



Queensland University of Technology
Brisbane Australia

This is the author's version of a work that was submitted/accepted for publication in the following source:

van Beest, Nick R.T.P., Dumas, Marlon, García-Bañuelos, Luciano, & La Rosa, Marcello

(2015)

Log delta analysis: Interpretable differencing of Business process event logs.

This file was downloaded from: <http://eprints.qut.edu.au/83018/>

© Copyright 2015 The Author(s)

Notice: *Changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published source:*

Log Delta Analysis: Interpretable Differencing of Business Process Event Logs

Nick R.T.P. van Beest¹, Marlon Dumas², Luciano García-Bañuelos², 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 identifying and 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 logs of business process executions. One class of problems addressed by process mining is that of *deviance mining* [1]: detecting and explaining differences between executions that lead to a positive outcome vs. those that lead to a negative outcome. This includes, for example, explaining what differentiates executions of a process that fulfills a service-level objective vs. those that do not, or what differentiates executions of a sales process that lead to a purchase vs. those that do not.

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 traces 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 time-frame. Manual comparison of these models is tedious and error-prone, calling for an automated method to distill differences that may explain the observed deviance.

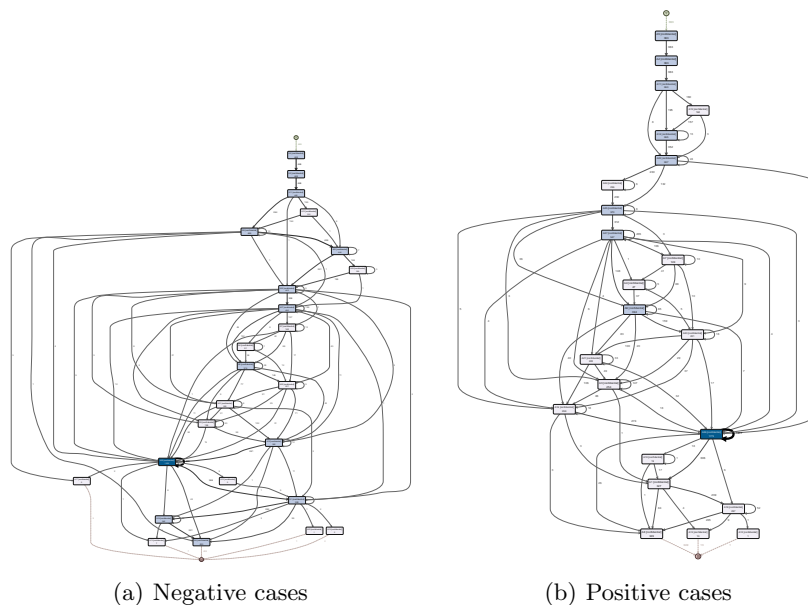


Fig. 1: Model discovered from a hospital log for positive and negative cases

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 in L_2 .* As the output is intended to inform business analysts, it is desirable that it is compact and interpretable. Accordingly, the paper proposes a log delta analysis method that 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.

The proposed method 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” (i.e. shared prefix). 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, but observed with lower frequency (or not at all) in the other log.

The proposal has been validated via artificial logs capturing different types of change patterns [7] as well as combinations thereof. The paper also reports on an evaluation on the hospital log from which the models in Fig. 1 are generated.

The paper is structured as follows. Section 2 discusses related work and introduces event structures. Section 3 presents the method, while Section 4 discusses its validation. Finally, Section 5 draws conclusions and discusses future work.

2 Background and related work

This section discusses previous work on deviance mining and introduces the notions of event structure and their differencing 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).

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 \mathcal{L} is the tuple $\mathcal{E} = \langle E, \leq, \#, \lambda \rangle$ 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,
- $\lambda : E \rightarrow \mathcal{L}$ is a labeling function.

We use $<$ to denote the irreflexive causality relation. The concurrency relation of \mathcal{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 $e, e', e'' \in E$.

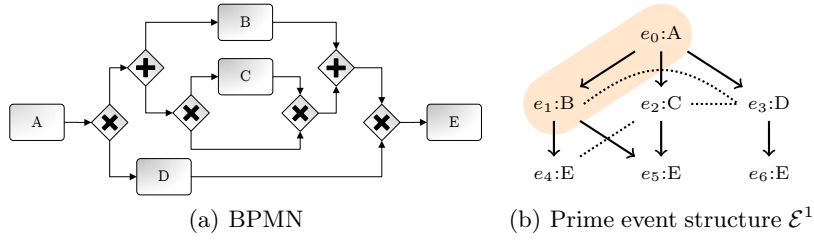


Fig. 2: Sample process model

For illustration, Fig. 11 presents side-by-side a BPMN process model and a corresponding PES \mathcal{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 the conflict relation is depicted by dotted edges. For the sake of 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 on an event structure (hereby called a *configuration*) is characterized by the set of events that have occurred so far. For instance, set $\{e_0:A, e_1:B\}$ – highlighted in Fig. 2(b) – is the configuration where tasks “A” and “B” have occurred. In this configuration, event $\{e_3:D\}$ can no longer occur because it is in conflict with $\{e_1:B\}$. On the other hand, events $\{e_2:C\}$ and $\{e_4:E\}$ can occur, but the occurrence of one precludes that of the other. Formally:

Definition 2 (Configuration). Let $\mathcal{E} = \langle E, \leq, \#, \lambda \rangle$ be a prime event structure. A configuration of \mathcal{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 $[e] = \{e' \mid e' \leq e\}$. Similarly, the (set of) strict causes of an event $e \in E$ is defined as $\llbracket e \rrbracket = [e] \setminus \{e\}$.

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

2.3 Comparison of event structures

In previous work, we presented a technique for differencing pairs of event structures [6]. This technique operates by performing a *Partial Synchronized Product* (PSP) of the event structures, which is in essence 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 they match, the simulation adds those 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 and this mismatch will be reflected in a *difference statement* that tells us that there

is a pair of matching configurations where an event can occur or a behavioral relation holds in one event structure, but not in the other. Having diagnosed the difference, the unmatched event is “hidden” and the simulation jumps to the next matching configurations.

Fig. 4 presents an excerpt of the PSP for \mathcal{E}^1 and \mathcal{E}^2 , shown in Fig. 11 and Fig. 3 respectively. Note that $MaxConf(\mathcal{E}^2) = \{\{f_0, f_1, f_2, f_4\}, \{f_0, f_3, f_5\}\}$. Clearly, all maximal configurations of \mathcal{E}^2 can be matched to configurations of \mathcal{E}^1 . The right-hand leaf node in the PSP illustrates the matching of configuration $\{e_0, e_1, e_2, e_5\}$ from \mathcal{E}^1 and $\{f_0, f_1, f_2, f_4\}$ from \mathcal{E}^2 . There, the set m records the fact that all the events in both configuration have been matched, lh records that none of the events from \mathcal{E}^1 (the one to the left of the “product”) has been hidden, and rh records that no event from \mathcal{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 \mathcal{E}^1 and \mathcal{E}^2 . The cloud in the top indicates that some states precedes 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 \mathcal{E}^2 (right-hand side model in the product) is hidden. Note that in the target box, m remains the same, i.e. no additional matching, whereas rh records now the hiding of f_2 :C. It is by composing the information in the states and edges associated to the moves “rhide C” and “match C” on the PSP that it becomes possible to diagnose that the “Task ‘C’ in model 1 can be skipped, whereas the same task is always executed in model 2”. The reader is referred to [6] for further details on the technique.

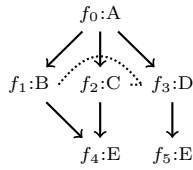


Fig. 3: PES \mathcal{E}^2

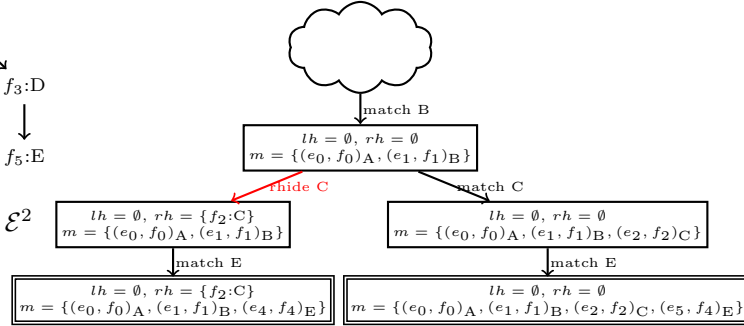


Fig. 4: Fragment of PSP of the PESs in Figs. 2(b) and 3

3 Log delta analysis method

In this section, we describe our approach to identify and verbalize differences between two logs. These logs can concern 2 logs with variance, 2 logs from different organizations or 1 log with 2 classes: 1 regular, 1 deviant.

First, the log files are categorized into two groups: regular logs and deviant logs. Subsequently, both the regular and deviant logs are transformed into a

Frequency-enhanced Prime Event Structure (FPES). An FPES represents, in addition to the Prime Event Structure, the branching frequencies between each pair of events in the log and a formal definition will be provided in Section 3.2. Finally, the FPESs are compared and the differences are verbalized into a set of statements, each indicating a difference between the regular and deviant logs. In Fig. 5, the approach is presented graphically.

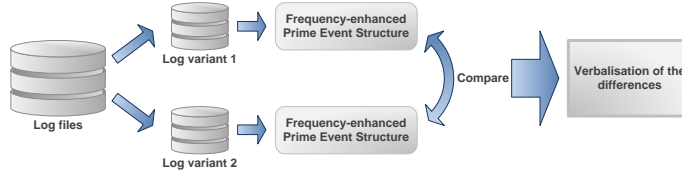


Fig. 5: Log delta analysis method

3.1 From logs to event structures

Event logs record the execution of a business process in the form of total orders. That is, tasks are recorded in sequences no matter if some tasks could have happened simultaneously. In this work, however, we advocate for an explicit representation of concurrency and, hence, use partial orders instead. We assume, however, that the concurrency relation observed on the set of tasks from a business process is given. Interestingly, several approaches to computing the concurrency relation have already been described in the literature [12, 13]. Here, we use the well-known alpha relations [13], but other approaches can be equally considered. Let us now introduce some additional notation.

Definition 3 (Event log, Trace). Let L be an event log over the set of labels \mathcal{L} , i.e. $L \in \mathbb{B}(\mathcal{L}^*)$. Let E be a set of event occurrences and $\lambda : E \rightarrow \mathcal{L}$ 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 $\sigma = \langle \lambda(e_0), \lambda(e_1), \dots, \lambda(e_{n-1}) \rangle$.

To illustrate the concepts, consider the event log presented in Fig. 6. The event log consists of a total of 10 traces, with 3 instances of trace t_1 (as specified in column “N”), 2 instances of t_2 , so on and so forth. To capture this fact, we should have used a notation like $e_{0,1,t_1}$ to denote the first event of the first instance of trace t_1 . For the sake of simplicity, however, we will only use the subscript associated to the order in σ , since the notation in the following refers only to a single trace. For instance, $\sigma = \langle A, B, C, E \rangle$ would refer to any of the three instances of t_1 , where $E_\sigma = \{e_0, e_1, e_2, e_3\}$ and $\{(e_0, A), (e_1, B), (e_2, C), (e_3, E)\} \subset \lambda$. Let us now turn our attention to alpha concurrency.

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. 6: Sample event log

Definition 4 (Alpha concurrency [13]). Let L be an event log over the set of event labels \mathcal{L} and $\sigma \in L$ be a log trace. A pair of tasks with labels $a, b \in \mathcal{L}$ are said to be in alpha directly precedes relation, denoted $A \prec_{\alpha(L)} B$, iff there exists a trace $\sigma = \langle \lambda(e_0), \lambda(e_1), \dots, \lambda(e_{n-1}) \rangle$ in L , such that $A = \lambda(e_i)$ and $B = \lambda(e_{i+1})$. We say that a pair of tasks $A, B \in \mathcal{L}$ are alpha concurrent, denoted $A \parallel_{\alpha(L)} B$, iff $A \prec_{\alpha(L)} B \wedge B \prec_{\alpha(L)} A$.

Note that alpha concurrency is defined over labels and not over event occurrences. Coming back to our example, we can see that $B \prec_\alpha C$ because of trace $t1 = \langle A, B, C, E \rangle$ and $C \prec_\alpha B$ because of trace $t2 = \langle A, C, B, E \rangle$. Hence $B \parallel_\alpha C$.

In the following, we assume there exists an oracle χ , which provides the concurrency relation \parallel_χ . For the purpose of the presentation, we will consider that $\parallel_\chi = \{(e, e') \mid \lambda(e) \parallel_{\alpha(L)} \lambda(e')\}$ for an event log L and its alpha concurrency relation $\parallel_{\alpha(L)}$. The following definition describes how, given a relation \parallel_χ , a trace can be transformed into a partially ordered run.

Definition 5 (Transformation of a trace into a partially ordered run).

Let L be an event log over the set of event labels \mathcal{L} and \parallel_χ be the concurrency relation provided by an oracle χ . Moreover, let E be a set of event occurrences, $\lambda : E \rightarrow \mathcal{L}$ a labelling function. We say that event e_i directly precedes event e_{i+1} , denoted $e_i \leq e_{i+1}$, iff there exists a trace $\sigma = \langle \lambda(e_0), \dots, \lambda(e_1), \dots, \lambda(e_{n-1}) \rangle$ in L with an order $i \in [0, n-1]$. Therefore, the 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_χ and the directly precedes relation \leq , where:

- E_π is the set of events occurring in σ ,
- \leq_π is the causality relation defined as $\leq_\pi = E_\pi^2 \cap (\leq^+ \setminus \parallel_\chi)^*$, and
- $\lambda_\pi : E_\pi \rightarrow \mathcal{L}$ is a labelling function, i.e. $\lambda_\pi = \lambda|_{E_\pi}$.

We write $\Pi_\chi(L)$ to denote the set of all partially ordered runs induced by \parallel_χ over the set of traces in L .

The critical issue in the transformation of a trace into a run is the computation of the causality relation \leq_π . Fig. 7 illustrates how this is done for $t_1 = \langle A, B, C, E \rangle$. First, Fig. 7(a) presents the direct precedes relation \leq for t_1 . Clearly, the relation is just a transposition of the inherent sequencing captured in the event trace. Fig. 7(b) presents the (irreflexive) transitive closure of \leq , that is, \leq^+ . Note that the blue edges in Fig. 7(b) correspond to the transitive relations. Just as for event structures, concurrency can be represented by the absence of edges (i.e. absence of order). Therefore, the set difference $\leq^+ \setminus \parallel_\chi$ will result in removing the edge connecting “B” with “C” as shown in Fig. 7(c). For the sake of readability, we can remove the edge connecting “A” with “E” (shown in grey color), which would correspond with the transitive reduction of the causality relation. The concurrency relation \parallel_π for a partially ordered run π can be derived from \leq_π , i.e. $\parallel_\pi = E_\pi^2 \setminus (\leq_\pi \cup \leq_\pi^{-1})$. Clearly, \parallel_π coincides with \parallel_χ .

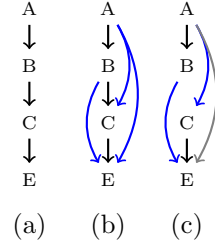


Fig. 7: Transformation of t_1 into π_1

A partially order run resembles a prime event structure, with the exception of not having any conflicting events. Clearly, the notion of configuration can be straightforwardly transferred to partially order runs. The absence of conflicts can be explained by the fact that a trace records the set of events that have actually been occurred in a computation. Fig. 8 presents the set of partially ordered runs $\{\pi_1, \pi_2, \pi_3\}$ that can be derived from the event log shown in Fig. 6 and its

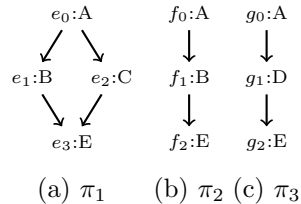


Fig. 8: Partially ordered runs of the event log in Fig. 6

corresponding alpha concurrency. Please note that 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 2 and 3 cases respectively. Without loss of generality, the number of cases associated with each event is not explicitly referred until Section 3.2.

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 the events in the local configuration of e have an equivalent event in the local configuration of e' . As is customary, we will use $[e]_\sim = \{e' \mid e \sim e'\}$ to denote the equivalence class of event e . With abuse of notation, we will use $[S]_\sim = \{[e']_\sim \mid e \in S\}$ 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. $[[e_i]_\sim]_\sim = [[e_j]_\sim]_\sim$.

We now formally 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 $\mathcal{E}(L)_\sim = \langle E_\sim, \leq_\sim, \#_\sim, \lambda_\sim \rangle$ s.t.

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

Let us now illustrate how the prime event structure for the set of runs in Fig. 8 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 $[e_0]_\sim = [f_0]_\sim = [g_0]_\sim = \emptyset$); 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 $[e_1]_\sim = \{e_0\}$ and $[f_1]_\sim = \{f_0\}$ 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. 9 depicts the entire PES induced

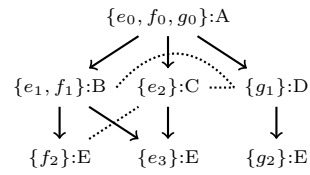


Fig. 9: PES induced by \sim over the runs in Fig. 8

by \sim over the set of runs in Fig. 8. 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. $[[e_3]]_{\sim} = \{[e_0]_{\sim}, [e_1]_{\sim}, [e_2]_{\sim}\}$ is different to $[[f_2]]_{\sim} = \{[e_0]_{\sim}, [e_1]_{\sim}\}$. 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. 9 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).

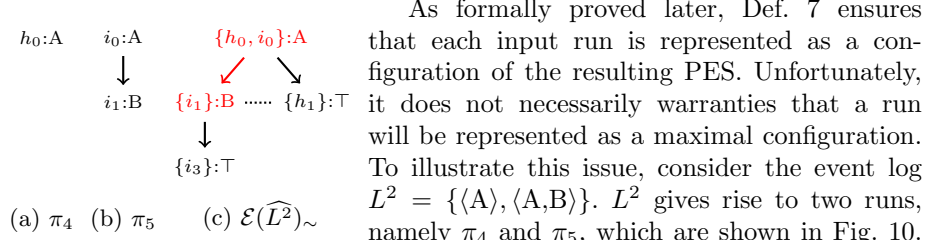


Fig. 10: Runs and PES for $\widehat{L^2}$. Moreover, the subgraph presented in Fig. 10 in red color corresponds to $\mathcal{E}(L^2)$. One can easily verify that $Conf(\mathcal{E}(L^2)) = \{\{[i_0]_{\sim}\}, \{[i_0]_{\sim}, [i_1]_{\sim}\}\}$ and $MaxConf(\mathcal{E}(L^2)) = \{\{[i_0]_{\sim}, [i_1]_{\sim}\}\}$. Since $h_0 \sim i_0$, we have that $\{[h_0]_{\sim}\}$ is also a configuration of $\mathcal{E}(L^2)$. Somehow, we can say that $\mathcal{E}(L^2)$ generalizes the behavior observed in L^2 . In order to fix this limitation, we append a fresh 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 $\sigma = \langle \lambda(e_0), \dots, \lambda(e_{n-1}) \rangle$ from an event log L , we build a new trace $\widehat{\sigma} = \langle \widehat{\lambda}(e_0), \dots, \widehat{\lambda}(e_{n-1}), \widehat{\lambda}(e_n) \rangle$ 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. 10(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: $\mathcal{E}(\widehat{L})$ is a lossless representation of L .

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 $\mathcal{E}(\widehat{L})_{\sim} = \langle E_{\sim}, \leq_{\sim}, \#_{\sim}, \lambda_{\sim} \rangle$ be the prime event structure induced by the equivalent relation \sim .*

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

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

- (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 $[[e]]_{\sim} = [[f]]_{\sim}$, that is, all strict causes of event e form also an equivalence class \sim . Therefore, $[[e]]_{\sim} \subseteq E_{\sim}$ (cf. Def. 7). Recall $|e\rangle = \{e' \mid e' <_{\pi} e\}$. 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 $[[f]]_{\sim} \subseteq [[e]]_{\sim}$. 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 $[[f]]_{\sim} = [[e]]_{\sim}$, by Def. 6(ii) it holds $f \sim e$, reaching contradiction. Conversely, if $[[f]]_{\sim} \neq [[e]]_{\sim}$, then it holds $\neg(f \sim e)$ and $[f]_{\sim} \notin [E_{\pi}]_{\sim}$. Hence, $[E_{\pi}]_{\sim}$ is conflict free.

Next, we prove that $[E_\pi]_\sim$ is a maximal configuration of $\mathcal{E}(\widehat{L})_\sim$. Let $z \in E_\pi$ be the artificial end event of π . We prove by contradiction. Assume there exists a run $\pi' \in \Pi(\widehat{L})$, with $z' \in E_{\pi'}$ being the corresponding artificial end event, s.t. $[E_\pi]_\sim \subseteq [E_{\pi'}]_\sim$, i.e. $[E_\pi]_\sim$ is not maximal w.r.t. \subseteq , and $[E_{\pi'}]_\sim \in \text{Conf}(E_\sim)$. Note that $[[z]]_\sim \subseteq E_\sim$ and also $[[z']]_\sim \subseteq E_\sim$. If $[[z]]_\sim = [[z']]_\sim$, then $z \sim z'$, which preserves maximality. Conversely, if $[[z]]_\sim \neq [[z']]_\sim$ (yet $[[z]]_\sim \subset [[z']]_\sim$), 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. \square

3.2 Enhancing event structures with frequencies

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 $\mathcal{E}(L)_\sim = \langle E_\sim, \leq_\sim, \#_\sim, \lambda_\sim \rangle$ 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 $\mathcal{F}(L)_\sim = \langle \mathcal{E}(L)_\sim, \mathcal{O}, \mathcal{P} \rangle$ where

- $\mathcal{O} : E \rightarrow \mathbb{N}$ 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. $\mathcal{O}([e]_\sim) = |[e]_\sim|$.
- $\mathcal{P} : E \times E \rightarrow [0, 1]$ 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:

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

Fig. 11 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. \mathcal{O}) on the input log. For instance, event $\{e_2\}:D$ occurs a total of 5 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 3 and 2 cases each, we have a total of 5 occurrences. The branching frequency (i.e. \mathcal{P}) is also shown Fig. 11, with labels close to the edge representing a direct causal relation (i.e. $<_{\sim}^{red}$). Note that including transitive causal relations during the verbalization would result in a large number of difference statements. Most of them, however, would be most likely redundant. That is why we only consider direct causal relations.

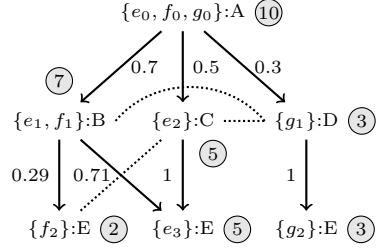


Fig. 11: FPES $\mathcal{F}(L)$

3.3 Comparison of frequency-enhanced prime event structures

The variant logs to be compared are each transformed into an FPES created according to Definition 8. Subsequently, Algorithm 1 is used to obtain the set of

frequency differences. As FPESs contain all event occurrences, repeated activities in a trace will occur separately in the FPES, while sharing the same label. Therefore, 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 to an activity, based on the frequencies of the event occurrences. 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 to 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).

Algorithm 1 Obtain frequency differences

```

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

```

Now a set of differences 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 different event structures from Fig. 2(b) and 3, 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 would result in the following: $(\langle e_0 \mapsto [A, 1] \rangle, \langle e_3 \mapsto [B, 1] \rangle, \langle f_0 \mapsto [A, 1] \rangle, \langle f_3 \mapsto [B, 1] \rangle, 0.5, 0.7) \in diffSet$. In our tool, this would be verbalized as follows: “*In variant 1, after the execution of A the frequency of branching to D is 0.5, while in variant 2, after the execution of A the frequency of branching to D is 0.7*”.

Note that some differences between occurrence frequencies 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 produced set of difference statements that removes those referring to frequency differences below a user-specified threshold.

4 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.

4.1 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 [14] that contains a representative set of control-flow patterns: sequence, choice, “task skipping”, parallelism and repetition (cf. Fig. 12). We generated 9 variants of this base model by manually introducing 1 of 9 simple change pattern into it, which are catalogued in [7]. These patterns span across 3 categories: “Insertion”, “Resequentialization” and “Optionalization” as shown in Table 1. We performed a 1000-traces simulation of the base model and each of its 9 variants. We then applied the proposed log delta analysis method to compare the log of the base model against the log of each of the 9 variants.

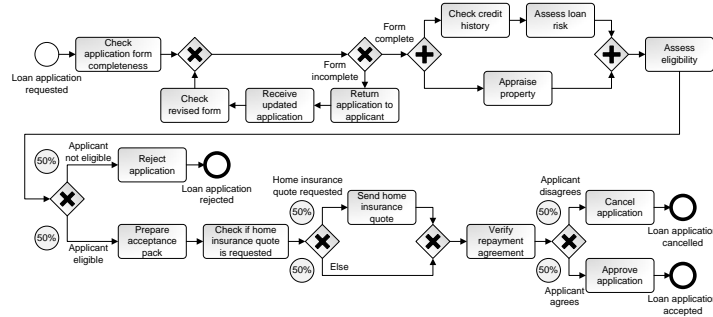


Fig. 12: Base model (branching probabilities are shown inside circles)

Insertion	Resequentialization	Optionalization
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.

Next, we generated 6 logs by combining the 9 simple change patterns into a composite (nested) changes. Specifically, we applied a randomly chosen change

⁴ Available at <http://www.apromore.org/platform/tools>

⁵ <http://bimp.cs.ut.ee>

	<p>I1 In variant 2, “Assess eligibility” occurs after “Assess loan risk” and “Appraise property”, while in variant 1 it does not occur.</p>
	<p>I2 In variant 2, “Assess loan risk” is repeated before “Approve application”, while in variant 1 it is not.</p>
	<p>I3 In variant 2, “Verify repayment agreement” is substituted by “Replaced activity”.</p>
	<p>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.</p>
	<p>R2 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.</p>
	<p>R3 In variant 1, the branching frequency to “Send home insurance quote” is 50.2%, while in variant 2, the branching frequency to “Send home insurance quote” is 24.7%.</p>
	<p>O1 In variant 1, “Assess loan risk” and “Appraise property” are in parallel, while in variant 2, “Assess loan risk” precedes “Appraise property”.</p>
	<p>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.</p>
	<p>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”.</p>

Table 2: Simple changes and their verbalization.

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 again 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 a new activity (“Insert”) then putting it in parallel with an existing activity (“Resequencing”) and then skipping the latter (“Optionalization”).

Results: Table 2 shows the diagnosis produced by the tool for each of the 9 variants corresponding to the simple changes. In all cases, the difference diagnosis matches the corresponding change pattern. Each diagnosis contains one state-

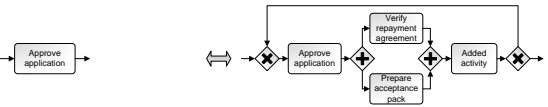
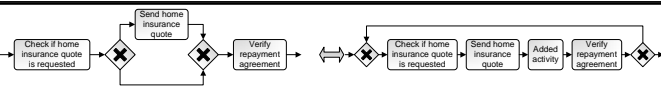
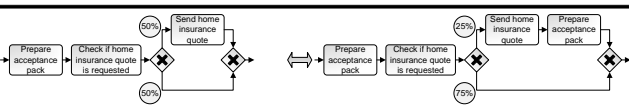
	
<p>IRO</p> <p>In variant 2, "Verify repayment agreement" is repeated after "Approve application", while in variant 1 it is not.</p> <p>In variant 2, "Prepare acceptance pack" is repeated after "Approve application", while in variant 1 it is not.</p> <p>In variant 2, "Added activity" occurs after "Verify repayment agreement" and "Prepare acceptance pack", while in variant 1 it does not occur.</p> <p>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%</p>	
<p>ORI</p>  <p>In variant 2 "Check if home insurance quote is requested" is repeated multiple times, while in variant 1 it is always executed.</p> <p>In variant 1 "Send home insurance quote" can be skipped, while in variant 1 it is always executed.</p> <p>In variant 2 "Send home insurance quote" is repeated multiple times, while in variant 1 it is always executed.</p> <p>In variant 2 "Verify payment agreement" is repeated multiple times, while in variant 1 it is always executed.</p> <p>In variant 2 "Added activity" occurs after "Send home insurance quote", while in variant 1 it does not occur.</p> <p>In variant 1, after the execution 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%.</p> <p>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%.</p> <p>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%.</p> <p>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%.</p>	
<p>RIO</p>  <p>In variant 2, "Prepare acceptance pack" is repeated after "Send home insurance quote", while in variant 1 it is not.</p> <p>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%.</p> <p>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%.</p> <p>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%.</p>	

Table 3: Composite changes and their verbalization.

ment per task affected by the change, e.g. in the case of R1 where 3 tasks are put in a loop, the diagnosis contains 3 statements. Note that in R3, the branching frequencies in the diagnosis do not exactly match the ones in the BPMN diagrams. This is due to the stochastic nature of the simulation (the actual frequencies in the logs do not match exactly the branching probabilities in the model).

Table 3 shows the difference statements 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 2), but in every case the diagnosis matches the corresponding change. We observe that 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 3.3).

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.

4.2 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 Sect. 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.

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 tasks⁶:

- IF “Nursing Progress Notes” > 7.5 THEN variant 1.
- IF “Nursing Progress Notes” ≤ 7.5 AND “Nursing Primary Assessment” > 1.5 THEN variant 2.
- IF “Nursing Progress Notes” ≤ 5.5 AND “Pre Arrival Note” ≤ 0.5 AND “Blood tests” ≤ 1.5 THEN variant 2.

Our method, on the other hand, 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 1st occurrence of “Pathology” the branching frequency to the 2nd occurrence of “Nursing Primary Assessment” is 15.0%, while in variant 2, after the 1st occurrence of “Pathology” the branching frequency to the 2nd 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.

5 Conclusion

The paper presented a method for diagnosing the differences between two event logs in the form of natural language statements capturing behavior present in one log but not in the other. This difference 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

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

explain differences between normal and deviant executions more compactly and precisely than using sequence mining techniques as proposed in previous work.

One limitation of the method is that it does not fully recognize cyclic behavior. Indeed, 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 being produced when cycles are at stake (cf. change R1 in Table 2). 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.

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. Cook, J.E., Wolf, A.L.: Event-base detection of concurrency. In: FSE’1998, ACM (1998) 35–45
13. 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
14. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.: Fundamentals of Business Process Management. Springer (2013)