Finding Process Variants in Event Logs

Alfredo Bolt [*], Wil M. P. van der Aalst, and Massimiliano de Leoni

Eindhoven University of Technology, The Netherlands

September 4, 2017

Abstract

The analysis of event data is particularly challenging when there is a lot of variability. Often, there are different variants of the same process, thus cluttering the overall representations of these processes, such as process models. Therefore, it is important to automatically detect process variants and support the analysis of individual variants and their comparison. Existing approaches can detect variants in very specific settings (e.g., changes of control-flow over time), or do not use statistical testing to decide whether a variant is relevant or not. In this paper, we introduce an unsupervised and generic technique to detect significant variants in event logs by applying existing, well-proven data mining techniques for recursive partitioning driven by conditional inference over event attributes. The discovered variants can be compared to spot differences in performance, resource utilization, etc. The approach has been fully implemented and is freely available as a ProM plugin. We evaluated our approach by successfully rediscovering deliberate performance issues injected to an artificial dataset. Finally, we validated our approach by applying it to a real-life event log obtained from a multinational Spanish telecommunications and broadband company, obtaining valuable insights directly from the event data.

1 Introduction

Every organization has to manage business processes (e.g., *order to cash*, *application to approval*). Business processes are what companies do whenever they deliver a service or product to a customer [1]. Business processes are not static: They have to adapt to constant environment changes (e.g., customer preferences, legal regulations, new competitors). Like any live species, companies (and their business processes) also evolve according to darwinian evolution: The best to adapt is the one that thrives. It is not uncommon for companies that the same business process has to adapt to different contexts simultaneously, which leads

[*]Corresponding Author: a.bolt@tue.nl

to variability in the behavior of such processes. Process variability is not only related to the *control-flow* perspective (e.g., a process may skip risk assessment steps for gold customers), but can also be related to other perspectives, such as *performance*. For example, if two branches of a company execute their processes in the same way (i.e., same control-flow) but there are huge performance differences between the branches, it is interesting to understand and explain such differences.

Organizations can record the execution of such business processes (including all of their variants) using Process-Aware Information Systems (PAIS) [2]. Such information systems record organization activities and can be used to extract event logs. Process mining is a relatively young research discipline that is concerned with discovering, monitoring, and improving real processes by extracting knowledge from event logs [3].

Most process *discovery* techniques (i.e., discovering process models from event logs) deal with process *control-flow* variability by combining all the observed *executions* (i.e., cases) of a process into a single process model. This results in what is known as *spaghetti models* (i.e., illegible process models). Also, process variability can be related to *performance* (e.g., throughput time of a process) or to other data attributes of events.

The process mining manifesto [4] proposes several challenges in process mining, one of them being: "(C2) Dealing with complex event logs having diverse characteristics" . One of the solutions related to this challenge is to partition complex event logs into sub-logs in order to reduce the variability and complexity. In such cases, the criteria used to partition the event log should be clear and usable for classifying future process executions.

In this paper, we consider a *process variant* as a group of executions of a process (i.e., a subset of cases of an event log) that share behavioral commonalities in terms of control-flow, performance and/or data attributes. A formal definition of a process variant is presented in Sec. 2.

As detailed in Section 6, several approaches for process variant detection have been proposed in literature [5, 6, 7, 8]. However, most approaches present at least one of the following drawbacks:

1. They focus on a single attributes (e.g, only control-flow changes over time)
2. Perform a brute-force analysis that retrieves many results that are not relevant.
3. Require many ad-hoc analysis steps to analyze the whole process.

In this paper, we propose a technique to detect relevant process variants in an event log using the control-flow, performance and context attributes of events in an interactive and exploratory way, where only relevant results are presented.

It is important to note that the type of analysis performed with our approach can also be achieved by combining other approaches and standard data mining techniques. However, such techniques require extensive and manual ad-hoc

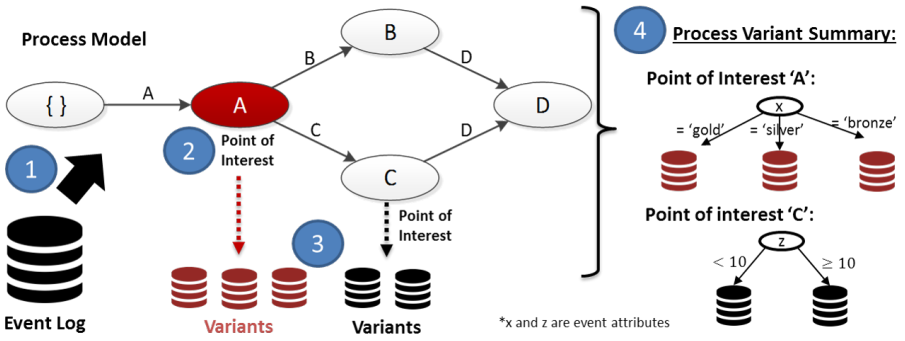


Figure 1: Overview and steps of our approach: (1) Given an event log, a process model is created. (2) Points of interest are identified in the process model. (3) For each point of interest, the set of cases that reach it is partitioned into process variants. (4) A summary of process variants is produced, where the splitting criteria and the resulting variants are shown for each point of interest.

parametrization and configuration to achieve the same results that our approach can obtain in a much easier way. We achieve this by leveraging on process models to identify points of interest in the process (e.g., a given *state* in the process). Then, the same variability analysis is automatically performed in each point of interest and the summarized results for the whole process are presented to the user as result.

Concretely, in this paper we use transition systems as process models because they have a low representational bias and because any process model (in any notation) that has executable semantics can be translated into a transition system.

Figure 1 illustrates the overview and steps of our approach. Note that our technique provides, for each point of interest, a clear partitioning criteria that allows one to easily identify and characterize process variants. The resulting process variants can be analyzed individually, but can also be compared using process comparison techniques such as [9]. Our approach has been implemented and evaluated in using both synthetic data (i.e., controlled experiment with a known ground truth) and a real case study.

The remainder of this paper is structured as follows: Section 2 introduces notations and concepts that will be used throughout the paper. Section 3 introduces our proposed approach and discusses how process variants can be obtained from event logs with the help of process models. Section 4 describes the freely available open-source software tool that implements our approach. Section 5 evaluates our approach through artificial experiments, and shows the usefulness of our approach by its application in a real case study. Section 6 discusses related work. Finally, Section 7 concludes the paper and discusses future work.

2 Preliminaries

In this section, we describe the basic concepts and ideas that will be used throughout this paper, namely event logs (and its components), process variants and transition systems.

2.1 Event Logs

Let $\mathcal{E}$ be the universe of events, $\mathcal{N}$ be the universe of attribute names and $\mathcal{V}$ be the universe of possible attribute values.

Events can have values for given attributes through the function $\# : \mathcal{N} \rightarrow (\mathcal{E} \nrightarrow \mathcal{V})$. For an attribute $a \in \mathcal{N}$, the partial function $\#(a) : \mathcal{E} \nrightarrow \mathcal{V}$, denoted as $\#_a$, can relate events to values of the attribute a. If an event e does not have a value for a, we denote this as $\#_a(e) = \perp$

The *domain* of the partial function $\#_a$ is denoted as $dom(\#_a)$ and it represents the subset of events of $\mathcal{E}$ that have a value in $\mathcal{V}$ for the attribute a. Commonly used event attributes are the `Event Class` (denoted as $\#_{class}$) which is used to define the type of event (e.g., activity name), and the `Case ID` (denoted as $\#_{case}$) that is used to group events into cases.

Let $\sigma \in \mathcal{E}^*$ be a trace. A trace records the execution of an *instance* of a process and is a finite sequence of events. The k^{th} event of a trace is denoted as $\sigma(k)$. The length of a trace is denoted as $|\sigma|$. The last event of a trace is denoted as $\sigma(|\sigma|)$. The prefix of a trace containing its first k events is defined by the function $pref^k \subseteq \mathcal{E}^* \rightarrow \mathcal{E}^*$. Note that $pref^0(\sigma) = \langle\rangle$. The set of all the prefixes of a trace σ is defined as $pref^\diamond(\sigma) = \bigcup_{k=0}^{|\sigma|} \{pref^k(\sigma)\}$.

For the sake of convenience, we introduce the partial function $tc : \mathcal{E}^* \nrightarrow \mathcal{P}(\mathcal{V})$ that maps traces to sets of event classes, where $\mathcal{P}$ represents the *powerset* function. For any trace σ, $tc(\sigma) = \{\#_{class}(e) | e \in \sigma\}$ defines the set of event classes that are related to the events in the trace.

Let $L \subseteq \mathcal{E}^*$ be an event log, i.e., a set of traces. Each event is unique and appears only once in one trace within the event log, i.e., for any event $e \in \mathcal{E} : \left|\left\{(\sigma, i) |\ \sigma \in L \ \wedge\ i \in \{1, ..., |\sigma|\} \ \wedge\ \sigma(i) = e\right\}\right| \leq 1$. The set of all the prefixes of traces of an event log L is defined as $P_L = \bigcup_{\sigma \in L} pref^\diamond(\sigma)$.[1] The set of all the events in an event log L is defined as $E_L = \bigcup_{\sigma \in L} \{e \in \sigma\}$. Table 1 shows an example of an event log represented in a table format. In this paper, we leverage on the same log augmentation techniques defined in [8] (i.e., trace manipulation operations) to extend events and cases with obtain additional attributes that are rarely recorded in event logs, such as the *elapsed time* of an event within its case, or the *next activity* to be executed in a case.

Now that events, traces and event logs are defined, we can formally introduce the concept or a process variant.

[1] For each trace σ, $\langle\rangle$ and σ are also considered to be a prefix of σ

Table 1: A fragment of an event log represented as a table: each row corresponds to an event (shown in the *event id* column) and each column corresponds to an event attribute. Events with the same *trace id* correspond to the same trace (i.e. process instance).

event id	**trace id**	**activity**	timestamp	next activity	elapsed time
1	1	A	28-12-2016 06:30	B	00:00
2	1	B	28-12-2016 06:45	C	00:15
3	1	C	28-12-2016 07:20	D	00:50
4	1	D	28-12-2016 08:05	-	01:35
5	2	A	29-12-2016 10:10	C	00:00
6	2	C	29-12-2016 10:30	B	00:20
7	2	B	29-12-2016 11:15	D	01:15
8	2	D	29-12-2016 12:10	-	02:00
9	3	A	30-12-2016 09:30	D	00:00
10	3	D	30-12-2016 09:40	-	00:10

Definition 1 (Process Variant) *Given an event log L, let $C = \{\#_{class}(e) | e \in E_L\}$ be the set of all the event classes observed in events of L. A process variant $V \subseteq L$ is a subset of traces of the event log such that $\exists_{c \in C} : c \in tc(\sigma)$ for all $\sigma \in V$.*

The traces in a process variant should have at least one common event class (e.g., activity). In this paper, we consider that traces without common activities relate to different processes, hence they cannot belong to the same variant.

The traces in a process variant also contain *similarities* in other event attributes. For example, a variant can be composed of traces in which an activity `B` is always executed by `John`, or by traces that have a *similar* (not necessarily equal) duration for an activity `D`.

Process variants also should have *differences* with respect to other process variants. For example, consider a process variant containing all the traces in which an activity `B` is executed by `John` and another process variant containing all the traces traces in which the same activity is executed by `Mary`. The traces in such process variants are similar to traces in the same variant, but are different to traces in other process variants.

2.2 Transition Systems

The first step in our approach (step 1 in Figure 1) is to create a process model from the event log.

Transition systems are very simple process models that are composed of *states* and of *transitions* between them. A transition is defined by an activity being executed, triggering the current state to move from a *source* to a *target* state. Prefixes of traces can be mapped to states and transitions using representation functions that define how these prefixes are interpreted.

The *state representation* function is defined as $r^s \in \mathcal{E}^* \rightarrow \mathcal{R}^s$ where $\mathcal{E}^*$ is the universe of possible trace prefixes and $\mathcal{R}^s$ is the set of possible representations of states. This function relates trace prefixes to states in a transition system.

The *activity representation* function is defined as $r^a \in \mathcal{E} \rightarrow \mathcal{R}^a$ where $\mathcal{E}$ is the set of possible events and $\mathcal{R}^a$ is the set of possible representations of activities (e.g. `activity name` or `event id`).

When using a state representation function r^s and an activity representation function r^a together, (prefixes of) traces can be related to transitions in a transition system. The activity and the source and target states of the transition can be identified using r^s and r^a. The set of all possible representations of transitions is defined as $\mathcal{R}^t \subseteq \mathcal{R}^s \times \mathcal{R}^a \times \mathcal{R}^s$. A transition $t \in \mathcal{R}^t$ is a triplet (s_1, a, s_2) where $s_1, s_2 \in \mathcal{R}^s$ are the source and target states and $a \in \mathcal{R}^a$ is the activity executed.

Definition 2 (Transition System) *Let $L \in \mathcal{E}^*$ be an event log, P_L the set of all the prefixes of traces of L, E_L the set of all the events of L, r^s a state representation function and r^a an activity representation function. A* transition system $TS^{(r^s,r^a,L)}$ *is defined as a triplet (S, A, T) where $S = \{s \in R^s \mid \exists_{\sigma \in P_L}\ s = r^s(\sigma)\}$ is the set of states, $A = \{a \in R^a \mid \exists_{e \in E_L}\ a = r^a(e)\}$ is the set of activities and $T = \{(s_1, a, s_2) \in S \times A \times S \mid \exists_{\sigma \in P_L \setminus \{\langle\rangle\}}\ s_1 = r^s(pref^{|\sigma|-1}(\sigma)) \wedge a = r^a(\sigma(|\sigma|)) \wedge s_2 = r^s(\sigma)\}$ is the set of valid transitions between states.*

Note that in this paper we only use the activity attribute to determine states and transitions. However, other event attributes can be used instead. For example, if a `resource` attribute is used in the state and activity representation functions, the resulting transition system will be a social network where states and transitions correspond to resources (e.g., employees) that execute events.

3 Finding Process Variants in Event Logs

There are many ways to partition an event log. The number of possible ways to partition an event log containing n traces is given by the n-th *Bell number*. As shown by Dobiński's formula [10]: $B_n = \frac{1}{e}\sum_{k=0}^{\infty}\frac{k^n}{k!}$, which is huge even for smaller n. However, most of these possible partitions are irrelevant for the analysis of a process because their resulting subgroups of traces do not correspond to process variants (see Def. 1). We can greatly reduce this number of partitions by leveraging on process models and event attributes.

The remainder of this section is organized as follows. Section 3.1 describes how points of interest can be defined in a transition system using event logs and discusses the guarantees that such points of interest provide (step 2 in Fig. 1). Section 3.2 describes how the cases that reach a given point of interest can be partitioned into *relevant* process variants by leveraging on event attributes (step 3 in Fig. 1). Finally, Section 3.3 describes how the summary of results is presented to the user (step 4 in Fig. 1).

3.1 Defining Points of Interest in a Transition System

The second step in our approach (step 2 in Figure 1) is to indeed identify points of interest in the process model. Given a transition system $TS^{(r^s,r^a,L)} =$

(S, A, T), we define $P \subseteq S \cup T$ as the set of *points of interest*.

Given an event log L and a transition system $TS^{(r^s, r^a, L)} = (S, A, T)$, every point of interest $p \in S \cup T$ can be related to a set of traces through the function $tr : (S \cup T) \rightarrow \mathcal{P}(L)$.

If a point of interest is a state, then the function tr is defined for s as $tr(s) = \{\sigma \in L | \exists_{\sigma' \in pref^{\diamond}(\sigma)} : r^s(\sigma') = s\}$. If a point of interest is a transition $t = (s_1, a, s_2)$, then the function tr is defined as $tr(s_1, a, s_2) = \{\sigma \in L | \exists_{\sigma' \in pref^{\diamond}(\sigma)} : s_1 = r^s(pref^{|\sigma'|-1}(\sigma')) \wedge a = r^a(\sigma'(|\sigma'|)) \wedge s_2 = r^s(\sigma')\}$. In other words, this function relates any state or transition with the traces that have a prefix that can be mapped into it.

As a consequence, for any point of interest p, all the traces $\sigma \in tr(p)$ are guaranteed to have at least one common event class (e.g., for a transition $t = (s_1, a, s_2)$, the event class is a , which is present in all the traces that are related to t). Trivially, the same applies for any subgroup of traces $g \subseteq tr(p)$

All the states and transitions of a transition system are initially considered as points of interest. However, not all points of interest are equally *relevant*. In this paper we define the relevance of a point of interest p in terms of the percentage of traces of the log L that reach it: $|tr(p)|/|L|$. For example, all the states and transitions that are reached by less than 5% of the traces in the event log can be considered as irrelevant because they rarely happen, and can be removed from P. In the implementation of our approach (see Sec. 4), the threshold of relevance is defined by the user. However, other alternative notions of relevance can be defined, and the users can arbitrarily select points of interest from $S \cup T$.

3.2 Finding Variants in a Point of Interest

The third step in our approach (step 3 in Figure 1) is to find process variants in the points of interest defined above. We do this by using Recursive Partitioning by Conditional Inference (RPCI) techniques [11] over event attributes. This technique is able to split a set of *instances* based on dependent and independent attributes (i.e., features, variables).

A trace cannot correspond directly to an instance because it may have several different values for the same attribute. For example, an `elapsed time` attribute can have different values for each event in the trace. In order to use RPCI (or any other partitioning or classification technique) each instance should have a single value for each of the attributes that will be used (or not have a value at all).

For this purpose, we choose the attribute values of a single event of a trace to represent it as an *instance*. The choice of which event should be used is related to the definition of points of interest discussed before.

Since we know that for a given point of interest p, any trace $\sigma \in tr(p)$ reaches it at some point, we could simply choose the last event of the smallest prefix of σ that reaches p. We define such set of events through the function $E : P \rightarrow (\mathcal{E}^* \rightarrow \mathcal{E})$, where for any point of interest p, $E(p)$ relates traces to a single events that

represents them. For example, if the point of interest is a state s, then a trace $\sigma \in tr(s)$ is related to the event $E(s)(\sigma) = \sigma\big(\min_{\sigma' \in pref^{\diamond}(\sigma)}(|\sigma'| \big| r^s(\sigma') = s)\big)$. Alternatively, we could choose the last event of the largest prefix that reaches p, or any other way that ensures that a trace is represented by a single event.

By doing this, we overcome the problem of traces reaching a point of interest more than once (e.g., in the presence of loops). This phenomenon can result in a trace being in two process variants simultaneously, like in [8] (we discuss this in detail in Sec. 6). For example, lets say that an activity `B` is executed first by `John` and then by `Mary` in the same trace. If process variants are defined based on who executes the activity `B` (i.e., `John` or `Mary`), then the trace would belong to both process variants simultaneously. This obviously causes problems when, for example, comparing process variants or aggregating performance.

Given a point of interest p, after the set of events that represent the traces in $tr(p)$ is defined as $E_p = \biguplus_{\sigma \in tr(p)} E(p)(\sigma)$, where every event $e \in E_p$ corresponds to an *instance*.

Partitioning Sets of Events:

In this paper, we rely on the event augmentation techniques proposed in [8]. Events are annotated with behavioral characteristics related to control-flow (e.g., the next activity after the event), performance (e.g., the elapsed time of an event within a trace) and other event attributes.

For each point of interest p, we aim to find the relevant partitions of its corresponding set of events (i.e., instances) E_p. For the remainder of this section, we will denote E_p as simply E. The initial number of possible partitions in a set of events E is given by $B_{|E|}$. The evaluation of all these possible partitions is still practically unfeasible. Therefore, we leverage on the available event attributes (defined in Section 2.1) to further reduce the number of partitions to be evaluated.

Let E be a set of events, and $A(E) = \{a \in \mathcal{N} | dom(\#_a) \cap E \neq \emptyset\}$ the set of event attributes associated with the events in E. For each attribute $a \in A(E)$, $n_a(E) = \{\#_a(e) | e \in dom(\#_a) \cap E\}$ defines the set of values of the attribute a over the set of events E.

We choose one of the event attributes $d \in A(E)$ as our *dependent attribute* (chosen by the user), for which we will reduce the variability by partitioning any combination of the other $A(E) \setminus \{d\}$ event attributes, namely *independent attributes*.

Our approach leverages on the Recursive Partitioning by Conditional Inference (RPCI) approach [11] to partition the set of events E. RPCI provides a unbiased selection and binary splitting mechanism by means of statistical tests of independence between the splitting attributes and the dependent attribute. It is based in the work of [12], which defines permutation-based independence tests using the asymptotic properties of linear statistics derived from arbitrary distributions. The specific independence tests used depend on the distributional characteristics of the dependent and independent attributes. The details of how RPCI works are out of the scope of this paper, and the reader is referred

to [11] for the specific mechanisms that RPCI uses to deal with different types of distributions and combinations of attributes.

In a nutshell, RCPI is described for a set of events E by the following steps:

1. Given a dependent attribute $d \in A(E)$, find the independent attribute $i \in A(E) \setminus \{d\}$ with the strongest significant correlation with d.

2. If such independent attribute i does not exist (i.e., no correlation is significant), stop the recursion. If it does exist, an optimal binary partition of the dependent attribute d is obtained, such that E is split into $E_1 \subset E$ and $E_2 = E \setminus E_1$.

3. Repeat step 1 and 2 for E_1 and E_2 recursively.

In step 1, the dependent attribute d is tested for independence w.r.t. each independent attribute in isolation. From the independent attributes for which the null hypothesis is rejected (i.e., they are significantly correlated to the dependent attribute), we select the one with the lowest p-value, which is obtained from the independence test. This lowest p-value indicates the strongest significant correlation between the dependent attribute and a independent attribute. Note that if no null hypothesis is rejected, then no splitting is done.

In step 2, if an independent attribute i is selected, an optimal binary partition is searched. A set of events E can be partitioned using the independent attribute i in different ways depending on the distribution of the attribute. For a numerical i, a single value is chosen, which acts as a "border" between the resulting subsets (e.g., the value 5 results in $E_1 = \{e \in E | \#_a(e) < 5\}$ and $E_2 = \{e \in E | \#_a(e) \geq 5\}$). For a categorical i, a set of values is chosen, as categories are not comparable (e.g., $E_1 = \{e \in E | \#_a(e) \in \{Gold, Silver\}\}$ and $E_2 = \{e \in E | \#_a(e) \notin \{Gold, Silver\}\}$). Note that RPCI provides several mechanisms to deal with missing values (see [11]).

RPCI uses the values of the independent attribute i to detect a binary partition of the set of events E into $E_1 \subset E$ and $E_2 = E \setminus E_1$ such that the difference between the value distributions of the dependent attribute d over the resulting subsets ($n_d(E_1)$ and $n_d(E_2)$) is maximized. This means that, in the worst case, $2^{|n_i(E)|-1} - 1$ possible partitions are evaluated.

Given the recursive nature of this approach, the exact total number of partitions to be evaluated depends on the characteristics and distributions of the attributes, and how many splitting attributes are correlated to the dependent attribute. If all the independent attributes are correlated to the dependent attribute and are used once, the total number of partitions to be evaluated is given by $\prod_{i \in A(E) \setminus \{d\}} (2^{|n_i(E)|-1} - 1)$, which is much smaller than $B_{|E|}$. Even if such independent attributes are used many times, the reduction of the number of possible partitions is still considerable. Note that this is an upper bound, because after every recursion the set of values of any independent attribute i will tend to decrease: $E_1 \subset E \Rightarrow n_i(E_1) \subseteq n_i(E)$.

In step 3, once an optimal partition is obtained, the whole process (steps 1 and 2) is repeated for the resulting partition E_1 and for E_2.

As a result of RPCI, a set of events E can be partitioned into a set of subsets $S_E = \{\lambda_1, ..., \lambda_n\}$. Every subset $\lambda \in S_E$ corresponds to a set of events. RPCI provides, for each $\lambda \in S_E$ a set of conditions that define it. These are presented to the user to help characterize a process variant.

Given the way that E was built and the nature of events, every event in λ is related to a different trace.

Therefore, $S_E = \{\lambda_1, ..., \lambda_n\}$ can be transformed into a set of process variants $V = \{v_1, ..., v_n\}$ of the same size where given an point of interest p, a variant v is defined as $v_i = \{\sigma \in tr(p) | \exists_{e \in \lambda_i} : e \in \sigma\}$ for any $i \in \{1, ..., n\}$. Therefore, the variants are guaranteed to be disjoint.

The approach discussed in this section is repeated for the sets of events related to each point of interest in the transition system defined by the user.

3.3 A Summary of Process Variants

According to RPCI, the traces related to a point of interest can be split into process variants or not, depending on the significance of the correlation between dependent and independent attributes. We present a summary of only the points of interest where process variants were found (see Sec. 4). For each point of interest, the splitting criteria obtained from RPCI is clearly presented, and the process variants are available to the user for other types of analysis.

4 Implementation

We have implemented our approach as a ProM [13] plugin named "Process Variant Finder" included in the *VariantFinder* package.[2] We use the R library ctree [14] to perform the statistical tests and partitioning of the sets of events. Therefore, a running instance of R is required. Figure 2 shows the interface of our tool.

The user needs to, first, specify the state and activity representation functions (shown as "TS Settings"). Then, the *dependent attribute*, and, later, the set of *independent attributes*. After several relevant process variants have been identified in states and transitions, a summary of points of interest with process variants is presented to the user (see panel 1 in Figure 2). When the user selects one of these points of interest, the tool shows the transition system and highlights the states or transitions that correspond to it (see panel 2 in Figure 2) and also describes the splitting criteria and value distributions of the dependent attribute for all the variants in a tree visualization (see panel 3 in Figure 2).

Our tool works with any combination of categorical (nominal and ordinal) and numeric (continuous and discrete) attributes. Hence, it is possible to use combinations of control-flow, time, resources, costs, customer details, etc.

[2]The reader can get this package via the ProM Package Manager.

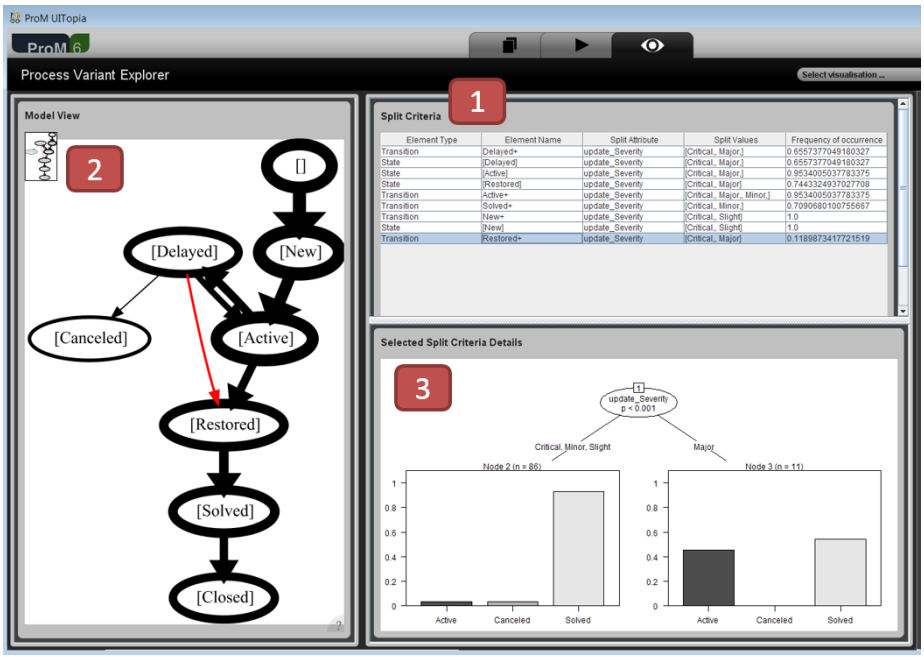


Figure 2: Screenshot of the "Process Variant Finder" tool in action. Panel (1) shows a list of the points of interest where process variants were detected. When a point of interest is selected, Panel (2) shows its location in the transition system (highlighted in red) and Panel (3) describes the details of the splitting criteria by visualizing it as a tree where each leaf of the tree is visualized as the distribution of the dependent attribute (for which we want to reduce the variability) for the subset of events that represent that variant.

5 Evaluation

Our approach was evaluated using two experiments. The first experiment is related to an artificial event log with artificially-induced performance variability for different resources participating in the process. For this event log, the ground truth (i.e., which resources are slower) is known, and our approach is used to re-discover the injected inefficiencies. The second experiment relates to a real case study in which we applied our approach to discover sources of variability and identify process variants among the event data of a Spanish broadband and telecommunication company.

5.1 Rediscovering Performance Variants in Artificial Event Data

This experiment refers to a generic *order-to-payment* process of a company that sells phones. In this process, an order is placed and an invoice is sent to the

customer. Then, the order is paid by the customer and the order is prepared and delivered to the customer. Finally, the payment is confirmed by the company. Order can also be cancelled at any time.

This artificially generated event log contains 10.000 cases and 127.216 events.[3] In average, each case contains 13 events. The process is executed by 31 different resources, of which three have a much slower performance (i.e., longer time to perform an activity) than others. These three slower resources are ironically named "Swift", "Speedy" and "Rush", and they perform different sets of activities. "Swift" participates only in the following activities: "send invoice", "confirm payment", "pay", "cancel order". "Speedy" participates only in "prepare delivery" and "place order". "Rush" only participates in "make delivery".

Since we know this ground truth, we want to evaluate whether our approach is able to detect which of those resources tend to be slower than the rest, and also pinpoint in which parts of the process does this performance difference occurs.

Concretely, we applied our approach into this event log by choosing the *time:duration* attribute (i.e., the time difference between an event and the previous one) as the dependent attribute, and *org:resource* (i.e., the resource that executes an event) as the independent attribute.

We were able to identify the all the deliberately injected differences in performance caused by the slow resources in specific activities. As an example, Figure 3 shows the performance differences in the "send invoice" activity (performed by "Swift" and other normal resources). Even though there are performance differences between normal resources (by design), the resource "Swift" is several orders of magnitude slower than all others. Note that the *Split Criteria* panel also shows all the other detected differences. The user has to click on any of them to visualize the detailed splitting conditions and highlight the parts of the transition system involved (in red).

5.2 Discovering Process Variants in Real Event Data: A Case Study

In this case study, we report on the results obtained by applying our approach to an event log provided by a Spanish broadband and telecommunications company. The provided event log refers to a *claim handling* process related to three services that this company provides, codenamed: Globalsim, SM2M and Jasper. In total, the event log contains 8296 cases (i.e., claims) processed between January 2015 and December 2016. Each claim has, on average 5 activities. Claims can have four severities: slight, minor, major and critical. In total, there are 40965 events in the event log.

Customers of the company *create* a claim which is *activated* by an employee of the company when he/she starts working on it. Claims with missing information can be *delayed*. If the service was interrupted, the first step is to work

[3] the event log is available in `https://www.dropbox.com/s/pa1j4062qf78lt5/event_log_orders_with_deliberate_performance_issues.rar?dl=0`.

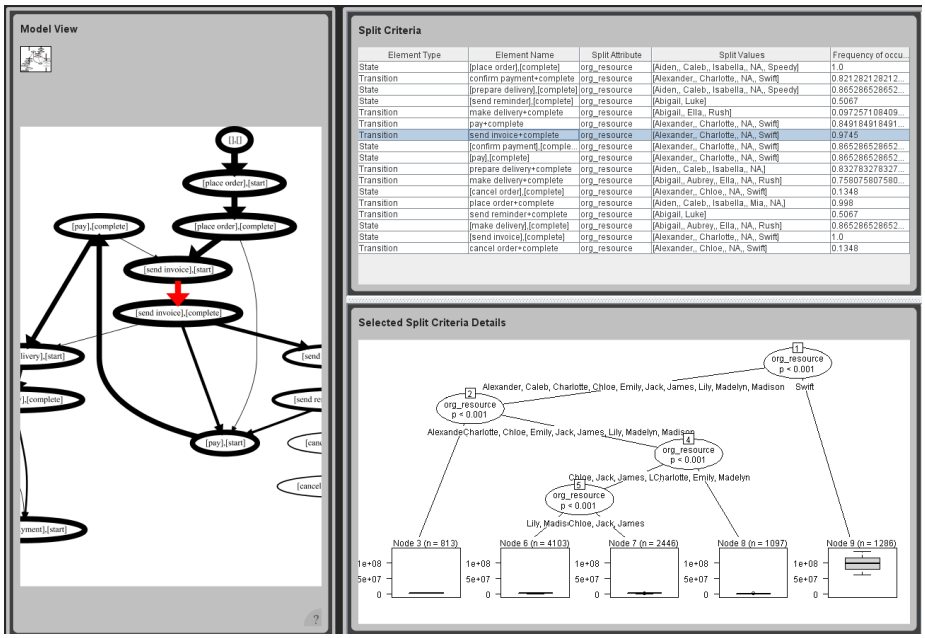


Figure 3: Example of process variants found. "Swift" is much slower that other resources than execute the activity "send invoice". The duration of an activity is measured in milliseconds. The dependent attribute is the duration of activities. The independent attribute is the resource.

on the *restoration* of the service. If there was no interruption, or the service has been restored, resources work on *solving* the problem that caused the claim. Once a problem has been solved, it is informed to the customer, which can *close* the claim. Customers can also *cancel* claims at any moment.

Figure 4 illustrates this process as a transition system, in which a state is defined by the last two activities executed in a prefix of a trace.

Note that activities such as "Active" can relate to different states depending on the previous activity executed in such trace. For example, the state "Active, New" represents claims that were created and then activated, whereas the state "Active, Delayed" refers to claims that were delayed at some point and then activated. Note that the labels of states are composed of the last activity in the prefix followed by the previous activity.

The company has established several Service Level Agreements (SLAs). An example is the *response SLA* (i.e., time until the claim becomes active). The response SLA has different values according to the service and severity. In a preprocessing step, we added a binary attribute "Check_resp" whose value can be (1) if the claim complied with its corresponding response SLA (depending on the service and severity) or (0) otherwise.

We used our approach to discover process variants in all states and transi-

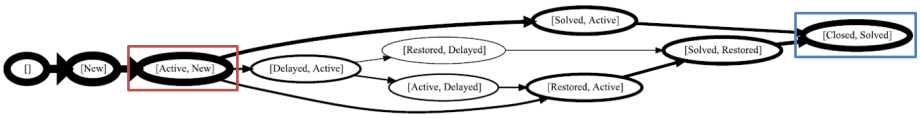


Figure 4: Transition system representing the claim handling process. States are defined by the last two activities executed in a trace prefix. Thickness represents frequency. States and transitions with a frequency of 5% of claims or less were filtered out. States "Active, New" and "Closed,Solved" are highlighted.

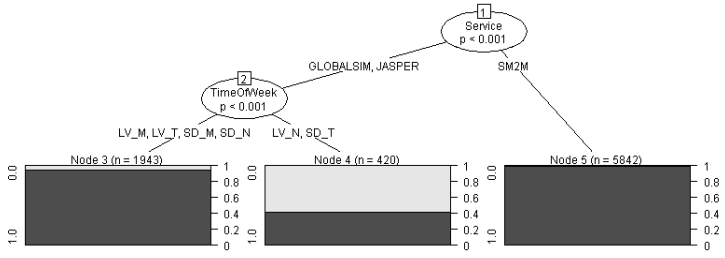
tions of the transition system shown in Figure 4. In every state and transition, we searched for relevant partitions using each available event attributes as a dependent attribute.

We were able to discover several partitions in many states and transitions. Because of space limitations, the remainder of this section discusses only a few partitions detected in the "Active, New" and the "Closed, Solved" states of the transition system presented in Figure 4 (highlighted in red and blue respectively).

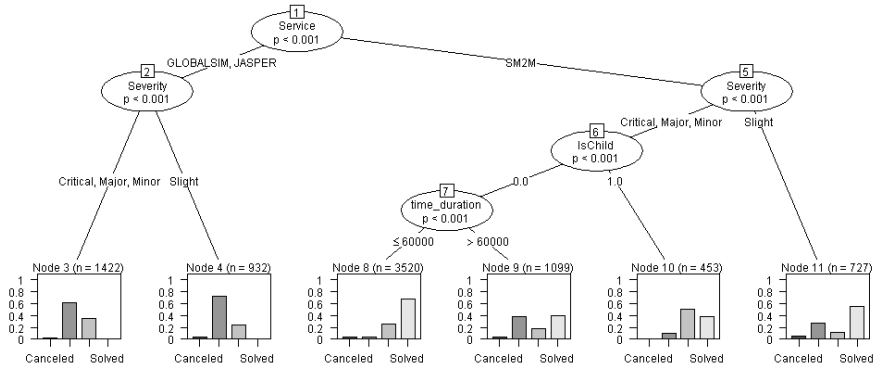
Figure 5 shows two relevant partitions detected in the "Active, New" state (i.e., when a claim has been created and then activated) where the splitting attributes and criteria are represented in a tree-fashion.

The first partition of the "Active, New" state (shown in Figure 5a) was obtained by selecting the "Check_resp" attribute (related to the *response SLA*) described above as the dependent attribute, and using all the other attributes as independent attributes. Therefore, the resulting variants of this partition can be considered as *context variants*. From the three resulting variants of this partition, we can observe that claims corresponding to the SM2M service (i.e., the right-hand branch in Figure 5a) have a very high response SLA score (i.e., almost all claims have "Check_resp" = 1). The remaining claims are subsequently split depending on the time of the week in which they were activated. Claims related to the Globalsim and Jasper services that were activated during weekday nights (i.e., LV_N) or weekend evenings (i.e., SD_T) are the ones that have the worst SLA compliance (i.e., the middle branch in Figure 5a), whereas claims from the same services that were activated in other times of the week had good compliance levels (i.e., the left-hand branch in Figure 5a). Domain experts from the company related this phenomenon to the fact that only claims related to the SM2M service are handled by the company's employees. The claims related to the other two services are handled by external providers. Also, the data exchange between the providers and the company is done in batches. This means that the external providers are not necessarily underperforming, and a real-time data exchange system would be necessary in order to correctly evaluate this SLA.

The second partition of the "Active, New" state (shown in Figure 5b) was obtained by selecting the "next activity" attribute (described in Section 2.1) as the dependent attribute, and using all the other attributes as independent



(a) Partition defining three *context* variants with differences in their compliance with their corresponding response SLA. The dependent attribute is the Response SLA compliance. All other attributes are considered as independent attributes.



(b) Partition defining six *Control-flow* variants with differences in the next activity to be executed. The labels in each bar chart are (from left to right): Canceled, Delayed, Restored, Solved. The dependent attribute is the next activity to be executed. All other attributes are considered as independent attributes.

Figure 5: Examples of partitions detected in the "Active, New" state.

attributes. Therefore, the resulting variants of this partition can be considered as *control-flow variants*. On the one hand, we can observe that the claims related to the Globalsim and Jasper services (i.e., first branch to the left in Figure 5b) have a higher tendency to get delayed than claims related to the SM2M service. This is accentuated in claims with a "Slight" severity. On the other hand, claims associated to the SM2M service (i.e., first branch to the right in Figure 5b) do not follow this pattern. From these claims, the ones that have a "Slight" severity are more likely to be immediately solved. Domain experts related this to the fact that slight severity claims usually do not involve an interruption of the service (thus, no restoration) and can be immediately solved.

More severe claims are divided whether they belong to a "parent claim" (1) or not (0). This is indicated by the "isChild" attribute (a claim can be subdivided into smaller claims, and such child claims naturally affect the SLAs of their parent claims). Claims that belong to a parent claim are more likely

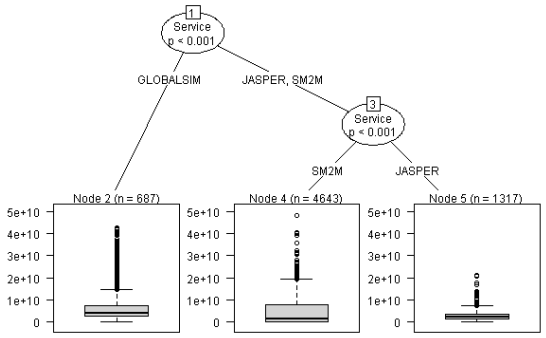


Figure 6: Performance variants detected in the "Closed, Solved" state. Elapsed time is measured in milliseconds and is presented as box plots for each variant. The dependent attribute is Elapsed Time. All other attributes are considered as independent attributes.

to become "restored". This make sense because bigger or more complex claims are more likely to have child claims, and are also more likely to have a service interruption. Claims that do not belong to any parent claim can be split into two main variants: the ones that take one minute or less to be activated and those that take more than one minute.[4] We can observe that the faster claims are more likely to be solved, but the slower ones get delayed more often. This could be related to "easier" claims being processed first.

Finally, Figure 6 shows performance variants detected in the "Close, Solved" state (i.e., when a claim has been solved and then closed) where the splitting attributes and criteria are represented in a tree-fashion. We can observe that claims related to the Globalsim service have the longest throughput time (i.e., the time between a claim is created until it is closed), followed by claims related to the Jasper service. Note that claims related to the S2M2 service are the fastest to be closed in average, but the time distribution is much more spread than claims related to the Jasper service. This can be observed on the position of quartiles in the box plots shown in Figure 6. Domain experts explained the fact that, in average, Globalsim claims took longer to be closed by the fact that there was a change in the management of this service in May 2016, which resulted, among other consequences, in the massive closeup of claims. Most of such claims were declared as "Solved" several months before, but were never officially closed. It is important to note that the company is only responsible for claims until they are solved, since the closing of a claim depends on the customer, hence it is not included in the company's SLAs.

Finally, we evaluated the impact of some of the detected partitions on the whole process. Note that this is not thoroughly discussed nor defined here since the strategies to relate variants and cases are out of the scope of this paper.

[4] 60000 milliseconds = 1 minute.

Figure 7: Transition system showing the *control-flow* comparison between claims of SM2M and claims of Globalsim and Jasper. The blue colors indicate that the frequency of occurrences of a state or transition is significantly higher in SM2M. Red colors indicates the opposite. Note that the structure of this transition system is the same as in Figure 4.

However, we explored a very simple scenario to illustrate the potential gains of combining the approach proposed in this paper with process comparison approaches.

We analyzed some of the control-flow variants defined by the partition shown in Figure 5b. This partition was detected in the "Active, New" state and is first characterized by the Service to which the claim belongs to (i.e., the independent attribute with the highest correlation to the dependent attribute). Note that "Service" is a case-level attribute, thus all the events in a case have the same value for this attribute. Also note that all the cases in the event log are associated to a service (i.e., there are no missing values). Accordingly, we split the event log into two sublogs: one with the claims that belong to the SM2M service, and another subset with the claims of the Globalsim and Jasper services. These two sublogs were compared in terms of control-flow frequency using the process comparison techniques described in [9]. The results of this comparison are illustrated in Figure 7. We can observe that, as expected, there is a significant difference in the control-flow frequency followed by claims of the different services. The state "Solved, Active" is colored blue, which means that the frequency of occurrence of such state is higher in SM2M (60% of the claims) than in Globalsim and Jasper (3% of claims). On the other hand, the state "Delayed, Active" is colored red, which means that the frequency of occurrence of such state is higher in Globalsim and Jasper (65% of the claims) than in SM2M (13% of claims). Note that there is a natural cascading effect in the red-colored states and transitions until claims are closed. In the last state of a claim ("Close, Solved" state), even if there are differences in the frequency of occurrence (75.8% for SM2M claims and 76.5% for Globalsim and Jasper claims), such differences are not significant. This is reflected by the fact that the state is not colored.

6 Related Work

Detecting process variants within an event log is not a new problem. Several authors have contributed to solving it from different perspectives. Even though these approaches may use different techniques, they all have a common goal: Split an event log into smaller event logs with less variability. Note that the con-

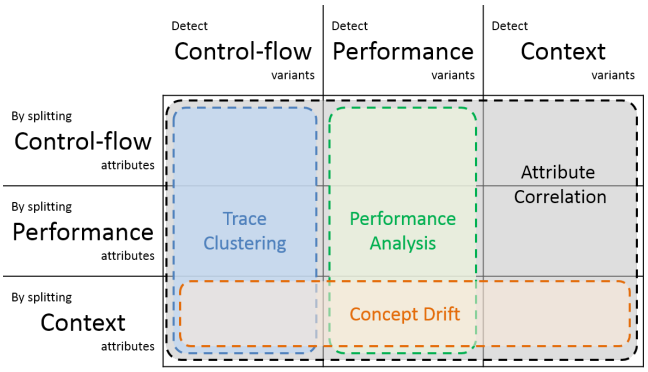


Figure 8: Categories of Related Works and their variant detection capabilities.

cept of variability may relate to control-flow, performance, customer types, etc. For example, trace clustering may be used to reduce the control-flow variability (i.e., group traces with similar activity execution sequences) or performance variability (i.e., group traces with a similar performance).

We grouped existing process variant detection techniques into four categories:(1) Concept drift detection, (2) Trace clustering, (3) Performance analysis, and (4) Attribute Correlation. The general variant detection capabilities of each category is illustrated in Figure 8.

Concept drift aims to detect when the relation between the input data and the dependent variable changes over time. An extensive survey on concept drift techniques in data mining is presented in [15]. Concept drift has been applied in process mining by several authors (see [16, 17, 5]). Concept drift techniques would allow one to find control-flow variability over time, which is only one of the possible combination of attributes that our approach can use.

Trace clustering techniques [6, 18, 19] usually focus on detecting different control-flow variants based on structural similarity. Notably, [20] uses trace clustering to detect process changes in time, extending the detection scope from purely control-flow to performance and context attributes as well. A downside of trace clustering approaches in general is that they are usually quite sensitive to parameterizations, such as the number of clusters to use. Also, the discovered clusters are often difficult to interpret from a business perspective.

Performance analysis techniques [7] focus on detecting changes in the performance of the process and characterizing them by using control-flow, performance and context attributes. This approach could be extended to also detect control-flow and context variants. However, it poses a scalability problem: all the combinations of attributes are evaluated in a brute-force fashion, which complicates its use in event logs with a high number of attributes.

Attribute Correlation techniques [8] are more general, as they aim to group cases depending on any event attributes. The approach proposed in this paper falls in this category. The only other approach that belongs in this category is the one presented in [8] that focuses on classifying specific selections of events

by building decision or regression trees with the attributes of such events that are later used to classify traces into process variants. Our approach is closely related to [8]. The similarities between our approach and [8] are:

- Behavioral features are annotated into events as extra attributes via trace manipulation functions.
- Selection of events are partitioned into subgroups using such event attributes.

The differences between these approaches are that in our approach:

- Process variants are always guaranteed to be disjoint (see Sec. 3.2). This is only guaranteed in [8] for the event filters EF_2 and EF_3, which select either the first or the last event of a trace respectively.
- The required configuration is simpler than in [8]: In our approach, given a transition system, the user only needs to select the dependent and independent attributes, and the same analysis is performed for all points of interest. In [8] an ad-hoc analysis use case needs to be manually designed for each point of interest.
- Our approach presents a summary of process variants in many points of the process. In [8], the result is a single decision tree describing variants in a single point of the process.
- Events are split using RPCI instead of Decision or Regression trees.

Arguably, if RPCI would be used in [8], then they could replicate the results provided by our approach in processes without loops (see the first difference), but it would require to manually configure several analysis use cases (see second and third differences).

Regarding the last difference, decision and regression trees have the advantage of clearly highlighting the splitting criteria, but they have two main drawbacks:

First, most decision and regression trees algorithms such as [21] calculate the correlation between the dependent attribute and all other attributes (using any correlation metric) and then choose the one that is most correlated to do the partitioning. Naturally, this leads to overfitting: as long as there is a positive information gain (regardless of its significance), there will be a partition. Decision tree algorithms usually incorporate pruning mechanisms to deal with this. However, decision trees have two drawbacks: they are usually biased towards selecting attributes with many split values, and they do not assess whether the correlation between the independent attribute and the dependent attribute is statistically significant. Several approaches like CHAID [22] and CART [23] incorporate statistical tests to define wether a partition is relevant, but they are limited to specific types of attribute distributions. CHAID does not work with numerical attributes (as it relies on CHI square tests). To overcome this, binning is applied, which has its own accuracy problems. CART does

not establish significant correlations between the independent attribute and the dependent attribute, so irrelevant partitions can be retrieved. In our approach, only attributes that are highly correlated to the dependent attribute are split.

Second, decision and regression trees aim to predict a value, whereas we aim to partition data. Hence, differences in minority classes would be overseen by approaches based on decision and regression trees. For example, consider an activity X which can be followed by A, B or C. The approach described in [8] would not find a difference between (A = 60%, B = 40%, C = 0%) and (A = 60%, B = 0%, C = 40%) because in both cases the majority class, hence, the actual predicted value, is A (60% of the cases). RPCI detects this because it focuses on the differences between the distributions instead of the expected (i.e., most probable) values.

Furthermore, to the best of our knowledge, only Recursive Partitioning by conditional Inference (RPCI) is able to perform unbiased selection of significantly correlated attributes with arbitrary distributions. This motivates the choice of using RPCI in this paper.

7 Conclusions

The problem of detecting process variants in event logs has been tackled by several authors in recent years. Many authors have successfully solved specific scenarios where the focus in on specific attributes, such as time. Some have even provided general solutions, but they fail to filter out irrelevant splits. To our knowledge, previous research has not tackled this problem in a general manner. This paper presents an approach that is able to detect relevant process variants in any available event attribute by splitting any other (combination of) event attributes. The approach has been implemented and is publicly available. We also demonstrated that the approach could rediscover designed performance issues in an artificially generated event log, where certain resources were related to poorer performance. We also were able to successfully identify points of process variability inside in a real-life event log and we were able to detect process variants without the use of domain knowledge, confirming such variability using process comparison techniques. Therefore, our approach provides a viable solution to process variant detection, even when no domain knowledge is available.

As future work we plan to work on analyzing the impact of detected variants in other parts of the process in an *a priori* fashion. Also, we aim to expand this work by ranking the detected relevant partitions (and their variants) based on partition quality criteria, such as those defined in [24].

References

[1] Dumas, M., La Rosa, M., Mendling, J., Reijers, H.: Fundamentals of Business Process Management. Springer-Verlag Berlin Heidelberg (2013)

[2] Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-Aware Information Systems: Bridging People and Software through Process Technology. John Wiley and Sons, Inc. (2005)

[3] van der Aalst, W.M.P.: Process Mining: Data Science in Action. 2nd edn. Springer-Verlag Berlin Heidelberg (2016)

[4] van der Aalst, W.M.P., et al: Process Mining Manifesto. In: Business Process Management Workshops of BPM 2011, Springer Berlin Heidelberg (2012) 169–194

[5] Maaradji, A., Dumas, M., La Rosa, M., Ostovar, A.: Fast and Accurate Business Process Drift Detection. In: Business Process Management: 13th International BPM Conference, Springer International Publishing (2015) 406–422

[6] Bose, R.P.J.C., van der Aalst, W.M.: Context Aware Trace Clustering: Towards Improving Process Mining Results. In: Proceedings of the 2009 SIAM International Conference on Data Mining. (2009) 401–412

[7] Hompes, B.F.A., Buijs, J.C.A.M., van der Aalst, W.M.P.: A Generic Framework for Context-Aware Process Performance Analysis. In: Proceedings of CoopIS 2016, Springer International Publishing (2016) 300–317

[8] de Leoni, M., van der Aalst, W.M., Dees, M.: A General Process Mining Framework for Correlating, Predicting and Clustering Dynamic Behavior based on Event Logs. Information Systems **56** (2016) 235 – 257

[9] Bolt, A., de Leoni, M., van der Aalst, W.M.P.: A visual approach to spot statistically-significant differences in event logs based on process metrics. In: Proceedings of 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016). Volume 9694., Springer International Publishing (2016) 151–166

[10] Dobiński, G.: Summirung der reihe $\sum \frac{n^m}{n!}$ für m = 1, 2, 3, 4, 5, ... Grunert's Archiv **61** (1877) 333–336

[11] Hothorn, T., Hornik, K., Zeileis, A.: Unbiased Recursive Partitioning: A Conditional Inference Framework. Journal of Computational and Graphical Statistics **15**(3) (2006) 651–674

[12] Strasser, H., Weber, C.: The Asymptotic Theory of Permutation Statistics. Mathematical Methods of Statistics **8** (1999) 220–250

[13] van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM Framework: A New Era in Process Mining Tool Support. In: Applications and Theory of Petri Nets. Volume 3536 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2005) 444–454

[14] Hothorn, T., Hornik, K., Zeileis, A.: ctree: Conditional Inference Trees. Cran. R_project (2015)

[15] Gama, J.a., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A Survey on Concept Drift Adaptation. ACM Comput. Surv. **46**(4) (2014) 44:1–44:37

[16] Seeliger, A., Nolle, T., Mühlhäuser, M.: Detecting Concept Drift in Processes Using Graph Metrics on Process Graphs. In: Proceedings of the 9th Conference on Subject-oriented Business Process Management, ACM (2017) 6:1–6:10

[17] Martjushev, J., Bose, R.P.J.C., van der Aalst, W.M.P.: Change Point Detection and Dealing with Gradual and Multi-order Dynamics in Process Mining. In: Proceedings of the 14th International Conference in Business Informatics Research (BIR), Springer International Publishing (2015) 161–178

[18] Weerdt, J.D., vanden Broucke, S., Vanthienen, J., Baesens, B.: Active Trace Clustering for Improved Process Discovery. IEEE Transactions on Knowledge and Data Engineering **25**(12) (2013) 2708–2720

[19] van Oirschot, Y.: Using Trace Clustering for Configurable Process Discovery Explained by Event Log Data. Master's thesis, Eindhoven University of Technology, Eindhoven, the Netherlands (2014)

[20] Hompes, B., Buijs, J.C.A.M., van der Aalst, W.M.P., Dixit, P., Buurman, J.: Detecting Change in Processes Using Comparative Trace Clustering. In: 5th International Symposium on data-driven process discovery and analysis (SIMPDA). (2015) 95–108

[21] Quinlan, J.R.: C4. 5: Programs for Machine Learning. Elsevier (2014)

[22] Kass, G.V.: An Exploratory Technique for Investigating Large Quantities of Categorical Data. Journal of the Royal Statistical Society. Series C (Applied Statistics) **29**(2) (1980) 119–127

[23] Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and Regression Trees. CRC press (1984)

[24] Montani, S., Leonardi, G.: Retrieval and clustering for supporting business process adjustment and analysis. Information Systems **40** (2014) 128–141