Data-Aware Process Mining: Discovering Decisions in Processes Using Alignments

Massimiliano de Leoni and Wil M.P. van der Aalst
Eindhoven University of Technology Eindhoven, The Netherlands
m.d.leoni@tue.nl, w.m.p.v.d.aalst@tue.nl

ABSTRACT

Process discovery, i.e., learning process models from event logs, has attracted the attention of researchers and practitioners. Today, there exists a wide variety of *process mining* techniques that are able to discover the control-flow of a process based on event data. These techniques are able to identify *decision points*, but do not analyze data flow to find rules explaining why individual cases take a particular path. Fortunately, recent advances in conformance checking can be used to *align* an event log with data and a process model with decision points. These alignments can be used to generate a well-defined classification problem per decision point. This way data flow and guards can be discovered and added to the process model.

Categories and Subject Descriptors

C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]: Process control systems

Keywords

Process Discovery, Machine-Learning Techniques, Business Process Data-flow Pespective

1. INTRODUCTION

Despite the focus in process-orientation in most organizations, few processes are fully controlled by software (e.g., a WFM or BPM system). IT systems are still data-centric and people have a lot freedom when executing tasks. This allows for flexibility, but also creates the need to analyze the processes as they are actually executed. Thanks to advances in process mining [9, 11] and the incredible growth of event data (cf. "Big Data" [5]), this is now possible. Process mining aims to *discover, monitor and improve real processes by extracting knowledge from event logs* readily available in today's information systems [9]. The two main types of process mining are Process discovery and Conformance Checking.

SAC'13 March 18-22, 2013, Coimbra, Portugal.

Given an event log consisting of a collection of traces (i.e., sequences of events), *process discovery* constructs a process model that "adequately" describes the observed behavior.

Given an event log and a process model, *conformance checking* diagnosed the differences between the observed behavior (i.e., traces in the event log) and the modeled behavior (i.e., execution sequences possible according to the model).

Lion's share of process mining research focuses on control-flow, i.e., the ordering of activities. In this paper we focus on data flow, e.g., we want to discover why particular cases take a particular path. In order to illustrate the importance of discovering the so-called *data-flow perspective*, let us consider the following example.

EXAMPLE 1. *A credit institute created a standardized process to deal with loans requested by clients. These loans can be used to buy small home appliances (e.g., fridges, TVs, high-quality digital sound systems). A customer can apply for a loan through a shop clerk. The clerk prepares the request by filling out the form and attaching documents that prove the capability to pay off the loan. Upon receiving a new request, the credit institute opens a new case. The initial step is to open the credit request by providing the information about the requester, i.e. the loan applicant, and the amount. Afterwards, the credit institute verifies the validity of the information provided. If the provided information is not valid, then the application is immediately rejected and stored in system along with informing the applicant. If the information is valid, a loan opening is made. Requests for small amounts are assessed differently than requests for large amounts (*Simple Assessment *versus* Advanced Assessment*), since, for large amounts, the loan assessment uses stricter constraints. If the decision is positive, then the application is accepted, stored in the system, and the applicant is informed. Moreover, the loan is opened. If the decision is negative, the applicant is informed about the decision. In the latter case, the decision is preliminary, because the applicant is allowed to renegotiate the loan and ask for a smaller loan. If no request for renegotiation is received, the decision becomes definitive: the applicant is informed and the negative result is stored in the system.*

Contemporary process mining techniques are able to discover models such as the Petri net in Figure 1 using only event data. Such models *only describe the control-flow perspective* and ignore data associated with the cases which are handled. However, the different decisions in the model highly depend on characteristics of the loan request (e.g., the size of the loan). *Therefore, one also needs to analyze the data-flow*

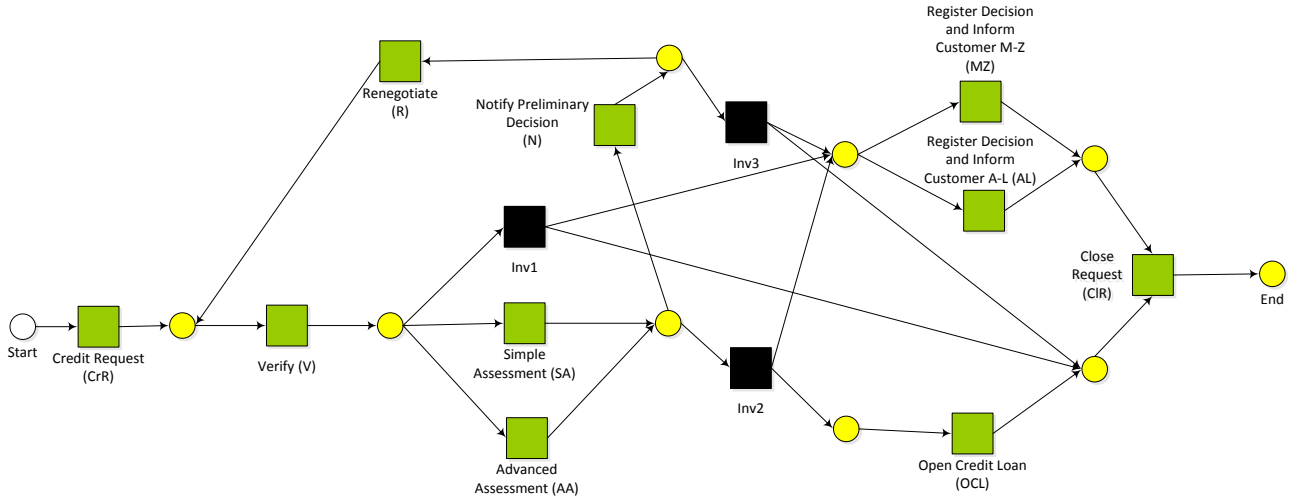


Figure 1: Petri net representation of the control-flow of the running example. The grey boxes represent the transitions that are associated with process tasks. The black boxes identify invisible transitions, i.e. transitions that do not correspond to actual pieces of work but necessary for routing purposes, whose executions are never recorded in event logs. *In brackets, there are the acronyms that are used to shorten the transition's names.* Circles are the so-called places, which may contain tokens. In order for a transition t to be executed, each input place (i.e., each place having outgoing arcs that enter t) needs to contain a token. When a transition fires, one token is removed from each input place and one token is put into each output place. Each Petri net used to model processes is characterized by two special places, the *start* and the *end* place (named *Start* and *End* in figure). In the initial state, there exists only one token in start place. A process instance is considered as concluded when a token is produced in the end place.

to discover the rules governing the choices in the process. To the best of our knowledge, the only approach that deals with the discovery of the data-flow is [7]. This approach was developed about 7 years ago and has several limitations.

Firstly, the decision mining approach in [7] can only partly deal with process models having invisible transitions. Invisible transitions do not correspond to actual pieces of work but are often necessary to model XOR-splits/joins and certain types of loops. As a consequence, the technique cannot discover conditions associated with XOR-splits and many loops. Secondly, the event log needs to fully conform to the modeled control-flow, i.e., the order according to which activities are executed can never differ from the idealized model. In fact, most control-flow discovery techniques treat the least frequent observed behaviors as noise and discard them. As result, even the discovered control-flow model does not fully conform to the event log. Also hand-made models rarely fully explain all observed behavior: It is not uncommon to encounter event logs where a number of cases cannot be fully replayed by the model from begin to end.

The approach in this paper *addresses these problems.* To do this, we use recent advances in *conformance checking using alignments* [1]. First, we discover the process control-flow, using one of the many process discovery techniques available today [9]. Then, we align the event log and control-flow, thus mitigating the effects of non-conformance, i.e., the observed behavior is squeezed into the Petri net without data. Once the alignment is computed, the *data-flow perspective can be discovered*, i.e., the read and write operations as well as the transitions guards. When mining mine the data-flow perspective, the most challenging task is to discover the guards. In this paper, we leverage on standard machine learning techniques to discover the guards. Each decision point can be seen as a *classification problem*: according to the values assigned to variables, a particular path (i.e., transition) is selected.

The proposed solution has been implemented as a plug-in for ProM that can be downloaded [10]. Moreover, we conducted experiments using synthetic process models and event logs, as well as real-life event logs. The experiments show that the discovered data-flows are surprisingly accurate, even in presence of event logs with non-conforming traces. Experiences using real-life logs show that this kind of data-aware process mining provides new and valuable insights.

Our data-aware process discovery technique is independent from the specific formalism used to describe the control-flow and data-flow perspectives. Therefore, BPMN, EPC or any other formalism can be employed to represent these perspectives. However, we use simple modeling language with clear semantics to explain our technique. In particular, we use a revisited version of *Petri nets with data* introduced in [8].

Section 2 discusses the syntax and the operational semantics of Petri nets with data. Section 3 introduces the notion of control-flow alignment. Our data-aware process discovery technique heavily relies on such alignments. Section 4 details the technique, whereas Section 5 reports the experimental results. Finally, Section 6 concludes the paper, delineating future research directions.

2. PETRI NETS WITH DATA

Before discussing syntax and semantics of Petri nets with data, we introduce classical Petri nets:

DEFINITION 1 (PETRI NET). *A Petri net is a triple* (P, T, F) *where*

- P *is a set of places;*
- T *is a set of transitions;*
- $F \subseteq (P \times T) \cup (T \times P)$ *is the flow relation describing the "arcs" between places and transitions (and between transitions and places).*

The preset of a transition t is the set of its input places: ${}^\bullet t = \{s \in P \mid (s,t) \in F\}$. The postset of t is the set of its output places: $t^\bullet = \{p \in P \mid (t,p) \in F\}$. Definitions of pre- and postsets of places are analogous. A marking of a Petri net is a multiset of its places, i.e., a mapping $M : P \to \mathbb{N}$. We say the marking assigns to each place a number of tokens. Firing a transition t in a marking M consumes a token from each of its input places ${}^\bullet t$, and produces a token for each of its output places $t^\bullet$. A transition t is enabled (it may fire) in M if there are enough tokens in its input places, i.e. iff $M \geq {}^\bullet t$.

A *Petri net with data* (DPN-net) is a Petri net in which transitions (modeling activities) can *read* and *write* variables. A DPN-net may use a finite set of process variables V and a function U that determines the domain of each variable. A transition modeling an activity is allowed to write (or update) a predefined subset of the process variables. A transition can have a data-dependent guard that blocks when it evaluates to false. Only if the guard evaluates to true and all input places are marked, a transition can fire. A guard can be any Boolean expression over V using logical operators such as conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$).

DEFINITION 2 (DPN-NET). *A Petri net with data (DPN-net)* $N = (P, T, F, V, U, R, W, G)$ *consists of:*

- *a Petri net* (P, T, F)*;*
- *a set* V *of variables;*
- *a function* U *that defines the values admissible for each variable* $v \in V$*, i.e. if* $U(v) = D_v$*,* D_v *is the domain of variable* v*;*
- *a read function* $R \in T \to 2^V$ *that labels each transition with the set of variables that it must read;*
- *a write function* $W \in T \to 2^V$ *that labels each transition with the set of variables that it must write;*
- *a guard function* $G \in T \to \mathcal{G}_V$ *that associates a guard with each transition.*[1]

EXAMPLE 1 (CONT.). *Figure 2 illustrates the data-flow perspective of the working example. When defining guards, we assume that string values can be lexicographically ordered and, hence it is possible to also use inequality operators (i.e.,* $<$ *and* $>$*) for strings.*

In order to provide an operational semantics, we introduce the concept of the state of a DPN-net:

[1]The guard is defined over (a sub set of) variables in V. If a transition t has no guard, we set $G(t) = \text{true}$.

Variable	Type
Amount	Non-negative Number
Decision	Boolean
Requester	String
Verification	Boolean

(a) definition of variables

Transition	Variables Written
Advanced Assessment	*Decision*
Credit Request	*Requester*, *Amount*
Simple Assessment	*Decision*
Renegotiate	*Amount*
Verify	*Verification*

(b) write/update operations

Transition	Guard
Inv1	*Verification* = false
Inv2	*Decision* = true
Notify Preliminary Decision	*Decision* = false
Simple Assessment	*Verification* = true $\wedge$ *Amount* > 10000
Advanced Assessment	*Verification* = true $\wedge$ *Amount* ≤ 10000
Register Decision and Inform Customer M-Z	*Requester* $\geq$ "*M*"
Register Decision and Inform Customer A-L	*Requester* $\leq$ "*L*"

(c) transition guards

Figure 2: The data-flow perspective for the running example. For clarity, here we omit to list the read operations, assuming that transitions only read the variables on which the respective guards are defined.

DEFINITION 3 (STATE OF A PETRI NET WITH DATA). *Given a DPN-net* $N = (P, T, F, V, U, R, W, G)$ *and let* $D = \bigcup_{v \in V} U(v)$*, the state of* N *is a pair* (M, A) *consisting of*

- *A marking* M *for Petri net* (P, T, F)
- *A function* A *that associates a value with each variable, i.e.* $A : V \to D \cup \{\perp\}$*, with* $A(v) \in U(v) \cup \{\perp\}$*. If a variable* v *is not given a value, we use the special symbol* $\perp$*, i.e.* $A(v) = \perp$*.*

In the initial state, there is only one token in a so-called start place $p_0 \in P$. A process instance is considered as concluded when a token is produced in a so-called end place $p_e \in P$. The *initial state* is (M_0, A_0) where $M_0(p_0) = 1$, $M_0(p) = 0$ for any other place p, and for all $v \in V$: $A_0(v) = \perp$.

In the remainder, $\mathsf{dom}(f)$ denotes the domain of some function f. The operational semantics can be introduced in term of valid transition firing and state transition:

DEFINITION 4 (VALID FIRING AND STATE TRANSITION). *Given a DPN-net* $N = (P, T, F, V, U, R, W, G)$*, a firing of a transition is denoted by a pair* (t, r, w) *where* $t \in T$*,* $r \subseteq V$ *is the set of variables that are read and* $w : V \not\to U$ *is the set of variables that are written with the respective values. A transition firing* (t, r, w) *is valid in state* (M, A) *if the following conditions hold:*

1. *each place in the preset of t contains at least one token, i.e. iff* $\forall p \in {}^\bullet t : M(p) > 0$*;*
2. *the transition reads and writes all and only the variables that it is prescribed to, i.e.* $r = R(t)$ *and* $\mathsf{dom}(w) = W(t)$*;*
3. *the value assigned to each variable is valid, i.e.* $\forall v \in \mathsf{dom}(w).\ w(v) \in U(v)$*;*
4. *the guard* $G(t)$ *evaluates true with respect to the assignment* A *of values to process' variables.*

A valid firing (t, r, w) *in state* (M, A) *leads to state* (M', A')*, where:*

$$M'(p) = \begin{cases} M(p) - 1 & \text{if } p \in {}^\bullet t \setminus t^\bullet; \\ M(p) + 1 & \text{if } p \in t^\bullet \setminus {}^\bullet t; \\ M(p) & \text{otherwise} \end{cases}$$

and

$$A'(v) = \begin{cases} A(v) & \text{if } v \in V \setminus W(t); \\ w(v) & \text{if } v \in W(t). \end{cases}$$

3. ALIGNING EVENT LOGS AND PROCESS MODELS

Before discovering the data-flow, we need to align the event log and control-flow process model, i.e., events in the log need to be related to transition executions. Please note that, before discovering the data-flow, the alignment can only consider the control-flow. Adriansyah et al. have developed powerful techniques to align models and logs and showed how to use these alignments for conformance checking [1]. Nevertheless, these techniques only focus on the name of process activity to which the event log refers, ignoring other attributes. However, data attributes are of the utmost importance for discovering data flow. Here, we discuss how the concept of alignment can be extended to incorporate data. First, we introduce the basic *alignment* concept.

A Petri net $P = (T, P, F)$ that describes a process relies on constructs such as parallel split nodes, synchronization nodes, decision/choice nodes, conditions, merge nodes, etc. Although we did not formalize this, we assume Petri nets with a defined initial and final state and consider all traces $\mathcal{D} \subseteq T^*$ (i.e., firing sequences) that start in the initial marking and end in the final marking. $\mathcal{D} \subseteq T^*$ fully describes the behavior of the process model, i.e., $\sigma_M \in \mathcal{D}$ is a complete trace.

An event log contains events associated with cases, i.e., process instances. Hence, a case can be described in terms of a trace σ_L, i.e., a sequence of events. Each event describes a log execution step and can be represented by a pair (a, ϕ) consisting of an activity a and a value assignment ϕ. Here we assume that activities directly correspond to transitions, i.e., $a \in T$. However, this can be relaxed if needed, e.g., multiple transitions referring to the same activity or activities that are described by multiple transitions, e.g., to denote the start and completion of the activity. $\phi \in V \not\to U$ is a function that assigns a value to some of the variables in V. $\Phi = V \not\to U$ is the set of all such functions. An **event log** $\mathcal{L}$ is a multiset of traces where each trace consists of events of the form (a, ϕ). In other words, $\mathcal{L} \in \mathcal{B}((T \times \Phi)^*)$.[2]

An alignment relates *moves in log* and to *moves in model* as explained in the following definition. Here, we explicitly indicate *no move* with $\gg$.

[2] $\mathcal{B}(X)$ the set of all multisets over X.

Definition 5 (Control-flow Alignment). *Let us denote* $S_L = (T \times \Phi) \cup \{\gg\}$ *and* $S_M = T \cup \{\gg\}$*. A pair* $(s_L, s_M) \in (S_L \times S_M) \setminus \{(\gg, \gg)\}$ *is*

- *a* move in log *if* $s_L \in (T \times \Phi)$ *and* $s_M = \gg$*,*
- *a* move in model *if* $s_L = \gg$ *and* $s_M \in T$*,*
- *a* move in both *if* $s_L = (a_L, \phi) \in (T \times \Phi)$*,* $s_M \in T$*, and* $a_L = a_M$*.*

$\Sigma = (S_L \times S_M) \setminus \{(\gg, \gg)\}$ *is the set of the* legal moves.

The alignment *of a log trace* $\sigma_L \in (T \times \Phi)^*$ *and a model trace* $\sigma_M \in T^*$ *is a sequence* $\gamma \in \Sigma^*$ *such that the projection on the first element (ignoring* $\gg$*) yields* σ_L *and the projection on the second element yields* σ_M *(ignoring* $\gg$*).*

If γ is an alignment of log trace σ_L and model trace σ_M, and if $\sigma_M \in \mathcal{D}$, it is called a **complete control-flow alignment** of σ_L and $\mathcal{D}$. An alignment of the event log $\mathcal{L}$ and the process model $\mathcal{D}$ is a multiset $\mathcal{A} \in \mathcal{B}(\Sigma^*)$ of alignments such that, for each log trace σ_L, there exists an alignment $\gamma \in \mathcal{A}$ of σ_L and $\mathcal{D}$. $\mathcal{A}$ is a multi-set because an event log may contain the same log trace σ_L multiple times, potentially resulting in multiple identical alignments.

In order to quantify the severity of a deviation, end users need to configure a cost function on the legal moves $\kappa \in \Sigma \to \mathbb{R}_0^+$. It may be defined differently for individual processes, since, generally speaking, the costs depend on the specific characteristics of the process. In most of scenarios, a **standard cost function** is applicable: $\forall (s_L, s_M) \in \Sigma$. $\kappa^{std}(s_L, s_M) = 1$ if $s_L = \ \gg$ or $s_M = \ \gg$; otherwise, $\kappa^{std}(s_L, s_M) = 0$. The cost of an alignment γ is defined as the sum of the costs of the individual moves in the alignment, i.e., $\mathcal{K}(\gamma) = \sum_{(s_L, s_M) \in \gamma} \kappa(s_L, s_M)$.

Given a log trace $\sigma_L \in \mathcal{L}$, the techniques of Adriansyah et al. [1] aim at finding a complete alignment of σ_L and a corresponding model trace $\sigma_M \in \mathcal{D}$ such that the alignment costs are minimal with respect to all $\sigma'_M \in \mathcal{D}$. In other words: there is no complete alignment of σ_L and $\mathcal{D}$ that has lower costs. A chosen alignment with the lowest cost is referred to as an *optimal alignment*.

Definition 6 (Optimal Control-flow Alignment). *Let* $\sigma_L \in \mathcal{L}$ *be a log trace and* $\mathcal{D}$ *a process model. Let* $\Gamma_{(\sigma_L, \mathcal{D})}$ *be the set of the complete alignments of* σ_L *and* $\mathcal{D}$*. A complete alignment* $\gamma \in \Gamma_{(\sigma_L, \mathcal{D})}$ *is an* optimal alignment *of* $\sigma_L \in \mathcal{L}$ *and* $\mathcal{D}$ *iff for all* $\gamma' \in \Gamma_{(\sigma_L, \mathcal{D})}$*:* $\mathcal{K}(\gamma') \geq \mathcal{K}(\gamma)$*.*

Example 1 (cont.). *Suppose we have a log trace:*

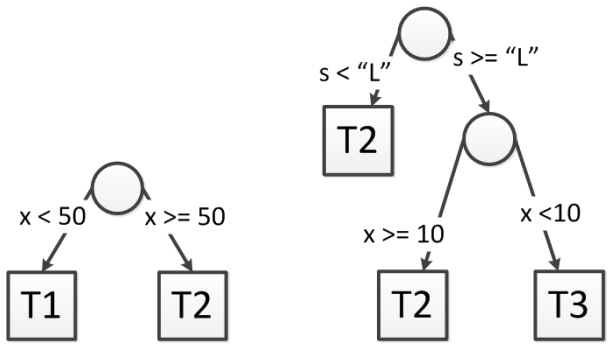
$$\sigma_L = \langle CrR\{A = 1000\}, V\{V = true\}, SA\{D = true\}, AA\{D = true\}, OCL, ClR \rangle$$

where activity and attribute names are shortened with the acronyms in brackets in Figure 1. If we use the standard cost function, an optimal control-flow alignment of σ_L *is:*

L:	$CrR\{A = 1000\}$	$V\{V = true\}$	$SA\{D = true\}$
P:	CrR	V	SA

L:	$AA\{D = true\}$	$\gg$	OCL	$\gg$	ClR
P:	$\gg$	$Inv2$	OCL	MZ	ClR

In order to find the optimal alignments as defined above, we first apply the technique described in [1] and already implemented in ProM. Then, we iterate over all log traces and enrich the log execution steps with the value assignments to attributes, as recorded in the attributes associated with events.



(a) (b)

Figure 3: Two decision trees describing estimators for (a) function $f_1 : \mathbb{N} \times \textbf{String} \rightarrow \{T1, T2\}$ and (b) function $f_2 : \mathbb{N} \times \textbf{String} \rightarrow \{T2, T3\}$.

4. DISCOVERY OF THE DATA-FLOW PROCESS PERSPECTIVE

The data-flow discovery technique presented in this paper takes a Petri net (P, T, F) without data and an event log $\mathcal{L}$ as input, along with the multi-set of optimal control-flow alignments Σ of (P, T, F) and $\mathcal{L}$. The outcome is a Petri net with data $N = (P, T, F, V, U, R, W, G)$, where our technique mines V, U, R, W and G. In the remainder, we say an event $(t, \phi) \in \mathcal{L}$, if there exists a trace $\sigma \in \mathcal{L}$ such that $(t, \phi) \in \sigma$.

We reasonably assume the set of variables of N are the set of variables defined in the event logs, i.e. $V = \{v \mid \exists (t, \phi) \in \mathcal{L} \text{ s.t. } v \in \mathsf{dom}(\phi)\}$. Given a variable $v \in V$, the values admissible for v are all those which have been observed in the log, i.e. $U(v) = \{u \mid \exists (t, \phi) \in \mathcal{L} : \phi(v) = u\}$. Regarding the write operations, we assume that a transition t writes a variable v if, according to the log, at least $X\%$ of times, events for t contain a value assignment to attribute v, where X can be customized by the users.

The most challenging task is to discover the guards of the transitions. Generally speaking, given a transition t, the respective guard $G(t)$ specifies the data-variables conditions that need to hold in order for t to be enabled to fire. In order to discover the guard for each transition, we first need to find the decision points. In our setting, each place p that has multiple outgoing transitions $t_1, \ldots, t_n$ forms a *decision point*. Indeed, these transitions are in "conflict": only one out of $t_1, \ldots, t_n$ is allowed to fire. Therefore, for each state (M, A) reachable from the initial state, $G(t_1) \wedge \ldots \wedge G(t_n)$ evaluates to false using the value's assignment A, and $G(t_1) \vee \ldots \vee G(t_n)$ evaluates to true.

In this paper, we assume that a transition t reads all the variables needed to evaluate $G(t)$. Obviously, if the event log contains additional information, we can use different approaches that allow for a more accurate discovery of the read operations. However, that is outside the scope of this paper.

4.1 Function Estimators

The discovery of the guards of the transitions associated with a decision point can be translated into the problem of finding the best estimator of a function.

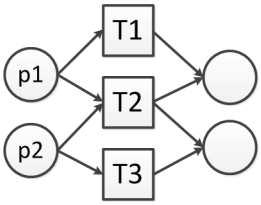


Figure 4: A fragment of a Petri net where a transition, i.e. T2, consumes tokens from two decision-point places.

Definition 7 (Function estimator). *Let $f : X_1 \times X_2 \times \ldots \times X_n \rightarrow Y$ be a function having a finite domain Y. An estimator of function f is a function $\psi_f : Y \rightarrow 2^{X_1 \times X_2 \times \ldots \times X_n}$, such that, for each $y \in Y$, $\psi_f(y)$ returns all input domain tuples for which the expected output is y.*

The function estimator is trained through a set of observations. An observation instance is a pair $(\overrightarrow{x}, y)$ where $\overrightarrow{x} \in X_1 \times X_2 \times \ldots \times X_n$ is the observed input and $y \in Y$ is the observed output. Given a set I of observation instances, for a practical use, the construction of a function estimator is abstracted as a function $\mathsf{buildFunctionEstimator}(I)$, which returns a function ψ_f such that, for each observed output $y \in Y$, $\psi_f(y) = expr$ where $expr$ is a expression that characterizes the input tuples $\overrightarrow{x}$ that are expected to lead y as output, i.e. $f(\overrightarrow{x}) = y$.

The function estimator can be easily built using many machine learning techniques. In this paper, we employ decision-tree building algorithms, specifically the C4.5 algorithm [6]. There are many reasons why to use decision-tree building algorithms to build a function estimator: the training data may contain missing attribute values and errors. Moreover, an expression can potentially contain disjunctions. Last but not least, the input domains are potentially defined over continuous domains.

Decision trees classify instances by sorting them down in a tree from the root to some leaf node. Each non-leaf node specifies a test of some attribute $x_1, \ldots, x_n$ and each branch descending from that node corresponds to a range of possible values for this attribute. In general, a decision tree represents a disjunction of conjunctions of expressions: each path from the tree root to a leaf corresponds to an expression that is, in fact, a conjunction of attribute tests. Each leaf node is associated one of the possible output values: if an expression e is associated with a path to a leaf node y, every input tuple for which e evaluates to true is expected to return y as output. Let us clarify through an example:

Example 2. *Let us suppose to have two functions: $f_1(x, s) = y_1$ and $f_2(x, s) = y_2$, where $x \in \mathbb{N}$, s is a string, $y_1 \in \{T1, T2\}$ and $y_2 \in \{T2, T3\}$. Let us suppose that we want to build the most accurate estimators ψ_{f_1} and ψ_{f_2} of f_1 and f_2, respectively. To build such estimators, we construct decision trees for f_1 and f_2. The two decision trees shown in Figure 3 were built using training sets for both f_1 and f_2. The decision tree on the left-hand side describes estimator ψ_{f_1}: $\psi_{f_1}(T1) = \{(x, s) \mid x < 50\}$ and $\psi_{f_1}(T2) = \{(x, s) \mid x \geq 50\}$. The decision tree on the right-hand side specifies ψ_{f_2} with $\psi_{f_2}(T3) = \{(x, s) \mid s \geq \text{"}L\text{"} \wedge x < 10\}$ and $\psi_{f_2}(T2) = \{(x, s) \mid s < \text{"}L\text{"} \vee (s \geq \text{"}L\text{"} \wedge x \geq 10)\}$.*

Algorithm 1: generateTransitionGuards

Data: $\mathbf{N} = (P, T, F)$ – A Petri net without data, $\mathcal{A}$ – A multi-set of optimal control-flow alignments of N and an event log
Result: A Guard Function $G : T \rightarrow \mathcal{G}$

```
1  Let I be a function whose domain is the set of places p s.t.
   | p• |> 1 and ∀p ∈ P s.t. | p• |> 1. I(p) = ∅.
2  foreach alignment ⟨(s_L^1, s_M^1), ..., (s_L^n, s_M^n)⟩ ∈ A do
3      Set function A such that dom(A) = ∅
4      for i ← 1 to n do
5          if s_M^i ≠≫ then
6              foreach p ∈ •s_M^i s.t. | p• |> 1 do
7                  I(p) ← I(p) ∪ (A, s_M^i)
8              end
9          end
10         if s_L^i ≠≫ ∧ s_M^i ≠≫ then
11             Let s_L^i = (a_L^i, φ_L^i)
12             foreach variable v ∈ dom(φ_L^i) do
13                 A(v) ← φ_L^i(v)
14             end
15         end
16     end
17 end
18 foreach place p ∈ P s.t. | p• |> 1 do
19     ψ_p ← buildFunctionEstimator(I(p))
20 end
21 foreach transition t ∈ T do
22     G(t) ← true
23     foreach place p ∈ •t s.t. | p• |> 1 do
24         G(t) ← G(t) ∧ ψ_p(t)
25     end
26 end
27 return G
```

4.2 Discovery of Guards

Algorithm 1 illustrates the steps to discover the guards of transitions. The input parameters are a Petri net without data (e.g. obtained by control-flow discovery algorithms) and a multi-set of optimal control-flow alignments. The output is the guard function G that is mined. Initially, in line 1, we initialize function I which is going to associate each decision-point place p with the set of observation instances that refer to execution of transitions in the postset of p. From line 2 to line 17, we replay all control-flow alignments to build the observation instances. While replaying, a function A keeps the current value's assignment to variables (line 3). In lines 5-9, for each move in the alignment that is in model or both, i.e. such that $s^i_M \neq \gg$, we create an observation instance where A and s^i_M are the input and output, respectively. This instance is also added to the set of instances $I(p)$ for place p. In fact, each move in log is a deviation and, thus, needs to be ignored. In lines 10-15, for each move in both, we update the current value's assignment, i.e. we rewrite function A.

Once all observation instances $I(p)$ have been built for each decision-point place p, we build the function estimator ψ_p (lines 18-20). Now the guards can be mined (see lines 21-26). For each transition t, the algorithm considers the function estimators $\psi_{p_1}, \ldots, \psi_{p_n}$ associated with all decision-point places $p_1, \ldots, p_n \in {}^\bullet t$. The guards of transition t is $G(t) = \psi_{p_1}(t) \wedge \ldots \wedge \psi_{p_n}(t)$.

Example 2 (cont.). *Let us suppose to have a fragment of Petri net as in Figure 4. In fact, f_1 and f_2 are the functions that, given the current value's assignment as input, supposedly return the transition that consumes a token from p_1 and p_2, respectively. Let us suppose that, after replaying the control-flow alignments, we discover that ψ_{f_1} and ψ_{f_2} are the most accurate function estimators. Therefore, the guards for T_1 and T_3 are $\psi_{f_1}(T1)$ and $\psi_{f_1}(T3)$, respectively. Since T_2 consumes tokens from both places, the corresponding guard is $\psi_{f_1}(T2) \wedge \psi_{f_2}(T2)$.*

Removed Events	Data-flow Conformance	Number of Guards Discovered
10%	1	7
20%	0.9999 - 1	7
25%	0.90 - 0.95	6
30%	0.80 - 0.90	6
35%	1	4
40%	1	2
50%	1	2

Table 1: Outcomes of the experiments to verify the level of robustness of the solution approach against event logs with deviations. The first column shows different values of percentages of events that have been removed. The second column illustrates the average data-flow conformance that has been observed during the experiments. The third column shows the number of guards that are discovered: the removal of more events cause the approach to discover a smaller number of guards.

5. IMPLEMENTATION AND EXPERIMENTAL EVALUATION

The *Data-flow Discovery* plug-in implements the approach just described. The plug-in is part of ProM, a generic open-source framework for process mining [10], and takes three objects as input: a Petri net, a log and the multi-set of alignments computed by the *Control-flow Conformance Checker* plug-in [1]. It returns a Petri net with data where the guards and the read and write operations are discovered by employing the techniques described in Section 4.

To evaluate the approach, we performed experiments to answer three questions: *(i)* the accuracy of discovered data-flows; *(ii)* how the accuracy is influenced by the number of control-flow deviations present in the event log; and *(iii)* whether the approach is applicable in real settings and which kind of insights can be gained.

To answer to the first two questions, we have employed Example 1; for the third question, we have used a real-life event log, taken from a Dutch Financial Institute, which is publicly available.

5.1 Experiments with a Synthetic Model and Event Log

In order to measure the accuracy of a discovered data-flow, we used Example 1 and generated an event log with 3000 traces by modeling the process in CPN Tools (`http://cpntools.org`) and subsequently simulating the model. The event log complies the control-flow in Figure 1 and the data-flow in Figure 2. With a synthetic event log, the evaluation lends itself to verify the accuracy of a discovered process data-flow, since the discovered data-flow can be compared with the target data-flow according to which the event log has been generated. In ProM, we employed the *Data-flow Discovery* plug-in, using the generated log and the same Petri net as in Figure 1. In less than 1 second, the plug-in discovered a DPN-net a with the same data-flow as Figure 2. Figure 5 shows how ProM visualizes the Petri net

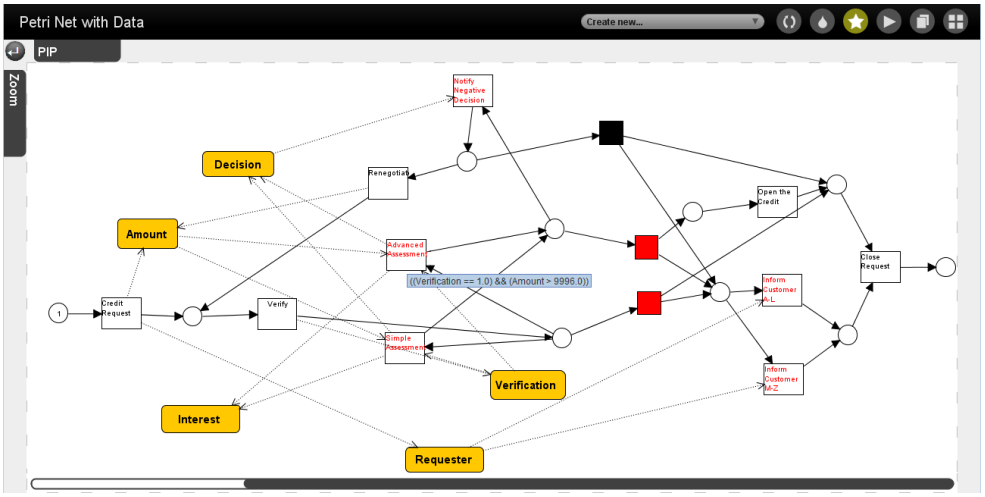


Figure 5: A screenshot of the *Data-flow Discovery* plug-in in ProM: the output of the plug-in is a Petri net with data. The white and black rectangles identify the visible and invisible transitions. The yellow "rounded rectangles" represent the variables defined in the process data-flow. The dotted arrows going in and out the yellow rectangles describe the write and read operations, respectively. When passing over a transition, a light-blue pop-up shows the possible guard. The transitions with red inscriptions or filled in red are those, visible or invisible, which are associated a guard.

with data discovered by the *Data-flow Discovery* plug-in.

Next we investigate how accuracy is influenced by event logs that are not fully conforming to the process control-flow. For this, we removed various percentages of events from the log and we used the resulting event log as input for the *Data-flow Discovery* plug-in. To finally check the accuracy of the discovered data-flow process model, we checked the data-flow conformance of the discovered process and the original log, i.e. the event log before removing the events. To measure the data-flow conformance, we used our implementation of a data-flow Conformance Checker, which extends [3] to deal with infinite domains. In the same way as [3], the data-flow conformance of a trace is measured with a value between 0 (none of the events is conforming) and 1 (all events are conforming).

Table 1 shows a summary of the results of the experiments after removing different percentages of events. For each value of percentage, we run the experiment 10 times, randomly removing an appropriate percentage of events. Up to 20%, removing events from the traces does not cause significant differences in the discovered guards. Indeed, in most of the runs of the experiments, the data-flow conformance is still 1 for each trace. Only in few cases, when removing 20% of the events, the average data-flow conformance of all traces is 0.9999. When removing 20-35% of events performance start to degrade gradually. As result, the function estimators are no longer completely accurate, and, hence, the discovered guards may contain some mistakes. Indeed, in a few experiment runs, the average value of data-flow conformance of each trace even reduces to 0.8. When removing 35%, 40% or 50% of events, the event log starts containing too little information and, hence, fewer guards are discovered, even though they are exactly as they would be expected. As result, the data-flow conformance is again 1. Nonetheless, the data-flow model is underfitting: since the guards are not discovered, too much behavior is allowed.

In conclusion, the solution approach that we propose is valuable, since it allows for discovering correct data-flows. And it can adequately deal with event logs with deviations, as well.

5.2 Validation Using a Real-life Event Log

Next, we evaluate the approach using a real-life log from a Dutch Financial Institute. The log is publicly available and can be downloaded using the following DOI `doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`. The original event log contains 13087 cases and 262200 events distributed over 36 activities ranging over a period from 1-Oct-2011 to 14-Mar-2012. The process represented in the event log is an application process for a personal loan or overdraft within a global financing organization. The application can be declined if it does not pass any checks. Applicants may be contacted for further information. The application is subjected to a final assessment upon which the application is either approved and activated, declined, or cancelled. Each case in the event log has an attribute $AMOUNT\_REQ$, which specifies the amount of loan/overdraft requested by a customer, and REG_DATE, which is the date when the request was submitted.

Using the insights reported in [2], we designed a model of the process control-flow, in form of Petri net (without data). Using the *control-flow conformance checking* plug-in available in ProM [1], we verified that this Petri net is a good representation of the control-flow, according to the real behaviors observed in the event log.

The control-flow model was used as input for our *Data-flow Discovery* plug-in. To have a more accurate validation, we split the event log in two sets. We randomly chose 6500 traces (i.e., 50% of traces the original log) to discover a DPN-net. Afterwards, we compute the data-flow conformance of the discovered DPN-net against the remaining traces to check the accuracy of the discovered model. From a performance viewpoint, the data-flow discovery was computed in few seconds, using 500 *MB* of memory, which shows that the

approach scales very well and is also applicable for quite large event logs. The average data-flow conformance was 0.85, from which we can derive the discovered DPN-net model is a good reflection of the reality.

The discovered data-flow allowed us to find recurring patterns in the management of the requests. The guard associated with activity *Accept Request* is $AMOUNT\_REQ \geq 3000 \wedge AMOUNT\_REQ < 49000$. It means that, usually, when a request is accepted, the amount requested is between 3000 and 49000 Euros. Similarly, activity *Cancel Request* is associated a guard $AMOUNT\_REQ < 3000 \vee AMOUNT\_REQ \geq 49000$, which means, when a request is cancelled, the amount requested is usually either small (< 3000) or large (≥ 49000). No guard has been discovered for activity *Decline Request*, which means that declining a request has nothing to do with the amount that is requested. From the analysis of the guards of other transitions in the model, we could only derive is that, when a request is provisionally accepted and, later, declined, the amount is generally less than 5350 Euros. Note that REG_DATE appears in no guards, which means that, to handle a request and determine the outcome, the procedure has not changed over time (i.e. no concept drift in the data-flow conditions).

To sum up, although the event log contains only two data attributes, we could derive an insightful process data-flow and, hence, interesting information about recurring behaviors. It is clear that the availability of more data attributes would have allowed for discovering a more detailed data flow. Therefore, we aim to make further investigations with richer logs.

6. CONCLUSION

In paper, we presented a novel technique for data-aware process mining. This is a neglected topic in literature as most process mining approaches focus on control-flow only. To the best of our knowledge, only the decision mining approach presented in [7] focuses on data-flow discovery. However, this approach has many limitations. For example, it cannot deal with event logs with deviating behavior and more complex control-flow constructs. To address these problems we first align log and model and only then apply decision-tree learning algorithms.

The proposed solution has been implemented in ProM and evaluated using both synthetic and real-life event logs. For example, this paper reports on experiences with an event log originating from Dutch financial institute. The experimental results show that the discovered data flows are accurate, even in presence of event logs with non-conforming traces (which is the case in most real-life event logs). Moreover, from a performance viewpoint, the data-flow can typically be mined in few seconds even for large event logs. Only if there are many different event attributes (say hundreds), the "curse of dimensionality" for traditional data mining approaches kicks in. Experiences with real-life logs have shown that the data-flow perspective can be discovered quickly and the results provide useful additional insights.

Nevertheless, there is still room for further improvements and extensions. In our current implementation, we can only discover guards that are conjunctions/disjunctions of expressions of the form variable-operator-constant (e.g. $x > 4$). Moreover, we only deal with the activity preconditions. It is also important to mine the post-conditions as they characterize the usual output of the activities' performance. We believe that we can find a solution for both problems by adapting techniques developed to detect invariants in software programs, e.g., Daikon [4] can discover such invariants but has not been applied in the context of process mining yet.

Acknowledgements. The authors thank J.C. Bose for his analysis of the real-life event log [2]. The research leading to these results has received funding from the European Community's Seventh Framework Program FP7/2007-2013 under grant agreement n° 257593.

7. REFERENCES

[1] A. Adriansyah, B. F. an Dongen, and W. M. P. van der Aalst. Conformance Checking Using Cost-Based Fitness Analysis. In *IEEE International Enterprise Distributed Object Computing Conference*, pages 55–64. IEEE Computer Society, 2011.

[2] R. P. J. C. Bose and W. M. P. van der Aalst. Process Mining Applied to the BPI Challenge 2012: Divide and Conquer While Discerning Resources. Technical Report BPM Center BPM-12-16, bpmcenter.org, 2012.

[3] M. de Leoni, W. M. P. van der Aalst, and B. F. van Dongen. Data- and Resource-Aware Conformance Checking of Business Processes. In *15th International Conference on Business Information Systems*, volume 117 of *LNBIP*, pages 48–59. Springer Verlag, 2012.

[4] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. The Daikon System for Dynamic Detection of Likely Invariants. *Science of Computer Programming*, 69(1–3):35–45, 2007.

[5] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. Byers. Big Data: The Next Frontier for Innovation, Competition, and Productivity. McKinsey Global Institute, 2011.

[6] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

[7] A. Rozinat and W. M. P. van der Aalst. Decision Mining in ProM. In *Proceedings of the 4th international conference on Business Process Management*, volume 4102 of *LNCS*, pages 420–425. Springer-Verlag, 2006.

[8] N. Sidorova, C. Stahl, and N. Trčka. Soundness Verification for Conceptual Workflow Nets With Data: Early Detection of Errors With the Most Precision Possible. *Information Systems*, 36(7):1026–1043, 2011.

[9] W. M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[10] H. M. W. Verbeek, J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. XES, XESame, and ProM 6. In *Information Systems Evolution*, volume 72 of *LNBIP*, pages 60–75, 2011.

[11] W. M. P. van der Aalst, et al. Process Mining Manifesto. In *Proceedings of Business Process Management Workshops 2011*, volume 99 of *LNBIP*. Springer Verlag, 2012.