Finding Process Variants in Event Logs (Short Paper)

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

Eindhoven University of Technology, Eindhoven, The Netherlands
{a.bolt,w.m.p.v.d.aalst,m.d.leoni}@tue.nl

Abstract. The analysis of event data is particularly challenging when there is a lot of variability. 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 approach has been fully implemented and is freely available as a ProM plugin. 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

Organizations can record the execution of business processes supported by process aware information systems into event logs [1]. Process mining is a relatively young research discipline that is concerned with discovering, monitoring, and improving real processes by extracting knowledge from event logs [1]. Processes are affected by variability that 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) there could still be performance differences between the branches.

In this paper, we briefly discuss a novel technique to detect relevant *process variants* (i.e., groups of process executions) 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. The full version of this paper containing extended discussions, formalizations and results is presented in [2].

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 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

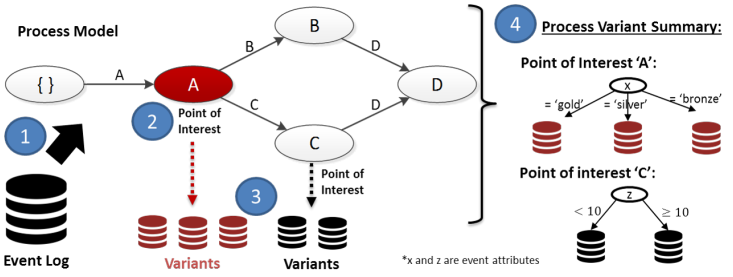

Fig. 1: Overview and steps of our approach: (1) Given an event log, a process model is created. (2) Points of interest are identified in the process model. (3) For each point of interest, the set of cases that reach it is partitioned into process variants. (4) A summary of process variants is produced,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

interest and the summarized results for the whole process are presented to the user as result. 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 [3]. Our approach has been implemented and evaluated in a real case study.

2 Preliminaries

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.

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. 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 $e \in \mathcal{E} : \left|\{(\sigma, i) |\ \sigma \in L\ \wedge\ i \in \{1, ..., |\sigma|\}\ \wedge\ \sigma(i) = e\}\right| \leq 1$.

A process variant $V \subseteq L$ is defined as a *set of traces*. The traces in a process variant also contain *similarities* in other event attributes. Process variants also should have *differences* with respect to other process variants. The traces in such process variants should be similar to traces in the same variant, but are different to traces in other process variants.

In this paper, we leverage on the same log augmentation techniques defined in [4] (i.e., trace manipulation operations) to extend events with obtain additional attributes, such as the *elapsed time* of an event within its case, or the *next activity* to be executed in a case.

The **first step** in our approach 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.

Definition 1 (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.*

3 Finding Process Variants in Event Logs

Defining Points of Interest in a Transition System (step 2): 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 $tr : (S \cup T) \to \mathcal{P}(L)$.

Finding Variants in a Point of Interest (step 3): We find process variants in the points of interest defined above by using Recursive Partitioning by Conditional Inference (RPCI) techniques [5] 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. 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.

Given a point of interest p, 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*. For each point of interest p, we aim to find the relevant partitions of its corresponding set of events (i.e., instances) E_p (denoted as simply E) based on their event attributes.

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 [5] 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. The details of how RPCI works are out of the scope of this paper, and the reader is referred to [5] 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.

As a result of RPCI, a set of events E can be partitioned into a set of subsets $S_E = \{\lambda_1, ..., \lambda_n\}$. 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. 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.

Given the way that E was built and the nature of events being unique, 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.

A Summary of Process Variants (step 4): 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. 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. A concrete visual representation of the summary is presented in [2]

4 Implementation & Case Study

We have implemented our approach as a ProM [6] plugin named "Process Variant Finder" included in the *VariantFinder* package.[1]

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

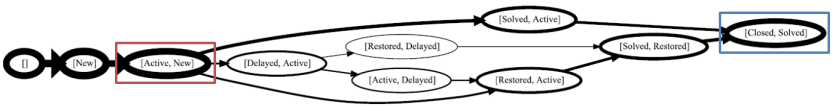

Fig. 2: Transition system representing the claim handling process. States are defined by the last two activities executed in a trace prefix. Thickness represents frequency. Points of interest (i.e., states and transitions) with a frequency of 5% of claims or less were filtered out. States "Active, New" and "Closed,Solved" are highlighted.

In this paper, we report on the results of a case study 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 correspond to traces of this process and 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 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 2 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. We used our approach to discover process variants in all states and transitions of the transition system shown in Figure 2. In every state and transition, we searched for process variants. Because of space limitations, the remainder of this section discusses only a few process variants detected in the "Active, New" and the "Closed, Solved" states of the transition system presented in Figure 2 (highlighted in red and blue respectively).

The variants detected in the "Active, New" state (shown in Figure 3) was obtained by selecting the `next activity` attribute (described in Section 2) as the dependent attribute, and using all the other attributes as independent 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 3) 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 3) do not follow this pattern. From these claims, the ones that have a "Slight" severity are

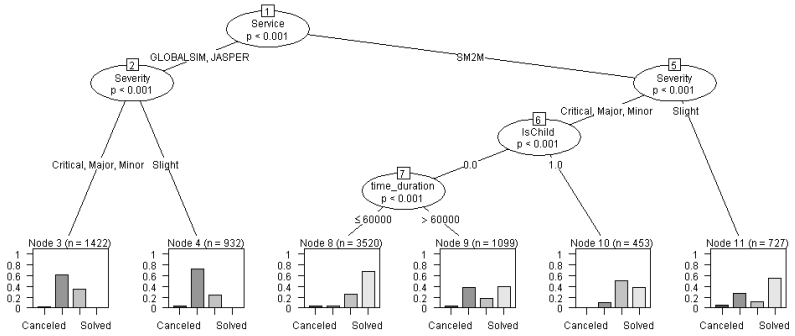

Fig. 3: Partition detected in the "Active, New" state, defining six *Control-flow* variants with differences in the "next activity" to be executed. The labels in each bar chart are (from left to right): Canceled, Delayed, Restored, Solved. The dependent attribute is the next activity to be executed. All other attributes are considered as independent attributes.

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). Claims that belong to a parent claim are more likely 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. 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.

Figure 4 shows *performance* process 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 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 4. 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

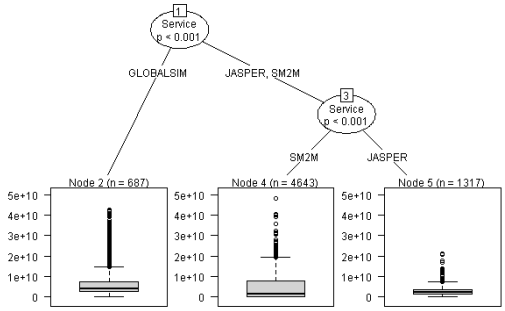

Fig. 4: Performance variants detected in the "Closed, Solved" state. Elapsed time is measured in milliseconds and is presented as box plots for each variant. The dependent attribute is Elapsed Time. All other attributes are considered as independent attributes.

are solved, since the closing of a claim depends on the customer, hence it is not included in the company's SLAs.

Naturally, the obtained process variants can be compared. We refer the reader to [2] for an comparative analysis of obtained process variants.

5 Related Work

We grouped existing process variant detection techniques into four categories: Concept drift detection, Trace clustering, Performance analysis, and Attribute Correlation. In this paper we only discuss Attribute Correlation approaches since it is the category in which our paper falls into. For a more detailed discussion of related work, we refer the reader to the full version of this paper [2].

Attribute Correlation techniques aim to group cases depending on any event attributes. The only other approach that belongs in this category (besides our approach) is [4], 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 [4].

The similarities between these approaches are: (1) Behavioral features are annotated into events as extra attributes via trace manipulation functions. (2) Selection of events are partitioned into subgroups using such event attributes.

The differences between these approaches are that, in our approach: (1) Process variants are always guaranteed to be disjoint (see Sec. 3). This is only guaranteed in [4] for the event filters EF_2 and EF_3, which select either the first or the last event of a trace respectively. (2) The required configuration is simpler than in [4]: 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 [4] an ad-hoc analysis use case needs to be manually designed for each point of interest. Therefore our approach presents

a summary of process variants in many points of the process. In [4], the result is a single decision tree describing variants in a single point of the process. (3) Events are split using RPCI instead of Decision or Regression trees.

Arguably, if RPCI would be used in [4], then they could replicate the results provided by our approach in processes without loops (see first difference), but it would require to manually configure several analysis use cases (see second differences).

6 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. This paper presents an approach that is able to detect relevant process variants in any available event attribute by automatically splitting any other (combination of) event attributes in many points of the process. The approach has been implemented and is publicly available. We 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.

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action. 2nd edn. Springer-Verlag Berlin Heidelberg (2016)
2. Bolt, A., van der Aalst, W.M.P., de Leoni, M.: Finding process variants in event logs. Research Report BPM-17-04, BPMCenter.org (2017)
3. 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 of Lecture Notes in Computer Science., Springer International Publishing (2016) 151–166
4. 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
5. Hothorn, T., Hornik, K., Zeileis, A.: Unbiased Recursive Partitioning: A Conditional Inference Framework. Journal of Computational and Graphical Statistics **15**(3) (2006) 651–674
6. 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