Decision Mining Revisited - Discovering Overlapping Rules

Felix Mannhardt[1,2], Massimiliano de Leoni[1], Hajo A. Reijers[3,1], Wil M.P. van der Aalst[1]

[1] Eindhoven University of Technology, Eindhoven, The Netherlands
[2] Lexmark Enterprise Software, Naarden, The Netherlands
[3] Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
f.mannhardt, m.d.leoni, h.a.reijers, w.m.p.v.d.aalst@tue.nl

Abstract. Decision mining enriches process models with rules underlying decisions in processes using historical process execution data. Choices between multiple activities are specified through rules defined over process data. Existing decision mining methods focus on discovering mutually-exclusive rules, which only allow one out of multiple activities to be performed. These methods assume that decision making is fully deterministic, and all factors influencing decisions are recorded. In case the underlying decision rules are overlapping due to non-determinism or incomplete information, the rules returned by existing methods do not fit the recorded data well. This paper proposes a new technique to discover overlapping decision rules, which fit the recorded data better at the expense of precision, using decision tree learning techniques. An evaluation of the method on two real-life data sets confirms this trade off. Moreover, it shows that the method returns rules with better fitness and precision in under certain conditions.

Keywords: Decision Mining, Process Mining, Overlapping Rules

1 Introduction

Organizations use process models representing their business processes for multiple reasons. Process models are used, for example, to document, specify, and analyze processes [1]. Generally, process models depict activities (i.e., units of work) and their dependencies in a graph representation, which specifies the order of activities in the process execution. During the execution of non-trivial processes, next to the ordering of activities, *decisions* between multiple alternative activities needs to be made. Those choices are explicitly modeled in process models as so-called *decision points*. A decision point specifies the alternatives available. An important challenge when using process models is to understand the decision that need to be made in a process, and the conditions under which certain alternative activities are performed. Awareness that modeling and analyzing decisions is key in process management is increasing. See for example the interest in the Decision Model and Notation (DMN) standard [2] supported by vendors such as Signavio and Camunda.

Process mining methods are able to discover process models with decision points by using event logs. Event logs contain information on performed activities (i.e., sequences

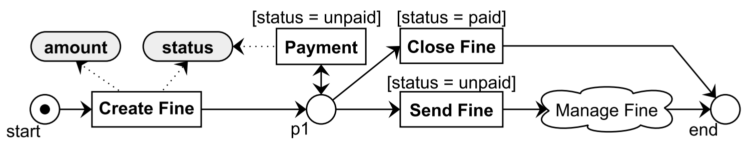


Fig. 1. Fragment of a process model taken from a road traffic fine management process with overlapping rules governing an exclusive choice based on the payment of the fine.

of events) and are available in today's information systems [1]. *Decision mining* aims to discover the rules that are underlying those decisions. Those rules are determined using data recorded by information systems that support the process [3]. Events in event logs used for decision mining need to contain process data, which was available when the activity was performed (i.e., attributes). Take, for instance, the process model fragment depicted in Fig. 1, which shows a simplified fragment of a road-traffic fine management process [4]. After creating the fine notice (`Create Fine`) and recording the amount (`amount`), an exclusive choice between three alternatives has to be made. Either a payment is received (`Payment`), possibly in multiple installments, the fine is closed (`Close Fine`), or the police sends a fine notice (`Send Fine`) and the process continues with further management of the fine. Please note, that only one out of these three alternatives can be taken, which is different to an inclusive choice that allows the execution of multiple alternatives. The rules depicted in Fig. 1 drive this choice. Activity `Close Fine` can only be executed if the value of attribute `status` is *paid*. Thus, the process can only finish directly if the fine has been paid in a timely manner. Rules defined over the process data are an integral part of this process. Traditionally, decision mining methods [3,5,6] use decision tree learning techniques such as C4.5 [7] to determine rules governing the process execution based on event logs. For example, attribute values recorded for `status` and `amount` are used as *feature*, while the choice between `Payment`, `Send Fine`, `Close Fine` is used as *target class*. Then, mutually exclusive rules for each activity are built using the obtained decision tree, thus, the choice at the decision point is completely determined by the values of `status` and `amount`.

Existing decision mining techniques for exclusive choices rely on the strong assumption that the rules attached to the alternative activities of a exclusive choice need to be *mutually exclusive*. However, business rules are often non-deterministic and this "*cannot be solved until the business rule is instantiated in a particular situation*" [8]. This ambiguity can occur due to conflicting rules or missing contextual information. For example, decisions taken by process workers may depend on contextual factors, which are not encoded in the system and, thus, not available in event logs. Moreover, even if those factors are encoded in the system event logs are often incomplete [9]. Without complete information in the event log, the mutually exclusive rules underlying decision-making cannot be discovered. Hence, this assumption is typically not met in reality. For example, the process model shown in Fig. 1, which is taken from a real-life process, contains overlapping rules for an exclusive choice. This means that on a decision point more than one of multiple activities may be executed under the same condition, i.e., when the same attribute values have been observed beforehand. In case the payment `status` is *unpaid*, the choice between `Payment` and `Send Fine` is deferred, i.e., an unpaid fine can be either paid directly or a notification is sent to the offender. The actual decision between both activities is not specified. It might depend on an unavailable contextual factor, e.g., some fines can be paid on-the-spot depending on the context. Please note that an exclusive choice with overlapping rules is different to an inclusive choice, which allows multiple alternatives activities to be executed in parallel. State of the art techniques [3,5,6] only use decision trees, and, hence, cannot discover this class of rules. For instance, current techniques fail to discover the decision rule in Fig. 1.

This paper proposes a technique that discovers *overlapping rules* in those cases that the underlying observations are characterized better by such rules. The technique is able to deliberately trade the precision of mutually-exclusive rules, i.e., only one alternative is possible, against fitness, i.e., the overlapping rules that are less often violated. In short, as in [3,5,6], our technique builds an initial decision tree based on observations from the event log. Then, for each decision tree leaf, the wrongly classified instances are used to learn a new decision tree leading to new rules. These new rules are used in disjunction with the initial rules yielding overlapping rules of the form $rule_1 \vee rule_2$. We evaluate our technique on two real-life data sets: an event log taken from a road traffic fine management process and an event log with pathways of patients in a hospital. The evaluation shows that our technique discovers overlapping rules in real-life data, and that those rules provide a better balance in terms of fitness and precision. For example, our technique discovers overlapping rules similar to the ones depicted in Fig. 1, whereas traditional method fail, e.g., to discover a rule for activity `Payment`.

As to the structure of this paper, we introduce necessary formalisms for process models and event logs (Sect. 2). Then, we present our discovery technique for process models with overlapping rules (Sect. 3). We evaluate our technique based on real-life event logs (Sect. 4). We discuss related work (Sect. 5), and conclude with a summary and sketched future work (Sect. 6).

2 Background

We present necessary preliminaries such as the formalism we use to represent process models and event logs, and criteria used to determine the quality of decision rules.

2.1 Process Model

Generally, our decision mining technique is independent of the formalism used to model the process, e.g., BPMN, UML-activity diagrams, EPCs or YAWL. We choose Data Petri Nets (DPN) [6] as modeling language because it has simple and clear semantics. A **DPN** is a Petri net [10] that is extended with variables (i.e., data attributes).

Definition 1 (Petri net). *A Petri net is a triple (P, T, F) with: P is a set of places P, T is a set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a set of flow relations that describe the bipartite graph between places and transitions.*

Transitions correspond to activities in the process. We denote the input places of a transition with $^\bullet t$ and the output transitions of a place with $p^\bullet$. The state of a Petri net is defined by its marking $M : P \to \mathbb{N}$ that assigns a number of tokens to each place. Executing a transition consumes one token from each of its input places and produces one token on each of its output places. A transition $t \in T$ can only can be executed (fired) when there is at least one token in every input place.

Definition 2 (DPN). *Denoted the universe of formulas over set V of variables with $Formulas(V)$, a DPN (P, T, F, V, U, W, G) is a Petri net with additional components, which describe the data perspective of the process model:*

- (P, T, F) *is a Petri net;*
- V *a set of variables;*
- U *the universe of possible variable values;*
- $W : T \to 2^V$ *a set of write operations for each transition;*
- $G : T \to Formulas(V)$ *a set of guard expressions (guards).*

Transitions update the values of variables through *write operations*. Furthermore, guards defined over the variables of the DPN further constrain when transitions may be executed. A transition in a DPN can be executed only if all its input places contain at least one token and the guard is satisfied by the current variable assignment. The state of a DPN is defined by both the marking and the current values of all its variables. The behavior of a DPN corresponds to all sequences of transition firings starting from an initial state to any final state. The initial state is made of the initial marking $M_I \in M$ and an empty set of variable values. Final states are all final markings $M_F \in M$. For sake of space we refer to [4] for a comprehensive introduction to DPNs.

Example 1. Fig. 1 shows a simplified process in the DPN notation. The process starts with executing transition `Create Fine`, which writes attribute `status`. When executing transition `Create Fine` a token is removed from the place *source* and a new token is put in place p_1. Now, a choice between the output transitions of p_1, `Payment`, `Send Fine`, and `Close Fine`, has to be made. Therefore, place p_1 is called *decision point*. As there are guards placed on all three transitions their enablement depends on the current assignment of attribute `status`. For example, both transitions `Send Fine` and `Payment` can only be executed when `status` is *unpaid*. As the guards of `Send Fine` and `Payment` overlap, the choice between both transitions is non-deterministic. Assuming transition `Create Fine` assigned the value *paid* to `status`, i.e., the fine has been paid directly, then, only transition `Close Fine` can be executed and the process ends by reaching the final marking of the DPN.

2.2 Event Log

An event log stores information about the executed activities in a process [1].

Definition 3 (Event Log). *Given a set of transitions T, variables V, and values U, we define an* ***event log*** $\mathcal{E}$ *as a collection of unique events [1,11]. Each event $e \in \mathcal{E}$ is associated with a set $values(e) \in (V \to U)$ containing the latest values of all*

Table 1. Event Log $\mathcal{E}$ with data attributes **status** and **amount**

Id	Case	Activity	Status	Amount
e_1	1	`Create Fine`	`unpaid`	30
e_2	1	`Payment`	`unpaid`	-
e_3	1	`Send Fine`	-	-
e_4	2	`Create Fine`	`unpaid`	30
e_5	2	`Send Fine`	-	-
e_6	3	`Create Fine`	`unpaid`	30
e_7	3	`Payment`	`paid`	-
e_8	3	`Close Fine`	-	-
...	...	...	...	

attributes recorded before the event occured starting with an initial value, and refers to the execution of transition $trans(e) \in T$. Moreover, $writes(e) \subseteq V$ is the set of variables that are written by event e.

Table 1 shows an event log for the process model introduced in Fig. 1. Each row represents a unique recorded activity execution (i.e., event) together with the produced data (i.e., attributes). Special attributes like an **id**, the case identifier **case**, and the activity name **activity** are recorded for each event. The location of an event in the case, i.e., the order in which the events occurred, is uniquely identified through the **id** attribute.

Example 2. For the example event log in Table 1 we can determine the transition corresponding to event e_2 as $trans(e_2) = \texttt{Payment}$. Moreover, we can obtain the value of all attributes at the moment when e_2 occurred as $values(e_2) = ((\texttt{status} = unpaid), (\texttt{amount} = 30))$. Finally, the variables written by e_2 are $writes(e_2) = \{\texttt{status}\}$.

2.3 Quality Criteria - Fitness and Precision

We use two criteria to determine the quality of the guards defined on the output transitions of a place $p \in P$ of a DPN given process executions recorded in an event log $\mathcal{E}$: place fitness and place precision. We denote the set of events for transitions in $p^\bullet$ with $\mathcal{E}_p = \{e \in \mathcal{E} \mid trans(e) = t, t \in p^\bullet\}$. We define **place fitness** based on the number of events in $\mathcal{E}_p$ for which the guard is violated:

$$fitness_{\mathcal{E},p} = 1 - \frac{|\{e \in \mathcal{E}_p \mid \text{Guard of } trans(e) \text{ is violated}\}|}{|\mathcal{E}_p|}$$

The place fitness linearly decreases with an increase of the fraction of transitions $t \in p^\bullet$ that fire violating the respective guard. The **place precision** of place $p \in P$ is defined as is based on the possible behavior at p defined by the DPN, and actual behavior observed in $\mathcal{E}_p$. We use functions $pos_p : \mathcal{E} \to \mathbb{N}$ and $obs_p : \mathcal{E} \to \mathbb{N}$ returning the possible and observed executions of transitions in $p^\bullet$ before an event $e \in \mathcal{E}$ occurred. Work [12] describes how to obtain values for pos_p and obs_p given an event log and a DPN. Using pos_p and obs_p, we define the place precision as:

$$precision_{\mathcal{E},p} = \frac{\sum_{e \in \mathcal{E}_p} |obs_p(e)|}{\sum_{e \in \mathcal{E}_p} |pos_p(e)|}$$

Given a data set that allows for precise disjunctive guards, the place precision gets lower when guards on the output transitions are more overlapping.

3 Discovery of DPN with Overlapping Decision Rules

Given an event log $\mathcal{E}$ containing information about process executions and a process model, the problem of decision mining can be regarded as that of discovering a DPN that characterizes the process: given a Petri net (P, T, F), we aim to discover the variables, write operations and guards of a DPN (P, T, F, V, U, W, G). Without loss of generality, we assume that the event log is defined over the same set T of transitions and the same set V of variables with the same universe U of values.

We make four assumptions on the input event log $\mathcal{E}$. First, we assume that ignoring the variables, all recorded process instances are *compliant* with regard to the process represented by model (P, T, F). Second, for each event $e \in \mathcal{E}$ the executed transition can be uniquely determined, i.e., there are no unobservable transitions and each event is mapped onto a single transition of the model. Third, we assume that events write attributes consistently, i.e, if an event writes an attribute $v \in V$, then all events corresponding to the same transition also write attribute v, i.e., for all $e, e' \in \mathcal{E} : \mathit{trans}(e) = \mathit{trans}(e') \Rightarrow \mathit{writes}(e) = \mathit{writes}(e')$. Fourth, we assume an initial value for each attribute. We show later in Sect. 3.3 that any event log can be transformed to an event log fulfilling these assumptions.

We discover the write operations, i.e. function W, as follows. For each $t \in T$, a variable $v \in V$ belongs to the set $W(t)$ if there exists an event $e \in \mathcal{E} : \mathit{trans}(e) = t$ that assigns a value to the variable, i.e. $v \in \mathit{writes}(e)$. We discover the guards $G(t)$ for each transition $t \in T$ using Alg. 1 as described in Sect. 3.1.

3.1 Overall Discovery Procedure

For each decision point $p \in P$ we construct a set of observation instances related to p to be used to discover the guards. In the remainder, given a set X, we denote the set of all multi-sets over a set X with $\mathbb{B}(X)$. Moreover, we use notation $X = [a^2, b]$ as short-hand notation to denote the multi-set $X = [a, a, b]$. Finally, we use $\uplus$ to denote the sum of two multi-sets, i.e., $X \uplus [b, c] = [a^2, b^2, c]$.

Definition 4 (Observation Instances). *Let p be a decision point of Petri net $N = (P, T, F)$, and let $p^\bullet = \{t_1, \ldots, t_m\}$ be the output transitions of p. We define function $I \in P \to \mathbb{B}((V \to U) \times T)$ returning the multi-set of observation instances for a decision point as:*

$$I(p) = \biguplus_{e \in \mathcal{E}, \mathit{trans}(e) \in p^\bullet} [(\mathit{values}(e), \mathit{trans}(e))]$$

For each event $e \in \mathcal{E}$ that refers to an output transition of p, i.e., $\mathit{trans}(e) \in p^\bullet$, the set of observation instances of p contains an instance $(\boldsymbol{x}, t) \in I(p)$, with $x \in (V \to U)$ being the observed values of the attributes, and $t \in T$ the observed transition. The values of $\boldsymbol{x}$ are obtained by taking the latest observed value for the attributes in preceding events.

Example 3. Given the process model introduced in Example 1 and the event log $\mathcal{E}$ introduced in Example 2 the multiset of observation instances for place p_1 is $I(p_1) = [((\texttt{status} = \mathit{unpaid}, \texttt{amount} = 30), \texttt{Payment})^2, ((\texttt{status} = \mathit{unpaid}, \texttt{amount} = 30), \texttt{Send Fine})^2, ((\texttt{status} = \mathit{paid}, \texttt{amount} = 30), \texttt{Close Fine})]$.

Algorithm 1: discoverGuards

```
Input: Petri net $(P, T, F)$, Compliant Event Log ($\mathcal{E}$), Minimum Instances ($\nu$), Merge Ratio ($\epsilon$)
Result: Guard function of the DPN $G$)
foreach $p \in P$ s.t. $|p^\bullet| > 1$ do
    $\psi_p \leftarrow$ buildEstimator$(p, I, \nu \cdot |I(p)|, \epsilon)$
end
foreach $t \in T$ do
    $G(t) \leftarrow$ true
    foreach $p \in {}^\bullet t$ do
        $G(t) \leftarrow G(t) \wedge \psi_p(t)$
    end
end
return $G$
```

Algorithm 1 describes the overall discovery method for the entire process model. The algorithm takes a Petri net (P, T, F) (i.e., without data) and an event log $\mathcal{E}$ as input and returns the guard function G of the DPN. Using the observation instances I_p, we build the guard function ψ_p for each decision point $p \in P$ through function `buildEstimator` as described in Algorithm 1. Having obtained the guard function, we assign each transition the conjunction of all rules obtained from their input places [6].

3.2 Discovering Overlapping Rules

Whereas the construction of $I(p)$ and Algorithm 1 share similarities with the previous work [6], our technique differs considerably in how the actual rules are obtained. Our contribution is a new algorithm that discovers guards that may be partially overlapping. Two or more transitions may be enabled for some state reachable in the DPN: $G(t_i) \wedge G(t_j)$ with $i \neq j$ may evaluate to `true`. The exact choice which transition is executed as next is non-deterministic.

Algorithm 2 describes how we discover overlapping guards for place p given the observation instances $I(p)$ and two user-defined parameters, the minimum number of instances n and the merge ratio ϵ. As our approach makes use of decision trees, we introduce a decision tree builder.

Definition 5 (C4.5 Decision Tree Builder). *Let O be a multi-set of observation instances over a set V of variables. Let $n \in \mathbb{N}$ be the minimum number of instances on a leaf for the splitting criterion in the decision tree induction. Function $\texttt{buildTree}_n(O) \in 2^{Formulas(V) \times T}$ returns the leaves of a C4.5 decision tree built using the supplied set of instances. A leaf predicts transition $t \in T$ under condition $expr \in Formulas(V)$.*

The rule for a leaf of the decision tree is obtained by taking the conjunction of all conditions represented by those nodes that are encountered on a path from the leaf up to the root node [6]. Initially all transitions are assigned placeholder guards $\psi(t) = void$. For notational convenience, we assume that for any $expr \in Formulas : expr = void \wedge expr = expr \vee expr$, i.e., $void$ does not influence the result (line 1).

First, a **base decision tree** $baseTree = \texttt{buildTree}_n(I(p))$ is built such that each leaf $(expr, t)$ of the decision tree corresponds to a rule $expr$ that predicts a single transition t as outcome (lines 2). Each discovered rule is added to $\psi(t)$ in disjunction to the

Algorithm 2: buildEstimator

```
Input: Place $(p)$, Observation Instances $(I)$, Minimum Number of Instances $(n)$, Merge Ratio $(\epsilon)$
Result: Guard Function $(\psi)$
$\psi : T \rightarrow Formulas \cup \{void\}$ s.t. $\forall t \in p^\bullet : \psi(t) = void$
foreach leaf: $(expr, t) \in buildTree_n\,(I(p))$ do
    $\psi(t) \leftarrow \psi(t) \vee expr$
    $\bar{I} \leftarrow [(\boldsymbol{x}, t') \in I(p) \mid t \neq t' \wedge expr$ evaluates to true for $\boldsymbol{x}]$
    $subTree \leftarrow buildTree_{n \cdot |\bar{I}|/|I|}\,(\bar{I})$
    if $|subTree| > 1$ then
        foreach subLeaf: $(subExpr, t') \in subTree$ do
            $\psi(t') \leftarrow \psi(t') \vee\ (expr \wedge subExpr)$
        end
    else
        $\{(\mathtt{true}, t')\} \leftarrow subTree$                                   // Majority vote $t'$
        if $|\bar{I}| > n\ and\ \frac{|\{(\boldsymbol{x},t) \in \bar{I} \mid\ t \neq t'\}|}{|\bar{I}|} < \epsilon$ then
            $\psi(t') \leftarrow \psi(t') \vee (expr)$
        end
    end
end
foreach $\{t \in p^\bullet \mid \psi(t) = void\}$ do $\psi(t) = \mathtt{true}$
return $\psi$
```

already discovered rules (line 3). We extract those instances $\bar{I}$ that have been wrongly classified by the base classifier: $\bar{I}$ contains all those instances $(\boldsymbol{x}, t') \in I(p)$ such that the predicted transition t' is different from the transition t in leaf l, i.e, $t' \neq t$ (line 4).

Next, we build an **new decision tree** $subTree$ based on those instances $\bar{I}$ that have been wrongly classified by the base classifier. Since the size of $\bar{I}$ can be significantly smaller than that of I, we scale down the parameter n to $n' = n \cdot |\bar{I}| / |I|$ (line 5). The idea is that the second decision tree $subTree$ can *further discriminate* between the observed transitions among the wrongly classified instances, thus, possibly introducing partial overlap with the existing rule. There are two possible cases:

1. an additional decision tree with more than one leaf is found;
2. a decision tree with a single leaf $(\mathtt{true}, t')$ is returned.

In the **first case** (line 6), we build rules for each leaf $subLeaf$ by taking the conjunction of the rule from the leaf of the first decision tree $expr$ and the newly discovered rule $subExpr$. Then, this conjunction is added to the already discovered rule for the predicted transition t' (line 7-9). In the **second case** the decision tree represents a majority vote, i.e, the transition that was most often observed within the wrong instances is predicted. In this case we add rule $expr$ to the existing guard $\psi(t')$ only under two conditions to avoid overfitting the data: First, the set of wrong instances is larger than the user-specified number of minimum instances n, and, second, the fraction of observation instance in $\bar{I}$ referring to t' is larger than the user-specified merge ratio ϵ (line 11-13). Finally, we assume that all transitions for which no rule could be found are always enabled (line 17).

Example 4. Given the multi-set of observations for place p_1 obtained from the events given in Table 1, we build the guard estimation function ψ_{p_1} using Algorithm 2. First a decision tree is built. Assume that this initial decision tree consists of two leafs

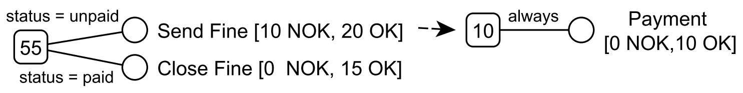


Fig. 2. Two decision trees that are discovered on the example data set. The number of instances is written in the root node (rectangle). The number of wrongly and correctly predicted instances is written next to leaf nodes (circle)

$l_1 = (\texttt{status} = \mathit{paid}, \texttt{Close Fine})$ and $l_2 = (\texttt{status} = \mathit{unpaid}, \texttt{Send Fine})$ as shown in Fig. 2. Ten of the 30 instances classified as `Close Fine` are wrongly classified. In all those instances transition `Payment` was observed. Those wrongly classified instances give evidence that also transition `Payment` is performed when `status` is unpaid. As depicted on the right-hand side of Fig. 2 an additional decision tree is built for those 10 instances. As the set of wrong observation instances $\bar{I}_{l_2}$ only contains instances for transition `Payment`, the additional decision tree only consists of one leaf that always predicts transition `Payment` by majority vote. If the user-defined threshold n is below the number of wrongly classified instances $|I_{l_2}| = 10$, then, the condition $\texttt{status} = \mathit{unpaid}$ will be added to the guard function of `Payment`. The same rule as shown in Fig. 1 is then discovered. Please note that this example is deliberately simplified. In a real-life setting event logs contain more than two attribute and, therefore, the additional decision would consist of multiple leafs.

3.3 Dealing with Real-life Event Logs

In Sect. 3 we have made several assumptions to explain the key idea: the event log fits the Petri net (P, T, F) perfectly, the set of attributes written by an event is consistent throughout the log, and every attribute value is initialized in the event corresponding to the first transition. Generally, real-life event logs do not satisfy this requirement. However, we can deal with these issues as shown next.

Non-compliant Event Log and Duplicate Transitions. An event log might contain events that cannot be matched reliably to a single transition in the DPN (e.g., in presence of noise or duplicate transitions). Similarly, for some transitions that are required according to the model no event has been recorded (e.g., when the recording is incomplete or for invisible routing transitions). Therefore, it might not be possible to determine $\mathit{trans}(e)$ for every event $e \in \mathcal{E}$. Using *alignment-based* techniques such as [4,11] we can determine a closest corresponding **process trace**, i.e., the sequence of transition executions leading to a final state of the DPN, for each *log trace*. Work [6] uses the same technique to deal with non-compliant event logs and shows that for reasonably compliant event logs the error introduced by such an alignment is negligible.

Inconsistent Attributes. In Sect. 3, we restricted the set of write operations for a transition to those variables that are consistently given a value by every event connected to this transition. In real-life event logs attribute values can be missing due to temporary recording errors leading to an inconsistent recording of attributes for a given transition.

Moreover, the *alignment-based* techniques might need to introduce artificial events for which the attribute values are unknown in case events are missing. Therefore we introduce a user-defined threshold K like in [6], but add a way to deal with missing values to it. A variable $v \in V$ is added to the set of write operations of a transition $t \in T$ when the variable is observed to be given a value by $K\%$ of the events e of transition t (i.e., $trans(e) = t$). As a result, attributes might be missing from the set of attributes written for an event $e \in \mathcal{E}$, i.e., $writes(e) \neq W(trans(e))$. Every time an event e does not assign a value to a variable v even though it should (i.e., $v \in W(trans(e))$), we assume $values(e)(v)$ to be $\diamond$. Symbol $\diamond$ indicates that the value is *missing*. Decision tree building algorithms, such as C4.5, can deal with such missing values.

Unassigned Attributes. Decision trees cannot deal with uninitialized attributes (similar to `NULL` values in databases). In real-life event logs, attributes might be uninitalized if some of the first events of the log's traces do not assign a value to all attributes. This issue can be mitigated by defining default values that are used when attribute have not taken on values yet. For example, for an attribute `approval` with values $\{0, 1\}$ in the event log we use -1 as default value. Hence, rules based on the default value can be discovered, e.g., reject claims with uninitialized `approval` < 0.

4 Evaluation

We evaluate our technique using two real-life data sets, and compare the obtained results to standard methods like decision tree induction algorithms. An implementation of our technique is available in the *MultiPerspectiveExplorer* [13] package of the open-source process mining framework ProM.

4.1 Evaluation Setup

Approaches. We compared the performance of our approach expressed in terms of place fitness and place precision with three other methods. We choose two methods at the extreme ends of the respective measure, and one method that naïvely introduces overlap. In total, we compared the following four approaches:

WO. The model without rules, i.e., the guard `true` is used for all transitions. This results in a perfect place fitness, no guard is violated.

DTF. The model with rules discovered by a decision tree as in work [6] using `false` as guard for transitions that are not observed in the tree. This method will always result in a perfect place precision as there is only one enabled transition.

DTT. The model with rules discovered by a decision tree as in work [6] using `true` as guard for transitions not observed in the tree. This method naïvely introduces overlap by enabling all those transitions.

DTO. The model with rules discovered by our approach as described in Sect. 3.1.

The DTF and WO methods are at the extreme ends of the respective measures. Our approach aims at providing better place fitness (i.e., less violated guards) at the expense of some place precision (i.e., multiple enabled transitions). Therefore, our approach should provide better place fitness than the DTF method together with better place precision than the a model without rules (WO). Method DTT is included to investigate whether our approach improves over a naïve method to introduce overlap.

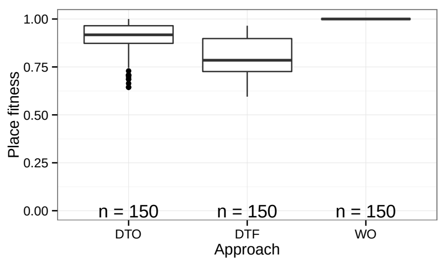
Event Logs and Process Models. We used two anonymized real-life data sets: *road fines* and *sepsis*. The road fines event log was taken from an information system handling road-traffic fines by an Italian local police force [14]. The road fines log contains more than 150,000 cases with approximately 500,000 events. There are 9 data attributes recorded including the fine amount and the payment amount. The sepsis event log contains events about the pathways of patients within a hospital that have a suspicion for sepsis, a life threatening condition typically caused by an infection. This event log contains data recorded during a full year resulting in 1056 cases with about 15,000 events. There are 39 data attributes recorded, e.g., the results of blood tests and information about the patient from checklists filled in by nurses. For both event logs, we obtained a normative process model of the control-flow, thus, without any guards. We used the control-flow of the process model presented in [4] for the road fines data set. For the sepsis data set, we created a model with help of domain experts from the hospital. Both models allow for most the behavior observed in the logs, but are lacking precision. We checked this using the fitness measure defined for DPNs in work [4] (road fines: 99.7%, sepsis: 98.6%) and the precision measure for DPNs proposed by [12] (road fines: 63.9% , sepsis: 16.5%). Therefore, both models are good candidates for adding precision through discovered rules.

Experimental Design. We performed experiments for every decision point for the road-fines and sepsis models, with the exception of four decision points of the sepsis model for which no technique was able to discover rules. We use the C4.5 implementation of the WEKA toolkit with the pruning feature activated; the merge ratio ϵ was set to 0.5 in all experiments. For each technique, we used 10 different values of *minimum number of instances* (`minInstances`) parameter that were equally distributed within a certain interval, which is determined as follows. The smallest value of the interval was chosen such that the discovered guards were not composed by more than 7 atoms. This choice was based on the assumption that guards with more than 7 atoms are too complex and humanly unreadable and, hence, of no business value. The upper bound of the interval was the smallest value that could still return a rule, i.e. larger values would return no rules. In fact, a too large value of the `minInstances` parameter would constrain the decision tree to be representative of so many instances that no reliable rule can be returned. It is worth observing that the interval potentially changed with varying the decision point and the technique (DTO, DTT and DTF) being considered.

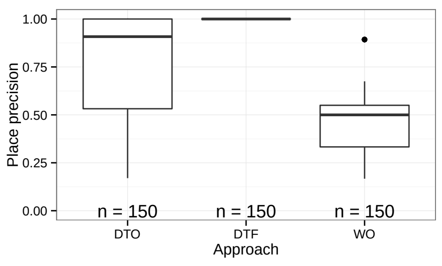
4.2 Results and Discussion

We conducted the experiments and recorded the obtained place fitness and place precision for 15 places and 10 parameter settings[4]. The boxplot in Fig. 3 shows the results. In the following paragraphs, we compare our method to the three other approaches.

[4] The data used for the evaluation is available under `http://purl.tue.nl/844997340832257`. For confidentiality reasons we cannot share the sepsis event log.



(a) Place fitness



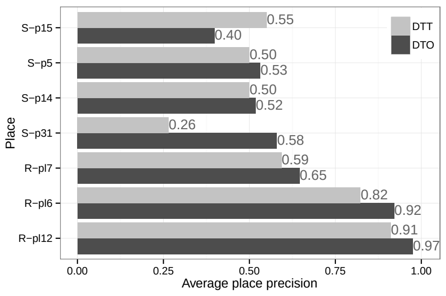
(b) Place precision

Fig. 3. Place fitness and local precision achieved by the proposed method (DTO) compared to the standard decision tree classifier (DTF), and the model without guards (WO)

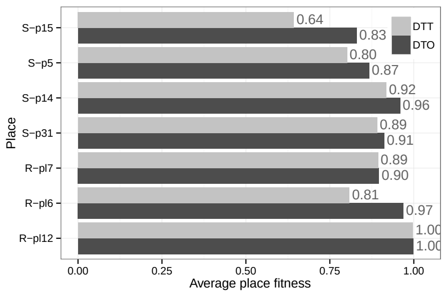
DTO vs. WO. Compared to WO, the results from our experiment (Fig. 3(b)) show clearly that DTO provides rules that increase the place precision against the process model without guards. The large spread of the obtained place precision indicates that, for some decision points and some parameter settings, our approach deliberately trades precision to obtain better fitting guards. This result is in line with the expectation that our approach returns overlapping rules that lose some precision for a better fitness.

DTO vs. DTF. The experimental results show that DTO discovers decision rules that lead to a better place fitness than the rules discovered by DTF (Fig. 3(a)) with, on average, a limited trade-off for lower precision. The outliers for DTO in Fig. 3(a) deserve some discussion. We inspected them and found that for some combinations of parameter settings and places, our approach failed to discover overlapping guards. It discovers the exact same rules as returned by DTT. Mostly, this happens for decision points with only two outgoing transitions and high settings of the `minInstances` parameter. This can be expected, as for decision trees with instances from two classes $\{A, B\}$ the wrong instances on a leaf $l = (expr, A)$ predicting transition A can only belong to the other transition B. Therefore, our approach will not discover a second decision tree with leafs predicting B, but rather use rule $expr$ from leaf l for the majority vote transition B. Alg. 2 only allows this if the number of instances for B is above the setting of the `minInstances` parameter. Therefore, for high settings of `minInstances` and decision points with two outgoing transitions our approach is unlikely to improve over the normal decision tree classifier approach.

DTO vs. DTT. We also compare DTO against DTT, which naïvely assumes the guard `true` for transitions that are not predicted by the decision tree. For this comparison, we specifically compare the results for decision points with more than two outgoing transitions, $|p^\bullet| > 2$, as the results obtained through DTT differ from the results of DTF only for those decision points. Fig. 4(b) shows the fitness and precision for those places averaged over 10 parameter settings as discussed in Sect. 4.1. Each place is given a name for reference. The results in Fig. 4(b) show that DTO is able to discover overlapping guards that fit the observations better: for all of the considered decision points, the



(a) Average place precision



(b) Average place fitness

Fig. 4. Average place fitness and place precision achieved by the DTO method compared to the DTT method. Only decision points with more than two choices are shown

decision rules returned by our approach increase the place fitness against those rules returned by DTT. Furthermore, the results show that for all except one decision point our approach discovers rules with a higher place precision. In other words, it discovers more precise guards without loosing fitness. In fact, for decision point `S-p31` our approach obtained an average place precision of 0.58 whereas the rule returned by the DTT approach scores only 0.26. Our approach discovers guards for all six outgoing transitions whereas DTT only discovers guards for three transitions. On the remaining three DTT uses `true` as a guard, i.e., always enabled. The only decision point for which DTO obtains a worse precision score than DTT is `S-p15`. Our approach discovers guards that correspond to `true` for all three alternatives. However, this is not necessarily a bad representation of the observed data. In fact, the guards discovered by DTT cause the lowest place fitness in our experiment 0.65, i.e., the discovered guards are wrong in one third of the cases.

Example. Fig. 5 shows a part of the DPN that we used for the sepsis data set. Table 2 shows the guards discovered by DTF, DTT, and DTO for the three alternative activities on decision point `S-p5`. All rules are based on two attributes: `Lactate` (`L`) and `Hypotensie` (`H`). DTF discovers the rule that patients with a lactate measurement (i.e., $\mathtt{L} > 0$) are generally admitted to normal care and patients without lactate measurement ($\mathtt{L} \leq 0$) leave the hospital. The guard for the admission to intensive care is returned as `false`. This leads to the situation where patients are never admitted to intensive care even if it is part of the model and observed. Obviously, this cannot be correct. DTF is unable to find a mutually-exclusive rule that includes this alternative activity given the information recorded in the event log. DTT discovers the same rules but naïvely assumes the guard `true` for admission to intensive care. Clearly, the DTT results are not satisfying as DTT would convey no rules about the admission of patients to intensive care. Our approach discovers that patients with a lactate measurement ($\mathtt{L} > 0$) can always be admitted to normal care. As an alternative to normal care, if attribute $\mathtt{H} = \mathtt{true}$ then patients can also be admitted to intensive care; otherwise, if $\mathtt{H} = \mathtt{false}$ patients

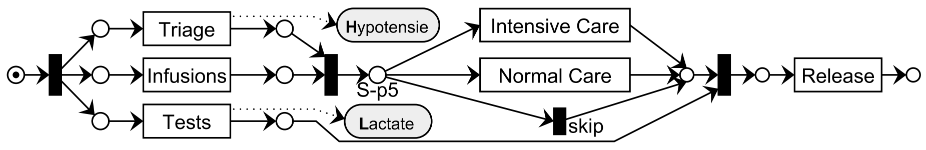


Fig. 5. Fragment of the process model used for the sepsis data set. After patients arrive in the hospital, a triage form is filled-in, the patient may receive infusions, and blood tests may be conducted. Then, patients are admitted either to normal care, intensive care, or are not admitted (skip). Two attributes relevant for this decision are recorded: `Lactat` and `Hypotensie`. For the sake of space, we depict sub processes `Infusions`, `Tests`, and `Release` as transitions

Table 2. Guards discovered by the compared approaches at decision point `S-p5`. We abbreviate attributes using their first letter, e.g., using `H` for `Hypotensie`.

Approach	**Normale Care**	**Intensive Care**	**Not Admitted**
DTF	$\texttt{L} > 0$	`false`	$\texttt{L} \leq 0$
DTT	$\texttt{L} > 0$	`true`	$\texttt{L} \leq 0$
DTO	$\texttt{L} > 0$	$\texttt{L} > 0 \wedge \texttt{H} = \texttt{true}$	$(\texttt{L} > 0 \wedge \texttt{H} = \texttt{false}) \vee \texttt{L} \leq 0$

leave the hospital. The guards for the activities overlap and the final decision is likely to be made on the basis of contextual factors, which are not encoded in the event log.

Limitations and threats to validity. The results show that our technique is successful in uncovering overlapping rules in processes from event logs, and that these rules provide in some cases a much better characterization of the observed behavior. Still, the proposed technique has some limitations and we evaluated our technique only with two real-life event logs. More experimental validation using event logs from different settings is required. An inherent limitation of our approach is that it can only use the majority vote to introduce overlapping guards for a decision point with two output transitions. This might cause the guard of one transition to be turned into the rule `true`, e.g. when the initial guards were based on a single condition.

5 Related Work

There are several approaches for decision mining given an event log with historical data about the process [1,3,5,6,15,16,17,18,19]. In all of these approaches, the decision mining problem is translated into a classification problem, and solved using classification analysis techniques such as C4.5 [7]. Work [20] proposes a technique based on discovering invariants rather than decision trees. Still, only mutually exclusive rules are considered. Most related from the traditional classification field to our approach is work about multi-label classification [21,22]. In a multi-label classification problem classes are not mutually exclusive, instances can be labeled with multiple classes, and the goal is to find the correct set of classes for unseen instances. Our setting is still different as we deal with instances that are only associated with one class, i.e., the executed transition. So, there is no work about discovering overlapping rules in the context of process models. Classifier chains methods are the closest to our work. They decompose the problem into multiple binary classification problems, one for each label [23]. This method assumes that instances are labeled with multiple classes. Also related to our work are methods for association rule mining [24], which could be used with attribute values as antecedent and the executed activity as consequent. The main problem of association-rule mining is that a potentially large set of rules is usually returned, failing to provide insights that are easy to interpret.

6 Conclusion

We propose a new technique for the discovery of overlapping rules in process models using event data. Existing techniques only return rules that assume completely deterministic decisions. This assumption rarely holds in reality. Our technique is the *first proposal* of a discovery technique that introduces overlapping rules. The technique aims to create process models that trade the precision of mutually-exclusive rules for the fitness of overlapping rules when the observed behavior gives evidence to such rules. The evaluation using several real data sets shows that our technique is able to produce models with overlapping rules that fit the observed behavior better without loosing too much precision. For some decision points, with more than 2 alternative activities, our technique returns rules that are both more fitting and more precise than the existing method [6]. As future work, we aim to investigate the application of other machine-learning techniques to decision mining. Moreover, we want to address limitations of decision mining techniques for data sets with imbalanced distributions of classes (e.g., sampling methods). We found that our technique helps to reveal rules when one transition is only observed for a small fraction of the cases, but a more thorough investigation of this phenomenon is needed.

References

1. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Object Management Group (OMG): Decision Model And Notation (DMN) Version 1.0 (2015) formal/2015-09-01.
3. Rozinat, A., van der Aalst, W.M.P.: Decision mining in ProM. In: Business Process Management. Volume 4102 of LNCS. Springer Berlin Heidelberg (2006) 420–425
4. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. Computing (2015) doi:10.1007/s00607-015-0441-1 (in press).
5. Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., Shan, M.C.: Business process intelligence. Computers in Industry **53**(3) (2004) 321 – 343
6. de Leoni, M., van der Aalst, W.M.P.: Data-aware process mining: Discovering decisions in processes using alignments. In: SAC'13, ACM (2013) 1454–1461
7. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)
8. Rosca, D., Wild, C.: Towards a flexible deployment of business rules. Expert Systems with Applications **23**(4) (2002) 385–394

9. Bose, R.P.J.C., Mans, R.S., van der Aalst, W.M.P.: Wanna improve process mining results? In: IEEE Symposium on Computational Intelligence and Data Mining. (2013) 127–134
10. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge University Press (1995)
11. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdiscip Rev Data Min Knowl Discov **2**(2) (2012) 182–192
12. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Measuring the precision of multi-perspective process models. In: Business Process Management Workshops - BPM 2015. (to appear).
13. Mannhardt, F., de Leoni, M., Reijers, H.A.: The multi-perspective process explorer. In: BPM Demo Session 2015. Volume 1418 of CEUR-WS.org. (2015) 130–134
14. de Leoni, M., Mannhardt, F.: Road traffic fine management process (2015) Eindhoven University of Technology. Dataset. doi:10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.
15. Jareevongpiboon, W., Janecek, P.: Ontological approach to enhance results of business process mining and analysis. Bus Process Manag J **19**(3) (2013) 459–476
16. Catalkaya, S., Knuplesch, D., Chiao, C.M., Reichert, M.: Enriching business process models with decision rules. In: Business Process Management Workshops. Volume 171 of LNBIP. Springer (2014) 198–211
17. Ghattas, J., Soffer, P., Peleg, M.: Improving business process decision making based on past experience. Decision Support Systems **59** (2014) 93 – 107
18. Bazhenova., E., Weske, M.: Deriving decision models from process models by enhanced decision mining. In: Business Process Management Workshops - BPM 2015. (to appear).
19. Dunkl, R., Rinderle-Ma, S., Grossmann, W., Fröschl, K.A.: A method for analyzing time series data in process mining: Application and extension of decision point analysis. In: CAiSE Forum 2014. Volume 204 of LNBIP., Springer (2014) 68–84
20. de Leoni, M., Dumas, M., García-Bañuelos, L.: Discovering branching conditions from business process execution logs. In: FASE. Volume 7793 of LNCS., Springer (2013) 114–129
21. Boutell, M.R., Luo, J., Shen, X., Brown, C.M.: Learning multi-label scene classification. Pattern Recognition **37**(9) (2004) 1757 – 1771
22. Tsoumakas, G., Katakis, I.: Multi-label classification: An overview. International Journal of Data Warehousing and Mining **2007** (2007) 1–13
23. Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier chains for multi-label classification. Machine Learning **85**(3) (2011) 333–359
24. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. SIGMOD Rec. **22**(2) (June 1993) 207–216