Model-based trace variant analysis of event logs

Mathilde Boltenhagen, Thomas Chatain, Josep Carmona

Model-Based Trace Variant Analysis of Event Logs

Mathilde Boltenhagen[a,*], Thomas Chatain[a], Josep Carmona[b]

[a] *Université Paris-Saclay, ENS Paris-Saclay, CNRS, LSV, Gif-sur-Yvette, France*
[b] *Universitat Politècnica de Catalunya, Barcelona, Spain*

Abstract

The comparison of trace variants of business processes opens the door for a fine-grained analysis of the distinctive features inherent in the executions of a process in an organization. The current approaches for trace variant analysis do not consider the situation where a process model is present, and therefore, it can guide the derivation of the trace variants by considering high-level structures present in the process model. In this paper we propose a fresh alternative to trace variant analysis, which proposes a generalized notion of trace variant that incorporates concurrency and iteration. This way, the analyst may be relieved from analysing trace variants that are essentially the same, if these aspects are disregarded. We propose a general algorithm for model based trace variant analysis which is grounded in encoding the problem into SAT, and a family of heuristic alternatives including a very light sampling technique that represents a good trade-off between quality of the trace variants identified, and the complexity of the analysis. All the techniques of the paper are implemented in two open-source tools, and experiments with publicly available benchmarks are reported.

Keywords:
Conformance Checking; Trace Variant Analysis; SAT; Process Mining.

*Corresponding author. *Phone number:* XXX
Email addresses: boltenhagen@lsv.fr (Mathilde Boltenhagen), chatain@lsv.fr (Thomas Chatain), jcarmona@cs.upc.edu (Josep Carmona)

1. Introduction

One of the key factors that explain the success of current process mining technology is the ability to extract from an event log, which can contain millions of traces, these representative traces that embed a particular way of executing a process. These representatives, known as *trace variants*, denote unique control-flow complete trajectories of a process model, i.e., each trace variant is a unique string formed by the concatenation of the sequence of tasks representing a process instance (or *case*). Thus, the analyst can then drill-down the analysis to be performed per trace variant, e.g., extracting insights arising from the other data attributes existing in the event log, so that one can see general rules that explain the traces represented by a certain variant (e.g., see [32]). Alternatively, other valuable information can be extracted by analyzing the differences between trace variants, a problem that has been considered recently both from the algorithmic perspective [41, 16, 2, 7, 40], but also for its application in certain domains [38, 37].

The event log partition by trace variants facilitates the analysis of real process executions, specially when a human is involved: by structuring a complex behavior into clusters, she can then work at a different abstraction level. In the ideal case, where few trace variants are identified, this process is clearly alleviated. However, when the process represented contains repetitions of certain activity fragments, the current approaches for trace variant analysis will be very sensitive to this phenomena. For instance, imagine the process of purchasing a certain item in an online shop; the fragment of the process that considers filling the form for the credit card number needs sometimes to be executed once, twice, three, ... several times. But all these different executions are not semantically different, they somehow, encode the buyer's ability in typing correctly her correct credit card number. A similar situation arises when certain fragments of a process can be executed concurrently; in this case, the process does not impose a particular ordering of the activities affected, although each ordering will be regarded as a different trace variant by current process mining tools.

In this paper we propose a radically different approach for trace variant analysis. We assume that a process model exists. This assumption is realistic in many contexts, e.g., in *Process-Aware Information Systems* (PAIS), process models are often available [20]. The quality of a process model can be first analyzed with *conformance checking* techniques [12], so that one can assume a certain validity and representativity of the process model used. The intuitive idea of our approach is to identify subnets of the process model that can serve as trace variant, so that different traces that are very distant when considered as words (like in the example of the credit card payment), will align similarly to the subnet representing a variant. Algorithmically, we compute the subnets corresponding the trace variants by encoding the problem as a SAT instance, following our previous work [14, 8]. Moreover, to enable the application of the technique for large problem instances, we present a sampling strategy. We provide some statistical guarantees that all traces sufficiently close to the model will be clustered after a certain number of trials.

The contributions of this paper are incorporated into two open-source tools, and the experiments provided both with synthetic and realistic benchmarks witness the ability in detecting meaningful trace variants corresponding to concurrent or repetitive process that can also be large and/or noisy.

1.1. A motivating example

Let us illustrate the main message of this paper with the process represented by the model shown in Figure 1. This is a synthetic process where, after some common activity is executed, 10 branches are possible, each one enabling at isolation concurrent and loop behavior. Once the executed branch terminates, a common activity is executed. Although this process has a very clear structure, it is very likely that an event log for this process models has a high number of trace variants. In fact, after simulating the model to obtain 500 traces, we obtained 411 trace variants in any of the commercial tools available. However, if our generalized notion of trace variants is used, which considers subnets of the process model instead of traces in the event log, 10 variants are considered

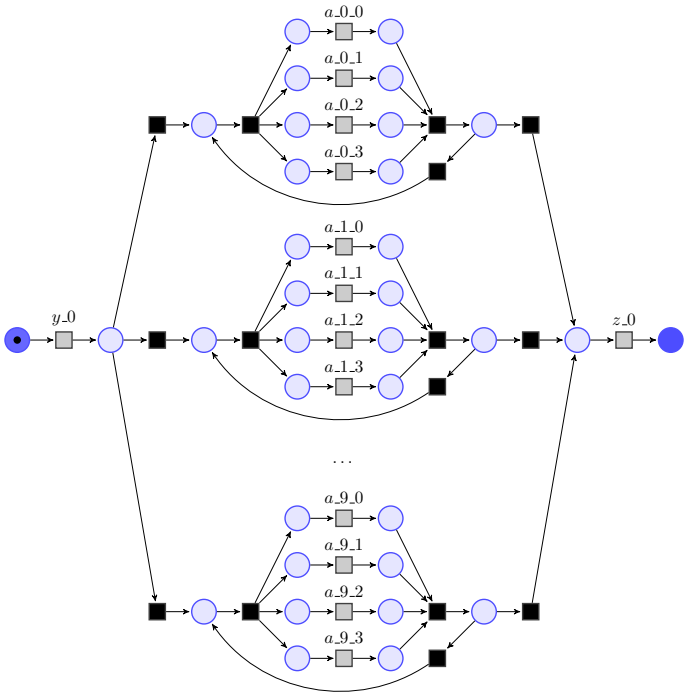


Figure 1: Petri Net of Motivation Example

(each for each subnet a_i, for $0 \leq i \leq 9$).

This paper is organized as follows: next section provides an overview of the current state-of-the-art approaches for trace variant analysis but also related topics. In section 3 the necessary preliminaries for this paper are provided. Then in section 4 we describe the different forms of trace variants proposed, whilst in section 6 it is formally shown how these can be computed. Strategies to fight the complexity of the problem are described in section 7. Section 8 reports experiments and section 9 concludes the paper.

2. Related work.

A complete and detailed review of trace variant analysis can be found in [40]. The work by van Beest et al. [41] relies on the product automaton of two event structures to distill all the behavioral differences between two process variants

$$\langle y\_0, a\_0\_0, a\_0\_1, a\_0\_2, a\_0\_3, z\_0 \rangle$$
$$\langle y\_0, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, z\_0 \rangle$$
$$\langle y\_0, a\_1\_2, a\_1\_0, a\_1\_3, a\_1\_1, z\_0 \rangle$$
$$\langle y\_0, a\_1\_3, a\_1\_1, a\_1\_2, a\_1\_0, z\_0 \rangle$$
$$\langle y\_0, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, z\_0 \rangle$$
$$\langle y\_0, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, z\_0 \rangle$$
$$\langle y\_0, a\_9\_0, a\_9\_3, a\_9\_1, a\_9\_2, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, z\_0 \rangle$$
$$\langle y\_0, a\_0\_1, a\_0\_1, a\_0\_2, a\_0\_2, a\_0\_3, z\_0 \rangle$$

Figure 2: Example of Artificial Log

from the respective event logs, and render these differences to end users via textual explanations. Cordes et al. [16] discover two process models and their differences are defined as the minimum number of operations that transform on model to the other. This work was extended in [2] to compare process variants using annotated transition systems. Pini et al. [31] contribute a visualization technique that compares two discovered process models in terms of performance data. The work in [47] proposes an extension of this work, by considering a normative process model alongside with event logs as inputs, and adding more data preparation facilities. This work is the only one in the literature that like us, considers the process model as input, but it uses the process model merely to compute alignments with the aim of computing performance information.

The works by Bolt et al. and Nguyen et al. are grounded on statistical significance. Bolt et al. [6] use an annotated transition system to highlight the differences between process variants. The highlighted parts only show different dominant behaviors that are statistically significant with respect to edge frequencies. This work was later extended in [7], by inducting decision trees for performance data among process variants. Nguyen et al. [30] encode process variants into *Perspective Graphs*. The comparison of perspective graphs results in a *Differential Graph*, which is a graph that contains common nodes and edges, and also nodes and edges that appear in one perspective graph only.

An approach very related to trace variant analysis is trace clustering. Several techniques have been proposed in the last decade for trace cluster-

ing [23, 22, 35, 9, 10, 45, 24, 18]. They can be partitioned into *vector space approaches* [23, 35, 18], *context aware approaches* [9, 10] and *model-based approaches* [22, 45, 24]. All the aforementioned clustering algorithms consider only the event log as input, and use different internal representations for producing the clusters. In contrast, in a recent paper [14], we presented a different view on clustering event log traces, by assuming that a process model exists. All the aforementioned techniques do not allow concurrency or loop behavior. Perhaps the closest work is [18], which extracts, together with the clusters, *Super-Instances* that are representatives of the discovered groups of traces. The method creates n-grams, i.e., sub-sequences of size n, which are considered as features in their vector-shape approach. They use a Principal Component Analysis (PCA) technique combined with the K-means algorithm, thus using the Euclidean distance. The Super-Instances are then mean of the input log, and can be used as representative, e.g., variants. In the experimental evaluation, we provide a detailed comparison with this closest work.

The use of an explicit characterization of concurrency has been considered recently in process discovery: the works in [34, 33] show how to improve the discovery of a process model by folding the initial unfolding that satisfies the independence relations given as inputs. In the area of conformance checking, the same phenomena has been observed: the work in [27] assumes traces are represented as partial order, thus allowing again an explicit characterization of concurrency in the problem formalization.

The trace clustering works also very similar to the one of this paper are [19, 28], where a transition system representing the event log is clustered, so that a set of simpler process models is generated. Tailored state-based properties that guarantee certain Petri net classes are used to guide the clustering, whereas in this work the computation of subnets is unrestricted. Our work is also related to [26] which clusters events and detects deviation. However, our work focuses on an existing model and the results may consider different directions like repairs while [26] gives a pre-processing of data.

Complexity of our works is related to [4] which demonstrates several methods

for distance between automata. Even for dynamic functions, the complexity have been proved PSPACE-complete and is even more complex for Petri nets which is formalism used in this paper.

3. Preliminaries

In this section, we introduce the main definitions required to understand the content of this paper from trace variants to model-based clustering.

3.1. Log Traces, Process Models, Alignment-based Clustering

Definition 1 (Log Trace). *A* log trace *over an alphabet of activity names Σ is a finite word $\sigma \in \Sigma^*$ that corresponds to an event sequence.*

A log is a collection of log traces. Fig. 2 shows an example of log traces of recorded behaviors.

Classically, some log traces appear many times in the log: this simply means that they correspond to frequent behavior. For this reason, one usually groups equivalent log traces together and consider them as several instances of the same log trace, called a *trace variant* [40]. In this paper we will propose richer notions of trace variants, obtained by clustering methods. The aim is to present to stakeholders a reduced number of trace variants, in order to make them easier to interpret.

Definition 2 (Trace Clustering). *Given a log L, a trace clustering over L is a partition over a (possibly proper) subset of the traces in L.*

Alignment-based trace clustering is a particular form of trace clustering: it relies on a model N of the observed system. We assume process models are described as Petri nets [29]. Formally:

Definition 3 (Process Model (Labeled Petri Net)). *A Process Model defined by a* labeled Petri net system *(or simply* Petri net*) is a tuple $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda \rangle$, where P is the set of places, T is the set of transitions (with $P \cap T = \emptyset$), $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, m_0 is*

the initial marking, m_f is the final marking, Σ is an alphabet of actions and $\lambda : T \to \Sigma \cup \{\tau\}$ labels every transition by an action or as silent.

The semantics of Petri nets is given in term of *firing sequences*. Given a node $x \in P \cup T$, we define its pre-set ${}^\bullet x \stackrel{\text{def}}{=} \{y \in P \cup T \mid (y, x) \in F\}$ and its post-set $x^\bullet \stackrel{\text{def}}{=} \{y \in P \cup T \mid (x, y) \in F\}$. A marking is an assignment of a non-negative integer to each place. A transition t is *enabled* in a marking m when all places in ${}^\bullet t$ are marked. When a transition t is enabled, it can *fire* by removing a token from each place in ${}^\bullet t$ and putting a token to each place in $t^\bullet$. A marking m' is *reachable* from m if there is a sequence of firings $\langle t_1 \dots t_n \rangle$ that transforms m into m', denoted by $m[t_1 \dots t_n \rangle m'$. The set of reachable markings from m_0 is denoted by $[m_0\rangle$. A Petri net is *k-bounded* if no marking in $[m_0\rangle$ assigns more than k tokens to any place. A Petri net is *safe* if it is 1-bounded. In this paper we assume safe Petri nets.

The idea of alignment-based trace clustering is to explicit the relation between log traces and full runs of N. Concretely, each cluster of log traces will be assigned a full run u of N, presented as the centroid of the cluster.

Definition 4 (Full Run). *A firing sequence $u = \langle t_1 \dots t_n \rangle$ such that $m_0[u\rangle m_f$ is called a* full run *of N. We denote by $Runs(N)$ the set of full runs of N.*

Given a full run $u = \langle t_1 \dots t_n \rangle \in Runs(N)$, the sequence of actions $\lambda(u) \stackrel{\text{def}}{=} \langle \lambda(t_1) \dots \lambda(t_n) \rangle$ is called a *(model) trace of N*. When the labeling function λ is injective, like in the model of Fig. 1, we sometimes identify the transition t with its label $\lambda(t)$. Then, full runs coincide with model traces. Examples for the model of Fig. 1 are $\langle y\_0, a\_0\_2, a\_0\_0, a\_0\_3, a\_0\_1, z\_0 \rangle$, $\langle y\_0, a\_1\_3, a\_1\_0, a\_1\_2, a\_1\_1, z\_0 \rangle$, $\langle y\_0, a\_9\_2, a\_9\_0, a\_9\_3, a\_9\_1, a\_9\_1, a\_9\_0, a\_9\_3, a\_9\_2, z\_0 \rangle$.

Definition 5 (Alignment-based Trace Clustering (ATC) [14]). *For a log L and a Petri net $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda \rangle$, an alignment-based trace clustering of L w.r.t. N is a tuple $\mathcal{C} = \langle \{u_1 \dots u_m\}, \chi \rangle$ where $u_1 \dots u_m$ ($m \in \mathbb{N}$) are full runs of N which serve as centroids for the clusters and*

Full Runs as Model-based Trace Variants	Traces
$\langle y\_0, a\_0\_0, a\_0\_1, a\_0\_2, a\_0\_3, z\_0\rangle$	$\langle y\_0, a\_0\_0, a\_0\_1, a\_0\_2, a\_0\_3, z\_0\rangle$ $\langle y\_0, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, z\_0\rangle$
$\langle y\_0, a\_1\_2, a\_1\_0, a\_1\_3, a\_1\_1, z\_0\rangle$	$\langle y\_0, a\_1\_2, a\_1\_0, a\_1\_3, a\_1\_1, z\_0\rangle$
$\langle y\_0, a\_1\_3, a\_1\_1, a\_1\_2, a\_1\_0, z\_0\rangle$	$\langle y\_0, a\_1\_3, a\_1\_1, a\_1\_2, a\_1\_0, z\_0\rangle$
$\langle y\_0, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, z\_0\rangle$	$\langle y\_0, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, z\_0\rangle$
$\langle y\_0, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, z\_0\rangle$	$\langle y\_0, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, z\_0\rangle$ $\langle y\_0, a\_9\_0, a\_9\_3, a\_9\_1, a\_9\_2, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, z\_0\rangle$
nc	$\langle y\_0, a\_0\_1, a\_0\_1, a\_0\_2, a\_0\_2, a\_0\_3, z\_0\rangle$

Table 1: Example of Alignment-based Trace Clustering (ATC) of log traces containing in log of Fig. 2 for a maximum distance to 2

$\chi : L \to \{\mathtt{nc}, u_1 \dots u_m\}$ *maps log traces either to the centroid of its cluster* $\chi(\sigma)$*, or to none of the clusters, denoted by* nc*.*

Each set $\chi^{-1}(u_i)$, for $i \in \{1 \dots m\}$, defines the cluster whose centroid is u_i. The set $\chi^{-1}(\mathtt{nc})$ contains the traces which are left non-clustered.

Tab. 1 shows a clustering of the traces of the log of Fig. 2 based on the model of Fig. 1.

Centroids of clustering is then a way to represent several log traces for a given distance. This novel idea relates to trace variants where variants are found in the model, introduced in this paper as *Model-based Trace Variants*.

Quality of an ATC

To get a good clustering as presented in Tab. 1, there exist several criteria that have been introduced in [8]. Obviously, a first criterion is the number of clustered traces: a good ATC should cluster as many traces as possible. A second criterion is the quality of the alignment between every log trace $\sigma \in L$ and the centroid of its cluster $u = \chi(\sigma)$ as we use them as model-based trace variants. This criterion can be quantified as:

$$\max_{\sigma \in L, \chi(\sigma) \neq \mathtt{nc}} dist(\sigma, \lambda(\chi(\sigma))) \tag{1}$$

where *dist* is a distance function between words. A good ATC minimizes this quantity.

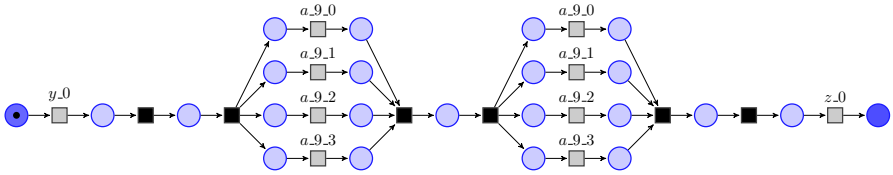


Figure 3: Example of a process of the Petri net in Fig. 1

In this paper, we use Levenshtein's edit distance, which is usually considered appropriate in Process Mining.

Definition 6 (Levenshtein's edit distance). *Levenshtein's edit distance* $dist_L(w1, w2)$ *between two words* w_1 *and* $w_2 \in \Sigma^*$ *is the minimal number of edits needed to transform* w_1 *to* w_2*. Editions can be substitutions to a letter by another one, deletions or additions of a letter in words.*

We will abuse notations, and write $dist_L(\sigma, u)$ for $dist_L(\sigma, \lambda(u))$, and $dist_L(u_1, u_2)$ for $dist_L(\lambda(u_1), \lambda(u_2))$. For example, the full run $\langle y\_0, a\_0\_0, a\_0\_1, a\_0\_2, z\_0\rangle$ and the log trace $\langle y\_0, a\_0\_0, a\_0\_1, a\_0\_2, a\_0\_3, z\_0\rangle$ have only one difference: the addition of a_0_3. They are at distance 1.

3.2. Partial-Order Semantics

In full runs of a process model, transition occurrences are totally ordered. However transitions can be handled in different orders for the same process in case of concurrency. In the model of Fig.2 traces $\langle y\_0, a\_1\_3, a\_1\_1, a\_1\_2, a\_1\_0, z\_0\rangle$ and $\langle y\_0, a\_1\_2, a\_1\_1, a\_1\_3, a\_1\_0, z\_0\rangle$ follow the same process but differ by the order of the transitions.

They can however be seen as two linearizations of a common representation based on partial-order runs which represents a *process*.

Definition 7 (Partial-Order Representation of Runs: Process [21]). *A (non-branching) process* $\mathcal{P}$ *of a Petri Net* $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda\rangle$ *is a tuple* $\mathcal{P} = \langle B, E, G, B_0, B_f, h\rangle$ *where:*

- (B, E, G, B_0, B_f) *is a non-branching, i.e., each place has a unique incoming and outgoing arc, finite, acyclic Petri Net,*

- *h is a function that relates $\mathcal{P}$ to N by mapping the elements of B (called conditions) to the places P of N and the elements of E (called events) to the transitions T of N. This function h is required to map bijectively pre- and post-sets of events to pre- and post-sets of transitions, which ensures that $\mathcal{P}$ simulates N.*

See e.g. [21] for a more detailed definition.

We write $Runs(\mathcal{P})$ for the set of full runs of the process $\mathcal{P}$. For every full run $\langle e_1 \dots e_n \rangle$ of a process $\mathcal{P}$ of a Petri net N, the sequence $u \stackrel{\text{def}}{=} \langle h(e_1) \dots h(e_n) \rangle \in T^*$ is called a *linearization* of $\mathcal{P}$. Every linearization of $\mathcal{P}$ is a full run of N.

Fig. 3 shows a process of the Petri net in Fig. 1. This process represents both full runs $\langle y\_0, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, z\_0 \rangle$ and $\langle y\_0, a\_9\_0, a\_9\_3, a\_9\_1, a\_9\_2, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, z\_0 \rangle$ which differ only by the order of concurrent transitions. Similarly, $\langle y\_0, a\_1\_3, a\_1\_1, a\_1\_2, a\_1\_0, z\_0 \rangle$ and $\langle y\_0, a\_1\_2, a\_1\_1, a\_1\_3, a\_1\_0, z\_0 \rangle$ are also represented by a single process. Hence, processes appear as a good representation for trace variants aware of concurrency.

4. Generalized Model-based Trace Variants

As presented above, full runs of process models can be extracted to represent log traces by using Alignment-based Trace Clustering (ATC). In this section, we want to go further and allow one to extract model-based trace variants aware of concurrent and loop behaviors contained in the process model.

4.1. Concurrency Aware Variants

By using processes as model-based trace variants, traces that only differ by the order of concurrent activities can be grouped and represented by the same variant. The process presented in Fig. 3 is then a unique variant for the log traces $\langle y\_0, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, z\_0 \rangle$ and $\langle y\_0, a\_9\_0, a\_9\_3, a\_9\_1, a\_9\_2, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, z\_0 \rangle$. Similarly to ATC,

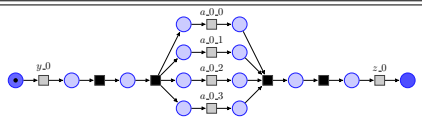
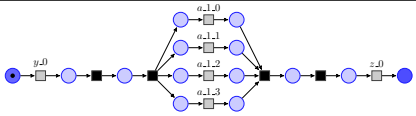
Processes as Model-based Trace Variants	Traces
	$\langle y\_0, a\_0\_0, a\_0\_1, a\_0\_2, a\_0\_3, z\_0\rangle$ $\langle y\_0, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, z\_0\rangle$
	$\langle y\_0, a\_1\_2, a\_1\_0, a\_1\_3, a\_1\_1, z\_0\rangle$ $\langle y\_0, a\_1\_3, a\_1\_1, a\_1\_2, a\_1\_0, z\_0\rangle$
process of Figure 3	$\langle y\_0, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, z\_0\rangle$ $\langle y\_0, a\_9\_0, a\_9\_3, a\_9\_1, a\_9\_2, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, z\_0\rangle$
process similar to Figure 3, not represented	$\langle y\_0, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, z\_0\rangle$
nc	$\langle y\_0, a\_0\_1, a\_0\_1, a\_0\_2, a\_0\_2, a\_0\_3, z\_0\rangle$

Table 2: Example of APOTC of traces contained in the log of Fig. 2. Maximum distance between clustered traces and their centroids is 0.

those process representatives can be found with a clustering method which gives process centroids that serve as model-based trace variants.

Definition 8 (Alignment and Partial Order based Trace Clustering (APOTC)). *An alignment and partial order based trace clustering (APOTC) of a log L w.r.t. a Petri net $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda\rangle$, is a tuple $\mathcal{C} = \langle\{\mathcal{P}_1 \dots \mathcal{P}_m\}, \chi\rangle$ where $\mathcal{P}_1 \dots \mathcal{P}_m$ ($m \in \mathbb{N}$) are processes of N which serve as centroids for the clusters and $\chi : L \to \{\mathtt{nc}, \mathcal{P}_1 \dots \mathcal{P}_m\}$ maps log traces either to the centroid of its cluster $\chi(\sigma)$, or to none of the clusters, denoted by* nc.

Like in the ATC method, the distance between traces and model-based trace variants, is minimized. The distance between a trace σ and a process $\mathcal{P}$ is defined by $dist_{\mathcal{P}}(\sigma, \mathcal{P}) \stackrel{\text{def}}{=} \min_{u \in Runs(\mathcal{P})} dist_L(\sigma, u)$. Then distances of APOTC minimize the following.

$$\max_{\sigma \in L, \chi(\sigma) \neq \mathtt{nc}} dist_{\mathcal{P}}(\sigma, \chi(\sigma)) \tag{2}$$

Tab. 2 shows an APOTC of log traces of Fig. 2 based on the model of Fig. 1. By using processes as model-based trace variants, one merges clusters which were separated in the ATC of Tab. 1 and identify richer variants.

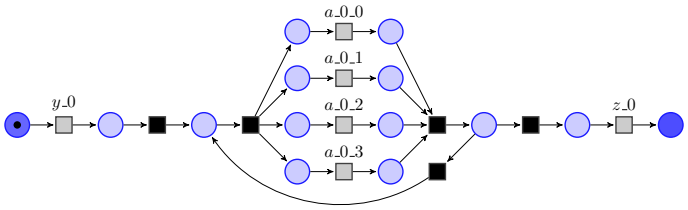


Figure 4: A subnet of the Petri Net in Fig. 1.

4.2. Concurrency and Loop Aware Variants

APOTC separates process arising from traces corresponding to different number of loop iterations, e.g., the traces $\langle y\_0, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, z\_0\rangle$ and $\langle y\_0, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, z\_0\rangle$. The issue is due of the fixed size of runs of processes, which do not represent loops. To overcome this limitation, we introduce subnets of models.

Definition 9 (Subnet of Petri net). *A* subnet *of a Petri net* $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda\rangle$ *is a Petri net* $\langle P, T', F_{|T'}, m_0, m_f, \Sigma_{|T'}, \lambda\rangle$ *with* $T' \subseteq T$, *and* $F_{T'} \stackrel{\text{def}}{=} F \cap (P \times T' \cup T' \times P)$.

Fig. 4 presents a subnet of the model of Fig. 1. Observe that our definition of subnets, based on selecting transitions, restricts the semantics of the net and cannot produce new behaviors. Formally:

Lemma 1. *Every full run (resp. process) of a subnet of a Petri net* N, *is a full run (resp. process) of* N.

We now formalize AMSTC, which consider subnets as centroids:

Definition 10 (Alignment and Model Subnet-based Trace Clustering (AMSTC)). *For a log* L *and a Petri net* $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda\rangle$, *an alignment and model subnet trace clustering, of* L *w.r.t.* N *is a tuple* $\mathcal{C} = \langle\{\mathcal{N}_1 \ldots \mathcal{N}_m\}, \chi\rangle$ *where* $\mathcal{N}_1 \ldots \mathcal{N}_m$ *are subnets of* N *which serve as centroids for the clusters and* $\chi : L \to \{\texttt{nc}, \mathcal{N}_1 \ldots \mathcal{N}_m\}$ *maps log traces either to the centroid of its cluster* $\chi(\sigma)$, *or to none of the clusters, denoted by* `nc`.

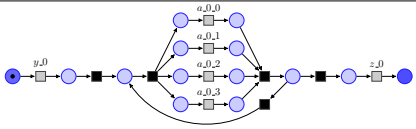
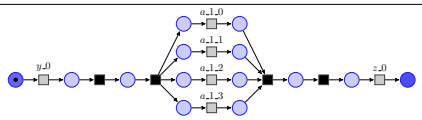
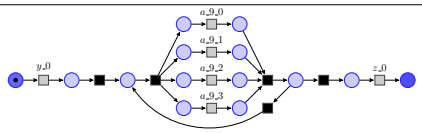
Subnets as Model-based Trace Variants	Traces
	$\langle y\_0, a\_0\_0, a\_0\_1, a\_0\_2, a\_0\_3, z\_0\rangle$ $\langle y\_0, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, z\_0\rangle$ $\langle y\_0, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, a\_0\_1, a\_0\_0, a\_0\_2, a\_0\_3, z\_0\rangle$
	$\langle y\_0, a\_1\_2, a\_1\_0, a\_1\_3, a\_1\_1, z\_0\rangle$ $\langle y\_0, a\_1\_3, a\_1\_1, a\_1\_2, a\_1\_0, z\_0\rangle$
	$\langle y\_0, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, z\_0\rangle$ $\langle y\_0, a\_9\_0, a\_9\_3, a\_9\_1, a\_9\_2, a\_9\_0, a\_9\_1, a\_9\_3, a\_9\_2, z\_0\rangle$
nc	$\langle y\_0, a\_0\_1, a\_0\_1, a\_0\_2, a\_0\_2, a\_0\_3, z\_0\rangle$

Table 3: Example of Alignment and Model Subnet-based Trace Clustering (AMSTC) of traces containing in log of Fig. 2 for a maximum distance to 0

Distances between a clustered trace σ and its centroid $\mathcal{N}$, used as model-based trace variants, is defined as $dist_{\mathcal{N}}(\sigma, \mathcal{N}) \stackrel{\text{def}}{=} \min_{u \in Runs(\mathcal{N})} dist_L(\sigma, u)$. Computing this distance corresponds to align traces to model. Alignment criterion of an AMSTC minimizes equation (3).

$$\max_{\sigma \in L, \chi(\sigma) \neq \mathtt{nc}} dist_{\mathcal{N}}(\sigma, \chi(\sigma)) \tag{3}$$

Tab. 3 shows an AMSTC of log traces of Fig. 2 based on the model of Fig. 1. The novel variant forms allow traces that only differ on concurrency and loops according to the model to be grouped together. Business analysis is sometimes too complicated due to the number of variants. With this approach, we help one to reduce the number of variants the stakeholder has to understand.

5. Complexity of Alignment-based Trace Clusterings

The search of model-based trace variants depends on the alignment-based trace clustering algorithms. Due to the constraints on the quality, i.e., *dist*, $|\mathtt{nc}|$, m ..., the complexity lies already in the existence of a clustering, and the specification of many quality constraints does not change the complexity.

For a non-empty log L and a model N, there exists an ATC $\mathcal{C}$ of L w.r.t. N having at least one clustered trace, iff N has a full run. Indeed, when no constraint is given about the quality criteria *dist*, m ..., any full run of N can serve as centroid, and any log trace can be assigned to any cluster. The same holds for APOTC, where centroids are processes of N, since N has a process iff N has a full run; it holds again for AMSTC, taking into account the constraint that any subnet used as centroid should have a full run.

Now, deciding if a model has a full run u, corresponds to checking reachability of the final marking. The problem of reachability in Petri nets is known to be decidable, but non-elementary [17], and still PSPACE-complete for safe Petri nets. But the complexity trivially drops to NP-complete[1] if a bound l is given (with l an integer coded in unary) on the length of u.

In practice, relevant clusterings will not use very long full runs (or processes for APOTC) as centroids. Typically, a bound l on the length of the full runs can be assumed, for instance 2 times the length of the longer log trace. Let us call l-bounded a trace clustering satisfying this constraint.

Theorem 1. *The problem of deciding, for a log L, a model N, an integer bound l, integers $m_{\max}$, $d_{\max}$, and a rational number $\mathtt{nc}_{\max}$, the existence of a l-bounded ATC (respectively APOTC, AMSTC) $\mathcal{C}$ of L w.r.t. N, having at most $m_{\max}$ clusters, a distance between the traces and their centroids lower or equal to $d_{\max}$ and a maximal number $\mathtt{nc}_{\max}$ of unclustered traces is NP-complete.*

Proof. As observed earlier, the problem is NP-hard even with the only constraint that at least one trace is clustered. It remains to show that it is in NP: indeed, if there exists a (l-bounded) clustering, there exists one with no more that $|L|$ clusters (forgetting empty clusters cannot weaken the quality criteria); and, by assumption, the size of centroids (defined as $|\sigma|$ for ATC, $|\mathcal{P}|$ for APOTC, number of transitions in the subnet for AMSTC) is bounded by l.

[1] NP-hardness can be obtained by reduction from the problem of reachability in a safe acyclic Petri net, known to be NP-complete [36, 15].

So, it is possible to guess a clustering $\mathcal{C}$ in polynomial time. For APOTC and AMSTC, one can also guess in P time the full run $u \in \mathit{Runs}(\chi(\sigma))$, for every clustered trace σ, which will achieve the $\mathit{dist}(\sigma, \chi(\sigma))$. Now, checking that $\mathcal{C}$ satisfies the constraints, only requires to compute Levenshtein's edit distances and minima over sets of polynomial size. This can be done in P time. □

6. SAT Encoding of Model-Based Trace Variants

In this section we show how we approach AMSTC definition with a SAT encoding to get model-based trace variants.

We encode the existence of an AMSTC of m clusters, to get m model-based trace variants, for net N and log L with maximal distance d, as a SAT formula of the equation (4). Formally, we check the existence of $\mathcal{C} = \langle\{\mathcal{N}_1 \ldots \mathcal{N}_m\}, \chi\rangle$ such as :

$$\bigwedge_{\sigma \in L} \chi(\sigma) \neq nc \Rightarrow \mathit{dist}_{\mathcal{N}}(\sigma, \chi(\sigma)) \leq d \tag{4}$$

and optimizes several quality criteria like the number of clustered traces. We develop them in Subsection 6.4.

Distance $\mathit{dist}_{\mathcal{N}}$ in (4) is the minimal Levenshtein distance between a trace and a model-based trace variant as subnet. This induces equation (5).

$$\bigwedge_{\sigma \in L} \chi(\sigma) \neq nc \Rightarrow \min_{u_\sigma \in \mathit{Runs}(\chi(\sigma))} \mathit{dist}_L(\sigma, u_\sigma) \leq d \tag{5}$$

We can now express Levenshtein distance with Hamming distance (dist_H) and skip actions ($\gg_m$ for model moves and $\gg_l$ for log moves), as :

$$\mathit{dist}_L(\sigma, u) = \min_{\substack{\sigma^{\gg_m}, u_\sigma^{\gg_l} \\ |\sigma^{\gg_m}| = |u_\sigma^{\gg_l}|}} \mathit{dist}_H(\sigma^{\gg_m}, u_\sigma^{\gg_l})$$

where $\sigma^{\gg_m}$ ranges over all the words of $(\Sigma \cup \{\gg_m\})^*$ obtained by inserting letters $\gg_m$ in σ and respectively $u_\sigma^{\gg_l}$ ranges over all the words of $(\Sigma \cup \{\gg_l\})^*$. Letters $\gg_m$ and $\gg_l$ represent deletions and insertions.

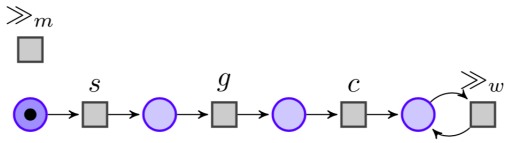


Figure 5: The net $((N_\sigma)^{\gg_m})$ used to produce the words $\sigma^{\gg_m}$ for a log trace $\sigma = \langle s, g, c\rangle$.

Technically, we obtain the words $\sigma^{\gg_m}$ for a log trace σ as runs of a Petri net $((N_\sigma)^{\gg_m})$, built from σ as illustrated in Fig. 5, which contains a transition $\gg_m$ for model moves. Similarly, the words $u_\sigma^{\gg_l}$ are obtained as runs of a modified subnet, $\chi(\sigma)^{\gg_l}$, which has an additional transition $\gg_l$ for log moves.

Hence, finding $\mathcal{C} = \langle\{\mathcal{N}_1 \ldots \mathcal{N}_m\}, \chi\rangle$ satisfying (5) amounts to finding $\mathcal{C} = \langle\{\mathcal{N}_1 \ldots \mathcal{N}_m\}, \chi, (\sigma^{\gg_m})_{\sigma\in L}, (u_\sigma^{\gg_l})_{\sigma\in L}\rangle$ satisfying

$$\bigwedge_{\sigma\in L} \begin{cases} \sigma^{\gg_m} \in \mathit{Runs}((N_\sigma)^{\gg_m}) & (\Phi_1) \\ u_\sigma^{\gg_l} \in \mathit{Runs}(\chi(\sigma)^{\gg_l}) & (\Phi_2) \\ \chi(\sigma) \neq nc \Rightarrow \mathit{dist}_H(\sigma^{\gg_m}, u_\sigma^{\gg_l}) \leq d & (\Phi_3) \end{cases} \tag{6}$$

The SAT encoding of (6) is a conjunction of three building blocks that are detailed in the next subsections.

6.1. (Φ_1): SAT Encoding of Log Traces

Log traces are encoded as sequential Petri nets noted N_σ to deal with model moves. The nets of traces contain an isolated $\gg_m$ transition that can fire at any instant. Moreover, a $\gg_w$ transition, for *"wait"*, is added at the end of the sequential nets to adapt the different sizes of traces. Fig. 5 shows the net of trace $\langle s, g, c\rangle$.

SAT encoding of Petri nets has been presented in [13] and require two kinds of boolean variables : transition firing and marking representations. We then declare $\tau^{\mathcal{L}}$ and $m^{\mathcal{L}}$ variables for the nets of log traces containing in $|L|$ and defined over alphabet Σ :

- $\tau^{\mathcal{L}}_{\sigma,i,a},\ for\ \sigma \in L,\ i = 1, \ldots, n\ and\ a \in \Sigma$ with n the limited size of run. Those boolean variables indicate that a transition of net of trace σ labeled by a fires at instant i. $\mathcal{L}$ allows one to differentiate net of log trace and the initial process model that also have τ variables.

- $m^{\mathcal{L}}_{\sigma,i,p},\ for\ \sigma \in L, i = 0, \ldots, n\ and\ p \in P_\sigma$ where P_σ is the set of places of net of trace σ. Those boolean variables represent the marking of the net of trace at instant i.

Then, any net of trace $N_\sigma = \langle P_\sigma, T_\sigma, F_\sigma, m_{\sigma_0}, m_{\sigma_f}, \Sigma, \Lambda \rangle$ has the following SAT clauses where n is the size of run:

- Initial marking of each net of trace:

$$\bigwedge_{\sigma \in L} \Big(\bigwedge_{p \in m_{\sigma_0}} m^{\mathcal{L}}_{\sigma,0,p} \Big) \wedge \Big(\bigwedge_{p \in P_\sigma \setminus m_{\sigma_0}} \neg m^{\mathcal{L}}_{\sigma,0,p} \Big) \tag{7}$$

- Final marking of each net of trace:

$$\bigwedge_{\sigma \in L} \Big(\bigwedge_{p \in m_{\sigma_f}} m^{\mathcal{L}}_{\sigma,n,p} \Big) \wedge \Big(\bigwedge_{p \in P_\sigma \setminus m_{\sigma_f}} \neg m^{\mathcal{L}}_{\sigma,n,p} \Big) \tag{8}$$

- One and only one t_i per net of trace j for each instant i:

$$\bigwedge_{\sigma \in L} \bigwedge_{i=1}^{n} \bigvee_{a \in \Sigma} \Big(\tau^{\mathcal{L}}_{\sigma,i,a} \wedge \bigwedge_{a' \in \Sigma \setminus t} \neg \tau^{\mathcal{L}}_{i,a'} \Big) \tag{9}$$

- The transitions are enabled when they fire:

$$\bigwedge_{\sigma \in L} \bigwedge_{i=1}^{n} \bigwedge_{a \in \Sigma} \Big(\tau^{\mathcal{L}}_{\sigma,i,a} \implies \bigwedge_{p \in {}^\bullet t} m^{\mathcal{L}}_{\sigma,i-1,p} \Big) \tag{10}$$

- Token game (for safe Petri nets):

$$\bigwedge_{\sigma\in L}\bigwedge_{i=1}^{n}\bigwedge_{a\in\Sigma}\bigwedge_{\substack{p\in P_\sigma\\ p\in t^\bullet\\ \Lambda(t)=a}}(\tau^{\mathcal{L}}_{\sigma,i,a} \implies m^{\mathcal{L}}_{\sigma,i,p}) \tag{11}$$

$$\bigwedge_{\sigma\in L}\bigwedge_{i=1}^{n}\bigwedge_{a\in\Sigma}\bigwedge_{\substack{p\in P_\sigma\\ p\in {}^\bullet t\setminus t^\bullet\\ \Lambda(t)=a}}(\tau^{\mathcal{L}}_{\sigma,i,a} \implies \neg m^{\mathcal{L}}_{\sigma,i,p}) \tag{12}$$

$$\bigwedge_{\sigma\in L}\bigwedge_{i=1}^{n}\bigwedge_{a\in\Sigma}\bigwedge_{\substack{p\in P_\sigma\\ p\notin {}^\bullet t\\ p\notin t^\bullet\\ \Lambda(t)=a}}(\tau^{\mathcal{L}}_{\sigma,i,a} \Rightarrow (m^{\mathcal{L}}_{\sigma,i,p} \Leftrightarrow m^{\mathcal{L}}_{\sigma,i-1,p})) \tag{13}$$

Then (Φ_1) is a conjunction of formulas (7) to (13).

6.2. (Φ_2)*: SAT encoding of Model Runs*

Now that nets of traces are encoded, we want to encode alignment between them and the process model which is also a Petri net. Any trace σ is aligned to the model and requires its own run of the model noted u_σ. We then encode in formula (Φ_2) $|L|$ times the process model with the following boolean variables :

- $\tau^{\mathcal{M}}_{\sigma,i,a},\ for\ \sigma \in L,\ i = 1,\dots,n\ and\ a \in \Sigma$ with n the limited size of run. Those variables indicate that model run of trace σ fires transition labeled by a at instant i.
- $m^{\mathcal{M}}_{\sigma,i,p},\ for\ \sigma \in L, i = 0,\dots,n\ and\ p \in P$ where P is the set of places of the process model. Those variables represent the required marking of the process model to get run u_σ.

The runs of the model follow the exact same Petri net encoding as the net of traces. Moreover, similarly to nets of traces, we added an isolated transition $\gg_l$ to represent log moves and a *wait* transition to deal with different sizes of traces.

Notice that due to the use of the SAT encoding of Petri nets, the model-based trace variants are forced to be sound and reach the final marking of the initial process model.

6.3. (Φ_3)*: SAT Encoding of Variants*

By using AMSTC, model-based trace variants are subnets $\mathcal{N}_1 \ldots \mathcal{N}_m$ ($m \in \mathbb{N}$). They are defined by transitions of model runs u_σ used to align clustered traces σ of L. This is defined with the conjunction (Φ_3) of (6) that we recall:

$$(\Phi_3): \quad \bigwedge_{\sigma \in L} \quad \chi(\sigma) \neq nc \Rightarrow dist_H(\sigma^{\gg_m}, u_\sigma^{\gg_l}) \leq d$$

In this subsection, we first present how the distances are encoded. Then, we detail the implication that incorporate clustered traces.

6.3.1. SAT Encoding of Distances

Aligning nets of traces and runs is obtained by computing the number of differences between fired transitions. We introduce $\delta^{\mathcal{M}}_{\sigma,i}$ and $\delta^{\mathcal{L}}_{\sigma,i} with\ \sigma \in L$ and $i = 1, \ldots, n$ boolean variables that represents model and log moves.

$$\bigwedge_{\sigma \in L} \bigwedge_{i=1}^{n} \bigwedge_{\substack{a \in \Sigma \\ a \neq \gg_l \\ a \neq \gg_m}} (\tau^{\mathcal{L}}_{\sigma,i,a} \wedge \neg \tau^{\mathcal{M}}_{\sigma,i,a}) \Leftrightarrow (\delta^{\mathcal{M}}_{\sigma,i} \wedge \delta^{\mathcal{L}}_{\sigma,i}) \tag{14}$$

$$\bigwedge_{\sigma \in L} \bigwedge_{i=1}^{n} \tau^{\mathcal{M}}_{\sigma,i,\gg_l} \Leftrightarrow \delta^{\mathcal{L}}_{\sigma,i} \tag{15}$$

$$\bigwedge_{\sigma \in L} \bigwedge_{i=1}^{n} \bigwedge_{a \in \Sigma} \tau^{\mathcal{L}}_{\sigma,i,\gg_m} \Leftrightarrow \delta^{\mathcal{M}}_{\sigma,i} \tag{16}$$

Axiom 14 forces two $\delta_{\sigma,i}$ variables to be `True` when two different activities are aligned which implies a distance to 2, this is equivalent of alignment cost. Indeed, in term of alignment, this situation is represented by a model and a log moves and costs 2.

The maximal distance d given in Equation (4) is implemented as *at_most_k* constraints, i.e., the number of variables δ to `True` is limited by d :

$$\bigwedge_{\sigma \in L} at\_most\_k(\sum_{i=1}^{n} \sum_{\Delta \in \{\mathcal{L}, \mathcal{M}\}} \delta_{\sigma,i}^{\Delta},\ d) \tag{17}$$

6.3.2. SAT Encoding of Clustered Traces

Every trace is either clustered in one of the m clusters and attached to a model-based trace variants either associated to the group entitled `nc`, for non-clustered traces. This is encoded with the following variables.

- InC_σ, for $\sigma \in L$ boolean variables that are `True` where trace σ is clustered.
- $\chi_{\sigma,k}$ for $\sigma \in L$ and $k = 0, \ldots, m$ boolean variables that encode which trace is in which cluster.

We then describe trace-cluster associations with the next SAT constraint :

$$\bigwedge_{\sigma \in L} InC_\sigma \quad \Leftrightarrow \quad \bigvee_{k1=0}^{m} (\chi_{\sigma,k1} \bigwedge_{\substack{k2=0 \\ k2 \neq k1}}^{m} \neg \chi_{\sigma,k2}) \tag{18}$$

If a trace is clustered, i.e. InC variable is `True`, transitions of its corresponding runs belong to the model-based trace variant of its cluster. We declare boolean variables that encode which transition belongs to which model-based trace variant.

- $c_{k,t}$ for $t \in T$ and $k = 0, \ldots, m$ with m the number of clusters. Those variables are `True` if transition t is in model-based trace variant k.

Equation 19 then describes model-based trace variant-transition associations.

$$\bigwedge_{\sigma \in L} InC_\sigma \Rightarrow (\bigwedge_{i=0}^{n} \bigwedge_{a \in \Sigma} \tau_{\sigma,i,a}^{\mathcal{M}} \Rightarrow \bigvee_{\substack{t \in T \\ \Lambda(t)=a}} \bigvee_{k=0}^{m} (\chi_{\sigma,k} \Rightarrow c_{k,t})) \tag{19}$$

Conjunction of expressions (14 to 19) forms (Φ_3).

6.4. Optimization criteria for AMSTC

The presented SAT formula accepts a large set of solutions. However, to get optimal model-based trace variants, we add three optimization criteria:

- Number of clustered traces should be maximized, i.e., number of non-clustered traces should be minimized.
- Inter-cluster distance, i.e., the distance between model-based trace variant, should be maximized.
- Distances between the traces and the model-based trace variant should be minimized.

This problem of maximizations is then a MaxSAT problem that uses weighted clauses.

6.4.1. Minimization of Non-Clustered Traces

First, we optimize the number of clustered traces by maximizing InC variables to `True` with the following MaxSAT formula :

$$\sum_{\sigma \in L} InC_{\sigma} * W1 \qquad \text{where } W1 \text{ is a weight} \quad (20)$$

6.4.2. Inter-cluster Distance Maximization

To maximize the inter-cluster distance defined in [8] we use the heuristic that inter-cluster distance is optimal when the number of common transitions between two model-based trace variants is minimized. We then introduce new boolean variables:

- $Common_{k1,k2,t}$ for $k1, k2 \in \{0, \ldots, m\}$, $k1 \neq k2$ and $t \in T$, are boolean variables describing common transitions between centroids of two clusters.

The minimization is found with the following MaxSAT problem where the idea is to set as many as possible $Common_{k1,k2,t}$ variables to `False` which reduces

the number of common transitions between model-based trace variants.

$$\sum_{k1=0}^{m} \sum_{\substack{k2=0 \\ k1 \neq k2}}^{m} \sum_{t \in T} \neg Common_{k1,k2,t} * W2 \qquad \text{where } W2 \text{ is a weight} \quad (21)$$

6.4.3. Minimization of Distances

Finally the minimization of differences can be encoded by the following MaxSAT clauses:

$$\sum_{i=1}^{n} \sum_{\sigma \in L} \sum_{\Delta \in \{\mathcal{L}, \mathcal{M}\}} \neg \delta_{\sigma,i}^{\Delta} * W3 \qquad \text{where } W3 \text{ is a weight} \quad (22)$$

6.4.4. Weights and Peculiarities in Implementation

This large MaxSAT formula is implemented in DarkSider[2] an Ocaml command line tool and `da4py`[3] a Python library. The two softwares use MaxSAT solvers to get optimal solutions and return centroids and associated traces.

In the implementations, the number of transitions per cluster is limited by a parameter, entitled $maxCSize$. Similarly to d, this threshold is encoded as at_most_k constraints [25].

We define the following priorities and the implications on the weights:

1. First, traces should be clustered: $W1 = maxD * W3 + maxCSize * W2$
2. Then, number of common transitions should be limited: $W2 = maxD * W3$
3. Finally, distances should be reduced: $W3$

7. A Sampling Algorithm to Deal with Large Logs

We propose a sampling algorithm that helps one to deal with large logs.

7.1. Main Sampling Idea

In practice, formulas produced by our encoding for AMSTC are too large for casual machines in term of memory space. To reduce the formula, we propose

[2] http://www.lsv.fr/ chatain/darksider
[3] https://github.com/BoltMaud/da4py

Algorithm 1, a sampling method that calls AMSTC only on samples. AMSTC returns a set of subnets which are the model-based trace variants $\mathcal{N}_i$ and their list of clustered traces. Then every trace of the entire log is aligned to each of the discovered variants and added to corresponding cluster if the alignment is sufficiently good. Then we iterate over the remaining traces to cluster with new model-based trace variants.

Algorithm 1: AMSTC Sampling Algorithm

Input : N, L, $sampleSize$, m, $maxCSize$, $maxD$, $maxTrials$

```
1  Clusters = {}
2  counter = 0
3  while |L| > 0 and counter < maxTrials do
4      sublog = randomSampling(L, sampleSize)
5      clustering = AMSTC(N, sublog, m, maxCSize, maxD)
6      if clustering is ∅ then
7          counter = counter + 1
8      else
9          LogAlignToCluster(clustering, L, maxD, Clusters)   ▷ below
10         counter = 0
11 if L > 0 then
12     Clusters[nc] = L                                       ▷ non-clustered traces
```

Output: Clusters : {modelBasedTraceVariant : clusteredTraces}

```
13 Function LogAlignToCluster(clustering, L, maxD, Clusters):
14     foreach cluster in clustering do
15         modelBasedTraceVariant = cluster.getVariant()
16         foreach l in L do
17             if alignmentCost(l, modelBasedTraceVariant) < maxD
                then
18                 Clusters[cluster].add(l)
19                 L.remove(l)
```

One can limit the number of model-based trace variants per loop, which also reduces the size of the SAT formula. Algorithm 1 takes as input a model N, a log L, a sample size $sampleSize$ to randomly select a sample of the entire log, a counter to stop the research of model-based trace variants when samples cannot be aligned anymore and parameters of the AMSTC algorithm, i.e. the number

of model-based trace variants m, the maximal number of transitions per cluster $maxCSize$ and the maximal distance between the traces and their variants.

7.2. Reducing Alignment Use with Casual Edit Distance between Traces

Alignment is much more expensive than Edit Distance between words. To limit the use of alignment, we propose another version that first uses edit distance between the clustered traces and the rest of the log. This heuristic allows one to considerably reduce the log before doing alignments.

Algorithm 2: Reducing Alignment Use of Algorithm 1 (lines 13 to 19)

```
1  Function LogAlignToCluster(clustering, L, maxD, Clusters):
2      foreach cluster in clustering do
3          clusteredTraces = cluster.getClusterTraces()
4          modelBasedTraceVariant = cluster.getVariants()
5          foreach l in L do
6              foreach trace in clusteredTraces do
7                  if editDistance(l, trace) < maxD then
8                      Clusters[cluster].add(l)
9                      L.remove(l)
10             if l is still unclustered then
11                 if alignmentCost(l, modelBasedTraceVariant) < maxD
                    then
12                     Clusters[cluster].add(l)
13                     L.remove(l)
```

As any trace is a maximal distance $maxD$ to its model-based trace variant, novel clustered trace is then at maximal distance to its centroid to $2 * maxD$ as the maximal distance between clustered traces and unclustered traces is also $maxD$. One can also consider to introduce another input for this purpose, thus raising the number of clustered traces.

7.3. Memoization of the calls to `alignmentCost`

Typically, in real life logs, traces corresponding to frequent behaviors occur many times in the log. This suggests an easy way to improve the efficiency of

Algorithm 1: it suffices to memoize the calls to the function `alignmentCost` which takes most of the computation time. Tab. 5e in the experiments section 8 shows the spectacular improvement obtained with this technique.

7.4. A Statistical Confidence for Sampling

Probability of Missing Clusterizable Traces

We focus on the situations where Algorithm 7 stops before clustering all the log traces which are sufficiently close to a run of the model (i.e. at distance $\leq maxD$), that we call clusterizable traces.

We quantify this probability as a function of the proportion of unclustered traces which are clusterizable. Let p be this proportion when the algorithm starts a series of iterations in order to find a nonempty clustering. As long as the clusterings fail, the unclustered traces remain the same and the proportion p does not change.

Now, a clustering fails precisely when no clusterizable trace is selected in the sample. The probability of this is $(1-p)^{sampleSize}$ (the sampled traces are selected independently one from the other).

Finally, if the algorithm starts a series of clusterings from a state where the proportion of unclustered traces which are clusterizable is p, the probability that it fails $maxTrials$ times to cluster traces (and then stops), is $(1-p)^{sampleSize \times maxTrials}$.

For example, assume 5% of the unclustered traces are clusterizable, the probability that the algorithm fails to detect them after 2 trials with $sampleSize = 10$, is $0.95^{2\times10} \approx 0,36$. Alter 10 trials, the probability drops to $0.95^{10\times10} \approx 0,006$.

If no clusterizable trace remains, i.e. $p = 0$, the trials fail (and then the algorithm terminates) with probability 1.

Wilson score interval

Equivalently to [3], the number of trials in the sampling method can be assessed with a statistical method. Wilson score interval [46] gives a lower bound

$\mathcal{L}_b$ and a upper bound $\mathcal{U}_b$ of probability δ to get a probability p of success on a sample of size n with a confidence α:

$$\frac{p + \frac{z^2}{2n} - z * \sqrt{\frac{p*(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} \leq \delta \leq \frac{p + \frac{z^2}{2n} + z * \sqrt{\frac{p*(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} \tag{23}$$

where z is the $1 - \frac{\alpha}{2}$ quantile of a standard normal distribution corresponding to the target error rate α.

In our sampling method, we are looking for the number of trials n such as the probability p to get success with our $AMSTC$ function, i.e., the sample provide new model-based trace variants, is bounded by δ with a confidence α. In other words, the probability to get new model-based trace variants with a maximal bound δ corresponds to the null hypothesis, i.e., $p = 0$, of the statistical interval. In our approach, we are looking for the minimal number of sample $\mathcal{N}_{min}$ that is sufficient for a given maximal bound δ and a confidence α. From the right side of Wilson score intervals (23), we obtain:

$$\mathcal{N}_{min} \geq \frac{z^2 * (1 - \delta)}{\delta} \tag{24}$$

To illustrate this inequality we give an example inspired from [3]. We want to find the minimum sample size required to be confident at 0.99 that a novel trial of $AMSTC$ would not give new model-based trace variants with a lower bound probability of 0.95. Then, we set the confidence α to 0.01 which implies $z = 2.58$ from the $1 - \frac{\alpha}{2}$ one-side quantile of the standard normal distribution. The lower bound probability that the novel trial of getting new clusters will fail corresponds to the upper bound to get a success, i.e., to get clusters. Then $\delta = 0.05$. From equation (24), we found that the minimal sample size is 127 with Wilson score interval.

8. Experimental Results

In this section, we present our different AMSTC implementations and show a set of experiments from small artificial logs to large real-life logs. As usual,

some traces corresponding to frequent behavior occur many times in the log. We write "number of classical trace variants" for the number of different log traces in a log, in reference to previous works on trace variants like [40]. The aim of the present paper is precisely to reduce this number of trace variants using our richer notion of trace variants obtained as centroids of trace clusters.

8.1. Tools

Implementation of the AMSTC method exists in two tools : DarkSider and `da4py`. DarkSider is an ocaml command line tool while `da4py` is a Python library. Thanks to [25], the most recent and effecient version of the clustering method is `da4py` and uses state-of-the-art SAT and MaxSAT solver. `da4py` also uses objects and functions like *alignment computation* from `pm4py` library [5].

As presented in [8], the tools use a heuristic on the length of the run to deal with long traces. The parameter $sizeOfRun$ alleviates the complexity of the problem when traces are long by computing only a prefix of the traces. In this paper, most of the traces can be fully computed (see the sizes of runs of Tab. 6).

The sampling methods and additive heuristics have been developed in `da4py` only. Then all the experiments of this paper have been run with `da4py` on a virtual machine with 12 CPU Intel Xeon 2.67GHz and 50GB RAM.

8.2. Event Logs

Experiments have been done on a set of 7 different logs shown in Tab. 4 from 9 to 41353 traces. First log has been presented in [8] to show how model-based clustering helps one to group traces and extracts deviated behaviors. Log presented in section 1.1 is also an artificial log for this purpose. All the other logs have been introduced in other context. We used 3 real-life logs of BPI challenges.

AMSTC algorithm extracts model-based variants that required a model as input. To show a large variety of different cases, we have done our experiments on models of literature that have been created in different ways for the respective logs. Tab. 4 indicates which design method have been used. Complexity of

Log	$\|L\|$	Number of Classical Trace Variants	$\|\Sigma\|$	$\forall_{\sigma \in L} max(\|\sigma\|)$	Model Discovery Method	$\|T\|$	$\|P\|$
Artificial L_1 of [8]	9	9	7	7	Hand written model [1]	8	6
Artificial L_{vrts} presented in Sec 1.1	500	411	40	36	Hand written model [1]	90	92
Artificial $L1$ of [39]	500	453	37	36	PLG2 tool	39	40
LoanA of [43]	500	100	16	37	Hand written model [2]	17	14
$BPI'2013_{cp}$	1487	183	7	35	Heuristic Miner [3]	25	18
$BPI'2013_{inc}$	7554	1511	13	123	Split Miner [3]	15	11
$BPI'2014_{f}$	41353	14948	9	167	Split Miner [3]	24	16

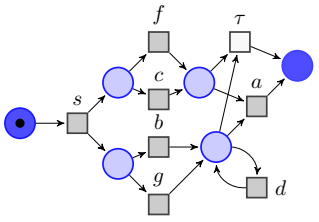
[1] Available at https://github.com/BoltMaud/da4py/examples
[2] Available at doi:10.4121/uuid:c1d1fdbb-72df-470d-9315-d6f97e1d7c7c
[3] From [1], available at https://doi.org/10.6084/m9.figshare.6376592.v1

Table 4: Event Logs Statistics and Used Discovered Models

AMSTC depends on the size of the event logs and size of the models which are detailed in the table.

Notice that models with choices, loops and concurrency have been preferred than linear models or mostly concurrent patterns. As our method extracts sound subnets, fully concurrent models cannot be divided in several subnets.

8.3. Qualitative Experiments



(a) Petri Net.

$\langle s, c, g \rangle$
$\langle s, c, g, d \rangle$
$\langle s, f, b, a \rangle$
$\langle s, f, f, a \rangle$
$\langle s, b, f, a \rangle$
$\langle s, g, f, d, d \rangle$
$\langle s, g, f, d, d, d, d \rangle$
$\langle g, c, f, s, d, d \rangle$
$\langle s, d, d, d \rangle$

(b) Log L_1

Figure 6: Motivation Example of [8]

This section aims at comparing complete AMSTC and the sampling method

outputs with different parameters. We used the log L_1 of [8] and its hand written process model which is small enough to be fully computed by the entire algorithm. The log contains 9 traces of maximal length to 7 and the model has 14 nodes (see Fig. 6).

Experiments have been done with a size of run to 7 and a maximal of trials to 2. Tab. 5 gives the results of the experiments. Each sub-table is an improvement of the previous one except table 5f which aims at showing consequence of the number of classical trace variants. Every line is an experiment of a specific setting and has been run 10 times. For descriptive results, like the number of clusters and traces, the most returned results are shown. Runtimes are means for the experiments of the selected results. We now give an analysis of the tables.

In sub-table 5a, we see that, given different distances and number of clusters, results differ. Raising the distance between the traces and the centroids allows to cluster more traces. However, a good distance threshold aims at partitioning the traces in more specific clusters and then get specific model-based trace variants.

In sub-table 5b, sampling method outputs are presented. For the exact same distribution of the traces, we proportionally get the same results of the complete AMSTC method. Then the exact same model-based trace variants are extracted.

Sub-table 5c aims at showing that our method can deal with strange trace variant frequency. In this experiment, the second trace of L_1 have been duplicated 82000 times while the other traces appear 1000 times each. We can see that the size of the clusters are indeed very different. However, notice that the number of clusters corresponds to the previous experiments. The same model-based trace variants have been extracted.

Sub-tables 5d and 5e contain heuristic improvements in term of runtimes.

Finally, we have added noise in log to raise the number of trace variants which is used by the heuristics. The number of clusters is then different which is expected.

Discussion: from this experiments, we see that the AMSTC method helps

$\|L\|$	Number of Classical Trace Variants	Sample Size	Maximal Distance (Number of Moves)	Number of Clusters per $AMSTC$	Number of Clusters	Traces Per Cluster			Un-clustered Traces	Runtime (secs)	Alignment Runtime (secs)
						Min	Max	Avg			
9	9	/	0	3	3	2	2	2	3	1.30	/
9	9	/	0	2	2	2	2	2	5	1.41	/
9	9	/	2	3	2	3	5	4	1	11.63	/

(a) Complete AMSTC on a small log (L_1)

90 000	9	10	0	2	3	20 000	20 000	20 000	30 000	1503.98	14095.16
90 000	9	10	2	2	2	30 000	50 000	40 000	10 000	984.90	975.45

(b) Sampling Method on a large Log ($L_1 * 10\ 000$)

90 000	9	10	0	2	3	2 000	83 000	29 000	3 000	595.37	587.83
90 000	9	10	2	2	2	3 000	86 000	44 500	1 000	473.42	465.52

(c) Clustering Effects on Specific Distribution of Traces of L_1 in large log

90 000	9	10	0	2	3	2 000	83 000	29 000	3 000	146.83	133.40
90 000	9	10	2	2	2	3 000	86 000	44 500	1 000	151.51	142.65

(d) Edit Distance Heuristic to reduce Alignment Runtime

90 000	9	10	0	2	3	2 000	83 000	29 000	3 000	7.36	0.16
90 000	9	10	2	2	2	3 000	86 000	44 500	1 000	8.45	0.07

(e) Memoization of `alignmentCost`

90 000	255	10	0	2	3	18 495	18 866	18 731	33 805	17.95	8.29
90 000	255	10	2	2	3	9 821	50 510	29 825	525	15.88	4.30
90 000	12 460	10	0	2	4	631	2 874	1 601	83593	962.82	935.65
90 000	12 460	10	2	2	3	15 366	35 964	22 855	21 433	572.87	560.10

(f) Clustering Effects on Noisy Log (Raising the Number of Classical Trace Variants)

Table 5: Comparison of AMSTC results for Different Parameters on log L_1 of [8].

one to extract good model-based trace variants. The sampling method efficiently works for large logs and specific distributions of trace variants. For noisy logs, more model-based trace variants are extracted. To reduce the number of model-based trace variants and un-clustered traces, one can change the maximal distance between trace and variants.

8.4. Quantitative Experiments

In this section, we present experimentation of the set of different logs presented in section 8.2. Settings presented in the table have been chosen by the author after some tests. The tests consisted on evaluating the distance between

the traces and the models, assessing the minimal size of the run depending on the traces and loop behaviors and counting an approximate maximal number of transitions per cluster. The sampling size have been chosen in a way that runtime is optimized. The number of clusters per loop was set to 2 to reduce the formula size. Finally, we set the number of trials of the sampling algorithm to 5 sequential fails.

Each experiment have been run 10 times except the last one because of long runtimes. Due to trace variants and the use of causal edit distance between traces, outputs of the same experiment are different. In Tab. 6 we show examples of clustering results that have returned the least unclustered traces. Notice that the runtimes are much larger because of the number of activities and trace variants in logs. Moreover, the models are also much larger than in the previous experiments.

Log	Number of Classical Trace Variants	Sample Size	Size of run	Maximal Number of Transition per Cluster	Maximal Distance (Number of Moves)	Number of Clusters	Number of Traces Per Cluster			Unclustered Traces	Runtime (secs)	Alignment Runtime (secs)
							Min	Max	Avg			
Artificial L_1 of [8]	9	5	5	5	0	3	2	2	2	3	4.02	0.13
Artificial L_{vrts} presented in Sec 1.1	411	5	15	9	0	12	22	54	42	0	2135.84	27.96
Artificial $L1$ of [39]	453	5	15	15	4	4	19	145	62	252	6681.47	666.23
LoanA of [11]	100	10	20	14	1	10	10	60	25	250	409.32	64.31
$BPI'2013_{cp}$	183	10	20	9	1	3	32	1121	451	134	245.29	42.33
$BPI'2013_{inc}$	1511	5	20	11	2	2	204	5981	3092	1369	4091.79	3646.47
$BPI'2014_f$	14948	5	20	15	2	6	257	21909	4344	15289	66709.70	66130.29

Table 6: Examples of AMSTC Outputs on a Set of 7 Logs

We can see that our running example presented in Sec 1.1 perfectly associates every trace of the log to a model-based trace variants. For 500 traces and 411 classical trace variants, the AMSTC method finds 12 model-based trace variants. Those subnet instances are then a good way to analyze the log traces separately.

We see that for more noisy logs containing in real-life data and a small distance between trace and variants, we are able to cluster a good number of traces in a very small number of clusters. We see from the number of classical

trace variants in the second column that the method is indeed able to group them in our more general model-based trace variants.

Discussion: in those experiments, we highlight how our method is able to extract a small number of model-based trace variants of real-life logs compared to the number of classical trace variants. We see that still a lot of traces remain unclustered but this is due to the distance between variants and traces which is intentionally small. The extracted model-based trace variants are well fitting to the clustered traces (maximal distance is always lower than 4 in Tab. 6).

8.5. Case Study

To present the value of our model-based variants, we present a case study on a real-life log and different process models. Then, we compare our work to the *Super-instances* from [18] which also aims at representing group of traces and are also found by using a clustering approach.

8.5.1. Event Log

We employed the real-life log from the Business Process Management Challenge of 2013 about the *closed problems* of Volvo management system. The event log contains 1487 log traces, 183 classical trace varitants, 7 activity steps (taking in account activity name and progress) and 4 main activity names. Tab. 7 shows the frequency traces containing the activity names. We see that some activities, like `Unmatched`, are much less frequent than other, like `Completed`. We also notice that activity `Queued` appears in many classical trace variants but those trace variants are not very frequent in the log. This simple table will help to get good intuition in the understanding of the results of the model-based variants and the super-instances from [18].

8.5.2. How Model Quality Impacts Model-based Trace Variants

By using the classical trace variants, i.e, the number of unique sequences, one obtains 183 instances for BPIC 2013 closed problems event log. For human analysis and business aspect, this number of instances is too large to be understood. Our AMSTC method helps one to get more representative variants.

Activity Label	Frequency of Traces Containing the Activity Label	Frequency of Classical Trace Variants Containing the Activity Label
Accepted	1.00	0.99
Completed	1.00	1.00
Queued	0.36	0.84
Unmatched	0.01	0.04

Table 7: Frequency of Traces and Classical Variants Containing the Different Activities

However, our method is based on an existing model. In Tab. 8, we show different clustering results for different model qualities.

Model	Fitness	Maximal Number of Transition per Cluster	Maximal Distance (Number of Moves)	Number of Clusters	Number of Traces Per Cluster			Unclustered Traces
					Min	Max	Avg	
Split Miner	0.98	10	0	4	1	917	251	485
			2	2	59	1377	718	51
Heuristic Miner	0.94	12	0	2	3	917	460	567
			2	2	17	1391	704	79
Inductive Miner	0.82	7	0	1	574	574	574	913
			2	1	1367	1367	1367	120

Models from [1], available at https://doi.org/10.6084/m9.figshare.6376592.v1

Table 8: AMSTC on BPIC 2013_{cp} Log and Different Models. Sample size is set to 15 and run size to 20.

We observe that the fitness of the model impacts the number unclustered traces. Moreover, we see that results of the last model of Tab. 8 do not give useful information in term of representatives. In fact, this model, learned with the inductive miner, is very simple and contains only a choice and a loop, making hard to split the model in model-based trace variants. Finally, we note that our clusters are very heterogeneous. This aspect can help one to understand the structure of the model by analyzing the model-based trace variants.

Discussion: in this experiment, we show how fitness impacts results of the ASMTC methods, thus giving different model-based trace variants. For larger distances to the centroids, one gets less variants but can cluster more traces from the event log. In opposite, for a distance to zero, the number of model-based variants raises but many traces are left unclustered, i.e, do not have a

representative.

8.5.3. Comparison of Clusters and Representatives

Our method extracts Model-based Trace Variants along with clustered traces. This aspect motivates us to compare our method to the Super-Instances of [18]. In this work, it also finds clusters from the log traces, and work with the centroids of them which they call *Super-Instances*. We see that both methods claims to get representatives of groups of traces. Moreover, they also use a clustering method, i.e., the K-means algorithm. The algorithm is based on the Euclidean distance between vectors of activities occurrence containing in the traces. The main differences between our approach and [18] are: first, our approach does not need as input the number of clusters, whilst [18] does. Second, [18] tries to obtain a balanced clustering, i.e., a clustering where the size of the computed clusters are as much similar as possible. In contrast, our method only focuses on finding good representatives of each cluster, regardless of its size. These differences are corroborated in the case study found below.

First, we used the Jaccard index [44] to compare two clusterings, $\mathcal{C}$ and $\mathcal{C}'$, defined by:

$$\mathcal{J}(\mathcal{C},\mathcal{C}') = \frac{n_{11}}{n_{11}+n_{01}+n_{10}}$$

where:
n_{11} is the number of pairs of items clustered together in $\mathcal{C}$ and in $\mathcal{C}'$
n_{10} is the number of pairs of items clustered together in $\mathcal{C}$ but in different clusters in $\mathcal{C}'$
n_{01} is the number of pairs of items in different cluster in $\mathcal{C}$ but in same clusters in $\mathcal{C}'$

In our study, the Jaccard index relates pairs of traces in the two clustering results, we also report how many pairs of traces have been clustered together in both clusterings (column n_{11} in Tab. 9), and how many traces were clustered together in a clustering but not in the other one (columns n_{01} and n_{10} in Tab. 9). A high value (≤ 1) of the Jaccard index indicates that the clustering are similar. The main of this study is then to study our clustering results of Tab.8 and the corresponding outputs of [18].

Cluster Sizes of AMSTC (see Tab. 8)			Jaccard Comparison				Cluster Sizes by using method of [18]		
Min	Max	Median	n_{11}	n_{10}	n_{01}	Index	Min	Max	Median
1	917	42	169577	253652	1560	0.399	82	496	212
59	1377	718	550325	398762	32081	0.561	458	978	718
3	917	470	210957	209032	1479	0.501	424	496	460
17	1391	704	562403	404478	7038	0.577	431	977	704
574	574	574	164451	0	0	1.000	574	574	574
1367	1367	1367	933661	0	0	1.000	1367	1367	1367

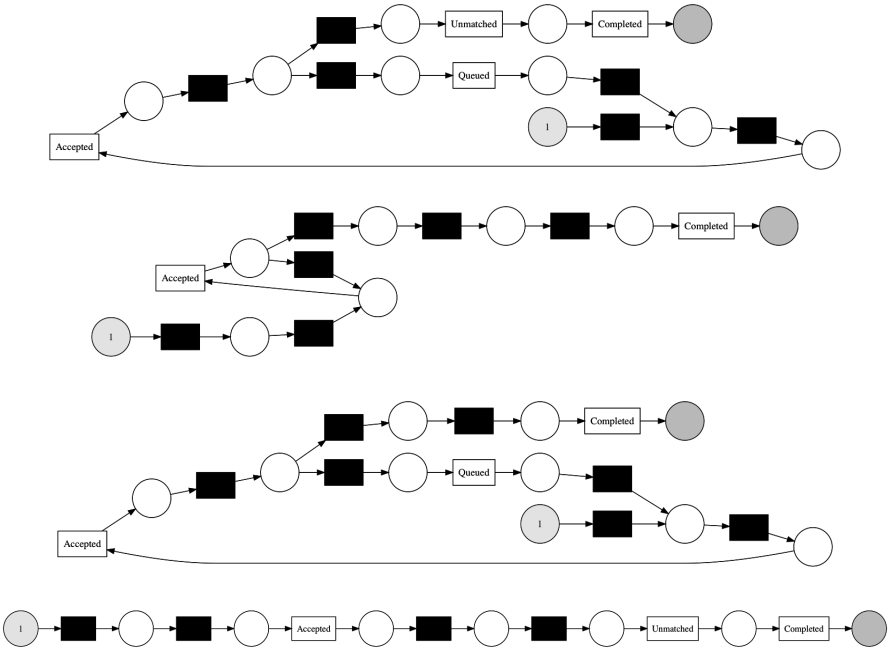
Table 9: Cluster Comparison between AMSTC and [18] results for the same traces and number of clusters. All the rows of this table have been produced based on the rows of Tab.8

.

First, we want to justify about the use of clustered traces only when instructing [18]: in order to compare the clusterings, which aim at grouping traces for their similarities, we claim that considering as a cluster the sets traces not clustered by our method would not give an appropriate comparison, since these unclustered traces are not related. So, to compare the clusters given by the two methods, we ran method of [18] only on the traces that have been clustered in Tab. 8.

In opposite of AMSTC, method of [18] requires, as input, the number of clusters. For each ASMTC presented in Tab. 8, we launched the method on the clustered traces and used the same number of clusters. For instance, for the first line, we ran the Super-Instance method on the 1002 clustered traces and set the number of clusters to 4. We give in Tab.9 an overview of the cluster sizes and the Jaccard Index. The two last rows of the table are not informative as their is only 1 cluster. For the first line, in which 4 clusters have been discovered, we see that the Jaccard Index is under 0.5, meaning that there are more pairs of traces clustered in different clusters by the clustering methods. For the other rows, we see that, the Jaccard index varies from 0.501 to 0.577 which is better than hazard for those lines which have only 2 clusters (see Tab. 8 for the number of clusters). Indeed, for 2 clusters, the probability of n_{11}, a pair of traces to be clustered together by both methods, is 1/4, while the addition of the probabilities of n_{10} and n_{01} is 1/2. We see that the Jaccard Index tells

us that our method have some similarities.



(a) Model-based Trace Variants

$\langle Accepted, Completed \rangle$
$\langle Accepted, Accepted, Completed \rangle$
$\langle Accepted, Queued, Accepted, Completed \rangle$
$\langle Accepted, Accepted, Accepted, Accepted, Accepted, Completed \rangle$

(b) Super-Instances

Figure 7: Representatives of Clusters

To conclude this section, we present in Fig. 7 the model-based trace variants of our approach and the super-instances of [18] of the first line of Tab.9. We remark that two of our representatives contain activity `Queued` and `Unmatched` while the super-instances only have once the activity `Queued`. This is due to the fact that method of [18] uses K-means that tries to get centroids which are means of the occurrences of activities and n-grams of activities. In our method, the model-based trace variants are subnets which can contain choices,

thus allowing activities that are not used in all the traces of a cluster. The variant is then more general than a sequence. However, our method can also get sequential net (see Fig. 7). Model-based trace variants are a good balance to represent the traces, especially when the distance to the traces is zero, i.e., the variants can replay the traces. This last aspect cannot be induced by using the super-instances.

Discussion: in this comparison, we aimed at showing the differences between the super-instances, which are sequences, and the model-based trace variants. We have seen that the present activities differ from a method to another. The AMSTC gives more complex structures, but the resulted representatives provide more information in terms of activity labels. Moreover, the semantics of Petri nets also provide information about the represented log traces. In addition to different structures, the two approaches have completely different underlying search algorithms. Our method tries to get subnets by using alignment between the model and the log traces. In [18], the distance is the Euclidean distance between vectors of occurrences and n-grams, i.e., sub-sequences of traces, and a balanced clustering is obtained if possible. Finally, we want to point out that our method does not require the number of clusters apriori, i.e., our method searches good representatives.

9. Conclusion and Future Work

This paper builds upon our previous work to propose a generalized version of trace variant. Our notion is based on the assumption that a normative process model exists, and then a trace variant, which represents a subset of log traces, can be created from the perspective of the process model (i.e., they can be processes or subnets of the initial process model), to describe better concurrency and iterative aspects that are blured when only looking at the event log. This generalization may alleviate the requirements for understanding the main behaviors in an event log, one of the first actions that is performed in a process mining project. Remarkably, we use different forms of alignments

to compute these generalized trace variants, and encode the problems into SAT to operationalize the techniques. Since for large instances the SAT problems to solve may be prohibitive, we propose a novel family of sampling techniques together with certain statistical guarantees on the number of trials to get a sound set of trace variants. Our experimental evaluation shows promising results in several aspects, included the capability to deal with larger problem instances, but also certain robustness in the presence of noise in the event log.

As future work, we see three main interesting directions to follow: first, studying the characteristics behind a certain clustering solution may help us to find better (SAT) encodings. Second, to explore different representations of the subnets found (e.g., regular expressions) may facilitate the interpretation by a human. Third, to investigate other applications of the clustering algorithm presented in this paper could be already envisioned; for instance, the repair of an imperative process model, by detecting those parts that are more declarative, thus transforming the initial model into a *hybrid process model* [42]: we believe our generalized approach for trace variants can be naturally suited for tackling this problem.

Acknowledgments.

This work has been supported by Farman institute at ENS Paris-Saclay and by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R.

References

[1] Adriano Augusto, Abel Armas-Cervantes, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, and Daniel Reißner. Abstract-and-compare: A family of scalable precision measures for automated process discovery. In Mathias Weske, Marco Montali, Ingo Weber, and Jan vom Brocke, editors, *Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings*, volume 11080 of *Lecture Notes in Computer Science*, pages 158–175. Springer, 2018.

[2] Nithish Pai Ballambettu, Mahima Agumbe Suresh, and R. P. Jagadeesh Chandra Bose. Analyzing process variants to understand differences in key performance indices. In *AISE*, pages 298–313. Springer, 2017.

[3] Martin Bauer, Arik Senderovich, Avigdor Gal, Lars Grunske, and Matthias Weidlich. How much event data is enough? a statistical framework for process discovery. In *International Conference on Advanced Information Systems Engineering*, pages 239–256. Springer, 2018.

[4] Michael Benedikt, Gabriele Puppis, and Cristian Riveros. Regular repair of specifications. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011*, pages 335–344, 2011.

[5] Alessandro Berti, Sebastiaan J van Zelst, and Wil van der Aalst. Process mining for python (pm4py): bridging the gap between process-and data science. *arXiv preprint arXiv:1905.06169*, 2019.

[6] Alfredo Bolt, Massimiliano de Leoni, and Wil M. P. van der Aalst. A visual approach to spot statistically-significant differences in event logs based on process metrics. In *AISE*. Springer, 2016.

[7] Alfredo Bolt, Massimiliano de Leoni, and Wil M.P. van der Aalst. Process variant comparison: Using event logs to detect differences in behavior and business rules. *Information Systems*, 74:53–66, 2018.

[8] Mathilde Boltenhagen, Thomas Chatain, and Josep Carmona. Generalized alignment-based trace clustering of process behavior. In *International Conference on Applications and Theory of Petri Nets and Concurrency*, pages 237–257. Springer, 2019.

[9] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Context aware trace clustering: Towards improving process mining results. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2009*, pages 401–412, 2009.

[10] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Trace clustering based on conserved patterns: Towards achieving better process models. In *Business Process Management Workshops, BPM 2009 International Workshops, Revised Papers*, pages 170–181, 2009.

[11] J.C.A.M. Buijs. Loan application example, 2013.

[12] Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. *Conformance Checking - Relating Processes and Models*. Springer, 2018.

[13] Thomas Chatain and Josep Carmona. Anti-alignments in conformance checking - the dark side of process models. In *Application and Theory of Petri Nets and Concurrency - 37th International Conference, PETRI NETS 2016, Proceedings*, pages 240–258, 2016.

[14] Thomas Chatain, Josep Carmona, and Boudewijn F. van Dongen. Alignment-based trace clustering. In *Conceptual Modeling - 36th International Conference, ER 2017, Proceedings*, pages 295–308, 2017.

[15] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. *Theor. Comput. Sci.*, 147(1&2):117–136, 1995.

[16] Carsten Cordes, Thomas Vogelgesang, and Hans-Jürgen Appelrath. A generic approach for calculating and visualizing differences between process models in multidimensional process mining. In *BPM Workshops*, pages 383–394. Springer, 2015.

[17] Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary (extended abstract). *CoRR*, abs/1809.07115, 2018.

[18] Pieter De Koninck and Jochen De Weerdt. Scalable mixed-paradigm trace clustering using super-instances. In *2019 International Conference on Process Mining (ICPM)*, pages 17–24. IEEE, 2019.

[19] Javier de San Pedro and Jordi Cortadella. Mining structured Petri nets for the visualization of process behavior. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 839–846, 2016.

[20] Marlon Dumas, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software Through Process Technology.* Wiley, 2005.

[21] J. Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28(6):575–591, 1991.

[22] Diogo R. Ferreira, Marielba Zacarias, Miguel Malheiros, and Pedro Ferreira. Approaching process mining with sequence clustering: Experiments and findings. In *Business Process Management, 5th International Conference, BPM 2007, Proceedings*, pages 360–374.

[23] Gianluigi Greco, Antonella Guzzo, Luigi Pontieri, and Domenico Saccà. Discovering expressive process models by clustering log traces. *IEEE Trans. Knowl. Data Eng.*, 18(8):1010–1027, 2006.

[24] Bart Hompes, Joos Buijs, Wil van der Aalst, Prabhakar Dixit, and Hans Buurman. Discovering deviating cases and process variants using trace clustering. In *Proceedings of the 27th Benelux Conference on Artificial Intelligence (BNAIC 2015)*, 2015.

[25] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. Pysat: A python toolkit for prototyping with sat oracles. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 428–437. Springer, 2018.

[26] Xixi Lu, Dirk Fahland, Frank JHM van den Biggelaar, and Wil MP van der Aalst. Detecting deviating behaviors without models. In *International Conference on Business Process Management*, pages 126–139. Springer, 2016.

[27] Xixi Lu, Dirk Fahland, and Wil M. P. van der Aalst. Conformance checking based on partially ordered event data. In *Business Process Management Workshops - BPM 2014 International Workshops, Eindhoven, Revised Papers*, pages 75–88, 2014.

[28] Andrey Mokhov, Jordi Cortadella, and Alessandro de Gennaro. Process windows. In *17th International Conference on Application of Concurrency to System Design, ACSD 2017*, pages 86–95, 2017.

[29] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–574, April 1989.

[30] Hoang Huy Nguyen, Marlon Dumas, Marcello La Rosa, and Arthur H.M. ter Hofstede. Multi-perspective comparison of business process variants based on event logs (extended paper). April 2018.

[31] A. Pini, R. Brown, and M. T. Wynn. Process visualization techniques for multi-perspective process comparisons. pages 183–197. Springer, 2015.

[32] Jonas Poelmans, Guido Dedene, Gerda Verheyden, Herman Van der Mussele, Stijn Viaene, and Edward Peters. Combining business process and data discovery techniques for analyzing and improving integrated care pathways. In *ICDM*, pages 505–517. Springer, 2010.

[33] Hernán Ponce de León, César Rodríguez, and Josep Carmona. POD - A tool for process discovery using partial orders and independence information. In *Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015)*, pages 100–104, 2015.

[34] Hernán Ponce de León, César Rodríguez, Josep Carmona, Keijo Heljanko, and Stefan Haar. Unfolding-based process discovery. In *Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Proceedings*, pages 31–47. Springer, 2015.

[35] Minseok Song, Christian W. Günther, and Wil M. P. van der Aalst. Trace clustering in process mining. In *Business Process Management Workshops, BPM 2008 International Workshops, Milano, Italy, September 1-4, 2008. Revised Papers*, pages 109–120, 2008.

[36] Iain A. Stewart. Reachability in some classes of acyclic Petri nets. *Fundam. Inform.*, 23(1):91–100, 1995.

[37] Suriadi Suriadi, Ronny S. Mans, Moe T. Wynn, Andrew Partington, and Jonathan Karnon. Measuring patient flow variations: A cross-organisational process mining approach. In *Asia Pacific BPM*. Springer, 2014.

[38] Jo Swinnen, Benoît Depaire, Mieke J. Jans, and Koen Vanhoof. A process deviation analysis – a case study. In *BPM Workshops*, pages 87–98. Springer, 2012.

[39] Farbod Taymouri and Josep Carmona. Model and event log reductions to boost the computation of alignments. In *Proceedings of the 6th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2016)*, pages 50–62, 2016.

[40] Farbod Taymouri, Marcello La Rosa, Marlon Dumas, and Fabrizio Maria Maggi. Business process variant analysis: Survey and classification, 2019.

[41] Nick R. T. P. van Beest, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Log delta analysis: Interpretable differencing of business process event logs. In *BPM*. Springer, 2015.

[42] Wil M. P. van der Aalst, Riccardo De Masellis, Chiara Di Francescomarino, and Chiara Ghidini. Learning hybrid process models from events - process discovery without faking confidence. In Josep Carmona, Gregor Engels, and Akhil Kumar, editors, *Business Process Management - 15th International Conference, BPM 2017, Barcelona, Spain, September 10-15, 2017,*

Proceedings, volume 10445 of *Lecture Notes in Computer Science*, pages 59–76. Springer, 2017.

[43] Seppe KLM vanden Broucke, Jorge Munoz-Gama, Josep Carmona, Bart Baesens, and Jan Vanthienen. Event-based real-time decomposed conformance analysis. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 345–363. Springer, 2014.

[44] Silke Wagner and Dorothea Wagner. *Comparing clusterings: an overview*. Universität Karlsruhe, Fakultät für Informatik Karlsruhe, 2007.

[45] Jochen De Weerdt, Seppe K. L. M. vanden Broucke, Jan Vanthienen, and Bart Baesens. Active trace clustering for improved process discovery. *IEEE Trans. Knowl. Data Eng.*, 25(12):2708–2720, 2013.

[46] Edwin B Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[47] M.T. Wynn, E. Poppe, J. Xu, A.H.M. ter Hofstede, R. Brown, A. Pini, and W.M.P. van der Aalst. Processprofiler3d: A visualisation framework for log-based process performance comparison. *DSS*, 100, 2017.