Log Delta Analysis: Interpretable Differencing of Business Process Event Logs

Nick R.T.P. van Beest[1,3], Marlon Dumas[2], Luciano García-Bañuelos[2], and Marcello La Rosa[3,1]

[1] NICTA, Australia
nick.vanbeest@nicta.com.au
[2] University of Tartu, Estonia
{marlon.dumas,luciano.garcia}@ut.ee
[3] Queensland University of Technology, Australia
m.larosa@qut.edu.au

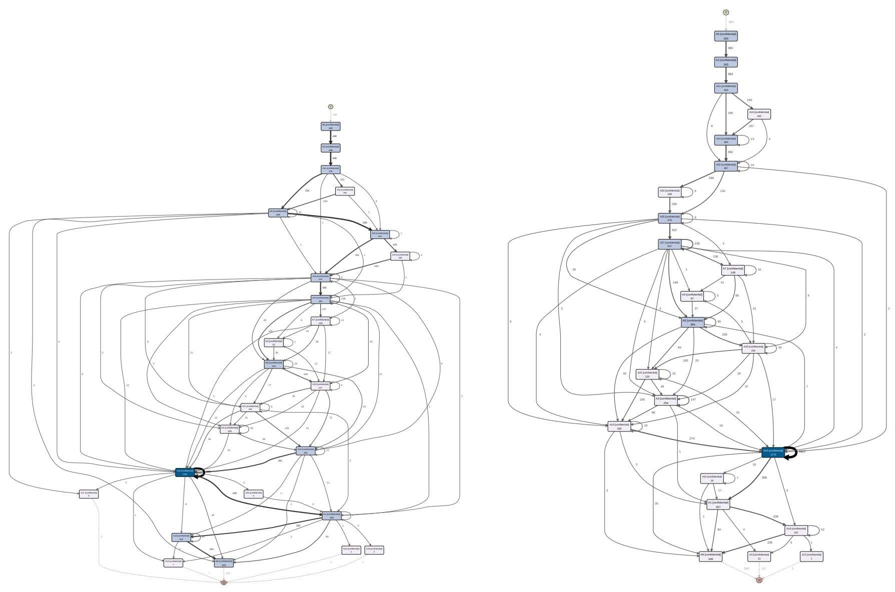
Abstract. This paper addresses the problem of explaining behavioral differences between two business process event logs. The paper presents a method that, given two event logs, returns a set of statements in natural language capturing behavior that is present or frequent in one log, while absent or infrequent in the other. This log delta analysis method allows users to diagnose differences between normal and deviant executions of a process or between two versions or variants of a process. The method relies on a novel approach to losslessly encode an event log as an event structure, combined with a frequency-enhanced technique for differencing pairs of event structures. A validation of the proposed method shows that it accurately diagnoses typical change patterns and can explain differences between normal and deviant cases in a real-life log, more compactly and precisely than previously proposed methods.

1 Introduction

Process mining is a family of methods to extract insights from business process execution logs. One problem type addressed by process mining methods is *deviance mining* [1]: understanding differences between executions that lead to a positive outcome vs. those that lead to a negative outcome, such as understanding what differentiates executions of a process that fulfill a service-level objective vs. those that violate it.

In previous case studies [2, 3], deviance mining has been approached using *model delta analysis*. The idea is to apply automated process discovery techniques to the traces of positive cases and to those of negative cases separately. The discovered process models are visually compared to identify distinguishing patterns. This approach does not scale up to complex logs. For example, Fig. 1 shows the models discovered by the Disco tool [4] for positive and negative cases of a patient treatment log at an Australian hospital – where a positive execution concerns a treatment that completes in less than a given timeframe. Manual comparison of these models is tedious and error-prone, calling for an automated method to distill differences that explain the observed deviance.

This paper approaches the problem of deviance mining via a *log delta analysis* operation defined as follows: *Given two event logs L_1 and L_2, explain the differences between the behavior observed in L_1 and that observed in L_2.* As the output is intended to inform business analysts, it should be compact and interpretable. Accordingly, the proposed method produces a set of simple statements, each capturing a behavior observed (frequently) in one log but not observed (or observed less frequently) in the other.



(a) Negative cases (b) Positive cases

Fig. 1: Model discovered from a hospital log for positive and negative cases

The proposal relies on a novel approach to losslessly encode an event log as an *event structure* [5]: a directed acyclic graph where nodes represent event occurrences sharing a common history. We enhance this representation with frequency information to capture how often a given event occurrence is observed in the log. Given the frequency-enhanced event structures of two event logs, the method calculates their differences based on an extended version of a technique for event structure differencing [6]. The latter step leads to a set of statements capturing behavior that is observed with some frequency in one log and with lower frequency (or not at all) in the other log.

The proposal has been evaluated on artificial logs capturing different types of change patterns [7] and combinations thereof, as well as on the hospital log from which the models in Fig. 1 are generated. The evaluation puts forward advantages of the proposed approach over an alternative method based on sequence classification.

The paper is structured as follows. Section 2 discusses related work and introduces event structures. Sections 3 and 4 present the construction of event structures from logs and their differencing for the purpose of log delta analysis. Section 5 discusses the evaluation while Section 6 draws conclusions.

2 Background and related work

This section discusses previous work on deviance mining and introduces the notion of event structure used in the rest of the paper.

2.1 Deviance mining

Approaches to deviance mining can be classified into two categories [1]: *model delta analysis* and *sequence classification*. As explained in Section 1, model delta analysis [2, 3, 8] requires manual comparison of automatically discovered process models. As such, it is error-prone and does not scale up to complex logs.

Sequence classification methods construct a classifier (e.g. a decision tree) that can determine with sufficient accuracy whether a given trace belongs to the positive or the negative class. The crux of these methods is the choice of features. In this respect, these methods fall into three categories: activity-based feature encoding, frequent sequence mining and discriminative sequence mining. In *activity-based feature encoding*, each trace is encoded as a vector containing one feature per activity referenced in the event log. The value of the feature corresponding to activity A is the number of times A appears in the trace. *Frequent sequence mining* methods [3, 9, 10] extract frequent patterns from the set of positive cases and that of negative cases separately. A possible pattern is that activity A occurs before activity B. Each pattern becomes a feature. The value of a feature for a trace is the number of times the pattern in question occurs in the trace. *Discriminative sequence mining* methods [11] operate similarly but extract patterns based on their discriminative power: a pattern is selected if it is a characteristic of positive cases but not of negative ones, or vice-versa.

In [1], we evaluated the above sequence classification methods on real-life logs. We found that a discriminative sequence mining method outperformed others (accuracy wise), but in all cases the obtained sets of rules were overly complex. For the patient treatment log in Section 1, between 106 and 130 rules are produced – each rule consisting of a conjunction of patterns possibly involving multiple activities. This observation motivates the development of a method to produce a compact set of statements explaining the differences between two groups of traces (e.g. positive vs. negative).

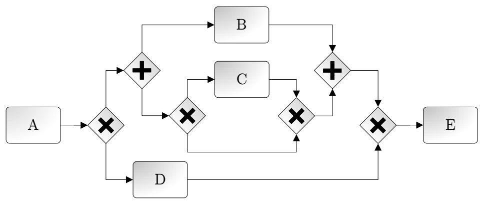
The problem of deviance mining could in theory be addressed based on the notion of behavioral profiles [12], where a process is represented via a matrix of behavioral relations between pairs of task labels. The idea would be to construct a behavioral profile for the normal cases and one for the deviant cases (as in model delta analysis), and then to calculate a difference between the two matrices. This approach however would be hindered by the fact behavioral profiles have limited expressive power: they sometimes fail to capture "task skipping" behavior and cannot distinguish an acyclic process with a similar process with an added cycle [12]. Thus the resulting difference statements would be incomplete. To avoid this limitation, the log delta analysis method adopts a more powerful representation of behavior, namely event structures.

2.2 Event structures

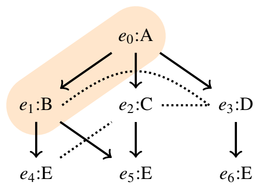
A Prime Event Structure (PES) [5] is a graph of events, where an event e represents the occurrence of an action (e.g. task) in the modeled system (e.g. business process). If a task occurs multiple times in a run, each occurrence is represented by a different event. The order of occurrence of events is defined via binary relations: i) *Causality* ($e < e'$) indicates that event e is a prerequisite for e'; ii) *Conflict* ($e \# e'$) implies that e and e' cannot occur in the same run; iii) *Concurrency* ($e \parallel e'$) indicates that no order can be established between e and e'.

Definition 1 (Labeled Prime Event Structure [5]). *A* Labeled Prime Event Structure *over the set of event labels $\mathscr{L}$ is the tuple $\mathscr{E} = \langle E, \leq, \#, \lambda \rangle$ where*
- *E is a set of events (e.g. tasks occurrences),*
- *$\leq \subseteq E \times E$ is a partial order, referred to as* causality*,*
- *$\# \subseteq E \times E$ is an irreflexive, symmetric* conflict *relation,*
- *$\lambda : E \rightarrow \mathscr{L}$ is a labeling function.*

We use $<$ to denote the irreflexive causality relation. The concurrency relation *of $\mathscr{E}$ is defined as $\parallel = E^2 \setminus (< \cup <^{-1} \cup \#)$. Moreover, the* conflict relation *satisfies the principle of conflict heredity, i.e. $e \# e' \wedge e' \leq e'' \Rightarrow e \# e''$ for $e, e', e'' \in E$.*



(a) BPMN



(b) Prime event structure $\mathscr{E}^1$

Fig. 2: Sample process model

For illustration, Fig. 10 presents side-by-side a BPMN process model and a corresponding PES $\mathscr{E}^1$. Nodes are labelled by an event identifier followed by the label of the represented task, e.g. "e_2:C" tells us that event e_2 represents an occurrence of task "C". The causality relation is depicted by solid arcs whereas conflict is depicted by dotted edges. For simplicity, transitive causal and hereditary conflict relations are not depicted. Every pair of events that are neither directly nor transitively connected are in a concurrency relation. Note that three different events refer to the task with label "E". This duplication is required to distinguish the different states where task "E" occurs.

A state of an event structure (a.k.a. *configuration*) is characterized by the set of events that have occurred so far. For example, set $\{e_0\text{:A}, e_1\text{:B}\}$ – highlighted in Fig. 2(b) – is the configuration where tasks A and B have occurred. In this configuration, event $\{e_3\text{:D}\}$ can no longer occur because it is in conflict with $\{e_1\text{:B}\}$. Meanwhile, events $\{e_2\text{:C}\}$ and $\{e_4\text{:E}\}$ can occur, but the occurrence of one precludes that of the other.

Definition 2 (Configuration). *Let $\mathscr{E} = \langle E, \leq, \#, \lambda \rangle$ be a prime event structure. A* configuration *of $\mathscr{E}$ is the set of events $C \subseteq E$ such that*

- *C is causally closed, i.e. $\forall e' \in E, e \in C : e' \leq e \Rightarrow e' \in C$, and*
- *C is conflict-free, i.e. $\forall e, e' \in C \Rightarrow \neg(e \# e')$.*

The local configuration *of an event $e \in E$ is the set $\lfloor e \rfloor = \{e' \mid e' \leq e\}$. Similarly, the* (set of) strict causes *of an event $e \in E$ is defined as $\lfloor e) = \lfloor e \rfloor \setminus \{e\}$.*

Set inclusion forms a partial order on configurations. We denote by $Conf(\mathscr{E})$ the set of all possible configurations of $\mathscr{E}$ and by $MaxConf(\mathscr{E})$ the subset of maximal configurations with respect to set inclusion. In the running example, $MaxConf(\mathscr{E}^1) = \{\{e_0, e_1, e_2, e_5\}, \{e_0, e_1, e_4\}, \{e_0, e_3, e_6\}\}$.

3 Constructing event structures from logs

In an event log, events are related via a total order induced by their timestamps.

Definition 3 (Event log, Trace). *Let L be an* event log *over the set of labels $\mathscr{L}$, i.e. $L \in \mathbb{B}(\mathscr{L}^*)$. Let E be a set of event occurrences and $\lambda : E \rightarrow \mathscr{L}$ a labelling function. An* event trace *$\sigma \in L$ is defined in terms of an order $i \in [0, n-1]$ and a set of events $E_\sigma \subseteq E$ with $|E_\sigma| = n$ such that $\sigma = \langle \lambda(e_0), \lambda(e_1), \ldots, \lambda(e_{n-1}) \rangle$.*

Trace	Ref	N
A B C E	t_1	3
A C B E	t_2	2
A B E	t_3	2
A D E	t_4	3

Fig. 3: Event log

Consider the event log in Fig. 3. The event log consists of 10 traces, with three instances of trace t_1 (cf. column "N"), two instances of t_2, etc. Herein, we write $\sigma = \langle \text{A,B,C,E} \rangle$ to refer to any of the three instances of t_1 such that $E_\sigma = \{e_0, e_1, e_2, e_3\}$ and $\{(e_0, \text{A}), (e_1, \text{B}), (e_2, \text{C}), (e_3, \text{E})\} \subset \lambda$.

To construct an event structure from a log, we start by transforming the log into a set of partially ordered runs by extracting concurrency relations between pairs of events. Several approaches have been proposed to extract concurrency relations between pairs of events from an event log [13, 14]. Here, we use the so-called *alpha concurrency* approach [14], but other approaches can be applied instead. Alpha concurrency is a relation over event labels appearing in a log. Specifically, two event labels a and b are alpha-concurrent if a is sometimes observed immediately after b and vice-versa.

Definition 4 (Alpha concurrency [14]). *Let L be an event log over the set of event labels $\mathscr{L}$ and $\sigma \in L$ be a log trace. A pair of tasks with labels $a, b \in \mathscr{L}$ are said to be in* alpha directly precedes relation*, denoted $A \prec_{\alpha(L)} B$, iff there exists a trace $\sigma = \langle \lambda(e_0), \lambda(e_1), \ldots, \lambda(e_{n-1}) \rangle$ in L, such that $A = \lambda(e_i)$ and $B = \lambda(e_{i+1})$. A pair of tasks $A, B \in \mathscr{L}$ are* alpha concurrent*, denoted $A \parallel_{\alpha(L)} B$, iff $A \prec_{\alpha(L)} B \wedge B \prec_{\alpha(L)} A$.*

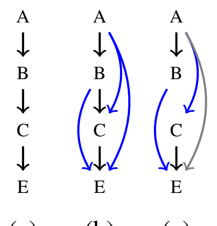
In the example log, B $\prec_\alpha$ C because of trace $t_1 = \langle$A,B,C,E$\rangle$ and C $\prec_\alpha$ B because of $t_2 = \langle$A,C,B,E$\rangle$, hence $B \parallel_\alpha C$. To construct partially ordered runs from traces, we take as input an oracle χ that defines a concurrency relation $\parallel_\chi$ over event occurrences (as opposed to event labels). Specifically, we use the concurrency relation $\parallel_\chi = \{(e, e') \mid \lambda(e) \parallel_{\alpha(L)} \lambda(e')\}$ for an event log L and its alpha concurrency relation $\parallel_{\alpha(L)}$. Given this, Def. 5 captures how traces are transformed into partially ordered runs.

Definition 5 (Transformation of a trace into a partially ordered run). *Let L be an event log over the set of event labels $\mathscr{L}$ and $\parallel_\chi$ be a concurrency relation. Moreover, let E be a set of event occurrences, $\lambda : E \to \mathscr{L}$ a labelling function. We say that event e_i* directly precedes *event e_{i+1}, denoted $e_i \lessdot e_{i+1}$, iff there exists a trace $\sigma = \langle \lambda(e_0), \lambda(e_1), \ldots, \lambda(e_{n-1}) \rangle$ in L with index $i \in [0, n-2]$. Thus, tuple $\pi = \langle E_\pi, \leq_\pi, \lambda_\pi \rangle$ is the* partially ordered run *corresponding to trace σ, induced by the concurrency relation $\parallel_\chi$ and the directly precedes relation $\lessdot$, where:*
- *E_π is the set of events occurring in σ,*
- *$\leq_\pi$ is the causality relation defined as $\leq_\pi = E_\pi^2 \cap (\lessdot^+ \setminus \parallel_\chi)^*$, and*
- *$\lambda_\pi : E_\pi \to \mathscr{L}$ is a labelling function, i.e. $\lambda_\pi = \lambda|_{E_\pi}$.*

$\Pi_\chi(L)$ is the set of partially ordered runs induced by $\parallel_\chi$ over the traces in L.

The crux of the transformation of a trace into a run is the computation of the causality relation $\leq_\pi$. Fig. 4 illustrates how this is done for $t_1 = \langle$A,B,C,E$\rangle$. First, Fig. 4(a) presents the direct precedes relation $\lessdot$ for t_1, which is directly derived from the sequential order in the event trace. Fig. 4(b) presents the (irreflexive) transitive closure of $\lessdot$, that is, $\lessdot^+$. Note that the blue edges in Fig. 4(b) correspond to the transitive relations.

The set difference $\lessdot^+ \setminus \parallel_\chi$ results in removing the edge connecting B with C as shown in Fig. 4(c). We can then remove the edge connecting A with E (shown in grey) by computing the transitive reduction of the causality relation. The concurrency relation $\parallel_\pi$ for a partially ordered run π is derived from $\leq_\pi$, i.e. $\parallel_\pi = E_\pi^2 \setminus (<_\pi \cup <_\pi^{-1})$. Observe that $\parallel_\pi$ coincides with $\parallel_\chi$.



(a) (b) (c)

Fig. 4: Transformation of t_1 into π_1

A partially ordered run resembles a prime event structure, with the exception that it does not have conflicting relations. This is because a run records the set of events that have actually occurred in an execution. Fig. 5 shows the set of partially ordered runs $\{\pi_1, \pi_2, \pi_3\}$ derived from the log in Fig. 3 and its corresponding alpha concurrency, where π_1 encodes the traces t_1 and t_2 and, therefore, is associated with 10 different cases. Similarly, π_2 encodes t_3, π_3 encodes t_4 and correspond to two and three cases respectively.

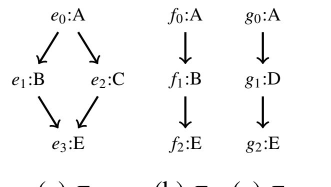


Fig. 5: Partially ordered runs of the event log in Fig. 3

The merging of runs $\Pi(L)$ to derive a prime event structure relies on an equivalence relation $\sim$. This relation partitions the set of events $E = \cup_{\pi \in \Pi(L)} E_\pi$, in a way that preserves the labelling of events as well as their "computation context". Labelling preserving implies that all the events in an equivalence class have the same label. The "computation context" is again related with a configuration. Informally, we require that if two events $e, e' \in E$ are equivalent, written $e \sim e'$, all events in the local configuration of e have an equivalent event in the local configuration of e'. As is customary, we write $[e]_\sim = \{e' \mid e \sim e'\}$ to denote the equivalence class of event e and for simplicity, we write $[S]_\sim = \{[e']_\sim \mid e \in S\}$ to denote the set of equivalence classes for all the events in the set S. The following definition formalizes the intuition above.

Definition 6 (Configuration-based prefix merging equivalence). *Let $e_i \in E_{\pi_i}$ and $e_j \in E_{\pi_j}$ be event occurrences in two different partially ordered runs. The* configuration-based prefix merging equivalence *is an equivalence relation $\sim$ over E, with the following properties:*

(i) $\sim$ is a reflexive, transitive and symmetric relation,
(ii) $e_i \sim e_j$ is label-preserving, i.e. $\lambda(e_i) = \lambda(e_j)$, and
(iii) $e_i \sim e_j$ is configuration preserving, i.e. $[\lfloor e_i)]_\sim = [\lfloor e_j)]_\sim$.

We now define a transformation to derive a prime event structure from an event log.

Definition 7 (Log-based Prime Event Structure). *Let L be an augmented event log. Let $\Pi(L)$ be its set of partially ordered runs. The* prime event structure *induced by equivalence relation $\sim$ is the tuple $\mathscr{E}(L)_\sim = \langle E_\sim, \leq_\sim, \#_\sim, \lambda_\sim \rangle$ s.t.*

– $E_\sim = \{ [e]_\sim \mid e \in \cup_{\pi \in \Pi(L)} E_\pi \}$,
– $\leq_\sim = \{ ([e]_\sim, [e']_\sim) \mid \exists \pi \in \Pi(L) : e \leq_\pi e' \}$,
– $\|_\sim = \{ ([e]_\sim, [e']_\sim) \mid \exists \pi \in \Pi(L) : e \|_\pi e' \}$,
– $\#_\sim = E_\sim^2 \setminus (\leq_\sim \cup \leq_\sim^{-1} \cup \|_\sim)$, *and*
– $\lambda_\sim = \{ ([e]_\sim, \lambda(e)) \mid [e]_\sim \in E_\sim \}$

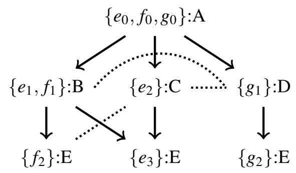
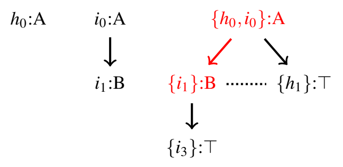


Fig. 6: PES induced by $\sim$ over the runs in Fig. 5

Let us now illustrate how the prime event structure for the set of runs in Fig. 5 is built. As usual, we assume that $\emptyset \in \sim$. It should be clear that $\{e_0, f_0, g_0\} \in \sim$: all those events share the label "A"; the events in the strict causes of each of those events form also an equivalence class (please consider that $\lfloor e_0) = \lfloor f_0) = \lfloor g_0) = \emptyset$); and the causality relation is preserved (this result is trivial because only one event has been considered so far). Note that $[e_0]_\sim = [f_0]_\sim = [g_0]_\sim = \{e_0, f_0, g_0\}$. Let us now consider the set of events sharing the label "B", namely $\{e_1, f_1\}$. Note that $\lfloor e_1) = \{e_0\}$ and $\lfloor f_1) = \{f_0\}$ and since $[e_0]_\sim = [f_0]_\sim$, we can conclude that $\{e_1, f_1\}$ is configuration preserving. Moreover, the equivalence class $\{e_1, f_1\}$ preserves causal order because $e_0 \leq_{\pi_1} e_1$ and $f_0 \leq_{\pi_2} f_1$. Fig. 6 depicts the entire PES induced by $\sim$ over the set of runs in Fig. 5. To further illustrate the concepts, let us consider the set of events sharing the label "E", namely $\{e_3, f_2, g_2\}$. Please note that the partition $\{e_3, f_2, g_2\}$ has to be refined because their corresponding configurations do not coincide, e.g. $[\lfloor e_3)]_\sim = \{[e_0]_\sim, [e_1]_\sim, [e_2]_\sim\}$ is different to $[\lfloor f_2)]_\sim = \{[e_0]_\sim, [e_1]_\sim\}$. Therefore, one equivalence class for each of those events is required, i.e. $\{[e_3]_\sim, [f_2]_\sim, [g_2]_\sim\} \subset \sim$. One can easily check that the PES in Fig. 6 is isomorphic to the PES in Fig. 2(b) and, hence, the sample event log could have been generated by executing the process model depicted in Fig. 2(a).

As formally proved later, Def. 7 ensures that each input run is represented as a configuration of the resulting PES. Unfortunately, it does not necessarily warranties that a run will be represented as a maximal configuration. To illustrate this issue, consider the event log $L^2 = \{\langle \text{A} \rangle, \langle \text{A,B} \rangle\}$. L^2 gives rise to two runs, namely π_4 and π_5, which are shown in Fig. 7. Moreover, the subgraph in red corresponds to $\mathscr{E}(L^2)$. One can easily verify that $Conf(\mathscr{E}(L^2) = \{\{[i_0]_\sim\}, \{[i_0]_\sim, [i_1]_\sim\}\}$ and $MaxConf(\mathscr{E}(L^2)) = \{\{[i_0]_\sim, [i_1]_\sim\}\}$. Since $h_0 \sim i_0$, we have that $\{[h_0]_\sim\}$ is also a configuration of $\mathscr{E}(L^2)$. Somehow, we can say that $\mathscr{E}(L^2)$ generalizes the behavior observed in L^2. In order to fix this limitation, we append an artificial end event to each trace in the input log, giving rise to an augmented log. Moreover, we use a special label, i.e. $\top$, to keep track of the artificial end events. Formally, for each $\sigma = \langle \lambda(e_0), \ldots, \lambda(e_{n-1}) \rangle$ from an event log L, we build a new trace $\widehat{\sigma} = \langle \widehat{\lambda}(e_0), \ldots, \widehat{\lambda}(e_{n-1}), \widehat{\lambda}(e_n) \rangle$ where e_n is a fresh event and $\widehat{\lambda} = \lambda \circ \{(e_n, \top)\}$. We write $\widehat{L}$ to refer to the augmented log of L. Fig. 7(c) presents the PES for log $\widehat{L^2}$. One can easily check that the artificial event h_1 preserves the maximality of the configuration corresponding to the run π_4. The following Theorem formalizes the intuition above and one of the major contributions of this work: $\mathscr{E}(\widehat{L})$ is a lossless representation of L.



(a) π_4 (b) π_5 (c) $\mathscr{E}(\widehat{L^2})_\sim$

Fig. 7: Runs and PES for $\widehat{L^2}$

Theorem 1 (Lossless representation). *Let $\Pi(\widehat{L})$ be the set of partially ordered runs of the augmented event log $\widehat{L}$, and $E = \cup_{\pi \in \Pi} E_\pi$ its corresponding set of events. Moreover, let $\mathscr{E}(\widehat{L})_\sim = \langle E_\sim, \leq_\sim, \#_\sim, \lambda_\sim \rangle$ be the prime event structure induced by the equivalent relation $\sim$.*

$$\textit{For every run } \pi \in \Pi(\widehat{L}) \textit{ it holds } [E_\pi]_\sim \in MaxConf(\mathscr{E}(\widehat{L})_\sim).$$

Proof. We first prove that $[E_\pi]_\sim$ is a configuration of $\mathscr{E}(\widehat{L})_\sim$.

- *(Causal closedness)* Take $e \in E_\pi$ and $f \in E \setminus E_\pi$ s.t. $e \sim f$. By Def. 6(iii), we have $[\lfloor e)]_\sim = [\lfloor f)]_\sim$, that is, all strict causes of event e form also an equivalence class $\sim$. Therefore, $[\lfloor e)]_\sim \subseteq E_\sim$ (cf. Def. 7). Recall $\lfloor e) = \{e' \mid e' <_\pi e\}$. Hence, $[E_\pi]_\sim$ is causally closed.
- *(Conflict freeness)* Take $e \in E_\pi$ and $f \in E \setminus E_\pi$ s.t. $\lambda(e) = \lambda(f)$ and $[\lfloor f)]_\sim \subseteq [\lfloor e)]_\sim$. Assume that $[f]_\sim \#_\sim [e]_\sim$. Recall $[E_\pi]_\sim$ is causally closed and hence consistent with $\leq_\sim$. Since $\|_\sim$ is derived from $\|_\pi$, by construction of $\#_\sim$, we require $[f]_\sim \neq [e]_\sim$ or equivalently $\neg(f \sim e)$. If $[\lfloor f)]_\sim = [\lfloor e)]_\sim$, by Def. 6(ii) it holds $f \sim e$, reaching contradiction. Conversely, if $[\lfloor f)]_\sim \neq [\lfloor e)]_\sim$, then it holds $\neg(f \sim e)$ and $[f]_\sim \notin [E_\pi]_\sim$. Hence, $[E_\pi]_\sim$ is conflict free.

Next, we prove by contradiction that $[E_\pi]_\sim$ is a maximal configuration of $\mathscr{E}(\widehat{L})_\sim$. Let $z \in E_\pi$ be the artificial end event of π. Assume there exists a run $\pi' \in \Pi(\widehat{L})$, with $z' \in E_{\pi'}$, s.t. $[E_\pi]_\sim \subseteq [E_{\pi'}]_\sim$, i.e. $[E_\pi]_\sim$ is not maximal w.r.t. $\subseteq$, and $[E_{\pi'}]_\sim \in Conf(E_\sim)$. Note that $[\lfloor z)]_\sim \subseteq E_\sim$ and also $[\lfloor z')]_\sim \subseteq E_\sim$. If $[\lfloor z)]_\sim = [\lfloor z')]_\sim$, then $z \sim z'$, which preserves maximality. Conversely, if $[\lfloor z)]_\sim \neq [\lfloor z')]_\sim$ (yet $[\lfloor z)]_\sim \subset [\lfloor z')]_\sim$), then $\neg(z \sim z')$ and $[z]_\sim \#_\sim [z']_\sim$. Moreover, if $\{[z]_\sim, [z']_\sim\} \subseteq [E_{\pi'}]_\sim$, then $[E_{\pi'}]_\sim$ is not conflict free and, therefore, not a configuration, reaching contradiction. Hence, $[E_\pi]_\sim$ is maximal. □

4 Comparing event structures

In this section, we describe our approach to identify and verbalize differences between two logs. These logs can concern two logs with variance, two logs from different organizations or one log with two classes: one regular, one deviant. The control-flow of both variants of the log is compared and verbalized in Section 4.1. Subsequently, the logs are transformed into a Frequency-enhanced Prime Event Structure (FPES) in Section 4.2, which allows to verbalize the branching frequency differences between the logs.

4.1 Control-flow comparison

In [6], we presented a technique for differencing pairs of event structures. This technique performs a *Partial Synchronized Product* (PSP) of the event structures, which is a synchronized simulation starting from the empty configurations. At each step, the events that can occur given the current configuration in each of the two event structures (i.e. the *enabled* events) are matched. If the events match, the simulation adds the matching events to the current configurations and continues. If an enabled event in the current configuration of one event structure does not match with an enabled event in the current configuration in the other event structure, a mismatch is declared. The unmatched event is "hidden" and the simulation jumps to the next matching configurations.

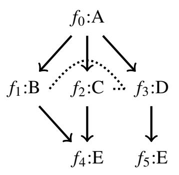
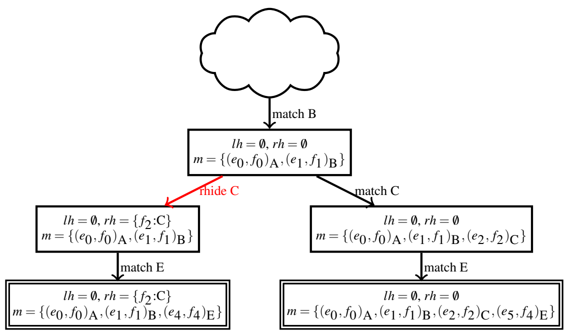


Fig. 8: PES $\mathscr{E}^2$



match B
$lh = \emptyset$, $rh = \emptyset$
$m = \{(e_0, f_0)_A, (e_1, f_1)_B\}$
rhide C
match C
$lh = \emptyset$, $rh = \{f_2$:C$\}$
$m = \{(e_0, f_0)_A, (e_1, f_1)_B\}$
$lh = \emptyset$, $rh = \emptyset$
$m = \{(e_0, f_0)_A, (e_1, f_1)_B, (e_2, f_2)_C\}$
match E
match E
$lh = \emptyset$, $rh = \{f_2$:C$\}$
$m = \{(e_0, f_0)_A, (e_1, f_1)_B, (e_4, f_4)_E\}$
$lh = \emptyset$, $rh = \emptyset$
$m = \{(e_0, f_0)_A, (e_1, f_1)_B, (e_2, f_2)_C, (e_5, f_4)_E\}$

Fig. 9: Fragment of PSP of the PESs in Figs. 2(b) and 3

Fig. 9 presents an excerpt of the PSP for $\mathscr{E}^1$ and $\mathscr{E}^2$, shown in Fig. 10 and Fig. 8 respectively. Note that $MaxConf(\mathscr{E}^2) = \{\{f_0, f_1, f_2, f_4\}, \{f_0, f_3, f_5\}\}$. Clearly, all maximal configurations of $\mathscr{E}^2$ can be matched to configurations of $\mathscr{E}^1$. The right-hand leaf node in the PSP illustrates the matching of configuration $\{e_0, e_1, e_2, e_5\}$ from $\mathscr{E}^1$ and $\{f_0, f_1, f_2, f_4\}$ from $\mathscr{E}^2$. There, the set m records the fact that all the events in both configurations have been matched, lh records that none of the events from $\mathscr{E}^1$ (the one to the left of the "product") has been hidden, and rh records that no event from $\mathscr{E}^2$ has been hidden. Similarly, the leaf node at the left-hand side corresponds to the best matching of configurations $\{e_0, e_1, e_4\}$ and $\{f_0, f_1, f_2, f_4\}$, respectively from $\mathscr{E}^1$ and $\mathscr{E}^2$. The cloud in the top indicates that some states precede to the matching of a pair of events sharing the label "B". The label on the edge from the cloud to the node just below records such matching. The configuration $\{e_0, e_1\}$ enables the occurrence of e_4:E, but that occurrence precludes the occurrence of e_2:C. This gives rise to a behavioral mismatch, that is resolved by hiding f_2:C. The red arrow in the PSP captures this hiding: the event f_2:C from $\mathscr{E}^2$ (right-hand side model in the product) is hidden. In the target box, *m* remains the same, i.e. no additional matching, whereas *rh* records the hiding of f_2:C. The reader is referred to [6] for further details on the technique.

In [7], a number of simple change patterns are catalogued (shown in Table 1). Each simple change will be used to identify the unique observation in the PSP, which is subsequently translated into a statement in natural language. However, change R3 concerns a branching frequency change instead of a control-flow change and will be discussed in Section 4.2. An overview of the control-flow changes is presented in Table 2.

Insertion	**R**esequentialization	**O**ptionalization
1. Add / remove	1. Loop	1. Parallel / sequence
2. Duplicate	2. Skip	2. Conditional / sequence
3. Substitute	3. Change branching frequency	3. Synchronize

Table 1: Change patterns applied to the base model to produce the variants.

I1	**PSP observation**: $match(\mathsf{a}) \rightarrow rhide(\mathsf{b})$, with no other branch showing $match(\mathsf{b})$. **Verbalization**: In variant 2, b occurs after a, while in variant 1 it does not.
I2	**PSP observation**: $match(\mathsf{a}) \rightarrow \ldots \rightarrow match(\mathsf{b}) \rightarrow rhide(\mathsf{a})$. **Verbalization**: In variant 2, after the occurrence of b, a is duplicated, while in variant 1 it is not.
I3	**PSP observation**: $match(\mathsf{a}) \rightarrow rhide(\mathsf{c}) \rightarrow lhide(\mathsf{b})$. **Verbalization**: In variant 2, after the occurrence of a, b is substituted by c.
R1	**PSP observation**: $match(\mathsf{a}) \rightarrow match(\mathsf{b}) \rightarrow rhide(\mathsf{a}) \rightarrow rhide(\mathsf{b})$. **Verbalization**: In variant 2, a is repeated multiple times, while in variant 1 it is not. In variant 2, b is repeated multiple times, while in variant 1 it is not.
R2	**PSP observation**: $lhide(\mathsf{a})$ in one branch and $match(\mathsf{a})$ in another branch. **Verbalization**: In variant 2, a can be skipped, while in variant 1 it cannot.
O1	**PSP observation**: $match(\mathsf{a}) \rightarrow lhide(\mathsf{b}) \rightarrow rhide(\mathsf{b})$. **PES observation**: (variant 1) $\mathsf{a} \parallel \mathsf{b}$; (variant 2) $\mathsf{a} \leq \mathsf{b}$. **Verbalization**: In variant 1, a and b are in parallel, while in variant 2, a precedes b.
O2	**PSP observation**: $rhide(\mathsf{a})$ in one branch and $lhide(\mathsf{b})$ in another branch. **Verbalization**: In variant 1, a precedes b, while in variant 2, a and b are mutually exclusive.
O3	**PSP observation**: $match(\mathsf{a}) \rightarrow match(\mathsf{b}) \rightarrow lhide(\mathsf{c}) \rightarrow rhide(\mathsf{c})$. **PES observation**: (variant 1) $\mathsf{a} \parallel \mathsf{b}$, $\mathsf{a} \parallel \mathsf{c}$, $\mathsf{b} \leq \mathsf{c}$; (variant 2) $\mathsf{a} \parallel \mathsf{b}$, $\mathsf{a} \leq \mathsf{c}$, $\mathsf{b} \leq \mathsf{c}$. **Verbalization**: In variant 1, a is in parallel with b and c, while in variant 2, a is in parallel with b.

Table 2: Translation from PSP observations to natural language expressions.

4.2 Frequency-enhanced comparison

In addition to control-flow variance, differences in branching probabilities are another type of variance that needs to be identified. To this end, we enhance the PES of a log with the branching frequencies as follows.

Definition 8 (Frequency-enhanced Prime Event Structure (FPES)). *Let $\mathscr{E}(L)_\sim = \langle E_\sim, \leq_\sim, \#_\sim, \lambda_\sim \rangle$ be the prime event structure induced by equivalence relation $\sim$ on the set of partially ordered runs $\Pi(L)$ of log L. A* frequency-enhanced prime event structure *is a tuple $\mathscr{F}(L)_\sim = \langle \mathscr{E}(L)_\sim, \mathscr{O}, \mathscr{P} \rangle$ where*

- *$\mathscr{O} : E \rightarrow \mathbb{N}$ is a function that associates an event $[e]_\sim$ with the number of times its event label occurs in the event log, and corresponds with the cardinality of the equivalence class, i.e. $\mathscr{O}([e]_\sim) = |[e]_\sim|$.*

– $\mathscr{P} : E \times E \to [0,1]$ *is a function that associates a pair of events* $[e_1]_\sim$ *and* $[e_2]_\sim$ *with the probability of occurrence of* $[e_2]_\sim$ *given that event* $[e_1]_\sim$ *has occurred. This function is defined as:*

$$\mathscr{P}([e_1]_\sim, [e_2]_\sim) = \begin{cases} \mathscr{O}([e_2]_\sim)/\mathscr{O}([e_1]_\sim) & \text{if } [e_1]_\sim <^{red}_\sim [e_2]_\sim \\ 0 & \text{Otherwise} \end{cases}$$

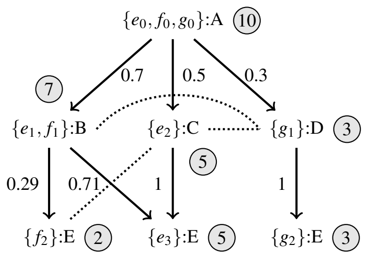


Fig. 10: FPES $\mathscr{F}(L)$

Fig. 10 presents the FPES for the log L that we have used as our running example. For each event in the graph there is a grey circle close to the event indicating the corresponding number of occurrences (i.e. $\mathscr{O}$) on the input log. For instance, event $\{e_2\}$:C occurs a total of five times. This value can be tracked back to the log as follows: e_2 comes from run π_1, which in turn comes from traces t_1 and t_2. Since t_1 and t_2 represent three and two cases each, we have a total of five occurrences. The branching frequency (i.e. $\mathscr{P}$) is also shown Fig. 10, with labels close to the edge representing a direct causal relation (i.e. $<^{red}_\sim$). Note that including transitive causal relations during the verbalization would result in a large number of difference statements. Most of them, however, would most likely be redundant. Therefore, we only consider direct causal relations.

Algorithm 1 Obtain frequency differences

```
1:  function OBTAINDIFFERENCES($\mathscr{F}_1$, $\mathscr{F}_2$)
2:      $diffSet \leftarrow \emptyset$
3:      $\bar{\lambda}_{\mathscr{F}_1} \leftarrow$ GETEVENTDEPTHLABELSET($\mathscr{F}_1$); $\bar{\lambda}_{\mathscr{F}_2} \leftarrow$ GETEVENTDEPTHLABELSET($\mathscr{F}_2$)
4:      $BP_1 \leftarrow$ GETAVERAGEBRANCHINGPROBABILITYSET($\mathscr{F}_1$, $\bar{\lambda}_{\mathscr{F}_1}$)
5:      $BP_2 \leftarrow$ GETAVERAGEBRANCHINGPROBABILITYSET($\mathscr{F}_2$, $\bar{\lambda}_{\mathscr{F}_2}$)
6:      for $e, e' \in E_{\mathscr{F}_1}$ s.t. $(\langle \bar{\lambda}_{\mathscr{F}_1}[e], \bar{\lambda}_{\mathscr{F}_1}[e'] \rangle \mapsto p_1) \in BP_1$ do
7:          for $f, f' \in E_{\mathscr{F}_2}$ s.t. $(\langle \bar{\lambda}_{\mathscr{F}_2}[f], \bar{\lambda}_{\mathscr{F}_2}[f'] \rangle \mapsto p_2) \in BP_2$ do
8:              if $\bar{\lambda}_{\mathscr{F}_1}[e] = \bar{\lambda}_{\mathscr{F}_2}[f] \wedge \bar{\lambda}_{\mathscr{F}_1}[e'] = \bar{\lambda}_{\mathscr{F}_2}[f'] \wedge p_1 \neq p_2$ then      ▷ Probability mismatch?
9:                  $diffSet \leftarrow diffSet \cup \{(\bar{\lambda}_{\mathscr{F}_1}[e], \bar{\lambda}_{\mathscr{F}_1}[e'], \bar{\lambda}_{\mathscr{F}_2}[f], \bar{\lambda}_{\mathscr{F}_2}[f'], p_1, p_2)\}$
10:             end if
11:         end for
12:     end for
13: end function
14: function GETAVERAGEBRANCHINGPROBABILITYSET($\mathscr{F}$, $\bar{\lambda}_{\mathscr{F}}$)
15:     $sums \leftarrow \emptyset$; $probs \leftarrow \emptyset$
16:     for $e, e' \in E_{\mathscr{F}}$ s.t. $e <^{red}_{\mathscr{F}} e'$ do
17:         $sums[\langle \bar{\lambda}_{\mathscr{F}}[e], \bar{\lambda}_{\mathscr{F}}[e'] \rangle] \leftarrow sums[\langle \bar{\lambda}_{\mathscr{F}}[e], \bar{\lambda}_{\mathscr{F}}[e'] \rangle] + \mathscr{P}_{\mathscr{F}}(e, e')$      ▷ $\emptyset$ is interpreted as 0
18:     end for
19:     for $e, e' \in E_{\mathscr{F}}$ s.t. $(\langle \bar{\lambda}_{\mathscr{F}}[e], \bar{\lambda}_{\mathscr{F}}[e'] \rangle \mapsto p) \in sums$ do
20:         $probs[\langle \bar{\lambda}_{\mathscr{F}}[e], \bar{\lambda}_{\mathscr{F}}[e'] \rangle] \leftarrow p \,/\, |\{e'' \in E_{\mathscr{F}} \mid e'' < e' \wedge \bar{\lambda}_{\mathscr{F}}[e''] = \bar{\lambda}_{\mathscr{F}}[e]\}|$
21:     end for
22:     return $probs$
23: end function
24: function GETEVENTDEPTHLABELSET($\mathscr{F}$)
25:     return $\{ \langle e \mapsto (\lambda_{\mathscr{F}}(e), |\{e' \mid e' \leq e \wedge \lambda_{\mathscr{F}}(e) = \lambda_{\mathscr{F}}(e')\}|) \rangle \mid e \in E_{\mathscr{F}} \}$
26: end function
```

The logs to be compared are each transformed into an FPES according to Def. 8. Subsequently, Algorithm 1 is used to obtain the set of frequency differences. As FPESs contain all event occurrences, repeated activities in a trace show up as separate events in the FPES with shared labels. Hence, we first create a set $\bar{\lambda}$ for each FPES, which holds each label along with the depth of the event occurrence in the respective run, determined based on the causality relation between each of these events (lines 3 and 25). As such, labels that are in conflict (and hence occur on different branches) will not be counted as consecutive occurrences. Function GETAVERAGEBRANCHINGPROBABILITYSET (lines 4, 5 and 14) calculates the average branching frequency of an activity based on the frequencies of occurrence of events. For instance, the branching activity may occur in multiple mutually exclusive branches, whereas the originating activity may also correspond to multiple event occurrences. As such, the frequencies of the respective event occurrences of a particular label are summed (line 17) and divided by the number of originating event occurrences that lead to an event with that label (line 20).

Next a set of difference statements can be created between events using the average frequency obtained. The differences are verbalized by referring to the frequency of branching between two activities in one variant, versus the same branching in the other variant. Consider the two event structures from Fig. 2(b) and Fig. 8, which we refer to as variant 1 and variant 2 respectively. We are interested in the frequency differences between event A and D. In variant 1, the frequency is 0.5, while in variant 2 the frequency is 0.7. This results in the following: $(\langle e_0 \mapsto [\mathrm{A},1]\rangle, \langle e_3 \mapsto [\mathrm{D},1]\rangle, \langle f_0 \mapsto [\mathrm{A},1]\rangle, \langle f_3 \mapsto [\mathrm{D},1]\rangle, 0.5, 0.7) \in \mathit{diffSet}$. Based on the *diffSet*, the branching frequency differences can be verbalized, as shown in Table 3.

R3	**FPES observation**: (variant 1) $(\mathsf{a},\mathsf{b}) = x_1$; (variant 2) $(\mathsf{a},\mathsf{b}) = x_2$. **Verbalization**: In variant 1, after the occurrence of a the branching frequency to b is <$x_1 * 100$>%, while in variant 2, after the occurrence of a the branching frequency to b is <$x_2 * 100$>%.

Table 3: Translation from FPES observations to natural language expressions.

Note that some differences between occurrence frequencies (R3) in the compared logs may be insignificant (cf. for example branching frequencies of 43.1% for variant 1 vs. 43.8% for variant 2). In addition, reported differences may include activities that only occur very rarely, e.g. in only 0.3% of all process instances. Accordingly, we apply a filter to the set of statements that removes those referring to frequency differences below a user-specified threshold.

Complexity analysis. The complexity of the approach has several elements. The transformation of the input event log into partially order runs (cf. Def. 5) is dominated by the computation of the transitive closure of the causality relation. This step has a complexity of $O(|\sigma_m|^3)$, where $|\sigma_m|$ is the length of the longest trace in the event log.

Prime event structures can be built using a partition-refinement approach via a breadth-first traversal of the graph induced by the direct causal relation of the event structure being computed. This traversal strategy guides the partitioning of events in a way that it eases the verification of the "preservation of configurations" required by $\sim$ (cf. Def. 6(ii)). Checking "preservation of configurations" is in worst case $O(|E|^2)$ and the breadth-first traversal is $O(|E| \cdot | \lesssim^{red} |)$. However, the complexity of this step is dominated by the computation of the transitive reduction of the causality relation in the resulting event structure, that is $O(| \lesssim^{red} |^3)$, which is used in subsequent steps.

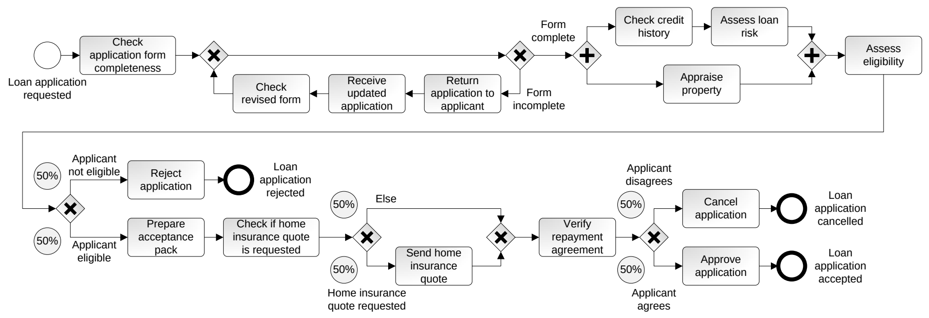
The above steps are polynomial. It turns out that the overall complexity of the log-delta analysis method is dominated by the computation of the PSP, which relies on the well-known A* heuristic search algorithm. The state space to be explored by A* is $O(3^{|Conf(E_1)| \cdot |Conf(E_2)|})$, since each configuration from E_1 is associated with each configuration from E_2 via three possible operations (i.e. *match*, *lhide* and *rhide*). However, this worst case complexity only occurs when the event structures are completely different. Conversely, when the event structures are identical, the heuristic search converges in linear time. In our setting, the input logs are expected to exhibit a high overlap in behavior and hence a reasonable performance.

5 Evaluation

We implemented the proposed method in the Apromore platform[4]. Using this implementation, we conducted a two-pronged validation. First, using synthetic datasets we assessed the method's ability to diagnose variations corresponding to typical process change patterns and combinations thereof. Second, using a real-life log, we qualitatively assessed the difference diagnosis produced by the method and compared it to rules produced using a sequence classification method.

5.1 Evaluation on synthetic logs

We generated synthetic logs by simulating BPMN process models using the BIMP simulator[5]. As a *base model*, we used a textbook example of a loan application process [15] comprising a representative set of control-flow patterns: sequence, choice, skipping, parallelism and repetition (cf. Fig. 11). Subsequently, we generated nine variants of this model by applying each of the simple change patterns (Table 1). We performed a 1000-traces simulation of the base model and each of its variants. Next, we applied the delta analysis method to compare the log of the base model against the log of each variant.



Loan application requested
Check application form completeness
Form complete
Form incomplete
Return application to applicant
Receive updated application
Check revised form
Check credit history
Assess loan risk
Appraise property
Assess eligibility
Applicant not eligible
50%
Reject application
Loan application rejected
50%
Applicant eligible
Prepare acceptance pack
Check if home insurance quote is requested
50%
Else
50%
Home insurance quote requested
Send home insurance quote
Verify repayment agreement
Applicant disagrees
50%
Cancel application
Loan application cancelled
50%
Applicant agrees
Approve application
Loan application accepted

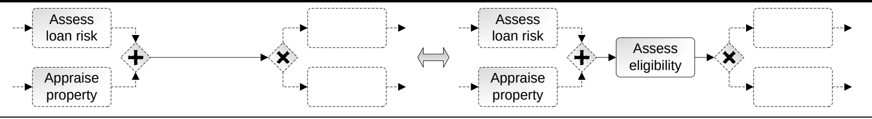
Fig. 11: Base model (branching probabilities are shown inside circles)

Next, we generated 6 logs by combining the 9 simple change patterns into composite (nested) changes. Specifically, we applied a randomly chosen change pattern from one of the three categories (say "I"), then nested a second pattern randomly chosen from another category (say "O") inside the fragment modified by the first pattern, and a third pattern randomly chosen from the last category ("R"). This led to one composite change (and corresponding log) for each permutation of the three categories. For example, a variant "IRO" was obtained by adding an activity ("Insert") then putting it in parallel with an existing activity ("Resequencing') and skipping the latter ("Optionalization").
Results: Table 4 shows the diagnosis produced for each of the nine variants corresponding to the simple changes. In all cases, the diagnosis matches the corresponding change pattern. Each diagnosis contains one statement per task affected by the change, e.g. in

[4] Available at `http://www.apromore.org/platform/tools`
[5] `http://bimp.cs.ut.ee`

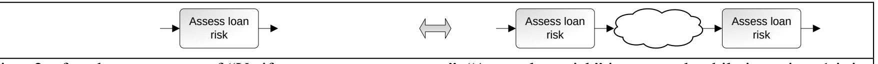
the case of R1 where the loop comprises three tasks, the diagnosis contains three statements. In R3, the branching frequencies in the diagnosis do not exactly match the ones in the BPMN diagrams, due to the stochastic nature of the simulation.

I1



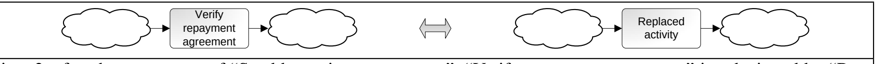
In variant 2, "Assess eligibility" occurs after "Assess loan risk" and "Appraise property", while in variant 1 it does not occur.

I2



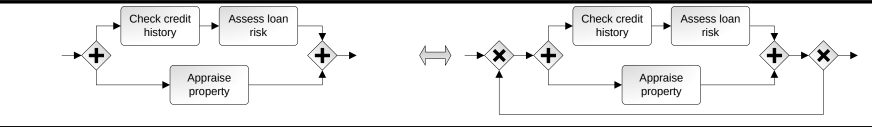
In variant 2, after the occurrence of "Verify repayment agreement", "Assess loan risk" is repeated, while in variant 1 it is not.

I3



In variant 2, after the occurrence of "Send home insurance quote", "Verify repayment agreement" is substituted by "Replaced activity".

R1

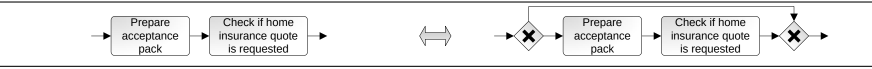


In variant 2, "Check credit history" is repeated multiple times, while in variant 1 it is not.

In variant 2, "Assess loan risk" is repeated multiple times, while in variant 1 it is not.

In variant 2, "Appraise property" is repeated multiple times, while in variant 1 it is not.

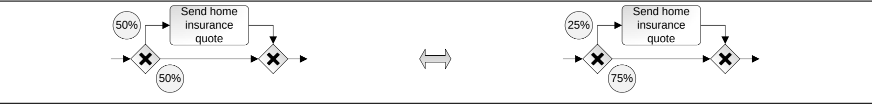
R2


Prepare acceptance pack
Check if home insurance quote is requested
Prepare acceptance pack
Check if home insurance quote is requested

In variant 2, "Prepare acceptance pack" can be skipped, while in variant 1 it cannot.

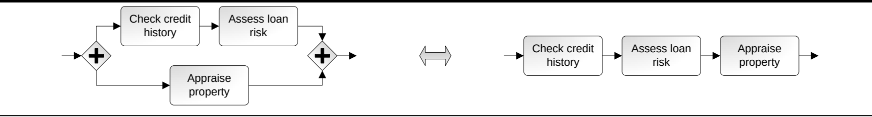
In variant 2, "Check if home insurance quote is requested" can be skipped, while in variant 1 it cannot.

R3



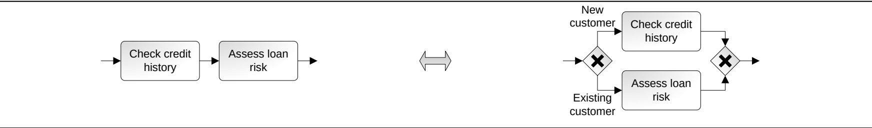
In variant 1, after the occurrence of "Check if home insurance quote is requested" the branching frequency to "Send home insurance quote" is 50.2%, while in variant 2, after the occurrence of "Check if home insurance quote is requested" the branching frequency to "Send home insurance quote" is 24.7%.

O1



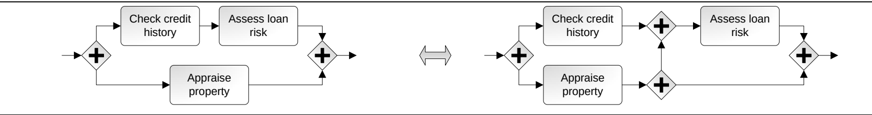
In variant 1, "Assess loan risk" and "Appraise property" are in parallel, while in variant 2, "Assess loan risk" precedes "Appraise property".

O2



In variant 1, "Check credit history" precedes "Assess loan risk", while in variant 2, "Check credit history" and "Assess loan risk" are mutually exclusive.

O3



In variant 1, "Appraise property" is in parallel with "Check credit history" and "Assess loan risk", while in variant 2, "Appraise property" is in parallel with "Check credit history".

Table 4: Simple changes and their verbalization.

Table 5 shows the diagnosis for three of the six composite changes. For space reasons, we omit the other composite changes (all results are packaged with the software tool). As expected, the composite changes lead to more difference statements than the simple changes (cf. Table 4), but in every case the diagnosis matches the corresponding change. Some difference statements refer to minor variations in frequencies (e.g. one branch is taken 48.1% of times in one variant and 49.2% in the other). This again comes from the stochastic nature of the simulation. Such spurious statements can be filtered by setting the frequency delta threshold to e.g. 10% (cf. Section 4.2).

Execution times: Each log comparison took between 10.06 and 11.18 seconds on a laptop with Intel i7 2.5GHz, running JVM 8 with 16GB of allocated memory.

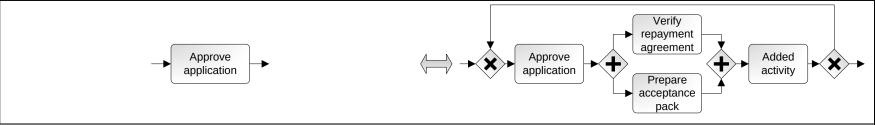
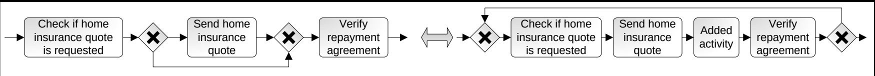
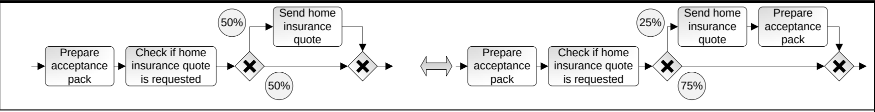
IRO	 Approve application Approve application Verify repayment agreement Prepare acceptance pack Added activity
In variant 2, after the occurrence of "Approve application", "Verify repayment agreement" is repeated, while in variant 1 it is not.	
In variant 2, "Prepare acceptance pack" is repeated after "Approve application", while in variant 1 it is not.	
In variant 2, "Added activity" occurs after "Verify repayment agreement" and "Prepare acceptance pack", while in variant 1 it does not occur.	
In variant 1, after the 1st occurrence of "Verify repayment agreement" the branching frequency to the 1st occurrence of "Approve application" is 48.1%; while in variant 2, after the 1st occurrence of "Verify repayment agreement" the branching frequency to the 1st occurrence of "Approve application" is 13.6%	
ORI	 Check if home insurance quote is requested Send home insurance quote Verify repayment agreement Check if home insurance quote is requested Send home insurance quote Added activity Verify repayment agreement
In variant 2 "Check if home insurance quote is requested" is repeated multiple times, while in variant 1 it is not.	
In variant 1 "Send home insurance quote" can be skipped, while in variant 2 it is always executed.	
In variant 2 "Send home insurance quote" is repeated multiple times, while in variant 1 it is not.	
In variant 2 "Verify payment agreement" is repeated multiple times, while in variant 1 it is not.	
In variant 2 "Added activity" occurs after "Send home insurance quote", while in variant 1 it does not occur.	
In variant 1, after the occurrence of "Check if home insurance quote is requested" the branching frequency to "Send home insurance quote" is 47.7%; while in variant 2, after the execution of "Check if home insurance quote is requested" the branching frequency to "Send home insurance quote" is 15.9%.	
In variant 1, after the 1st occurrence of "Verify repayment agreement" the branching frequency to the 1st occurrence of "Approve application" is 48.1%; while in variant 2, after the 1st occurrence of "Verify repayment agreement" the branching frequency to the 1st occurrence of "Approve application" is 32.0%.	
In variant 1, after the execution of "Send home insurance quote" the branching frequency to the 1st occurrence of "Verify repayment agreement" is 100.0%; while in variant 2, after the execution of "Send home insurance quote" the branching frequency to the 1st occurrence of "Verify repayment agreement" is 42.8%.	
In variant 1, after the 1st occurrence of "Verify repayment agreement" the branching frequency to the 1st occurrence of "Cancel application" is 51.9%; while in variant 2, after the 1st occurrence of "Verify repayment agreement" the branching frequency to the 1st occurrence of "Cancel application" is 23.2%.	
RIO	 50% Send home insurance quote Prepare acceptance pack Check if home insurance quote is requested 50% 25% Send home insurance quote Prepare acceptance pack Prepare acceptance pack Check if home insurance quote is requested 75%
In variant 2, after the occurrence of "Send home insurance quote", "Prepare acceptance pack" is repeated, while in variant 1 it is not.	
In variant 1, after the execution of "Check if home insurance quote is requested" the branching frequency to the 1st occurrence of "Verify repayment agreement" is 52.3%; while in variant 2, after the execution of "Check if home insurance quote is requested" the branching frequency to the 1st occurrence of "Verify repayment agreement" is 71.5%.	
In variant 1, after the 1st occurrence of "Verify repayment agreement" the branching frequency to the 1st occurrence of "Approve application" is 48.1%; while in variant 2, after the 1st occurrence of "Verify repayment agreement" the branching frequency to the 1st occurrence of "Approve application" is 49.2%.	
In variant 1, after the 1st occurrence of "Verify repayment agreement" the branching frequency to the 1st occurrence of "Cancel application" is 51.9%; while in variant 2, after the 1st occurrence of "Verify repayment agreement" the branching frequency to the 1st occurrence of "Cancel application" is 50.8%.	

Table 5: Composite changes and their verbalization.

5.2 Evaluation on real-life logs

We evaluated the method on the sub-logs of positive and negative cases of the patient treatment process discussed in Section 1 (Fig. 1). Variant 1 (448 cases, 7329 events) corresponds to the negative (slow) cases, while variant 2 (363 cases, 7496 events) corresponds to the positive cases. The logs cover a period of 1.5 years.

Results: An evaluation of sequence classification methods using this log is presented in [1], where it is shown that sequence classification methods require between 106 and 130 statements to explain the differences between these sub-logs. In contrast, our method requires 48 statements to explain all differences (without filtering) and 42 statements with a frequency delta threshold of 20%. Moreover, the statements produced by sequence classification approaches produce rules referring to the number of occurrences of a given event or event pattern in a variant, without specifying where exactly the difference occurs. For example, using the approach in [9], the following statements are produced – where "Nursing Progress Notes", "Nursing Primary Assessment", etc. refer to the number of occurrences of the corresponding tasks[6]:

- IF "Nursing Progress Notes" $>$ 7.5 THEN variant 1.
- IF "Nursing Progress Notes" $\leq$ 7.5 AND "Nursing Primary Assessment" $>$ 1.5 THEN variant 2.
- IF "Nursing Progress Notes" $\leq$ 5.5 AND "Pre Arrival Note" $\leq$ 0.5 AND "Blood tests" $\leq$ 1.5 THEN variant 2.

Meanwhile, our method produces statements that point to the exact state in the process where a behavioral difference occurs. For example:

- In variant 1, "Nursing Primary Assessment" is repeated after "Medical Assign Start" and "Triage Request", while in variant 2 it is not.
- In variant 2, "Blood tests" occurs after "Triage Request", while in variant 1 it does not occur.
- In variant 1, after the 1^{st} occurrence of "Pathology" the branching frequency to the 2^{nd} occurrence of "Nursing Primary Assessment" is 15.0%, while in variant 2, after the 1^{st} occurrence of "Pathology" the branching frequency to the 2^{nd} occurrence of "Nursing Primary Assessment" is 33.3%.

Execution time: The comparison of the two variants of the hospital log took 7.82 seconds, which is in the same order of magnitude as in the synthetic logs.

6 Conclusion

The paper presented a method for diagnosing the differences between two event logs via natural language statements capturing behavior present in one log but not in the other. This diagnostics is built on top of a lossless encoding of logs in the form of frequency-enhanced event structures. Based on this encoding, the method detects and diagnoses mismatching behavior, specifically: (i) events that occur in one log but not in the other; (ii) events occurring with different frequencies; (iii) events repeated in one log but not in the other; and (iv) behavioral relations that hold in one log but not in the other.

The validation on synthetic logs shows that the method accurately diagnoses typical change patterns, while the validation on a real-life log shows that it can explain differences between normal and deviant executions more compactly and precisely than sequence classification techniques considered in prior work (over 60% fewer statements).

[6] Even though the number of occurrences is always an integer, some rules contain decimals because the decision tree learning algorithm may use decimals as split thresholds.

A limitation of the method is that it does not fully recognize cyclic behavior. While the method detects that an activity occurs multiple times in traces of one log but not in those of the other, it does not identify the boundaries of cycles. This leads to multiple difference statements concerning the same cycle (cf. change R1 in Table 4). Another limitation is that the method treats the input log as consisting of sequences of event labels, ignoring timestamps and event payloads. Hence, directions for future work include designing cycle-aware, temporal and data-aware extensions of the method.

The current transformation from traces to partially ordered runs relies on the alpha concurrency oracle. While useful in relatively simple scenarios, this oracle can sometimes confuse concurrency with loops [14]. Another direction for future work is to comparatively evaluate this oracle against alternative ones such as the one in [13].

Finally, we plan to evaluate the method with domain experts so as to assess the usefulness of the generated statements for understanding deviance in practical scenarios.

Acknowledgments. NICTA is funded by the Australian Government. This research is funded by the Australian Research Council Discovery Project DP15010-3356 and the Estonian Research Council.

References

1. Nguyen, H., Dumas, M., La Rosa, M., Maggi, F.M., Suriadi, S.: Mining business process deviance: A quest for accuracy. In: OTM 2014, Springer (2014) 436–445
2. Suriadi, S., Wynn, M.T., Ouyang, C., ter Hofstede, A.H.M., van Dijk, N.J.: Understanding process behaviours in a large insurance company in Australia: A case study. In: CAISE 2013, Springer (2013) 449–464
3. Lakshmanan, G.T., Rozsnyai, S., Wang, F.: Investigating clinical care pathways correlated with outcomes. In: BPM 2013, Springer (2013) 323–338
4. Günther, C.W., Rozinat, A.: Disco: Discover your processes. In: BPM 2012 Demos, CEUR (2012) 40–44
5. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri Nets, Event Structures and Domains, Part I. TCS **13** (1981) 85–108
6. Armas, A., Baldan, P., Dumas, M., García-Bañuelos, L.: Behavioral comparison of process models based on canonically reduced event structures. In: BPM 2014, Springer (2014) 267–282
7. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features: Enhancing flexibility in process-aware information systems. DKE **66**(3) (2008) 438–466
8. Partington, A., Wynn, M.T., Suriadi, S., Ouyang, C., Karnon, J.: Process mining of clinical processes: Comparative analysis of four australian hospitals. ACM TMIS (2014) In press.
9. Bose, R.P.J.C., van der Aalst, W.M.P.: Abstractions in process mining: A taxonomy of patterns. In: BPM 2009. Springer (2009) 159–175
10. Swinnen, J., Depaire, B., Jans, M.J., Vanhoof, K.: A process deviation analysis – a case study. In: BPM 2012 Workshops, Springer (2012) 87–98
11. Lo, D., Cheng, H., Han, J., Khoo, S.C., Sun, C.: Classification of software behaviors for failure detection: A discriminative pattern mining approach. In: KDD 2009, ACM (2009) 557–566
12. Weidlich, M., Mendling, J., Weske, M.: Efficient Consistency Measurement Based on Behavioral Profiles of Process Models. IEEE TSE **37**(3) (2011) 410–429
13. Cook, J.E., Wolf, A.L.: Event-base detection of concurrency. In: FSE'1998, ACM (1998) 35–45
14. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE TKDE **16**(9) (2004) 1128–1142
15. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.: Fundamentals of Business Process Management. Springer (2013)