a corresponding decision model supplies the decision-making system with input data. Next, the decision is "made", and the result is returned to the invoking activity.

As availability of data grows, organisations integrate it more firmly into the decision-making process. However, companies struggle to find optimal ways of data acquisition, conditioned upon performance requirements and cost pressures [1]. Our interviews with industry partners from the banking domain revealed a challenge, the solution of which can improve the process of data acquisition for decision making. Namely, there exist real-world situations requiring decisions, when a set of information inputs is available, and acquisition of only a subset of it might be enough to make a decision. For example, in a process of assessment of a credit or an insurance application, depending on the case, there may be questions to an applicant which could be asked firstly in order to make a decision faster. So far, the existing literature on modeling and executing decisions complementary to business processes [15, 13, 5], including the Decision Model and Notation (DMN) [11], does not provide guidelines on how to optimally collect the data needed for decision making. Thus, optimal acquisition of input data for decision making remains an open question in the context of processes.

To address the described problem, in this paper we investigate the question of acquisition of decision inputs utilizing decision tables [11]. Our contribution is an approach for executing a decision model during instantiation of the associated process model aimed at (1) reducing the number of inputs to be acquired; (2) finding an optimal order of acquisition of decision inputs. In particular, we firstly propose to exclude pre-existing process instance data from the set of inputs to be acquired during decision execution. Next, we present an algorithm for constructing a decision tree which stores an optimal ordering of acquisition of inputs needed for decision execution. As optimization criteria we use costs associated with inputs acquisition, and predictions about decision outcomes.

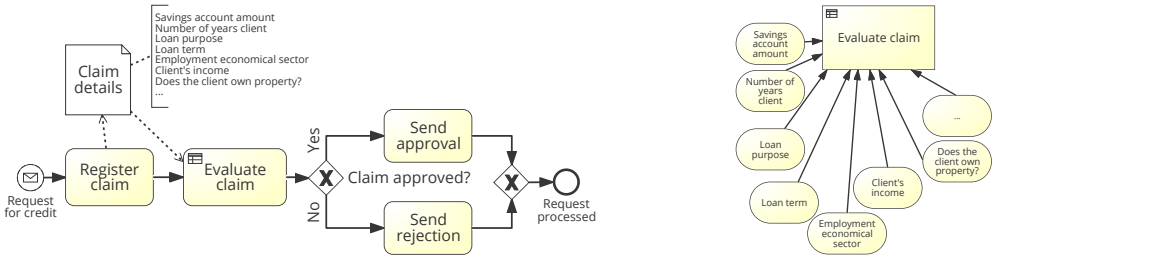
The paper remainder is structured as follows. Section 2 introduces a motivating example of a credit application process. In Section 3 we identify factors influencing inputs acquisition during decision execution. Section 4 presents a problem statement and our approach for optimal inputs acquisition during decision execution. The approach is based on the algorithm of conversion of decision tables into optimal decision trees, presented in Section 5. We evaluate the approach by applying it to the presented example in Section 6. Further, we discuss related work and conclude the paper.

2. MOTIVATING EXAMPLE

Our motivating example is a business process of credit-risk assessment in banks. Data acquisition is an essential step in this process,

SAC 2017,April 03-07, 2017, Marrakech, Morocco

http://dx.doi.org/10.1145/3019612.3019615



(a) BPMN process model (b) DMN decision requirements diagram (DRD)

Figure 1: Process and decision models corresponding to the example of assessment of credit applications in banks

as banks need to collect and interpret a large amount of information for deciding whether to accept or reject an application. But also, data acquisition requires time and resources, and therefore should be kept to a minimum unless it is absolutely necessary for taking a decision. A step-by-step approach that is generally followed while assessing a personal loan [7] is presented in Fig. 1a by a BPMN process model. Upon receiving a request for credit, bank workers collect and register application data, such as *Savings account amount, Number of years client, Loan purpose*, etc. This data is assigned to the attributes of the data object *Claim details*, which is used further as an input to the decision activity *Evaluate claim*. The process continues with execution of this decision followed either by *Send approval* or *Send rejection* activity. The depicted process model does not contain the decision rules for the evaluation process, as the paradigm of separation of process and decision logic [15, 13], also prescribed by the DMN standard, states that decision logic should be modeled separately from a process model.

INPUTS \ RULES	R1	R2	R3	R4	R5
I1: Savings account amount	<= 15 EUR	<= 15 EUR	<= 15 EUR	<= 15 EUR	> 15 EUR
I2: Number of years client	<=3	> 3	<=3	<=3	> 3
I3: Loan purpose	-	-	Not a house	Not a house	House
I4: Employment economical sector	Not sector C	Sector C	Sector C	Not sector C	-
I5: Loan term	>= 1 year	>= 1 year	>= 1 year	>= 1 year	< 1 year
I6: Client's income	>= 720 EUR	>= 720 EUR	-	-	< 720 EUR
I7: Does the client own property?	No	Yes	No	Yes	-
OUTPUT Credit Rating	A	B	C	D	E

Figure 2: Example decision table for credit rating determination

DMN defines the *decision requirements* and the *decision logic* levels. The decision requirements show how decisions depend on each other and what input data they require. In our use case, the decision activity *Evaluate claim* from Fig. 1a references a corresponding decision requirements diagram (DRD) shown in Fig. 1b. This DRD consists of a decision node *Evaluate claim* and a finite set of data nodes representing inputs needed for decision making, such as *Savings account amount*, *Number of years client*, etc. Thereby, the attributes of the data object *Claim details* in the BPMN process model correspond to the decision inputs in the DRD. A decision in the DRD may additionally reference the decision logic level where the decision output is determined by some predefined logic. Often, the decision logic is expressed with the help of decision tables. An example of the decision table for the presented case is depicted in Fig. 2; it is based on real decision rules communicated to us by our industry partner. The rows in the depicted decision table are *inputs*, which correspond to the decision inputs in the DRD. The columns group the input states, thereby providing *rules* for credit rating determination. The bottom row contains the possible decision *outputs*; in our case those are values of assigned credit rating.

When the process from Fig. 1a is instantiated and the claim details are registered, the DRD from Fig. 1b is invoked, followed by evaluation of the application according to the decision table from Fig. 2. The amount of questions the clients need to answer in a loan application can be really high: in [2], two real-life credit-risk assessment data sets are presented which are based on questionnaires consisting of 45 inputs and 105 inputs respectively. Our interviews with industry partners revealed that along with having so many pieces of information to be obtained, the lenders do not have any guidelines on how to prioritize the questions asked to applicants. Thus, they just consequently collect and verify all the input data required for making a decision. In such a way, acquisition of decision-related process data can be a very time-consuming activity.

Our interviews lead us to the idea that the process of obtaining decision-related data can be minimized and prioritized during process execution. For example, the answer for such question as *Number of years client* should exist in the information system of the bank prior to the application, and there is no need to ask the applicant this question. Moreover, as it can be seen from the decision table in Fig. 2, depending on the information about the applicant, it might be sufficient to have just a subset of answers for the questions in order to make a decision. For example, imagine that the bank gets to know that the applicant's *Savings account amount* is equal to 100 EUR, the *Number of years client* is equal to 5 years, that the *Loan purpose* is "House", and that the *Client's income* is 700 EUR. Then, Rule 5 from the table can be applied, even if the answers for the questions *Employment economical sector* and *Does the client own property?* are unknown. Usually, the banks spend some time and resources associated with inputs verification which could be skipped in this case. In such a way, the bank would save related time and resources by not asking the unnecessary questions.

To sum up, if companies had guidelines on how exactly to minimize and prioritize acquisition of decision-related data, they could save time and resources, and increase customer satisfaction from quicker service. To the best of our knowledge, the existing works do not tackle this issue. In order to fulfil the discovered gap, we firstly explore factors influencing prioritization of inputs acquisition during decision execution in the next section.

3. FACTORS INFLUENCING EXECUTION OF DECISIONS IN PROCESSES

According to [15, 13, 6], managing decisions complementary to

business processes can be broken down into the following stages of a decision management lifecycle (see Fig. 3): (1) *Decision modeling*, when decisions are identified and modelled; (2) *Decision configuration*, when decisions are implemented on enterprise platforms; (3) *Decision execution*, incorporating the actual decision enactment; and (4) *Evaluation phase*, when decisions can be assessed and improved. Optimization of decision logic in *Decision modeling* phase would be relevant for minimization of inputs to be acquired for decision execution, but it was already explored in earlier works. For example, decision rules can be discovered automatically from large credit-risk data sets, and then, the application of classification will reduce the number of decision inputs [2, 4]. In case if the decision rules are modeled manually, minimization of inputs in decision tables can be achieved through tables contractions which is well explored in [14]. Exploration of factors influencing prioritization of inputs acquisition during decision execution which might refer to the phases of *Decision configuration* and *Decision evaluation* are left out of the scope for now, but they might be considered in future works. Thereby in this paper we focus influencing prioritization of inputs acquisition during decision execution which corresponds to *Decision execution* phase of the decision management lifecycle.

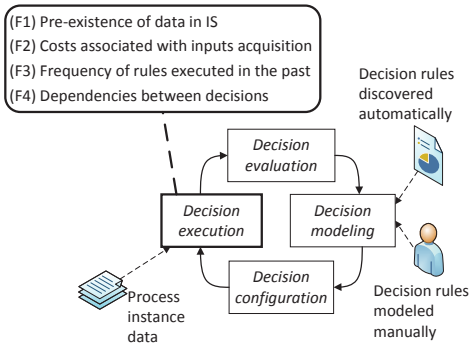


Figure 3: Overview of factors influencing *Decision execution* phase

(F1) A priori knowledge about process instance. Consideration of data which exists in an enterprise information system prior to process instantiation [6] can reduce the number of inputs to be acquired for decision making. For example, sometimes it is the case that some data about a credit applicant can already pre-exist in the information system of the bank prior to an application.

- Example 1. *Employment status* is new information for the bank, such question definitely needs to be asked;
- Example 2. *Number of years client* should exist in the information system of the bank prior to the application. There is no need to ask this question, as the values can be obtained from the information system automatically.

During process execution, such questions as in Example 2 can be eliminated, which leads to a reduced number of questions and influences the order of rest of questions.

(F2) Costs associated with decision inputs acquisition. Being one of the main process performance indicator [6, 9], costs associated with inputs verification should be considered as an optimization criterion for inputs acquisition.

- Example 1: *Loan purpose* is new information for the bank and it does not have to be verified, so no costs are involved;
- Example 2: *Number of years client* exists in the information system, the average verification time is 15 minutes, which is associated with certain costs.

(F3) Frequency of rules executed in the past. In reality, different decision outcomes have different probabilities. For comparing different ordering of questions to be asked, it is useful to consider the relative frequency of fired decision rules. The historical frequency of rules can be treated as probability that a concrete rule is triggered in comparison to all the rules in a decision table [12, 9].

(F4) Dependencies between decisions. For real-world problems, using only decision table to hold all the knowledge is not sufficient. One should be able to structure the decision logic into a set of interrelated tables. [14] distinguishes between two kinds of subtables:

- Condition tables using the outputs of some decision tables as inputs to others;
- Action tables triggered for evaluation by certain outputs of another decision tables.

Thus, consideration of these inter-tabular dependencies should be taken into account as influencing decision execution.

Although the list of factors influencing the decision execution is not exhaustive and can be continued (e.g., resource utilization associated with the procedure of inputs acquisition [6] etc.), we limit the scope of the paper by the four crucial factors presented earlier. With that, our work should serve as a basis for further multi-criterion extensions taking into account concrete business requirements.

4. DYNAMIC MINIMIZATION AND PRIORITIZATION OF DECISION INPUTS

In this section we firstly formalize the problem in Section 4.1. Further, Section 4.2 presents our approach for minimization and prioritization of inputs acquisition during decision execution.

4.1 Formalized Problem Statement

A *process model* is a tuple $PM = (N, C, \alpha)$, where $N = T \cup G$ is a finite non-empty set of control flow nodes comprising disjoint 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.

A *decision requirement diagram* is a tuple $DRD = (D_{dm}, ID, IR)$ 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 $IR \subseteq D_{dm} \cup ID \times D_{dm}$ representing the information requirements, thereby $(D_{dm} \cup ID, IR)$ is a directed acyclic graph.

A *decision table DT* is a tuple (I, O, R), where $I = \{I_1, ..., I_v\}$ is a finite non-empty set of input variables *(inputs)*, $O = \{O_1, ..., O_s\}$ is an output variable *(output)*, and $R = \{R_1, ..., R_n\}$ is a finite non-empty set of mappings (*decision rules*) which relate a subset of inputs to an output:

$$\forall\, i \in [1;\ n]\ R_i : \bigwedge_{j=1}^{w} (I_j\ op_j\ q_j) \longrightarrow O_l, 1 \leq w \leq v, 1 \leq l \leq s \quad (1)$$

where $op_1, ..., op_w$ are comparison predicates, $q_1 \in dom(I_1)$, ... , $q_w \in dom(I_w)$ are constants, $v, s, n, j \in \mathbb{N}_{>0}$

A decision rule example for the decision task *Evaluate claim* in Fig. 1a would be:

$$Amount > 1000,\ Age < 30,\ Status = single \longrightarrow Approved = No \quad (2)$$

We use a term *decision activity* for an activity $t \in T$ that has a rule task type. A collective term *decision model D* is used for an aggregate of a decision requirement diagram *DRD* and a corresponding decision table *DT*. Further in this section, we propose

a scheme of optimal execution of decision process assuming that given is a process model *PM* containing exactly one decision activity *t* referencing a DRD. It is also assumed that this DRD consists of one decision associated with a decision table *DT*. Fig. 1a and 1b are examples of such interconnected process and decision models. In further sections, we provide guidelines on dealing with a more complicated case where the process can contain several decisions.

Problem statement. Given is a process model *PM* containing a decision task *t* that references a decision model *D*, which consists of a decision requirements diagram *DRD* referencing a decision table *DT*. Assuming that during the instantiation of the process *PM* the execution of the decision model *D* is fired, we would like to find out: (1) how to reduce the number of elements in the set of inputs *A* for the decision table *DT*, and (2) what is the optimal ordering of acquisition of a set of inputs *A* from the decision table *DT*?

4.2 Our Approach

For addressing the problem described above, we propose an approach depicted by the scheme in Fig. 4 describing steps of decision inputs acquisition during instantiation of a process and a corresponding decision table. This approach corresponds to the *Decision execution* phase of decision management lifecycle from Fig. 3.

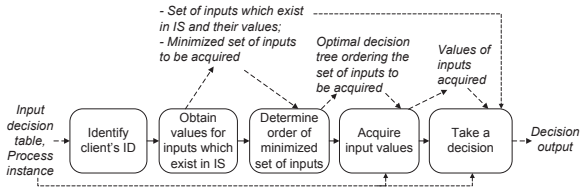


Figure 4: Outline of our proposal for acquiring decision inputs

Let *A* be a set of integer number of inputs to be acquired (e.g., questions asked to an applicant by a bank worker) in "as-is" decision process. Firstly, a decision table incorporating decision rules and considering these inputs should be designed at *Decision modeling* stage of the decision management lifecycle. As discussed, application of by-design optimization techniques [14] might decrease the set of inputs *A* to a reduced set of inputs which we will denote as $\tilde{A}$. The latter set will be an actual set of inputs depicted by input data object *Decision table* for our approach depicted in Fig. 4.

Inputs minimization. For minimizing the number of decision inputs to be acquired during decision execution, we propose to exclude process instance data which exist in the enterprise information system (factor F1). For achieving that, the first step of our approach for decision executing from Fig. 4 is identification of client's ID (or any other key data) associated with the process or decision instance. Upon identifying the client's ID, the values for inputs which pre-exist in the information system (IS) of an enterprise should be obtained. As a result of this step, a possibly reduced set of inputs $\tilde{A_1} \in \tilde{A}$ is obtained from the set of inputs $\tilde{A}$ of the input decision table. Subsequently, the set of values $V(\tilde{A_1})$ from the corresponding domains is obtained, as this information is taken from the information system of an enterprise. Immediately, a set $\tilde{A_2}$ of inputs that do not pre-exist in the system, and which is needed to be acquired further is obtained: $\tilde{A_2} = \tilde{A} \setminus \tilde{A_1}$.

Prioritization of inputs acquisition. The acquisition of inputs during decision execution (i.e., $V(\tilde{A_2})$) can be prioritized based on optimality criteria associated with certain costs and probabilities of decision rules (see factors F2 and F3). As one of the most comfortable way to represent the ordering of inputs acquisition for the goal of making a decision is decision trees [16], we propose to construct a decision tree which orders the set of inputs to be acquired in an optimal way. The decision nodes in the output decision tree should be inputs from the set $\tilde{A_2}$, and the end nodes should be the decision outcomes which can be obtained from the input decision table. In the next section, we develop an algorithm determining the optimal ordering of decision inputs based on techniques that obtain sequential testing procedures for computer implementations [12, 8]. We show thereby that without examining all of the possible decision trees equivalent to a given decision table, our algorithm still leads to the guaranteed best answer by using such optimality criteria as minimizing number of inputs to be inquired, associated costs, and frequency of decision rules executed in the past. The algorithm yields a decision tree representing the optimal ordering of inputs to be acquired during execution of the activity *Acquire input values* from Fig. 4. As a result of it, the set of values $V(\tilde{A_2})$ from the corresponding domains of set of inputs $\tilde{A_2}$ is obtained. This follows up with the actual decision taking activity.

5. ACQUISITION OF DECISION INPUTS

Our proposal for acquiring decision inputs during execution of processes is based on the idea of prioritization of inputs acquisition by constructing a decision tree from the given decision table. In Section 5.1 we discuss the premises for our proposed algorithm and direct a reader through the rest of the section.

5.1 Discussion of the Algorithm Base

At first glance, it might seem useful to turn a decision table into a learning problem, where the rules would represent sample data. The goal would be to classify inputs into a decision tree according to the degree of their influence on the table outputs. An idea would be to apply then the machine learning algorithms such as C4.5 for decision tree learning [16]. In these algorithms, an output decision tree is obtained by splitting the sample data set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner. As this procedure is appropriate for constructing the output decision tree in our case, we reuse it with some adaptations of parameters, as described further in Algorithm 1 in Section 5.2. However, there is a problem of applying the decision tree learning techniques at a full extent. In particular, their choice of the next node to be inserted into the output decision tree at each step of the recursive procedure is not applicable for our case. It is generally not recommended to use machine learning techniques on small sets of input data [16]. With that, usual decision tables contain at most several tens of rules. Also, the decision tree learning techniques incorporate not only classification of existing data values, but also predictions of possible new data values. In contrast, in our case the decision tables are fixed, and no *new* decision rules or inputs can emerge during process execution. Because of this, the formulas used for calculating information gain when a new node is inserted into the output decision tree, are not applicable. That being said, there exist earlier works which are based on Reinwald and Soland's algorithm on conversion of limited-entry decision tables to optimal computer programs [12, 8]. We adapt this approach for finding the next best node during the output decision tree construction from decision tables in Algorithm 2 (see Section 5.3). We adapt the existing algorithm by using factor F2 (costs associated with decision inputs acquisition) and factor F3

(frequency of rules executed in the past) as optimization criteria.

The presented algorithms take into account that decision tables could be of two types: with limited, and extended entries [8]. *Decision tables with extended entries* consist of rules represented by expressions of the general type I_i *op* q_i. Algorithm 1 determines optimal acquisition of decision inputs for the case of *decision tables with limited entries*, where the expressions I_i op_i q_i corresponding to tables can be evaluated either to true ('Y') or false ('N'). An entry depicted by dash ($-$) means that the input value does not influence the decision rule, by that two rules are combined into one. The guidelines for the case of decision tables with extended entries are provided in Section 5.4. Afterwards, Section 5.5 provides guidelines on further adaptations of the presented algorithms for specifics of DMN decision requirements diagrams and process-level decision logic. We assume that the input decision tables are complete, correct, free of ambiguities, and the hit policies of firing the decision rules are single first [11]. Also, the input decision tables are assumed to have vertical orientation, where inputs are represented by rows and rules are represented by columns.

5.2 A Case of Limited-Entry Decision Tables

Algorithm 1 presents a procedure BUILD_DECTREE taking as inputs a decision table *DT*, a vector of probability distribution *P* for its rules, and a vector of expected costs *C* associated with inputs acquisition, as well as auxiliary variables *parentInd* and *nextNode* needed as passing arguments for recursive construction of an output decision tree. The algorithm output is a binary tree storing the ordering of inputs which can be obtained by tracing a simple path from the root down to a leaf. More particularly, the output decision tree is a tuple $T = (nodes, leaves, parents)$ consisting of three arrays which are assigned at the k-step of the algorithm with values:

$$nodes[k] = \begin{cases} x, \text{ an index of input } I_x \text{ detected as the next node to be inserted;} \\ 0, \text{ if all the inputs of a rule were inserted in the tree.} \end{cases}$$

$$leaves[k] = \begin{cases} O_y, \text{ a rule output, if all the inputs of a rule were inserted in the tree;} \\ 0, \text{ otherwise (tree path has not reached the end).} \end{cases}$$

$parents[k] = z$, an index of the parent input of the chosen input I_x.

Prior to procedure execution, the first elements of arrays *nodes*, *leaves*, *parents*, as well as *parentInd* and *nextNode* are set to zeros. The elements assignment is done by function Update_DecTree(*int nodeCount*, *int nodeInd*, *int leafInd*, *int parentInd*) which assigns either an index of a node *nodeInd*, or an output *leafInd*, as well as an index of its parent *parentInd* to the current *nodeCount*-element of the tree arrays. The algorithm starts with identification of the number of inputs and rules in the decision table *DT* passed as a procedure argument. A boolean variable *onlyDashes* reflects if all the table entries are dashes. Further, as the procedure builds the tree recursively, stop conditions need to be identified. The first stop condition (Line 5) checks if all the table inputs values are dashes, or if the table consists only of one input and one rule. In both cases, all rule outputs are assigned as output tree leaves. The second stop condition (Line 8) is met if the table has exactly one input, and several rules that consist not only of dashes. If that is the case, this input is recorded as a tree node through the value of the function Initial_Ind(nextNode) returning the original index of the node which is passed to the procedure as the next node to be inserted; all the possible outputs are assigned as leaves.

If the stop conditions are not met, the procedure proceeds in Line 14 with identification of the next best node to be assigned to the tree by procedure NEXT_NODE (see Section 5.3). Then, the initial index number of this node returned by function *Initial_Ind* is recorded

Algorithm 1 Creation of an output decision tree for a limited-entry decision table

```
1:  procedure BUILD_DECTREE(decisionTable DT, outcomeProbabilities P,
    inputCosts C, int parentInd, int nextNode)
2:      inputNum ← number of inputs in DT
3:      rulesNum ← number of rules in DT
4:      onlyDashes(DT) ← procedure checking if all input values of DT are dashes
5:      if onlyDashes=true or
        (onlyDashes=false and inputNum=1 and rulesNum=1) then
6:          for l from 1 to rulesNum do
7:              Update_DecTree(nodeCount, 0, Output(l), parentInd)
8:      else if onlyDashes = false and inputNum = 1 and rulesNum > 1 then
9:          Update_DecTree(nodeCount, Initial_Ind(nextNode), 0, parentInd)
10:         temp ← nodeCount − 1
11:         for l from 1 to rulesNum do
12:             Update_DecTree(nodeCount, 0, Output(l), temp)
13:     else
14:         nextNode ← NEXT_NODE(DT, P, C)                ▷ See Algorithm 2
15:         Update_DecTree(nodeCount, Initial_Ind(nextNode), 0, parentInd)
16:         parentInd ← nodeCount − 1
17:         DT^r = Reorder_Table(DT, nextNode)
18:         (tableY, outputProbY, costsY) = Get_TableY(DT^r)
19:         BUILD_DECTREE(tableY, outputProbY, costsY, parentInd, nextNode)
20:         (tableN, outputProbN, costsN) = Get_TableN(DT^r)
21:         BUILD_DECTREE(tableN, outputProbN, costsN, parentInd, nextNode)
```

as the tree node, *leaves* is assigned with zero as no rule output is reached at this moment, and *parents* is assigned by the index of a node – parent of *nextNode* in the output tree. Next, the corresponding index of a parent node is assigned in Line 16 as this node will become a parent at the next procedure call to the parent. Function Reorder_Table should include three steps: (1) The row which has the index number corresponding to the *nextNode* index number becomes the first row ; (2) For each rule which has a *dash* as the first input value, two new rules are generated which contain the same input values as the initial rule, but instead of the first elements (dashes) these two new rules contain 'Y' and 'N' respectively; (3) The resulting table is reordered so that all the columns containing 'Y' ('N') values as their first input elements are grouped on the left (right) side of the table. Thereby, we enable an easy extraction of subtables from the input table corresponding to the 'Y' and 'N' paths which will branch from the current node recorded in the tree.

Such reordering of the table allows further extraction of a new table *tableY* from the initial decision table *DT* which represents a subtable obtained by storing only those columns of the original which have the values 'Y' as their first elements, thereby extracting these values so that the size of the new matrix is reduced by 1. Then, the procedure is recursively called passing the new decision table *tableY* with corresponding probabilities *outputProbY* and costs *costsY* associated with the new inputs. Thereby, the algorithm enters the 'Y' branch of the decision tree calls itself recursively until a stop condition is met. Then, the recursion stops and the algorithm returns to the state from which it started constructing the 'Y' branch and then it proceeds by applying analogous steps for the 'N' branch. The procedure corresponds to standard binary tree creation and guarantees inclusion of all the inputs in the output decision tree.

5.3 Finding the Next Best Node

Below we present a procedure NEXT_NODE which is called during execution of procedure BUILD_DECTREE each time a new node to be included into the output decision tree has to be identified. The inputs to the procedure are the decision table *DT* obtained at some step of Algorithm 1, a vector of probability distribution *P* for its rules, and a vector of expected costs *C* associated with inputs acquisition. In this procedure, we firstly identify the number of in-

Algorithm 2 Finding the next best node during construction of an output decision tree

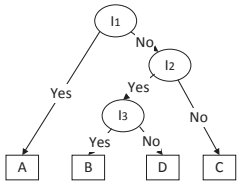
```
1:  procedure NEXTBESTNODE(decisionTable DT, outcomeProbabilities P,
    inputCosts C)
2:      inputNum ← number of inputs in DT
3:      rulesNum ← number of rules in DT
4:      worstCosts ← sum(C)
5:      savePotential ← initialize as a zero-vector with the size of (1:rulesNum)
6:      for i from 1 to inputNum do
7:          for j from 1 to rulesNum do
8:              if DT[i,j] = '-' then
9:                  savePotential[i] ← savePotential[i] + P[j]* C[i]
10:     for i from 1 to inputNum do
11:         bestPotentialThisFirst[i]=worstCosts-sum(savePotential)+savePotential(i)
12:     return nextBestNode ← min(bestPotentialThisFirst)
```

puts and rules in the input table. Further in Line 4, we calculate what would be the most expensive decision tree associated with *worstCosts* which can potentially be constructed, that is equal to a sum of all the costs $sum(C)$. In order to find whether this tree can be improved, Lines 6–9 we calculate the value of a saving potential variable for each input $i \in [1{:}inputNum]$ assuming that it is acquired at the end of all the tree branches. That will be the case if some rule $j \in [1{:}rulesNum]$ does not require the input to be acquired (as indicated by the dash in Line 8). The probability $P[j]$ of such a rule is to be taken into account as to consider the percentage cases where this rule will be obeyed (Line 9). Next, in Lines 10–11 the procedure calculates the potential costs benefit that each input can bring, if it is included into the tree as the next node. The expected costs of a decision tree starting with any input should not be better than the difference between the potentially highest costs *worstCosts* and highest savings $sum(savePotential)$, which should be summed up with the saving potential value *savePotential* for each input. In Line 12, the procedure returns the index number of the input corresponding to the minimum of vector *bestPotentialThisFirst* as acquiring this input in the next step of the procedure of the decision tree generation will minimize the expected costs. In [12] it is proven formally on example of conversion of any given limited-entry decision table into a computer program that such method of finding the optimal next node in a decision tree yields the lowest execution costs.

INPUTS \ RULES	R1	R2	R3	R4
I1	Y	N	N	N
I2	-	Y	N	Y
I3	-	Y	-	N
OUTPUT	A	B	C	D

COSTS
10 EUR
10 EUR
10 EUR

OUTPUT PROBABILITIES	0.500	0.125	0.250	0.125



(a) Input data (b) Output decision tree

Figure 5: Demonstration of the work of Algorithms 1 and 2

An example of the inputs for both algorithms presented is depicted in Fig. 5a: a decision table shown at the top left corner of the figure, a vector of probability distribution for its rules, and a vector of expected costs associated with inputs acquisition. The output decision tree generated for this decision table by Algorithms 1 and 2 is shown in Fig. 5b. The tree stores the ordering of inputs to be acquired during decision execution, which represents a simple path from the root down to a leaf during inputs acquisition.

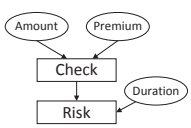
5.4 A Case of Extended-Entry Decision Tables

The algorithms presented considered the decision tables with limited entries, serving as a ground base for further investigations as similar tactics apply to the extended case. For the case of decision tables with extended entries, the values of rule expressions I_i *op* q_i are of a general type. With that, the left (right) branch of the node in the output decision tree shows that this expression is evaluated to *true* (*false*), as in the limited case. To convert the case of decision tables with extended entries to the limited case, the Algorithm 1 should be extended with the considerations presented below.

- According to [8], the frequency of occurrence of the repeated expressions I_i *op* q_i in rows of the tables should be added to the optimality criteria for finding the best next node during the output decision tree construction;
- The inputs ordering also can be obtained by tracing a simple path from the root down to a leaf. With that, if a path contains consequential nodes I_j op_j q_j where I_j corresponds to the same input, in the output ordering the duplicated values should be replaced by an unique index of the input I_j.

5.5 A Case of Interconnected Decision Tables

Until now we considered only the case when a process references a decision model containing exactly one decision table. However, in real processes some decisions depend on each other, which can be represented by interconnected decision tables. Consider an extension of the loan assessment example which contains two decision tasks *Decide check* and *Evaluate risk*. The DMN decision requirements diagram and corresponding decision tables for this case (see Fig. 6) show that two decisions *Check* and *Risk* are interconnected as the output of decision *Check* is used as an input for decision *Risk*.



Amount	Premium	Check
> 50	false	Full check
≤ 50	false	Standard check
-	true	No check

Duration	Check	Risk
≤ 10	No check	0
> 10; ≤ 20	Standard check	1
> 10; ≤ 20	Full check	2
> 20	Standard check	3
> 15	Full check	4

Figure 6: Extended example of credit risk assessment

The presented algorithms of finding optimal ordering of inputs acquisition for decision tables can as well be adapted for the case of interconnection tables where an output of some decision is an input to another decision. For this case, an extended decision table representing an aggregation of several atomic decision tables should be done by substituting consequently the inputs in one decision table which represent outputs of another decision table by corresponding decision rules. For example, the *Check* table from Fig. 6 can be merged with the *Risk* table with the resulting inputs *Duration, Amount, Premium*, and the output *Risk*. With that, the values of the *Check* input in the decision table at the right side of the picture should be replaced with the corresponding decision rules over inputs *Amount* and *Premium* obtained from the *Check* decision table. The inputs of such aggregated extended table represent a sequence for which the optimal acquisition order has to be identified. This can be done manually, e.g., by an enterprise analyst, as this step needs to be determined in accordance to business requirements.

6. EVALUATION

For evaluating our scheme of process-level decision execution, we implemented the introduced algorithms and conducted a series of experiments in the Matlab R2014a[1] environment, which we present below using the same denotation as in Section 4.2. Our implementation is available at http://blindedForAnonymity. As input data we

[1]http://www.mathworks.com/matlab

NR	NAME	TYPE
1	Identification number	continuous
2	Loan amount	continuous
3	Amount on purchase invoice	continuous
4	Loading percentage over credit	continuous
5	**Loan term**	**continuous**
6	**Loan purpose**	**nominal**
7	Use code (private or professional loan)	nominal
8	Monthly payment	continuous
9	**Savings account amount**	**continuous**
10	Other loan expenses	continuous
11	**Client's income**	**continuous**
12	Number of years employed	continuous
13	Number of years in residence country	continuous
14	Applicant age	continuous
15	Applicant type	nominal
16	Marital status	nominal
17	Number of years since last house move	continuous
18	Code of loyalty saver	nominal
19	**Does the client own property?**	**nominal**
20	**Number of years client**	**continuous**
21	Number of years since last loan	continuous
22	Number of checking accounts	continuous
23	Number of term accounts	continuous
24	Number of dependents	continuous
25	Number of mortgage loans	nominal
26	**Employment economical sector**	**nominal**
27	Employment status	nominal
28	Title/salutation	nominal

Figure 7: Attributes for the credit-risk data set

considered real-life credit-risk data set Bene 1 which was also used in [2]. The set consists of $A{=}28$ questions asked to person applying for a loan (see Fig. 7). In [2], a decision table construction with the help of network rule extractions is presented, thereby many questions are eliminated as statistically they are not important for evaluation procedure. The first step in our scheme of decision execution includes by-design optimization of decision table which is out of the scope of the present paper, therefore we rely upon the output results of the research presented in [2]. This optimization yields $\tilde{A}{=}7$ questions highlighted in Fig. 7 which need to be asked to a customer in order to assess the client's eligibility for credit. Our scheme from Fig. 4 prescribes execution of the activity *Obtain information about client from IS* followed by detecting whether some data about the client already pre-exists in information system of the bank. For identifying the questions which might pre-exist in the bank IS, we conducted an interview with 10 experts who identified themselves as bank managers or analysts specializing in credit-risk and underwriting processes. The experts were asked which questions from the set $\tilde{A}$ (highlighted in Fig. 7), according to their opinion and experience, normally exist in the information system of the bank prior to the application. The results showed that such data as *Savings account amount* and *Number of years client* normally exist in the bank IS prior to credit application. Therefore, the set $\tilde{A_1}$ consisting of these two questions can be excluded from the set of inputs $\tilde{A}$ needed to be acquired from a client, as corresponding values $V(\tilde{A_1})$ can be immediately obtained from the bank IS. With that, the set of rest of questions to be asked $\tilde{A_2}{=}\tilde{A} \setminus \tilde{A_1}$ was identified: *Loan term, Loan purpose, Client's income, Does the client own property?,* and *Employment economical sector.*

The next step of our scheme from Fig. 4 is to *Determine ordering of minimized set of questions* which takes as input the decision table consisting of minimized set of questions. For defining such decision table, we took as basis the decision rules extracted from Bene 1 data set in [2]. The modelled table is presented in Fig. 8; it is a decision table corresponding the case of a single decision table with limited entries (only two options of inputs values are possible,

INPUTS \ RULES	R1	R2	R3	R4	R5	COSTS
I1: Loan purpose	-	-	Not a house	Not a house	House	5 EUR
I2: Employment economical sector	Not sector C	Sector C	Sector C	Not sector C	-	5 EUR
I3: Loan term	>= 1 year	>= 1 year	>= 1 year	>= 1 year	< 1 year	5 EUR
I4: Client's income	>= 720 EUR	>= 720 EUR	-	-	< 720 EUR	10 EUR
I5: Does the client own property?	No	Yes	No	Yes	-	10 EUR
OUTPUT Credit Rating	A	B	C	D	E	
OUTPUT PROBABILITIES	0.25	0.22	0.28	0.19	0.32	

Figure 8: Decision table for credit rating determination

in case the input value is not significant, the dash is used). The decision output is *Credit Rating* with possible values depicted in the figure. For assuming the associated costs, our hypothesis was that inputs acquisition is connected with verification costs. The expert survey acknowledged it and showed that verification of such data as *Loan purpose, Employment economical sector,* and *Loan term* would be verified by a bank resource in average in 10 minutes. Assuming a bank worker's salary of 30 EUR per hour, the average costs associated with acquiring of these inputs are 5 EUR. For the activities *Client's income* and *Does the client own property?*, the experts indicated an average verification time of 20 minutes, which would result in average costs of 10 EUR. The probability distribution of decision rules was generated randomly for the presented case, but in real cases such information should be obtained from considering historical frequencies of rules execution.

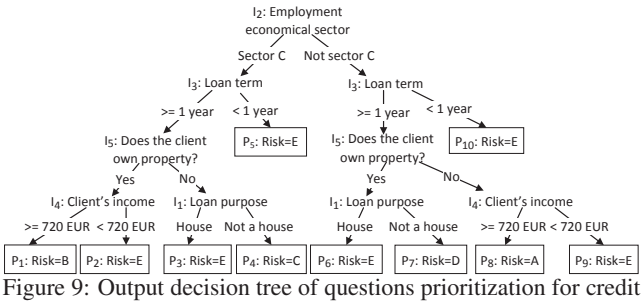


Figure 9: Output decision tree of questions prioritization for credit rating assessment: each path represents a sequence of questions

According to the next step our algorithm, the generation of optimal decision tree of questions can be done with the help of Algorithm 1. The resulting decision tree of questions generated by the algorithm is shown in Fig. 9. The picture presents a decision tree which stores the ordering of inputs to be acquired during execution of a decision process when a client applies for the loan. The ordering of questions can be obtained by tracing a simple path from the root down to a leaf, where the leaf value represents a final decision on credit rating to be done in accordance with decision rules.

Total costs evaluation. Generation of the output tree of ordering of questions allows us to estimate the total cost of decision execution. The cost of each path *Pi* from the root down to a leaf at Fig. 9 is equal to sum of costs associated with every input I_j on the path. For example, for the path P_1 the total costs are equal to sum of costs for inputs I_2, I_3, I_5, and I_4, which is equal to $5{+}5{+}10{+}10{=}30$ [*EUR*]. If no input prioritization were done, the costs would be equal to the sum of costs associated to all inputs (sum of all values in the column *Costs* in Fig. 8), which would be then equal to 35 [*EUR*].

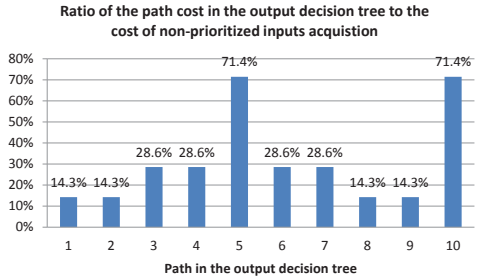


Figure 10: Reduction of costs due to optimized data acquisition

In such a way, the ordering of inputs acquisition results in costs reduction of $1-30/35 \approx 0.143$, or 14.3%. The costs reduction calculated in the same way for all the paths in the output decision tree compared to the costs of non-prioritized inputs acquisition are depicted in Fig. 10. It can be seen that acquisition of inputs in the order of nodes on any path of the decision tree is more beneficial than non-prioritized inputs acquisition. The average costs of inputs acquisition are equal to the weighted sum of costs multiplied with the rule probabilities, which is $7[EUR]$ for non-prioritized, and ≈ 6.59 $[EUR]$ for prioritized rules execution, so the average costs reduction takes the value of about 3.64%. Hereby, the optimal decision tree is very dependent on the two factors: condition cost and decision rule frequency. Although the value of the average costs reduction already shows that a performance improvement during decision execution is achieved, it is clear that in case if the probability of the path associated with the lowest costs in the output decision tree (as for paths P_5 and P_{10} from Fig. 8) significantly higher than the probabilities of the paths associated with lowest costs in the output decision tree (as for paths P_1 and P_8 from Fig. 8), the average costs reduction will be significantly higher.

7. RELATED WORK

In comparison to the extensive literature on translation of decision tables into programs to which we refer when constructing the optimal decision tree of decision inputs acquisition [12, 8], the literature on optimal execution of process-level decisions has not developed so well yet as it only introduces the concept of decision modeling and execution complementary to processes without providing any guidelines on acquisition of decision inputs [15, 3, 5]. In [13], prioritization of decisions taken in the process is done by their ranking through a scorecard technique, which can be useful for prioritizing decision nodes in decision requirements diagrams, but not for decision logic level as the technique does not apply to decision rules. For finding more of modeling and execution guidelines for the case of interconnected decision tables, one can refer to [14]. The techniques presented in these papers can be combined with our approach for decision execution. Further factors influencing prioritization of inputs acquisition during decision execution, such as resources utilization or user satisfaction of software systems, can be found in [9]. The case-related factors can also be taken into account, e.g., approval accuracy for credit or interest revenue per account for credit assessment can be found in [10].

8. CONCLUSION

In this paper we propose a scheme of optimal inputs acquisition for operational decisions represented as decision tables during their execution which allows to (1) minimize the number of decision inputs; (2) order the decision inputs to be acquired in an optimal way. For minimizing the number of decision table inputs we consider a priori knowledge about process instance existing before process execution. Our approach to executing process-level decisions guarantees the optimal execution of decision tables by using such optimality criteria as minimizing number of inputs to be inquired, associated costs, and frequency of decision rules executed in the past. We demonstrated our approach on the example of credit-risk assessment by showing that the number of questions to be asked to an applicant in order to make a decision can be reduced significantly. Moreover, the costs associated with acquisition of answers for questions can be reduced by prioritizing questions asked, according to the presented algorithms. The approach can be also applied in a broader domain of business processes in which a decision process is executed over a big amount of data inputs. For such cases, further case-dependent factors influencing prioritization and ordering of inputs acquisition during decision execution might be considered, which could be a multi-criterion extension of our work.

9. REFERENCES

[1] The Evolving Role of Data in Decision-Making. In *A report from the Economist Intelligence Unit*. The Economist, 2013.

[2] 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), 2003.

[3] P. Baltzan and A. Phillips. *Business Driven Information Systems*. McGraw-Hill Higher Education, 2013.

[4] E. Bazhenova, S. Buelow, and M. Weske. Discovering decision models from event logs. Lecture Notes in Business Information Processing, Vol. 255:237–251. Springer, 2016.

[5] T. Debevoise and J. Taylor. *The MicroGuide to Process Modeling and Decision in BPMN/DMN*. CreateSpace Independent Publishing Platform, 2014.

[6] M. Dumas, M. Rosa, J. Mendling, and H. Reijers. *Fundamentals of Business Process Management.* Springer, 2013.

[7] T. Gestel and B. Baesens. *Credit Risk Management*. Oxford University Press, 2009.

[8] E. Humby. *Programs from decision tables*. MacDonald, London, 1973.

[9] B. Killingsworth and M. McLeod. Factors affecting the acceptance and feasibility of expert systems in business. In *Proceedings of the Information Resource Management Association Conference*. Idea Group Publishing, 1992.

[10] K. J. Leonard. The development of credit scoring quality measures for consumer credit applications. *Int. Journal of Quality & Reliability Management*, 12(4):79–85, 1995.

[11] OMG. Decision Model And Notation (DMN), v. 1.1, 2016.

[12] L. T. Reinwald and R. M. Soland. Conversion of limited-entry decision tables to optimal computer programs. *J. ACM*, 13(3):339–358, July 1966.

[13] J. Taylor. *Decision Management Systems*. IBM Press, 1st edition, 2011.

[14] J. Vanthienen, C. Mues, and A. Aerts. An illustration of verification and validation in the modelling phase of KBS development. *Data Knowl. Eng.*, 27(3):337–352, Oct. 1998.

[15] B. Von Halle and L. Goldberg. *The Decision Model*. Taylor and Francis Group, 2010.

[16] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., 3rd edition, 2011.