Visual Drift Detection for Event Sequence Data of Business Processes

Anton Yeshchenko, Claudio Di Ciccio, Jan Mendling, and Artem Polyvyanyy

Abstract—Event sequence data is increasingly available in various application domains, such as business process management, software engineering, or medical pathways. Processes in these domains are typically represented as process diagrams or flow charts. So far, various techniques have been developed for automatically generating such diagrams from event sequence data. An open challenge is the visual analysis of drift phenomena when processes change over time. In this paper, we address this research gap. Our contribution is a system for fine-granular process drift detection and corresponding visualizations for event logs of executed business processes. We evaluated our system both on synthetic and real-world data. On synthetic logs, we achieved an average F-score of 0.96 and outperformed all the state-of-the-art methods. On real-world logs, we identified all types of process drifts in a comprehensive manner. Finally, we conducted a user study highlighting that our visualizations are easy to use and useful as perceived by process mining experts. In this way, our work contributes to research on process mining, event sequence analysis, and visualization of temporal data.

Index Terms—Sequence data, Visualization, Temporal data, Process mining, Process drifts, Declarative process models.

1 Introduction

Event sequence data is increasingly available in various application domains and the design of suitable analysis techniques is an ongoing research challenge. Research by Aigner et al. [2], [3], provides an excellent overview of time-oriented visualizations concluding that most available techniques plot temporal data in a *continuous* way. Examples of this visualization type are the Time Line Browser [4], History Flow [5], ThemeRiver [6], and TimeNets [7]. Various domains such as business process management, software engineering, and medical pathways use process diagrams, flow charts, and similar models to describe temporal relations between *discrete* activities and events [8]. Techniques from process mining are concerned with generating such visual models from event sequence data [9].

Business process management is a discipline concerned with organizing activities and events in an efficient and effective way [10]. To this end, business processes are analyzed, designed, implemented, and monitored. A business process in this context can be a travel request or an online order of a textbook. *Event sequence data* plays an important role in process analysis. An individual case of a textbook order by the first author on the 4th of April is also referred to as a *trace*, and a multiset of such traces is called an *event log*. In process mining, process discovery algorithms have proven to be highly effective in generating process models from event logs of stable behavior [9]. However, many processes are not stable but change over time. In data mining, such change over time is called *drift*. Furthermore, to the detriment of *process analysts*, drift is a concept that has been addressed only to a limited extent in BPM.

- *Anton Yeshchenko and Jan Mendling are with the Vienna Universtiy of Economics and Business, Vienna, Austria*
  E-mails: anton.yeshchenko@wu.ac.at, jan.mendling@wu.ac.at
- *Claudio Di Ciccio is with the Sapienza University of Rome.*
  E-mail: claudio.diciccio@uniroma1.it
- *Artem Polyvyanyy is with the University of Melbourne.*
  E-mail: artem.polyvyanyy@unimelb.edu.au

Recent works have focused on integrating ideas from research on concept drift into process mining [11], [12], [13], [14], [15]. The arguably most advanced technique is proposed in [16], where Maaradji et al. present a framework for detecting process drifts based on tracking behavioral relations over time using statistical tests. A strength of this approach is its statistical soundness and ability to identify a rich set of drifts, making it a suitable tool for validating if an intervention at a known point in time has resulted in an assumed change of behavior. However, a key challenge remains. In practice, the existence of different types of drifts in a business process is not known beforehand, and analysts are interested in distinguishing what has and what has not changed over time. This need calls for a more fine-granular analysis as compared to what recent techniques have offered.

In this paper, we present a *design study* [17] on how to support process analysts with visualizations to better understand drift phenomena [18] associated with business processes. Specifically, we develop a novel system for process drift detection, named Visual Drift Detection (VDD), which addresses the identified research gap. Our system aims to support process analysts by facilitating the *visual analysis* [19] of process drifts. Figure 1 schematically illustrates the main visual cues it offers to the users to this end. We integrate various formal concepts grounded in the rigor of temporal logic, DECLARE constraints [20], [21] and time series analysis [22]. Key strengths of our system are clustering of declarative behavioral constraints that exhibit similar trends of changes over time and automatic detection of changes in terms of drift points. For each of these analysis steps, we provide different visualizations, including the Extended Directly-Follows Graph, the Drift Map, Drift Charts, and various measures to indicate the type of drift. These features allow us to detect and explain drifts that would otherwise remain undetected by existing techniques. The paper presents an evaluation that demonstrates these capabilities.

The remainder of the paper is structured as follows.

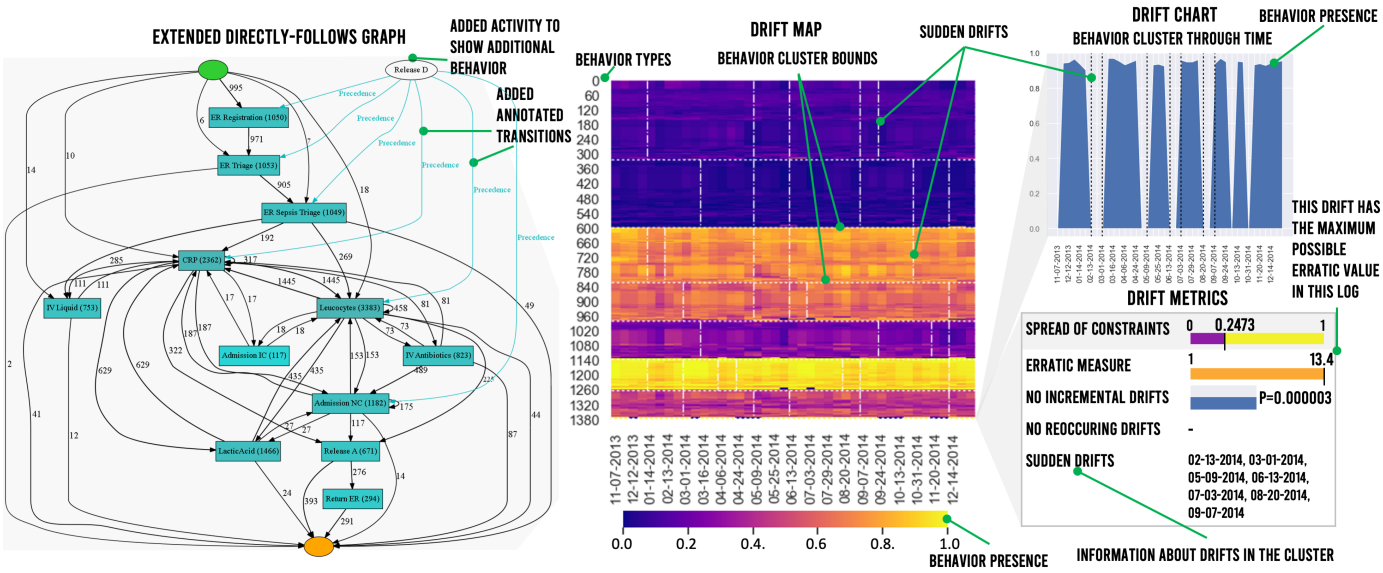


Figure 1: The Visual Drift Detection (VDD) approach visualization (here using as input the Sepsis event log [1]). In the center, a Drift Map shows the degree to which clusters of behaviour change over time (on the x axis). The intensity of the color indicates the confidence associated to the behavioral constraints (on the y axis). Vertical dashed lines signal drift points. On the top-right corner, a Drift Chart depicts the oscillations of the confidence values that determine the drift points of a cluster. On the bottom-right corner, Drift Metrics document the detected erratic behavior. On the left-hand side, the extended Directly Follows Graph illustrates the behavior of the cluster as a workflow diagram.

Section 2 illustrates the problem of process drift detection and formulates five requirements for its analysis. Then, Section 3 states the preliminaries. Section 4 presents the concepts of our drift detection system, while Section 5 evaluates the system using benchmark data and a user study. Finally, Section 6 summarizes the results and concludes with an outlook on future research.

2 Process Drift Analysis

In this section, we discuss the analysis of drift phenomena for business processes. First, Section 2.1 illustrates an example of drift in a business process. Section 2.2 then characterizes the specific analysis task of the analysts and identifies requirements for supporting process analysts for visually inspecting drift.

2.1 Drift in Business Processes

Business processes are collections of inter-related events and activities involving a number of actors and objects [10]. They define the steps by which products and services are provided to customers. Arguably, any work or business operation can be understood as a business process, though more specific terms are used in different industries: manufacturing processes, clinical pathways, service provisioning, or supply chains [23]. Analyzing and improving these processes is difficult due to their complexity and their division of labour with separate agents being responsible for different activities.

As an example of a business process, consider the log of a hospital on handling sepsis patients [1] displayed by our system in Fig. 1. The diagram on the left-hand side is a Directly-Follows Graph showing potential sequences of the process. One individual patient is a case of this process, and his or her sequences through the process is a trace. The process typically starts with the registration and admission of the patient with ER Registration. A first diagnosis is performed with the ER Triage activity followed by an ER Sepsis Triage. The patients suspected of sepsis are treated with infusions of antibiotics and intravenous liquid (IV Antibiotics and IV liquid). The majority of the patients are admitted to the normal care ward (Admission NC), while some are admitted to intensive care (Admission IC). In some cases, the admission type changes during the treatment process. At the end of the treatment, and due to different reasons, patients are dispatched (with Release A-D activities).

The hospital is now interested in this question: *Has the process of treating sepsis patients changed over time, and which parts of it now work differently than in the past?* The described problem is typical for many process domains. The objective is to explain the change of the process behavior in a dynamically changing non-stationary environment based on some *hidden context* [24]. The data mining and machine learning community use the term *concept drift* to refer to any change of the conditional distribution of the output given a specific input. Corresponding techniques for *concept drift detection* identify drift in data collections, either in an *online* or *offline* manner, with applications in prediction and fraud detection [25].

Recently, the availability of event logs of business processes has inspired various process mining techniques [9]. Those techniques mainly support process monitoring and analysis. Classical process mining techniques have implicitly assumed that logs are not sensitive to time in terms of systematic change [9]. For instance, sampling-based techniques explicitly build on this assumption for generating a process model with a subset of the event log data [26]. A significant challenge for adopting concept drift for process mining is to represent behavior in a time-dependent way. The approach

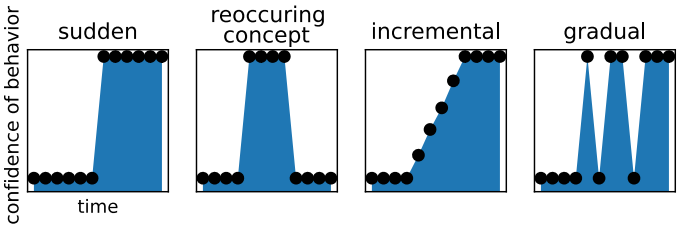


Figure 2: Different types of drifts, cf. Fig. 2 in [24].

reported in [16] uses causal dependencies and tracks them over time windows. Support for more detailed analysis is limited so far. Specifically relevant is the question if a process show concept drift and which of its activities relate to it.

Prior research on data mining has described different archetypes of drift (see Fig. 2). We use the example of the sepsis process to illustrate the potential causes of drifts. A *sudden drift* is typically caused by an intervention. A new guideline could eliminate the need to conduct triage in two steps, as it is currently done. As a result, we would not see second triage events in our log in the future. An *incremental drift* might result from a stepwise introduction of a new type of infusion. A *gradual drift* may stem from a new guideline to consider intensive care already for patients with less critical symptoms. Finally, a *reoccurring drift* might result from specific measures taken in the holiday season from July to August when inflammations are more likely due to warm weather. Existing process mining techniques support these types of drifts partially.

The following are four illustrative cases from the sepsis process:

10 Jan. 2014: ⟨ER Registration, ER Triage, ER Sepsis Triage, IV Antibiotics, Release A⟩

15 Jan. 2014: ⟨ER Registration, ER Triage, ER Sepsis Triage, IV Antibiotics, Release A⟩

04 Feb. 2014: ⟨ER Registration, ER Triage, IV Antibiotics, Release A⟩

06 Feb. 2014: ⟨ER Registration, ER Triage, IV Antibiotics, Release A⟩

We observe a sudden drift here due to the introduction of a new guideline. After 04 Feb. 2014, the sepsis triage is integrated with the general triage step. Therefore, in formal terms, from case 3 onwards, the behavioral rule that the sepsis triage occurs after the general triage abruptly decreases in the number and share of observations. Several rule languages with a rich spectrum of behavioral constraints have been proposed [27], [28], [29]. In rule languages based on linear temporal logic such as DECLARE, we can formally state that the rule ALTERNATERESPONSE(ER Triage, ER Sepsis Triage) drops in confidence. We will make use of such rules in our technique.

2.2 Analysis Tasks of Process Analysts

We frame our design study in the tasks of process analysts. *Process analysts* are typically in charge of steering process improvement projects, gathering information about current process performance, modeling the process as-is, analyzing weaknesses and changes over time, developing redesign options, and bringing them into implementation [10]. The analysis of changes based on the evidence brought by event logs entails the challenge of detecting and understanding process drifts. Such a complex task with data requires interactive support to explore and investigate various aspects of the information source at hand [9]. Based on the experience gained in projects with industry partners, we identified five requirements for process drift analysis [30]:

R1. Identify drifts: The points at which a business process undergoes drifts should be identified based on precise criteria;

R2. Categorize drifts: Process drifts should be categorized according to their types;

R3. Drill down and roll up analysis: Process drifts should be characterized at different levels of granularity, e.g., drifts that concern the entire process or only its parts;

R4. Quantitative analysis: Process drifts should be associated with a degree of change, a measure that quantifies to which extent the drift entails a change in the process;

R5. Qualitative analysis: Process drifts should convey changes in a business process to process analysts effectively.

Table 1: Process drift detection in process mining.

Approach	R1	R2	R3	R4	R5
ProDrift [15], [16]	+	+/-	-	-	-
TPCDD [14]	+	-	-	-	-
Process Trees [31]	+	-	-	-	+
Performance Spectra [11]	-	-	+/-	-	+
Comparative Trc. Clustering [12]	-	-	-	+	+
Graph Metrics On Proc.Graphs [13]	+	-	-	+	+
Eventpad [32]	+	-	-	-	+
ViDX [33]	+	-	+/-	+/-	+
Eventthread3 [34]	-	-	+	+	+
VDD (this paper)	**+**	**+**	**+**	**+**	**+**

Table 1 provides an overview of the state-of-the-art methods for process drift analysis with reference to the five listed requirements. Notice that collectively these methods address (at least partially) all the requirements, whereas each method addresses only a subset. In particular, R2 and R3 remain mostly open challenges.

Approaches like ProDrift [16] and Graph Metrics on Process Graphs [13] put an emphasis on requirement R1. The evaluation of ProDrift in [16] shows that sudden and gradual drifts are found accurately, thus partly addressing requirement R2, although with a reported high sensitivity to the choice of method parameters. ProDrift relies on the automatic detection of changes in business process executions, which are analyzed based on causal dependency relations studied in process mining [35]. The Tsinghua Process Concept Drift Detection approach (TPCDD) [14] uses two kinds of behavioral relationships: direct succession and weak order. The approach computes those relations on every trace, so as to later identify the change points with the help of clustering. The only type of drift that TPCDD detects is sudden drift.

The other approaches emphasize requirement R5. The approach based on Process Trees [31] uses ProDrift for drift detection, and aims at explaining how sudden drifts influence process behavior. To this end, process trees are built for pre-drift and post-drift sections of the log and used to explain the change. The Performance Spectra approach [11]

focuses on drifts that show seasonality. The technique filters the control-flow and visualizes identified flow patterns. It is evaluated against a real-world log, in which recorded business processes show year-to-year seasonality. A strength of the Comparative Trace Clustering approach [12] is its ability to include non-control-flow characteristics in the analysis. Based on these characteristics, it partitions and clusters the log. Then, the differences between the clusters indicate the quantitative change in the business processes, which addresses requirement R4. The Graph Metrics on Process Graphs approach [13] discovers a first model, called a reference, using the Heuristic Miner on a section of the log [9]. Then, it discovers models for other sections of the log and uses graph metrics to compare them with the reference model. The technique interprets significant differences in the metrics as drifts. The reference model and detection windows get updated once a drift is detected.

Works that emphasize the visualization analysis of drifts for event sequence data mainly approach change as a type of anomaly. Eventpad [32] allows the users to import event sequences for interactive exploration by filtering the visual representation using constraints and regular expressions. The overview provided by the system helps to uncover change patterns. Eventpad supports the requirements R1 and R5. The ViDX system [33] offers an interactive visualization system to discover seasonal changes. Note that Performance Spectra [11] build on similar design ideas. The user of the ViDX system can select the sequences that are considered normal and the system highlights the sequences that deviate from this norm. The system also supports a calendar view, which helps to identify where drifts happen in the timeline. The system supports requirements R1, R5, and partially R3 and R4. The EventThread3 system [34] relies on an unsupervised anomaly detection algorithm and the interactive visualization system to uncover changes in event sequence data. Seven connected views allow the analyst to inspect the flow-based overview of the event sequence data with additional information on anomalous sequences. The system supports the thorough analysis of anomalous behavior (requirements R3, R4 , and R5) but neither identifies the exact point in time in which the change of behavior happened, nor classifies the changes.

Beyond these specific works on process drift, there are numerous related works on the visualization of event sequence data [36], [37]. The *summarization of event sequence data* can be supported by visual representations of different types. Chen et al. [38] use several connected views including a raw sequence representation and an abstraction based on the minimum description length principle. The work by Gou et al. [39] splits the event data into threads and stages. In this way, they summarize complex and long event sequences. Zhang et al. [40] combine the raw event sequence visual representation with annotated line plots together with custom infographics emphasizing use-case related characteristics. Wongsuphasawat et al. [41] introduce an interactive event sequence overview system called LifeFlow, which builds upon the Icicle plot and represents temporal spacing within event sequences. Monroe et al. [42] present the event sequence analysis system EventFlow, which offers different types of aggregation and simplification. Law et al. [43] introduce an interactive system that supports flexible analysis of event sequences by combining querying and mining. Wongsuphasawat and Gotz [44] extend the directed graph event sequence representation with colored vertical rectangles used as the transitions between events. Tanahashi and Ma [45] describe design considerations for visualizing event sequence data. This work concerns the usage of color and layout when designing visualizations. Other papers explain how to effectively visualize the *alignment of sequences*. Albers et al. [46] present a hierarchically structured visual representation for genome alignments. Cappers et al. [47] visualize event sequences aligned by user-defined temporal rules. Malik et al. [48] introduce the cohort comparison system CoCo, which uses automated statistics together with a user interface for exploring differences between datasets. Zhao et al. [49] introduce a novel visualization system based on the matrices arranged in a zig-zagging pattern that allows for less overlapping edges than common Sankey based visualizations. Xu et al. [33] achieve visualization of changes and *drifts* in event sequence data through compound views consisting of Marey's graph, line plots, bar charts, calendar views, and custom infographics.

This discussion, summarized in Table 1, witnesses that none of the state-of-the-art methods covers the full set of the five requirements of visualizing process drifts. The approach described in the following addresses this research gap.

3 Preliminaries

This section defines the formal preliminaries of our approach. Section 3.1 gives an overview of the event log, the main input data type used in process mining. Sections 3.2 and 3.3 describe process representation languages: the former introduces the directly-follows graphs for procedural models, and the latter illustrates the representation of the process. Sections 3.3 and 3.4 discuss the DECLARE specification and the techniques to discover and simplify those models from event logs, respectively. Section 3.5 describes time series clustering, and Section 3.6 illustrates change point detection methods, which are the main instruments of our approach.

3.1 Event log

Event logs capture actual execution sequences of business processes. They represent the input for process mining techniques. An event log L (*log* for short) is a collection of recorded traces that correspond to process enactments. In this paper, we abstract the set of activities of a process as a finite non-empty alphabet $\Sigma = \{a, b, c, \ldots\}$. Events record the execution of activities, typically indicating the activity and the completion timestamp. A trace σ is a finite sequence of events. For the sake of simplicity, we shall denote traces by the sequence of activities they relate to, $a_i \in \sigma, 1 \leqslant i \leqslant n$, sorted by their timestamp. In the following examples, we also resort to the string-representation of traces (i.e., $\sigma = a_1 a_2 \cdots a_n$) defined over Σ. Case 1 of the sepsis process from Section 2.1 is an example of a trace. An event log L is a multiset of traces, as the same trace can be repeated multiple times in the same log: denoting the multiplicity $m \geqslant 0$ as the exponent of the trace, we have that $L = \{\sigma_1^{m_1}, \sigma_2^{m_2}, \ldots, \sigma_N^{m_N}\}$ (if $m_i = 0$ for some $1 \leqslant i \leqslant N$ we shall simply omit σ_i). The size of the log is defined as

$|L| = \sum_{i=1}^{N} m_i$ (i.e., the multiplicity of the multiset). Cases 1-4 of the sepsis process in Section 2.1 constitute an example of event log of size 4. The size of the sepsis log is 1050 [1]. A sub-log $L' \subseteq L$ is a log $L' = \{\sigma_1^{m'_1}, \sigma_2^{m'_2}, \ldots, \sigma_N^{m'_N}\}$ such that $m'_i \leqslant m_i$ for all $1 \leqslant i \leqslant N$. A log consisting of cases 1-3 from the example log L in Section 2.1 is a sub-log of L.

3.2 Directly-Follows Graph

The first output that process mining tools generate for providing an overview of a business process is the Directly-Follows Graph (DFG, also referred to as *process map*). Given an event log L, a DFG is a tuple $G(L) = (A_L, \mapsto_L, A_L^{start}, A_L^{end})$ [9], [10]. In a DFG, each node in set A_L represents an activity class, and each arc denotes a tuple in the directly-follows relation $\mapsto_L$ *discovered* from the event log. Figure 1 shows a DFG of the sepsis log on the left-hand side. For instance, for a specific patient we observe that the ER Triage activity is followed by ER Sepsis Triage, resulting into a corresponding tuple in the directly-follows relation. Each arc is annotated with a number representing frequency of occurrence in the event log to indicate the importance of that transition between tasks in the process. $G(L)$ explicitly encodes start and end of the discovered process with sets of activities A_L^{start}, A_L^{end}, respectively. DFGs are known to be simple and comprehensive [50], [51]. Indeed, they are used as a visual overview for processes both in open-source and commercial process mining tools like Fluxicon Disco[1] and Celonis[2], and pm4py [52]. They are also used as an intermediate data structure by several process discovery algorithms [50], [53].

As shown in [54], the complexity of DFG mining is linear in the number of traces ($O(|L|)$) and quadratic in the number of activities ($O(|\Sigma|^2)$).

3.3 DECLARE modeling and mining

Fine-granular behavior of a process can be represented in a declarative way. A declarative process specification represents this behavior by means of *constraints*, i.e., temporal rules that specify the conditions under which activities may, must, or cannot be executed. In this paper, we focus on DECLARE, a well-known standard for declarative process modeling [20] based on linear temporal logic. DECLARE provides a *repertoire* of template constraints [55], [56]). Examples of DECLARE constraints are RESPONSE(a, b) and CHAINPRECEDENCE(b, c). The former constraint applies the RESPONSE template on tasks a and b, and states that if a occurs then b must occur later on within the same trace. In this case, a is named *activation*, because it is mentioned in the if-clause, thus triggering the constraint, whereas b is named *target*, as it is in the consequence-clause [56]. CHAINPRECEDENCE(b, c) asserts that if c (the activation) occurs, then b (the target) must have occurred immediately before. Given an alphabet of activities Σ, we denote the number of all possible constraints that derive from the application of DECLARE templates to all activities in Σ as $\#_{\text{cns}} \subseteq O(\Sigma^2)$ [56]. For the sepsis log, $\#_{\text{cns}} = 3424$. Table 2 shows some of the templates of the DECLARE repertoire, together with the examples of traces that satisfy (✓) or violate (×) them.

Declarative process mining tools can measure to what degree constraints hold true in a given event log [57]. To that end, diverse measures have been introduced [58]. Among them, we consider here *support* and *confidence* [21]. Their values range from 0 to 1. In [21], the support of a constraint is measured as the ratio of times that the event is triggered and satisfied over the number of activations. Let us consider the following example event log: $L = \{\sigma_1^4, \sigma_2^1, \sigma_3^2\}$, having σ_1 = baabc, σ_2 = bcc, and σ_3 = bcba. The size of the log is $4 + 1 + 2 = 7$. The activations of RESPONSE(a, b) that satisfy the constraint amount to 8 because two a's occur in σ_1 that are eventually followed by an occurrence of b, and σ_1 has multiplicity 4 in the event log. The total amount of the constraint activations in L is 10 (see the violating occurrence of a in σ_3). The support thus is 0.8. By the same line of reasoning, the support of CHAINPRECEDENCE(b, c) is $\frac{7}{8} = 0.875$ (notice that in σ_2 only one of the two occurrences of c satisfies the constraint). To take into account the frequency with which constraints are triggered, confidence scales support by the ratio of traces in which the activation occurs at least once. Therefore, the confidence of RESPONSE(a, b) is $0.8 \times \frac{6}{7} \approx 0.69$ because a does not occur in σ_2. As b occurs in all traces, the confidence of CHAINPRECEDENCE(b, c) is 0.875.

As shown in [21], [59], the computation of constraint measures on an event log L is performed efficiently as the mining algorithms have a complexity that is *(i)* linear with respect to the number of traces, $O(|L|)$, *(ii)* quadratic to the total number of events, $O(\sum_{\sigma \in L} |\sigma|^2)$, and *(iii)* linear to the number of constraints, $O(\#_{\text{cns}})$, hence quadratic with respect to the number of activities in the event log as $\#_{\text{cns}} \subseteq O(\Sigma^2)$. This complexity corresponds to that of mining Directly-Follows Graph (DFG), as previously discussed in Section 3.2.

3.4 Subsumption of DECLARE rules

For one event log, there are typically a large amount of DECLARE constraints. Efficient abstraction can be achieved by pruning out constraints that are subsumed by others. To this end, we outline here the concept of *subsumption* for declarative constraints, and its impact on the support and confidence measures. For technical details, see [56]. As it can be noticed in Table 2, CHAINPRECEDENCE imposes a stricter rule on the process execution than ALTERNATEPRECEDENCE, which in turn exerts a stricter rule than PRECEDENCE: for example, C = PRECEDENCE(b, c) requires that every occurrence of c is preceded at some point before by b; C' = ALTERNATEPRECEDENCE(b, c) adds to the statement of C that no other c can recur between c and the preceding occurrence of b; on top of that, C'' = CHAINPRECEDENCE(b, c) excludes that *any* other task between c and the preceding b occurs (not just c). As a consequence, every trace that satisfies C'' is also compliant with C', and every trace that satisfies the latter, in turn, complies with C. For example, let $L' = \{\sigma_4^2, \sigma_5^1, \sigma_6^3\}$ be an event log in which σ_4 = bccabc, σ_5 = bacabc, and σ_6 = bcaabc. σ_4 satisfies C but does not comply with either of C' and C''. σ_5 satisfies C and C' but not C''. Finally, σ_6 satisfies C, C' and C''. Notice that it is not possible to find an example of trace satisfying, e.g., C and C'' but not C'. We say that C'' is subsumed by C' and C' is

1. https://fluxicon.com/disco/
2. https://www.celonis.com/

Table 2: Example DECLARE constraints.

Constraint	Explanation	Examples			
ATMOSTONE(a)	If a occurs, then it occurs at most *once*	✓bcc	✓bcac	×bcaac	×bcacaa
RESPONSE(a, b)	If a occurs, then b occurs eventually after a	✓baabc	✓bcc	×bcba	×caac
ALTERNATERESPONSE(a, b)	If a occurs, then b occurs eventually afterwards, and no other a recurs in between	✓cacb	✓abcacb	×caacb	×bacacb
CHAINRESPONSE(a, b)	If a occurs, then b occurs immediately afterwards	✓cabb	✓abcab	×cacb	×bca
PRECEDENCE(a, b)	If b occurs, then a must have occurred before	✓cacbb	✓acc	×ccbb	×bacc
ALTERNATEPRECEDENCE(a, b)	If b occurs, then a must have occurred before and no other b recurs in between	✓cacba	✓abcaacb	×cacbba	×abbabcb
CHAINPRECEDENCE(a, b)	If b occurs, then a occurs immediately beforehand	✓abca	✓abaabc	×bca	×baacb
NOTSUCCESSION(a, b)	a occurs if and only if b does not occur afterwards	✓bbcaa	✓cbbca	×aacbb	×abb

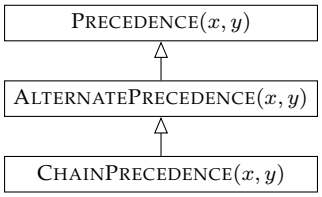


Figure 3: The subsumption relation restricted to CHAINPRECEDENCE(x, y), ALTERNATEPRECEDENCE(x, y), and PRECEDENCE(x, y)

subsumed by C. Subsumption enjoys the properties of transitivity, reflexivity, and anti-symmetry, thus being a partial order. The repertoire of DECLARE constitutes a *subsumption hierarchy*. Figure 3 depicts the fragment of subsumption hierarchy related to the aforementioned constraints as an is-a relation. Interestingly, the subsumption hierarchy among constraints induces a partial order also on the sub-multisets of traces in an event log, the homomorphism being the relation with respect to constraints: considering the example above, $\{\sigma_4^2, \sigma_5^1, \sigma_6^3\}$ satisfies C, $\{\sigma_5^1, \sigma_6^3\}$ satisfies C', and $\{\sigma_6^3\}$ satisfies C''. Therefore, by definition, support and confidence are monotonically non-decreasing along the subsumption hierarchy [56]. On L', e.g., we have that the support of C'', C', and C is 0.71, 0.85, and 1.0, respectively. Their confidence coincides with support as c (the activation) occurs in all traces, for simplicity. We shall take advantage of this property to reduce the number of constraints to represent the behavior of identified clusters.

An array of algorithms have been introduced to automatically detect and remove redundant constraints. The techniques described in [21], [60] resort to auxiliary data structures that are heuristically optimized for the repertoire of DECLARE, and require linear time with respect to the number of constraints, $O(\#_{\text{cns}})$. In [56] a general and more effective approach for declarative languages has been proposed. It first creates a priority list for the elimination of possibly redundant constraints ($O(\#_{\text{cns}} \cdot \log_2(\#_{\text{cns}}))$) and then linearly scans that list for redundancy checking. The check is based on the incremental comparison of the finite-state automata underlying the process model and the constraints. We resort to techniques optimized for DECLARE as a pre-processing phase pruning the vast majority of redundancies and operate with the small-sized automata of DECLARE constraints for the final removal of redundancies.

3.5 Time series clustering

Plotting the confidence and support of different DECLARE constraints over time produces a time series. A *time series* is a sequence of ordered data points $\langle t_1, t_2, \ldots, t_d \rangle = T \in \mathbb{R}^d$ consisting of $d \in \mathbb{N}^+$ real values. The illustrations of drift types in Figure 2 are in essence time series. A *multivariate time series* is a set of $n \in \mathbb{N}^+$ time series $D = \{T_1, T_2, \ldots, T_n\}$. We assume a multivariate time series to be piece-wise stationary except for its *change points*.

In our approach, we take advantage of the time series clustering algorithms. Time series clustering is an unsupervised data mining technique for organizing data points into groups based on their similarity [61]. The objective is to maximize data similarity within clusters and minimize it across clusters. More specifically, the *time-series clustering* is the process of partitioning D into non-overlapping clusters of multivariate time series, $C = \{C_1, C_2, \ldots, C_m\} \subseteq 2^D$, with $C_i \subseteq D$ and $1 \leqslant m \leqslant n$, for each i such that $1 \leqslant i \leqslant m$, such that homogeneous time series are grouped together based on a *similarity measure*. A *similarity measure* $\text{sim}(T, T')$ represents the distance between two time series T and T' as a non-negative number. Time-series clustering is often used as a subroutine of other more complex algorithms and is employed as a standard tool in data science for anomaly detection, character recognition, pattern discovery, visualization of time series [61]. As discussed in [61] the hierarchical clustering computation is polynomial in the number of time series (which, in turn, is proportional to the number of constraints), hence $O(|D|^3) = O(\#_{\text{cns}}^3)$.

3.6 Change point detection

Change point detection is a technique for identifying the points in which multivariate time series exhibit changes in their values [22]. Let D^j denote all elements of D at position j, i.e., $D^j = \{T_1^j, T_2^j, ..., T_n^j\}$, where T^j is a j-th element of time series T. The objective of change point detection algorithms is to find $k \in \mathbb{N}^+$ changes in D, where k is previously unknown. Every element D^j for $0 < j \leqslant k$ is a point at which the values of the time series undergo significant changes. Change points are often represented as vertical lines in time series charts.

To detect change points, the search algorithms require a *cost function* and a *penalty* parameter as inputs. The former describes how homogeneous the time series is. It is chosen in a way that its value is high if the time series contains many change points and low otherwise. The latter is needed

to constrain the search depth. The supplied penalty should strike a good balance between finding too many change points and not finding any significant ones. Change point detection is a technique commonly used in signal processing and, more in general, for the analysis of dynamic systems that are subject to changes [22]. In the worst case, the change point detection algorithm has a quadratic performance [62] in the number of time series in the cluster $O(|D|^2) = (\#^2_{\text{cns}})$.

4 A System for Visual Drift Detection

In this section, we introduce the VDD system. Its overall design idea is to cut the log into time windows and compute the confidence of behavioral constraints on the traces within those windows, so that the results can be visualized over time. Figure 4 illustrates the steps of VDD for generating the visualizations.

Step 1: Mining Directly-Follows Graph as an overview. In the first step, we mine a DFG from an input event log to get an overview of the behavior captured in the log.

Step 2: Mining constraints windows. Concurrently with the first step, we split the log into sub-logs. From each sub-log, we mine the set of DECLARE constraints and compute their confidence. As a result, we obtain several time series.

Step 3: Clustering Time Series. In this step, we cluster those time series into groups of constraints that exhibit similar confidence trends (henceforth, *behavior clusters*).

Step 4: Visualizing Drifts. In this step, we detect drift points for the whole log and each cluster separately. We plot drift points in Drift Maps and Drift Charts to effectively communicate the drifts to the user.

Step 5: Detecting Drift Types. In this step, we use an array of methods to further analyze drift types. We employ multivariate time series change point detection algorithms to spot *sudden drifts* in both the entire set of constraints and in each cluster. We use stationarity analysis to determine if clusters exhibit *gradual drifts* and autocorrelation plots to check if *reoccurring drifts* are present. While Step 4 is concerned with estimating *the extent* of drift presence, Step 5 is intended to show and *explain* those drifts.

Step 6: Understanding drift behavior. In the final step, we present semantic information on the identified drifts. Step 6 produces a minimized list of constraints and a projection of these constraints onto the Directly-Follows Graph to explain the behavior in the drift cluster.

In the following, we detail these steps.

4.1 Mining Directly-Follows Graph as an Overview

The first step takes as input a log L and produces the Directly-Follows Graph (DFG). The DFG includes an arc $a \xrightarrow{n} a'$ if a sub-sequence $\langle a, a' \rangle$ is observed in any traces of the log (n indicates the total number of such observations). The process analyst typically starts the analysis by exploring the paths of the DFG.

4.2 Mining constraints windows

Performed in parallel with the mining of the DFG, this step takes as input a log L and two parameters (win_{size} and win_{step}). It returns a multivariate time series D based on the confidence of mined DECLARE constraints.

In this step, we first sort the traces in the event log L by the timestamp of their start events. Then, we extract a sub-log from L as a window of size $\text{win}_{\text{size}} \in \mathbb{N}^+$, with $1 \leqslant \text{win}_{\text{size}} \leqslant |L|$. Next, we shift the sub-log window by a given step ($\text{win}_{\text{step}} \in \mathbb{N}^+$, with $1 \leqslant \text{win}_{\text{step}} \leqslant \text{win}_{\text{size}}$). Notice that we have sliding windows if $\text{win}_{\text{step}} < \text{win}_{\text{size}}$ and tumbling windows if $\text{win}_{\text{step}} = \text{win}_{\text{size}}$. Thus, the number of produced sub-logs is equal to: $\#_{\text{win}} = \left\lfloor \frac{|L| - \text{win}_{\text{size}} - \text{win}_{\text{step}}}{\text{win}_{\text{step}}} \right\rfloor$. Having win_{size} set to 50 and win_{step} set to 25, $\#_{\text{win}}$ is 39 for the sepsis log.

For every sub-log $L_j \subseteq L$ thus formed ($1 \leqslant j \leqslant \#_{\text{win}}$), we check all possible DECLARE constraints that stem from the activities alphabet of the log, amounting to $\#_{\text{cns}}$ (see Section 3.3). For each constraint $i \in 1..\#_{\text{cns}}$, we compute its confidence over the sub-log L_j, namely $\text{Conf}_{i,j} \in [0, 1]$. This generates a time series $T_i = (\text{Conf}_{i,1}, \ldots, \text{Conf}_{i,\#_{\text{win}}}) \in [0, 1]^{\#_{\text{win}}}$ for every constraint i. In other words, every time series T_i describes the confidence of all the DECLARE constraints discovered in the i-th window of the event log. The multivariate time series $D = \{T_1, T_2, \ldots, T_{\#_{\text{cns}}}\}$ encompasses the full spectrum of all constraints. Next, we detail the steps of slicing the DECLARE constraints and explaining the drifts.

4.3 Clustering Time Series

The third step processes the previously generated multivariate time series of DECLARE constraints D to derive a set C of clusters exhibiting similar confidence trends. For instance, if we observe confidence values over five time windows for RESPONSE(a, b) as $(0.2, 0.8, 0.9, 0.8, 0.9)$ and for CHAINPRECEDENCE(b, c) we have $(0.23, 0.8, 0.9, 0.9, 0.9)$, it is likely that the two time series for these constraints might end up in the same cluster due to their small difference. The aim of this step is to identify drift points at a fine-granular level. To this end, we use time-series clustering techniques [61] for grouping together similarly changing pockets of behavior of the process. Each time series describes how one constraint changes its confidence over time. By clustering, we find all the time series that share similar trends of values, hence, we find all similarly changing constraints. We use *hierarchical clustering*, as it is reportedly one of the most suitable algorithms when the number of clusters is unknown [61]. As a result, we obtain a partition of the multivariate time series of DECLARE constraint confidence values into behavior clusters.

4.4 Visualizing Drifts

The fourth step generates visual representations of drifts. To this end, we construct a graphical representation called Drift Map. Drift Maps depict clusters and their constraints' confidence measure evolution along with the time series and their drift points. We allow the user to drill down into every single cluster and its drifts using dedicated diagrams that we call Drift Charts.

Drift Maps (see Fig. 1, in the center) plot all drifts on a two-dimensional canvas. The visual representation we adopt is inspired by [19]. The x-axis is the time axis, while every constraint corresponds to a point on the y-axis. We add vertical lines to mark the identified change points, i.e., drift

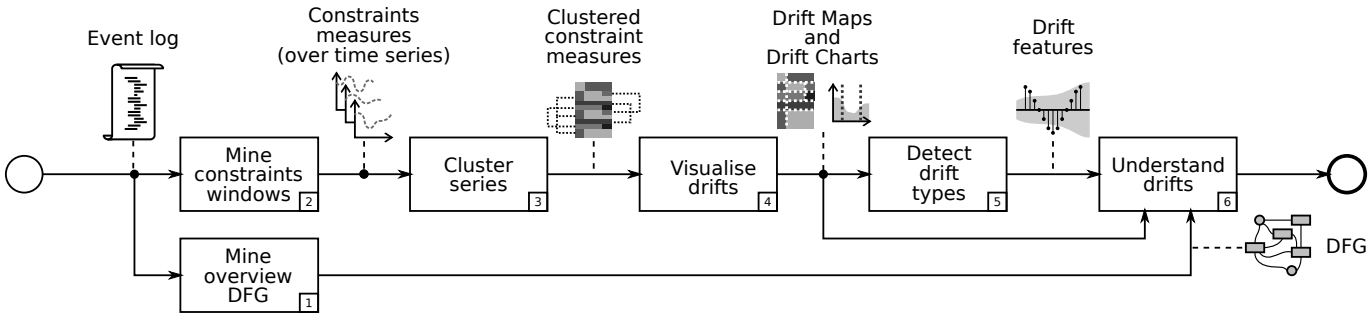


Figure 4: The Visual Drift Detection approach.

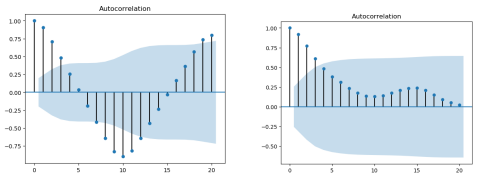
points, and horizontal lines to separate clusters. Constraints are sorted by the similarity of the confidence trends. The values of the time series are represented through the plasma color-blind friendly color map [19] from blue (low peak) to yellow (high peak). To analyze the time-dependent trend of specific clusters, we build Drift Charts (see Fig. 1, on the right). They have time on the x-axis and average confidence of the constraints in the cluster on the y-axis. We add vertical lines as in Drift Maps.

Drift Maps offer users a global overview of the clusters and the process drifts. Drift Charts allow for a visual categorization of the drifts according to the classification introduced in [24], as we explain next. These visualizations help the analyst determine if drifts exist at all, which kind of pattern they exhibit over time, and which kind of behavior is stable or drifting.

We use autocorrelation plots to identify the process changes that follow a seasonal pattern, namely the *reoccuring concept* drift. Autocorrelation is determined by comparing a time series with the copy of itself with a lag (delay) of some size [63]. Autocorrelation plots are useful to discover seasonality in the data. The vertical axis in the plot shows the correlation coefficient between elements. The horizontal axis shows the size of the lag between the time series and its copy, refer to Fig. 5. The cosine-wave shaped graph in Fig. 5(a) shows a high seasonality as the peaks share the same value, while the x-axis indicates the steps needed for the season to reoccur. The plot exhibits a seasonal behavior that changes every 10 steps from positive to negative correlation. This means that the values in the time series in step 0 are the opposite of those in step 10 and match those in step 20. Figure 5(b), in contrast, shows the graph with an autocorrelation suggesting that the time series does not exhibit seasonality. We determine whether the step lags are significantly autocorrelated via statistical time series analysis [63]. We classify only significant autocorrelations as an evidence of reoccurring drifts.

4.5 Detecting Drift Types

After clustering the behavior of the log, we support the analyst with visual cues to understand and categorize the drifts within behavior clusters. To this end, we calculate several measures and display them in our visualization system (see Drift Metrics in Fig. 1). These measures are introduced for guiding the analyst in the analysis of the drifts. First, we aid visual analysis by providing a ranking of drifts to assist in



(a) Example of auto-correlated time series (b) Example of low autocorrelation

Figure 5: Example of autocorrelation plots (sepsis log).

focusing on the interesting clusters and filter out outliers. We do so by computing the erratic measure (Section 4.5.1). Then, we categorize drifts using time series coefficients to identify sudden drifts (Section 4.5.2). The sudden drifts are highlighted on Drift Charts and summarized as a list of timestamps indicating when they happened. We then report on statistics that aids in the identification of incremental and gradual drifts (Section 4.5.3).

4.5.1 *Finding erratic behavior clusters*

As we are interested in the extent to which the confidence levels of constraints change over time, we calculate the following measures.

First, to quantify the overall behavior change in the log we introduce a measure we name *range of confidence*. This measure shows what the average change of the value of DECLARE constraint is in the whole log. We compute this measure as follows. For all constraint time series $T_i = (T_{i,1}, \ldots, T_{i,\text{win}_\text{size}})$, where $1 \leqslant i \leqslant |D|$, we calculate the difference between maximum and minimum values. Then, we average the difference on the number of time series:

$$\text{Spread}(D) = \frac{\sum_{i=1}^{|D|} \max(T_i) - \min(T_i)}{|D|} \quad (1)$$

Second, to find the most interesting (*erratic*) behavior clusters, we define a measure inspired by the idea of finding the length of a poly-line in a plot. The rationale is that straight lines denote a regular trend and have the shortest length, whilst more irregular wavy curves evidence more behavior changes, and their length is higher. We are, therefore, mostly interested in long lines.

We compute our measure as follows. We calculate for all constraints i such that $1 \leqslant i \leqslant \#_{\text{cns}}$ the Euclidean distance $\delta : [0,1] \times [0,1] \to \mathbb{R}_+$ between consecutive values in the time series $T_i = (T_{i,1}, \ldots, T_{i,\text{win}_\text{size}})$, i.e., $\delta(T_{i,j}, T_{i,j+1})$ for

every j s.t. $1 \leqslant j \leqslant \text{win}_{\text{size}}$. For every time series T_i, we thus derive the overall measure $\Delta(T_i) = \sum_{j=1}^{\text{win}_{\text{size}}-1} \delta(T_{i,j}, T_{i,j+1})$. Thereupon, to measure how erratic a behavior cluster is, we devise the *erratic measure* as follows:

$$\text{Erratic}(C) = \sum_{i=1}^{|C|} \sqrt{1 + (\Delta(T_i) \times \#_{\text{win}})^2} \quad (2)$$

The most erratic behavior cluster has the highest Erratic value.

4.5.2 Detect sudden drifts: Change point detection

For each cluster of constraints, we search for a set of *sudden drifts*. This means that we look for a set of $k \in \mathbb{N}^+$ change points in the time series representing a drifting cluster. To detect change points, we use the *Pruned Exact Linear Time (PELT)* algorithm [62]. This algorithm performs an exact search, but requires the input dataset to be of limited size. Our setup is appropriate as, by design, the length of the multivariate time-series is limited by the choice of parameters win_{size} and win_{step}. Also, this algorithm is suitable for cases in which the number of change points is unknown a priori [22, p. 24], as in our case. We use the *Kernel cost function*, detailed in [22], which is optimal for our technique, and adopt the procedures described in [62] to identify the optimal *penalty* value.

4.5.3 Detect incremental and gradual drifts: Stationarity

Stationarity is a statistical property of a time series indicating that there is no clear tendency of change over time. It is useful in the context of time series analysis to suggest the presence of a pronounced trend. Here, we rely on *parametric tests* as a rigorous way to detect non-stationarity. One of the most used techniques are the Dickey-Fuller Test and the Augmented Dickey-Fuller Test [64]. It tests the null hypothesis of the presence of a unit root in the time series. If a time series has a unit root, it shows a systematic trend that is unpredictable and not stationary.

In particular, we use the Augmented Dickey-Fuller test to detect *incremental* and *gradual drifts*. Those drifts represent a slow change that goes undetected by change point detection algorithms. If a time series is non-stationary, this signifies that there is a trend in time series. Combined with the analysis of the Drift Charts and the erratic measure, we can differentiate between the incremental and gradual drift. Non-stationary time series with a smoothly increasing Drift Chart represent an incremental drift. A Drift Chart that shows erratic behavior (or such that the erratic measure is large) indicate a gradual drift. The highlighted cluster in Fig. 2 is stationary as suggested by the Augmented Dickey-Fuller test. This means there is no clear trend in the drift.

4.6 Understanding drift behavior

Sections 4.2 to 4.5 describe techniques that provide various insights into drifts in the event log. However, knowing that a drift exists and that it is of a certain type is not sufficient for process analysis. Explanations are required to understand the association between the evidenced drift points and the change in the behavior that led to them. In this section, we describe the two visual aids that we employ to explain that association.

Table 3: Example of constraints present in the drift.

Cluster	Constraint	Activity 1	Activity 2
1	PRECEDENCE	Leucocytes	Release D
	PRECEDENCE	CRP	Release D
	PRECEDENCE	ER Triage	Release D

4.6.1 List of DECLARE constraints

The first report that we generate is the list of DECLARE constraints that are associated with drifts of a selected cluster. To this end, we use the DECLARE subsumption algorithm described in Section 3.4. Reporting these constraints together with the analysis and plots from previous sections help to understand *what* part of the process behavior changes over time and *how*.

Once a highly erratic drift with a seasonal behavior is found, we look up the constraints associated with that drift. For the sepsis case in Fig. 1, e.g., we detect the constraints summarized in Table 3. That drift relates to PRECEDENCE constraints indicating that before *Release D* can occur, *Leucocytes*, *CRP* and *ER Triage* must occur.

4.6.2 Extended Directly-Follows Graph

The process analyst also benefits from a graphical representation of the drifting constraints. To this end, we build upon the Directly-Follows Graphs (DFGs) as shown in Fig. 2 on the left-hand side. Our technique extends the DFG with additional arcs that convey the meaning of the DECLARE constraints. We distinguish three general types of constraints: *immediate* (e.g., CHAINPRECEDENCE(a,b), imposing that b can occur only if a occurs immediately before), *eventual* (e.g., SUCCESSION(a,b), dictating that, if a or b occur in the same trace, b has to eventually follow a), and negated (e.g., NOTSUCCESSION(a,b), imposing that a cannot follow b). We annotate them with green, blue, and red colors, respectively. This way, the user is provided with an overview of the log and which parts of the business process are affected by drifts.

4.7 Computational Complexity

As discussed in Section 3, *Step 1* involves DFG mining algorithms that are linear in the number of traces ($O(|L|)$) and quadratic in the number of activities ($O(|\Sigma|^2)$). *Step 2*, that is, mining constraint windows, is linear in the number of traces ($O(|L|)$) and quadratic in the number of activities ($O(|\Sigma|^2)$) too. The subsumption of DECLARE constraints runs in $O(\#_{\text{cns}} \cdot \log_2(\#_{\text{cns}}))$ where $O(\#_{\text{cns}}) \subseteq O(|\Sigma|^2)$. *Step 3*, clustering time series, is polynomial in the number of time series and, therefore, of constraints ($O(\#^3_{\text{cns}})$). *Step 4*, sudden drift detection, runs in $O(\#^2_{\text{cns}})$ in the worst case. The tasks of detecting gradual drifts and reoccurring drifts are constant operations, as they are performed on the averaged time series. Finally, *Step 5*, understanding drift behavior, has the same asymptotic complexity as *Step 1*. We note that all the applied computations present at most polynomial complexity.

5 EVALUATION

This section presents the evaluation of our visualization system. This evaluation represents the deploy step that

Table 4: Event logs used in the evaluation.

Origin	Event log	Related work
Synthetic	ConditionalMove	ProDrift 2.0 [15]
Synthetic	ConditionalRemoval	ProDrift 2.0 [15]
Synthetic	ConditionalToSequence	ProDrift 2.0 [15]
Synthetic	Loop	ProDrift 2.0 [15]
Real-world	Italian help desk[1]	Process Trees [31]
Real-world	BPI2011[3]	ProDrift 2.0 [15]
Real-world	Sepsis[7]	-

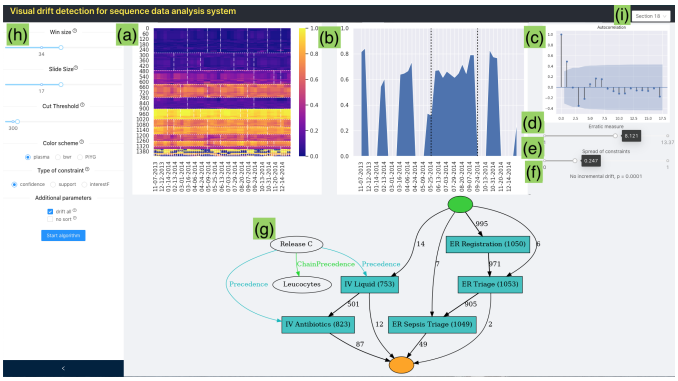


Figure 6: The user interface of the VDD system, running on the Sepsis event log [1]. (a) Drift Map. (b) Drift Chart. (c) Autocorrelation plot. (d) Erratic measure. (e) Spread of constraints view. (f) Incremental drifts test. (g) Extended Directly-Follows Graph. (i) Behavior cluster selection menu.

completes the core phase of the design study methodology by [17]. Section 5.1 describes our implementation. Using this implementation, our evaluation focuses on the following aspects. Section 5.2 evaluates our drift point detection technique for its capability to rediscover change points induced into synthetic logs. Section 5.3 presents insights that our system reveals on real-world cases. Section 5.4 presents experimental results on computational complexity. Section 5.5 summarizes findings from a user study with process mining experts who evaluated the visualizations of our system on a real-world event log. With this part of the evaluation, we focus on target users, their questions and their measurements [65]. Finally, Sections 5.6 and 5.7 discuss how our system addresses the requirements for process drift detection and limitations of the approach, respectively.

5.1 Implementation and user interaction

For the implementation of our approach, we integrate several state-of-the-art techniques and tools. To discover DECLARE constraints, we use MINERful[3] because of its high performance [21]. For change point detection, we integrate the *ruptures* python library[4]. For time series clustering, we resort to the *scipy* library[5].

To attain the most effective outcome, we tune the clustering parameters such as the weighted method for linking clusters (distance between clusters defined as the average between individual points) and the correlation metric (to find individual distances between two time-series). To enhance Drift Map visualizations, we sort the time series of each cluster by the mean squared error distance metric. We implemented both the Drift Map and Drift Chart using the python library *matplotlib*.[6] For the Augmented Dickey-Fuller test and autocorrelation we use the *statmodels* python library[7]. To discover the Directly-Follows Graph, we extended the pm4py process mining python library[8] [52]. Our overall system is implemented in Python 3. Its source code and the parameters used for our experiments are publicly available.[9]

We found that varying the window size affects the results only marginally. Experimenting with parameters, we observed that producing sub-logs out of 60 windows provided a good balance between detail and stability of the results. Therefore, we recommend the following set-up for the involved parameters: $\text{win}_{\text{step}} = \frac{|L|}{60+1}$, and $\text{win}_{\text{size}} = 2 \cdot \text{win}_{\text{step}}$ for smooth visual representation.

We use *hierarchical clustering* for time series clustering, as it is reportedly one of the most suitable algorithms when the number of clusters is unknown [61]. We found that the Ward linkage method and the Euclidean distance function produce the best results. To detect change points, we use the *Pruned Exact Linear Time (PELT)* algorithm [62]. This algorithm performs an exact search but requires the input dataset to be of limited size. Our setup is appropriate as by design the length of the multivariate time-series is limited by the choice of parameters win_{size} and win_{step}. Also, this algorithm is suitable for cases in which the number of change points is unknown a priori [22, p. 24], as in our case. We use the *Kernel cost function*, detailed in [22], which is optimal for our technique, and adopt the procedures described in [62] to identify the optimal *penalty* value.

The VDD system web application is shown in Fig. 6. We describe the tool and user interaction in detail in the demo paper [66] and in the walk-through video.[10] The user starts with uploading an event log file. Then, she can tune analysis parameters including win_{step}, win_{size}, DECLARE constraint type, cut threshold for hierarchical clustering, as well as look-and-feel parameters such as the color scheme, as shown in Fig. 6(h). Default values are suggested based on the characteristics of the input log. Multiple views are displayed and updated in the maun panel Fig. 6(a-g). The user can select the behavior cluster to focus on Fig. 6(i), thus triggering an update in the other views Fig. 6(b-g).

The application of our system with a multi-national company highlights the importance of such exploratory analysis strategies. Understanding changes over time is of key importance to process analysts to identify factors of change and effects of management interventions into the process. The user interaction of our system supports the visual identification of drifts and helps to drill down into the behavior that is associated with those drifts, thereby helping

3. https://github.com/cdc08x/MINERful
4. https://github.com/deepcharles/ruptures
5. https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html
6. https://matplotlib.org/
7. http://www.statsmodels.org/
8. http://pm4py.org, https://github.com/pm4py
9. https://github.com/yesanton/Process-Drift-Visualization-With-Declare
10. https://youtu.be/mHOgVBZ4Imc

the analysts formulate and validate hypotheses about factors of change in the process.

5.2 Evaluation on synthetic data

For our evaluation, we make use of synthetic and real-world event logs.[11][12][13] In this way, we can compare the effectiveness of our approach with earlier proposals. Table 4 summarizes the event logs used in the evaluation and indicates which prior papers used these logs.

To demonstrate the accuracy with which our technique detects drifts, we first test it on synthetic data in which drifts were manually inserted, thereby showing that we accurately detect drifts at the points in which they occur. We compare our results with the state-of-the-art algorithm ProDrift [15] on real-world event logs.

Ostovar et al. [15] published a set of synthetic logs that they altered by artificially injecting drifting behavior: ConditionalMove, ConditionalRemoval, ConditionalToSequence, and Loop.[15] Figure 7 illustrates the results of the application of the VDD technique on these logs. By measuring *precision* as the fraction of correctly identified drifts over all the ones retrieved by VDD and *recall* as the fraction of correctly identified drifts over the actual ones, we computed the F-score (harmonic mean of precision and recall) of our results for each log. Using the default settings and no constraint set clustering, we achieve the F-score of 1.0 for logs ConditionalMove, ConditionalRemoval, ConditionalToSequence, and 0.89 for the Loop log. When applying the cluster-based change detection for the Loop log, we achieve an F-score of 1.0. he Drift Chart in Fig. 7(f) illustrates the trend of confidence for the most erratic cluster for the *Loop* log. The Drift Map for the *Loop* log is depicted in Fig. 7(e). In contrast to [15] we can see which behavior in which cluster contributes to the drift.

5.3 Evaluation on real-world data

Next, we evaluate our system with three real-world event logs. In the next subsections we describe all processing steps for each of the logs.

5.3.1 Sepsis log

The sepsis log describes the cases of patients affected by sepsis. This condition occurs to some patients as a response to infections. The process that generated this log captures the activities executed from the registration of the patients until they are discharged. Prior process mining techniques offer limited insights into this log [1]. We use the processing steps and the multiple outputs of our system to get an understanding of changes in this log over time.

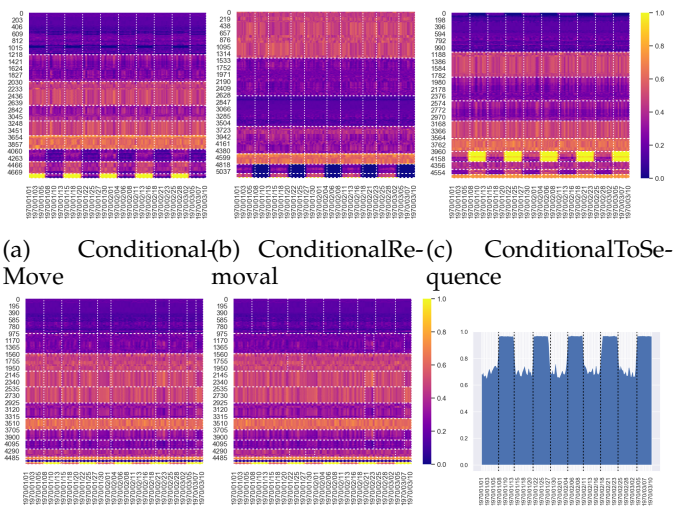
Step 1: Mining Directly-Follows Graph as an overview. The directly-Follows Graph from this log shows 12 activities. The most frequent activity is *Leucocytes* with 3386 instances, followed by the activity *CRT* with 3262 occurrences. In contrast, the activity Admission IC only occurred 117 times. The main path through the DFG is ER registration → ER Triage → ER Sepsis Triage → CRT → Leucocytes → Admission NC. Next to this frequent path, various variants exist that correspond to the less frequent behavior – including, e.g., the Addmission IC and IV Liquid activities.

Step 2: Mining DECLARE constraints windows. For determining the win_{size} and $\#_{\text{win}}$ we consider the number of traces in the log. The log contains 1050 cases spanning over the period of time of a year and four months. We chose a win_{size} of 50 and a win_{step} of 25.

Step 3-4: Finding Drifts and Visual Drift Overview. Figure 9(a) depicts the overall drift behavior. The Drift Map hardly shows any strong patterns of change over time. Apparently, most of the major clusters of behavior do not evidence drifts. Drilling down into clusters with less constraints offers us insights into quite erratic behavior though (see Fig. 9(b) and Fig. 9(c)).

Step 5: Detecting Drift Types. Using the Augmented Dickey-Fuller test, we test the hypothesis that there is a unit root present in the data. If so, the time-series is considered to be non-stationary. The analysis of cluster 8 and cluster 12 shows a p value of 3×10^{-6} and 7.7×10^{-5}, respectively, suggesting that the data does not have a unit root, i.e., it is *stationary*. This means that the behavior does not have an upward or downward trend of change.

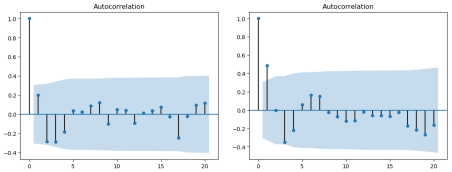
The autocorrelation plots shown in Fig. 8 display negative correlation in steps 2-3 and positive autocorrelation in steps 6-7 – see Fig. 8(a) and Fig. 8(b). That means that there is significant seasonality in the data.



(a) ConditionalMove (b) ConditionalRemoval (c) ConditionalToSequence

(d) Loop (e) Loop, drifts by cluster (f) Loop, most erratic cluster

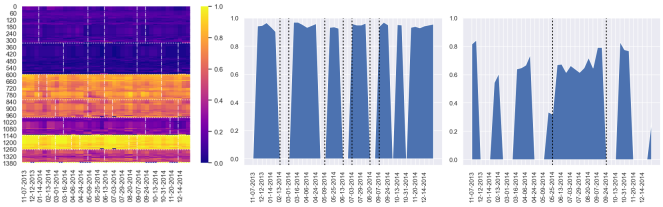
Figure 7: Evaluation results on synthetic logs.



(a) Cluster 8 (b) Cluster 10

Figure 8: Autocorrelation plots for the sepsis log.

11. https://doi.org/10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb
12. https://doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07
13. https://doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54 (pre-processed as in [15])
14. https://data.4tu.nl/repository/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460
15. http://apromore.org/platform/tools



(a) Drift Map (b) Drift in cluster 8 (c) Drift in cluster 12

Figure 9: Sepsis VDD visualizations.

Table 5: Sepsis log erratic clusters.

Drift number	Erratic measure
without drift	40.000
11	245.305
12	324.841
10	415.008
7	417.135
9	495.795
8	534.815

Step 6: Understanding drift behavior. In order to understand the behavior behind some of the drifts we discovered in previous steps, we explore their list of constraints and the derived extended DFG. Based on the inspection of the Drift Map in Fig. 9(a) and the erratic measures in Table 5, we focus on the drifts in Fig. 9(b) and Fig. 9(c).

Table 6 shows the DECLARE constraints of these clusters. We observe that the drifts are related to specific activities, namely Release C for cluster 8 and Release D for cluster 12. We conclude that there are reoccurring drift patterns indicating, thus there are seasonal factors affecting Release C and a Release D. We highlight the process behavior that is subject to drifts via the extended Directly-Follows Graphs. Figure 10 shows the extended DFG highlighting the activities involved in the drift behavior of cluster 8. For this case, we observe that activity Release D was executed after several activities in certain parts of the timeline, as shown in Fig. 9(b).

Table 6: Sepsis log constraints.

Cluster	Constraint	Activity 1	Activity 2
8	PRECEDENCE	IV Antibiotics	Release C
	ALTERNATEPRECEDENCE	IV Antibiotics	Release C
	PRECEDENCE	IV Liquid	Release C
	ALTERNATEPRECEDENCE	IV Liquid	Release C
	ALTERNATEPRECEDENCE	Leucocytes	Release C
12	CHAINPRECEDENCE	CRP	Release D
	PRECEDENCE	IV Antibiotics	Release D
	ALTERNATEPRECEDENCE	IV Antibiotics	Release D
	PRECEDENCE	IV Liquid	Release D
	ALTERNATEPRECEDENCE	IV Liquid	Release D
	PRECEDENCE	LacticAcid	Release D
	ALTERNATEPRECEDENCE	LacticAcid	Release D

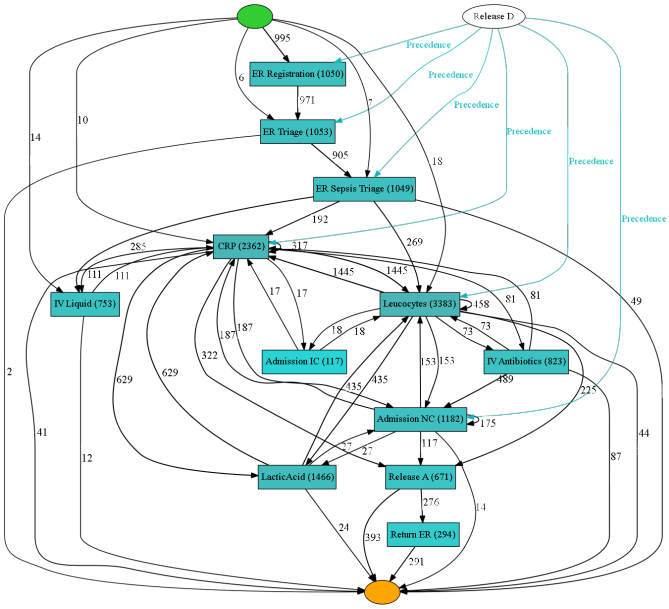


Figure 10: Extended Directy-Follows Graph for cluster 8, derived from the sepsis log.

5.3.2 *Italian help desk log*

Next, we focus on the event log of the help-desk of an Italian software company. It covers all steps from the submission of a support ticket to its closing. Figure 11 provides an overview.
Step 1: Mining Directly-Follows Graph as an overview. The Directly-Follows Graph of this log displays 9 activities. While activity Take in charge ticket occurred 5059 times, activity Schedule intervention only occurred 5 times. The main path through the DFG is Assign seriousness → Take in charge ticket → Resolve ticket → Closed. Other variants are evidenced though, corresponding to the the observation of anomalies (Create SW anomaly activity), waiting (Wait activity), or requests for an upgrade (Require upgrade activity).
Step 2: Mining DECLARE windows. This log contains 4579 cases that are evenly distributed over the period of four years. We set the win_{size} to 100 and the win_{step} to 50.
Step 3-4: Finding Drifts and Visual Drift Overview. Based on the mined DECLARE constraints, the Drift Map is generated. Figure 13 shows the overview of the drifts in the log. For the overall set of clusters, there are three major drift points detected. Figure 13(b) shows a more fine-granular series of drift points, which can be observed within separate clusters. There are also many drifts that signify unregular behaviour and are probably outliers (such as drifts 9, 10 and 11 in Fig. 13(b)). In step 5 we inspect them in detail.
Step 5: Detecting Drift Types. Our system correctly detects sudden drifts in the Italian help desk log, identifying the same two drifts that were found by ProDrift [31], approximately in the first half and towards the end of the time span. As illustrated by the VDD visualization in Fig. 13(a), we additionally detect another sudden drift in the first quarter. By analyzing the within-cluster changes (Fig. 13(b)), we notice that the most erratic cluster contains an outlier, as is shown by the spikes in Fig. 13(c).

We check for reoccurring drifts based on autocorrelation. The visualizations in Fig. 12 show the autocorrelation plots of different clusters together with their Drift Charts. Cluster

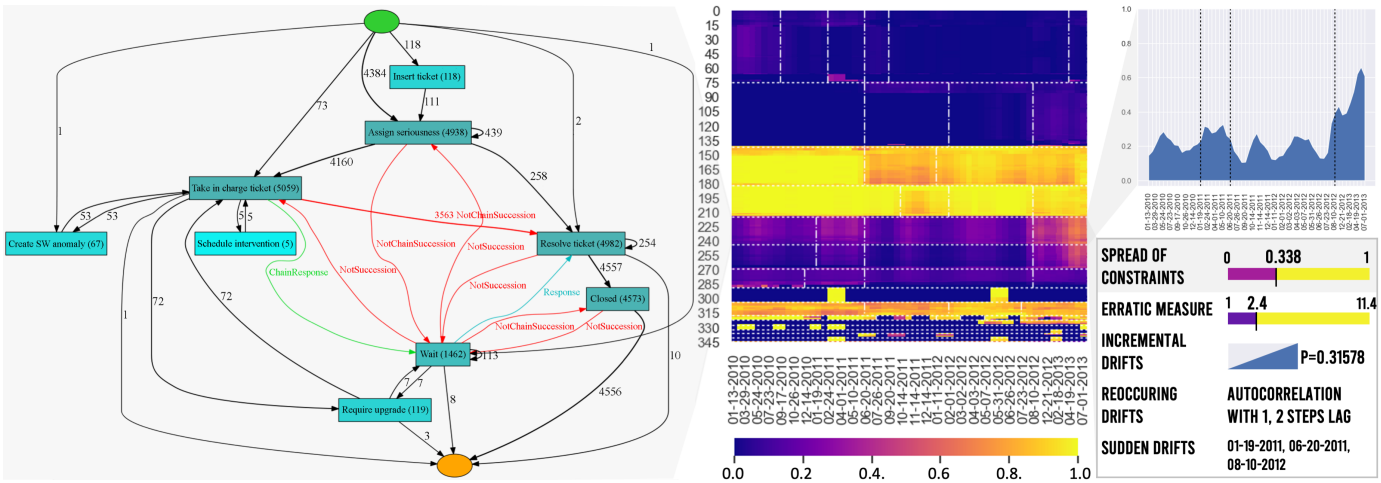
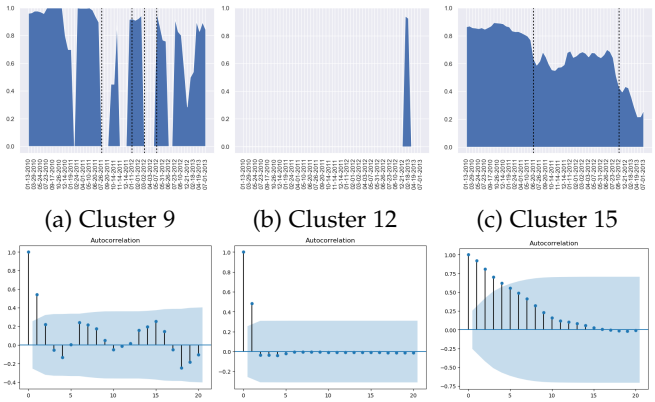


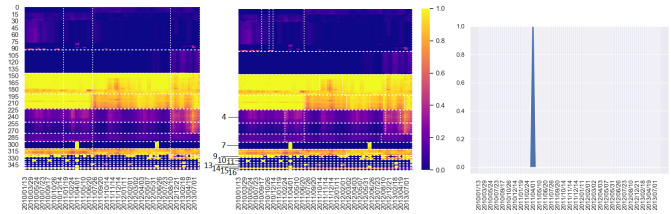
Figure 11: The VDD system visualization of the Italian help desk event log.



(a) Cluster 9 (b) Cluster 12 (c) Cluster 15

(d) Autocorrelation in cluster 9 (e) Autocorrelation in cluster 12 (f) Autocorrelation in cluster 15

Figure 12: Italian help desk autocorrelation results

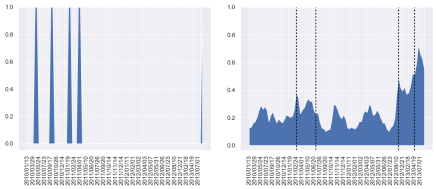


(a) Overall change points (b) Drifts by cluster (c) Most erratic cluster

Figure 13: Italian help desk log VDD visualizations.

Table 7: Italian help desk log erratic clusters.

Drift number	Erratic **measure**	**A-Dickey-Fuller p-value**
without drift	89.000	
9	681.466	0.001
11	578.792	0.001
14	394.138	0.000
13	386.377	0.130
7	287.538	0.316
10	256.339	0.960
16	174.017	0.080
12	166.638	0.900
4	139.403	0.316



(a) Erratic=578.792 for cluster 11 (b) Erratic=139.403 for cluster 4

Figure 14: Italian help desk log detailed clusters.

9 (Fig. 12(a)) shows seasonality, while clusters 12 and 15 (Figs. 12(b) and 12(c)) do not.

Based on the Augmented Dickey-Fuller test, we discover that some of the clusters exhibit incremental drift. For example, cluster 15 has a p-value of 0.980 45 indicating a unit-root, which points to non-stationarity. Indeed, we find an incremental drift with an associated decreasing trend, as shown in Fig. 12(c). The result alongside the erratic measures are shown in Table 7. They highlight that cluster 9 has the most erratic drift behavior.

Step 6: Understanding drift behavior. We further investigate the most erratic cluster. Figure 12(a) shows that its behavior of a *reoccurring drift* type. During the times of peaks, the activity Create SW anomaly always had Take in charge ticket executed immediately beforehand. Also, we observe that the Assign seriousness activity was executed before Create SW anomaly, and no other Create SW anomaly occurred in between. We further analyze other clusters with erratic behavior as shown in Table 7. Figure 14 shows the drift for cluster 11 and cluster 4. The corresponding constraints are listed in Table 8. Figure 14(a) has four spikes, where Schedule intervention activities occurred. Immediately before Schedule intervention, Take in charge ticket occurred. Also, Assign seriousness occurred before Schedule intervention. We notice, however, that this cluster shows *outlier* behavior, due to

Table 8: Italian ticket log constraints; including minimum, maximum, and average confidence.

Cluster	Constraint	Activity 1	Activity 2	Min	Max	Mean
1	ChainPrecedence	Take in charge ticket	Create SW anomaly	0.0	100.0	42.8
	AlternatePrecedence	Assign seriousness	Create SW anomaly	0.0	100.0	49.0
11	ChainPrecedence	Take in charge ticket	Schedule intervention	0.0	100.0	9.9
	AlternatePrecedence	Assign seriousness	Schedule intervention	0.0	100.0	9.9
4	ChainResponse	Take in charge ticket	Wait	9.4	69.6	23.2
	NotSuccession	Resolve ticket	Wait	10.0	77.2	26.0
	NotSuccession	Wait	Assign seriousness	10.0	78.0	26.6
	NotSuccession	Wait	Take in charge ticket	9.8	73.3	22.1
	AlternateResponse	Assign seriousness	Wait	9.0	72.3	23.8
	AlternateResponse	Wait	Closed	8.3	61.4	22.5
	AlternateResponse	Wait	Resolve ticket	8.3	61.4	22.8
	AtMostOne	Wait		9.8	68.6	25.1

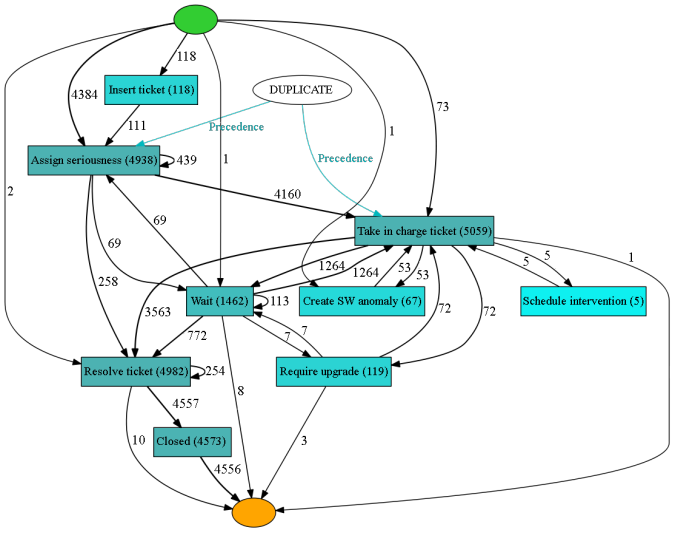


Figure 15: Extended Directly-Follows Graph of cluster 12 in the Italian help desk log.

its rare changes.

Figure 14(b) shows a *gradual* drift until June 2012, and an *incremental* drift afterward. We notice that all constraints in the cluster have Wait either as an activation (e.g., with AlternateResponse(Wait, closed)) or as a target (e.g., with ChainResponse(Take in charge ticket, Wait)).

Finally, we look at cluster 12 with its one-spike drift in Fig. 12(b). The corresponding eDFG in Fig. 15 shows that this behaviour relates to a Take in charge ticket and Assign seriousness.

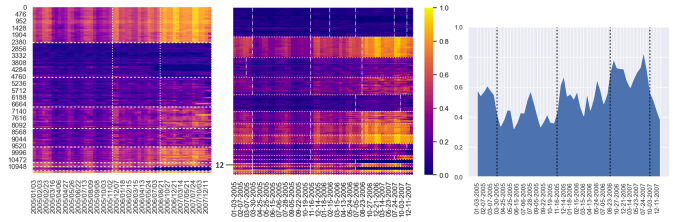
5.3.3 BPI2011 event log

BPI2011 is the log from the the Gynaecology department of a hospital in the Netherlands.

Step 1: Mining Directly-Follows Graph as an overview. The Directly-Follows Graph includes 34 activities. It is shown in Fig. 19. The paths of the cases are largely different, such that no clear main path can be identified.

Step 2: Mining Declare windows. This log contains 1142 cases spanning over a period of three years and four month. We chose the win_{size} of 40 and the win_{step} of 20 in our analysis.

Step 3-4: Finding Drifts and Visual Drift Overview. Figure 16(a) shows the Drift Map of the *BPIC2011* event log. As in [15], two drifts are detected towards the second half of the time span of the log. However, in addition, our technique



(a) Overall change points (b) Drifts by cluster (c) Most erratic cluster (cluster 12)

Figure 16: BPI2011 VDD visualizations.

Table 9: BPI2011 erratic clusters.

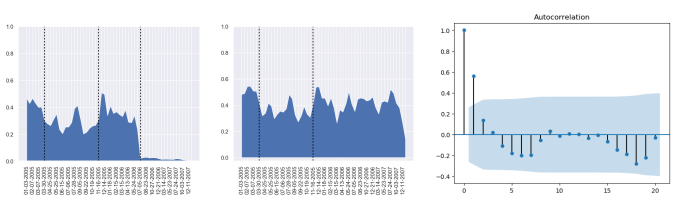
Drift number	Erratic measure	A-Dickey-Fuller p-value
without drift	55	
12	221.791 552 8	0.060 885
6	220.990 778 5	0.479 091
8	215.308 957 5	0.546 296
7	214.006 770 7	0.887 760
1	205.843 939 9	0.760 080

identifies drifting behavior at a finer granularity. Figure 16(b) shows the drifts pertaining to clusters of constraints. The trend of the confidence measure for the most erratic cluster is depicted in Figure 16(c).

While the Drift Map shows that most of the drifts display increasing trends for the plots at the end of the event log timeline, Fig. 17(a) highlights the opposite direction. The most erratic cluster is characterized by a confidence values that decrease from the beginning of the timeline and decreases afterwards.

Step 5: Detecting Drift Types. To better understand a particular drift, we further examine the constraints that participate in the drift. We explore statistical properties of the discovered drifts. We use the erratic measure to identify the strongest drifts and run sudden drift detection in order to identify the drift types. Sudden drifts are visible in Fig. 16(a) that correspond to those found in [15]. Moreover, we are able to discover the sudden drifts for each individual cluster of behavior as shown by vertical lines in Figs. 16(b) and 16(c).

Running the autocorrelation analysis reveals that most of the drifts do not show seasonality. An exception is cluster 15. Its autocorrelation graph (Fig. 17(c)) and Drift Chart (Fig. 17(b)) exhibit seasonality. The Augmented Dickey-Fuller test Table 9 evidences that all of the most erratic clusters are non-stationary. This means that there is a constant change in



(a) Drift Chart of cluster 13 (b) Drift Chart of cluster 15 (c) Autocorrelation plot of cluster 15

Figure 17: BPI2011 visualizations.

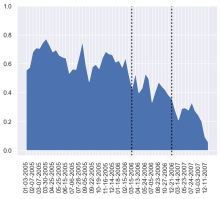


Figure 18: Cluster 16 of BPIC2011

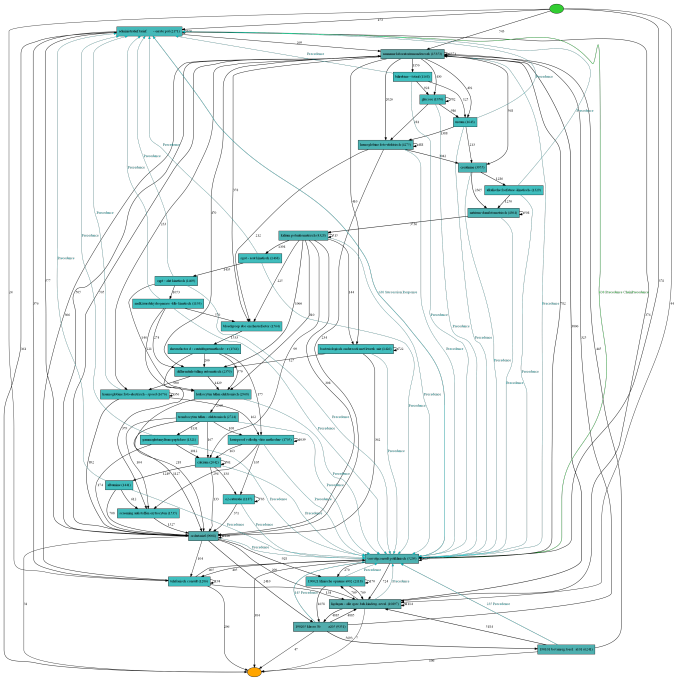


Figure 19: Cluster 16 of BPIC2011

the process behavior, thus we can conclude that those drifts are incremental.

Step 6: Understanding drift behavior. Figure 18 illustrates the drift chart of cluster 16, which we consider for the annotation of the extended DFG in Fig. 19. Apparently, the majority of the drifts in this cluster refer to activity *vervolgconsult poliklinisch*, which is subject to PRECEDENCE constraints with several other activities.

5.4 Computational Performance

We have tested the computational performance of the system. We used a MacOS system, equipped with 2.4 GHz Dual-Core Intel Core i5 and 8 GB of RAM. Table 10 shows the wall-clock time needed for our system to process each data set, and the basic data set characteristic. To determine the computational performance we used parameters applied in our tests from Sections 5.2 and 5.3.

We have measured the computation time of the different steps of the algorithm. First, we measured the time needed to extract time series from the data, cluster, perform change point detection, visualize Drift Map and Drift Charts (Steps 2-4 of our algorithm from Section 4). Second, we measured the time to build extended DFGs for each cluster (Steps 1 and 6). Third, we measured the time employed by the system to generate autocorrelation plots, finding erratic and spread of

Table 10: Characteristics of the event logs and wall-clock time performance of the system expressed in seconds. CM stands for Conditional Move, CR for Conditional Removal, CS for Conditional to Sequence, IHD for Italian Help Desk logs.

Event log	CM	CR	CS	Loop	IHD	BPI2011	Sepsis
$\#_{\text{seq}}$	9998	9999	2999	9999	4579	1142	1050
av.seq.l.	22.27	23.10	23.04	23.13	4.66	98.31	14.49
$\#_{\text{act}}$	27	28	28	27	14	33	16
Steps 2-4	34.98	34.16	27.83	34.77	34.21	48.77	17.39
Steps 1, 6	20.98	18.88	8.94	18.16	5.48	194.19	4.22
Step 5	11.24	10.84	11.86	10.93	10.76	15.04	8.29
Total	**67.24**	**63.87**	**48.63**	**63.82**	**50.45**	**258.02**	**29.90**

constraints measures, and performing the stationarity tests (Step 5).

The tests show that our system if mostly affected by the number of activities, $\#_{\text{act}}$, and the average length of the sequences in the DFG. This parameter is a key factor for the complexity of the extended DFG, as the rendering of the graph appears to be the most costly operation due to the number of DECLARE constraints that need to be visualized for some of the clusters. Indeed, the *BPI2011* event log required the highest amount of time for all steps. The Italian help desk log needed the lowest time to complete all calculations, as $\#_{\text{act}}$ and average sequence length is the lowest of other datasets.

5.5 User Evaluation

The previous part of the evaluation highlights the accuracy of our drift detection and visualization. Our system is designed to meet the requirements of business process analysts. The objective of our user evaluation is to collect evidence in order to judge to which extent the requirements have been effectively addressed. To this end, we conducted a user study with 12 process mining experts who are familiar with different tools and approaches for visualizing business process event logs.

The participants were introduced to the data set of the helpdesk case described in Section 5.3.2 together with its Directly-Follows Graph. Then, the participants learned about the four major visualization techniques of our system (extended directly-follows graph, drift map, drift chart, and drift measures). We collected quantitative and qualitative data via a survey with a Likert scale and open questions.

Our quantitative evaluation builds on the established technology acceptance model [67], [68]. This model posits that the two major antecedents of technology adoption are perceived usefulness and ease of use. In our context, this essentially means that if process analysts perceive our visualization system to be easy to use and to provide useful insights, they would likely want to use it in their daily work. The user perceptions of ease of use and usefulness were assessed using the established psychometric measurement instrument with 5 and 6 question items per construct, respectively [67].

The results of the technology acceptance assessment are presented in Fig. 20. We observe that both ease of use and

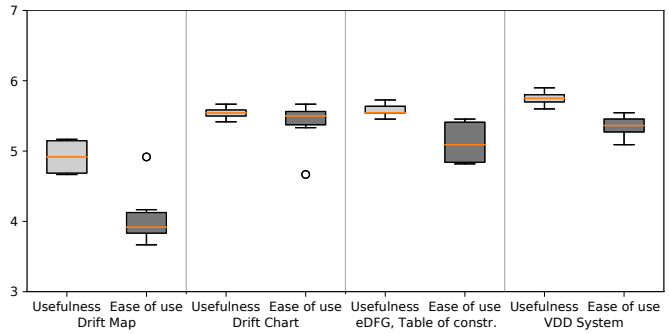


Figure 20: Boxplots of perceived ease of use and perceived usefulness according to the user study

usefulness are perceived to be close to each other, with usefulness being consistently stronger. Both measurement scales have a high average between 5 and 6, indicating that the users agree that the system is easy to use and useful. The exception is the ease of use of the drift map, which was partially judged to be difficult.

Beyond the quantitative assessment, we also collected qualitative feedback on the different visualizations of our system. Regarding the *drift map*, participant P7 states that it "*visualizes in one picture a great amount of detailed information in a useful way. It allows perceiving the changes of all the behavior without query for each of them.*" Participant P2 suggests that it "*would be nice to add the meaning of clusters.*" To address this point, we added the feature to filter the drift map for constraints that relate to a specific activity. Regarding the *drift chart*, Participant P6 notes that it "*is very easy to understand. It clearly shows the compliance of the cases with certain constraints and how it evolves over time.*" Participant P5 suggests some indication "*if less/more traces comply with rules.*" To address this point, we added absolute numbers showing how many cases relate to this chart. Regarding the *extended DFG*, Participant P8 emphasizes that "*I like that they provide details of specific constraints. I like to visually see the process. I like the enhanced process model.*" Participant P5 highlights that "*I see a risk of information overload.*" We address this point by offering a functionality to filter the eDFG. Regarding the overall system, different participants expressed their perceptions on usefulness by emphasizing that the system "*provides very powerful means to explore the process change*" (P6). Participant P8 states that "*I like to see the three visualizations together.*" Participant P5 concludes that the information provided by the system "*certainly improves the accuracy of decisions.*"

5.6 Discussion

Our method addresses all the five requirements for process drift detection presented in Section 2.2 as follows:

- R1 We evaluated our method with the synthetic logs showing its ability to identify drifts precisely;
- R2 We developed a visualization approach based on Drift Maps and Drift Charts for the classification of process drifts and have shown its effectiveness for real-world logs. Our enhanced approach based on change point detection has yielded an effective way o automatically discover the exact points at which *sudden* and *reoccurring concept* drifts occur. The indicative approximation of long-running progress of *incremental* and *gradual* drifts was also found. *Outliers* were detected via time series clustering;
- R3 Using clustering, Drift Map, and Drift Charts, the method enables the drilling down into (rolling up out) sections with a specific behavior (general vs. cluster-specific groups of constraints);
- R4 We introduced, and incorporated into our technique, a drift measure called Erratic that quantifies the extent of the drift change;
- R5 To further qualitatively analyze the detected drifts, VDD shows how the process specification looks before and after the drift (as a list of DECLARE constraints, refer to Table 8).

5.7 Limitations

In this section, we outline the future work directions defined by the limitations of our system.

We noticed that irregularly sampled data could affect the analysis. Our approach splits a log into windows of a fixed number of traces. The irregular data could produce graphs that have unevenly spaced timeticks. Taking into account the time ranges instead of the number of traces will affect our analysis. Different strategies for splitting the log should be investigated in future work.

When interacting with the VDD system, an analyst manually identifies seasonal drifts based on the autocorrelation graphs and explores incremental drifts based on Drift Charts. Future work will aim at automating both these tasks.

As demonstrated in Section 5.4, the performance of the system allows for the handling of industrial datasets. However, this performance is achieved for the offline setting, when the necessary information is precomputed, and does not extend to the online setting, as new input data will trigger an overall recalculation. Extending the system to online settings is another avenue for future work.

For datasets with a large number of possible activities and a significant number of drifts, the performance of the system could be further improved by prioritizing DECLARE constraints that get rendered as DFGs.

Finally, the choices of algorithms for clustering and change-point detection could be informed by the input data. In the case of a large dataset, faster clustering algorithms could be selected. The analysis of such choices on the system's performance is future work.

6 CONCLUSIONS

In this paper, we presented a visual system for the detection and analysis of process drifts from event logs of executed business processes. Our contributions are techniques for fine-granular process drift detection and visualization. The different visualizations of our system integrate extended Directly-Follows Graphs, DECLARE constraints, the Drift Maps and Drift Charts plus several metrics and statistics for determining types of drift.

We evaluated our system both on synthetic and real-world data. On synthetic logs, we achieved an average F-score of 0.96 and outperformed all the state-of-the-art

methods. On real-world logs, the technique describes all types of process drifts in a comprehensive manner. Also, the evaluation reported that our technique can identify outliers of process behavior. Furthermore, we conducted a user study, which highlights that our visualizations are easy to interact with and useful, as perceived by process mining experts.

ACKNOWLEDGMENTS

This work is partially funded by the EU H2020 program under MSCA-RISE agreement 645751 (RISE_BPM). A. Polyvyanyy was in part supported by the Australian Research Council Discovery Project DP180102839. C. Di Ciccio was partly supported by the MUR under grant "Dipartimenti di eccellenza 2018-2022" of the Department of Computer Science at Sapienza University of Rome.

REFERENCES

[1] F. Mannhardt and D. Blinde, "Analyzing the trajectories of patients with sepsis using process mining," in *RADAR+EMISA@CAiSE*, ser. CEUR Workshop Proceedings, vol. 1859. CEUR-WS.org, 2017, pp. 72–80.

[2] W. Aigner, S. Miksch, W. Müller, H. Schumann, and C. Tominski, "Visual methods for analyzing time-oriented data," *IEEE transactions on visualization and computer graphics*, vol. 14, no. 1, pp. 47–60, 2007.

[3] W. Aigner, S. Miksch, H. Schumann, and C. Tominski, *Visualization of time-oriented data*. Springer Science & Business Media, 2011.

[4] S. B. Cousins and M. G. Kahn, "The visual display of temporal information," *Artificial Intelligence in Medicine*, vol. 3, no. 6, pp. 341–357, 1991.

[5] F. B. Viégas, M. Wattenberg, and K. Dave, "Studying cooperation and conflict between authors with history flow visualizations," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2004, pp. 575–582.

[6] S. Havre, E. Hetzler, P. Whitney, and L. Nowell, "Themeriver: Visualizing thematic changes in large document collections," *IEEE transactions on visualization and computer graphics*, vol. 8, no. 1, pp. 9–20, 2002.

[7] N. W. Kim, S. K. Card, and J. Heer, "Tracing genealogical data with timenets," in *Proceedings of the International Conference on Advanced Visual Interfaces*, 2010, pp. 241–248.

[8] J. Mendling, *Metrics for process models: empirical foundations of verification, error prediction, and guidelines for correctness*. Springer Science & Business Media, 2008, vol. 6.

[9] W. M. P. van der Aalst, *Process Mining - Data Science in Action*. Springer, 2016.

[10] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.

[11] V. Denisov, E. Belkina, and D. Fahland, "BPIC'2018: Mining concept drift in performance spectra of processes," 2018.

[12] B. Hompes, J. C. A. M. Buijs, W. M. P. van der Aalst, P. Dixit, and H. Buurman, "Detecting change in processes using comparative trace clustering," in *SIMPDA 2015*, 2015, pp. 95–108.

[13] A. Seeliger, T. Nolle, and M. Mühlhäuser, "Detecting concept drift in processes using graph metrics on process graphs," in *S-BPM*, 2017, p. 6.

[14] C. Zheng, L. Wen, and J. Wang, "Detecting process concept drifts from event logs," in *OTM CoopIS*, 2017, pp. 524–542.

[15] A. Ostovar, A. Maaradji, M. La Rosa, A. H. M. ter Hofstede, and B. F. van Dongen, "Detecting drift from event streams of unpredictable business processes," in *ER*, 2016, pp. 330–346.

[16] A. Maaradji, M. Dumas, M. La Rosa, and A. Ostovar, "Detecting sudden and gradual drifts in business processes from execution traces," *IEEE TKDE*, vol. 29, no. 10, pp. 2140–2154, 2017.

[17] M. Sedlmair, M. D. Meyer, and T. Munzner, "Design study methodology: Reflections from the trenches and the stacks," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2431–2440, 2012.

[18] X. Wang, W. Chen, J. Xia, Z. Chen, D. Xu, X. Wu, M. Xu, and T. Schreck, "ConceptExplorer: Visual analysis of concept driftsin multi-source time-series data," *CoRR*, vol. abs/2007.15272, 2020.

[19] C. Ware, *Information visualization: perception for design*. Elsevier, 2012.

[20] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg, "Declarative workflows: Balancing between flexibility and support," *CS - R&D*, vol. 23, no. 2, pp. 99–113, 2009.

[21] C. Di Ciccio and M. Mecella, "On the discovery of declarative control flows for artful processes," *ACM TMIS*, vol. 5, no. 4, pp. 24:1–24:37, 2015.

[22] C. Truong, L. Oudre, and N. Vayatis, "Selective review of offline change point detection methods," *Signal Process.*, vol. 167, 2020.

[23] C. dos Santos Garcia, A. Meincheim, E. R. F. Junior, M. R. Dallagassa, D. M. V. Sato, D. R. Carvalho, E. A. P. Santos, and E. E. Scalabrin, "Process mining techniques and applications-a systematic mapping study," *Expert Systems with Applications*, 2019.

[24] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 44:1–44:37, 2014.

[25] A. Tsymbal, "The problem of concept drift: definitions and related work," *Computer Science Department, Trinity College Dublin*, vol. 106, no. 2, p. 58, 2004.

[26] M. Bauer, A. Senderovich, A. Gal, L. Grunske, and M. Weidlich, "How much event data is enough? A statistical framework for process discovery," in *CAISE*, 2018, pp. 239–256.

[27] D. Deutch and T. Milo, "A structural/temporal query language for business processes," *J. Comput. Syst. Sci.*, vol. 78, no. 2, pp. 583–609, 2012.

[28] A. Polyvyanyy, M. Weidlich, R. Conforti, M. La Rosa, and A. H. M. ter Hofstede, "The 4C spectrum of fundamental behavioral relations for concurrent systems," in *Petri nets*. Springer, 2014, pp. 210–232.

[29] J. Prescher, C. D. Ciccio, and J. Mendling, "From declarative processes to imperative models," in *SIMPDA*, ser. CEUR Workshop Proceedings, vol. 1293. CEUR-WS.org, 2014, pp. 162–173.

[30] A. Yeshchenko, C. D. Ciccio, J. Mendling, and A. Polyvyanyy, "Comprehensive process drift detection with visual analytics," in *ER*, ser. Lecture Notes in Computer Science, vol. 11788. Springer, 2019, pp. 119–135.

[31] A. Ostovar, S. J. J. Leemans, and M. L. Rosa, "Robust drift characterization from event streams of business processes," *ACM Trans. Knowl. Discov. Data*, vol. 14, no. 3, pp. 30:1–30:57, 2020.

[32] B. C. M. Cappers, P. N. Meessen, S. Etalle, and J. J. van Wijk, "Eventpad: Rapid malware analysis and reverse engineering using visual analytics," in *VizSEC*. IEEE, 2018, pp. 1–8.

[33] P. Xu, H. Mei, L. Ren, and W. Chen, "Vidx: Visual diagnostics of assembly line performance in smart factories," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 291–300, 2017.

[34] S. Guo, Z. Jin, Q. Chen, D. Gotz, H. Zha, and N. Cao, "Visual anomaly detection in event sequence data," in *BigData*. IEEE, 2019, pp. 1125–1130.

[35] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: discovering process models from event logs," *TKDE*, vol. 16, no. 9, pp. 1128–1142, 2004.

[36] W. Aigner, S. Miksch, H. Schumann, and C. Tominski, *Visualization of Time-Oriented Data*, ser. Human-Computer Interaction Series. Springer, 2011.

[37] Y. Guo, S. Guo, Z. Jin, S. Kaul, D. Gotz, and N. Cao, "Survey on visual analysis of event sequence data," *CoRR*, vol. abs/2006.14291, 2020.

[38] Y. Chen, P. Xu, and L. Ren, "Sequence synopsis: Optimize visual summary of temporal event data," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 45–55, 2018.

[39] S. Guo, K. Xu, R. Zhao, D. Gotz, H. Zha, and N. Cao, "Eventthread: Visual summarization and stage analysis of event sequence data," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 56–65, 2018.

[40] Y. Zhang, K. Chanana, and C. Dunne, "Idmvis: Temporal event sequence visualization for type 1 diabetes treatment decision support," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 512–522, 2019.

[41] K. Wongsuphasawat, J. A. G. Gómez, C. Plaisant, T. D. Wang, M. Taieb-Maimon, and B. Shneiderman, "Lifeflow: visualizing an overview of event sequences," in *CHI*. ACM, 2011, pp. 1747–1756.

[42] M. Monroe, R. Lan, H. Lee, C. Plaisant, and B. Shneiderman, "Temporal event sequence simplification," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 12, pp. 2227–2236, 2013.

[43] P. Law, Z. Liu, S. Malik, and R. C. Basole, "MAQUI: interweaving queries and pattern mining for recursive event sequence exploration," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 396–406, 2019.

[44] K. Wongsuphasawat and D. Gotz, "Exploring flow, factors, and outcomes of temporal event sequences with the outflow visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2659–2668, 2012.
[45] Y. Tanahashi and K. Ma, "Design considerations for optimizing storyline visualizations," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2679–2688, 2012.
[46] D. Albers, C. N. Dewey, and M. Gleicher, "Sequence surveyor: Leveraging overview for scalable genomic alignment visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 2392–2401, 2011.
[47] B. C. M. Cappers and J. J. van Wijk, "Exploring multivariate event sequences using rules, aggregations, and selections," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 532–541, 2018.
[48] S. Malik, F. Du, M. Monroe, E. Onukwugha, C. Plaisant, and B. Shneiderman, "Cohort comparison of event sequences with balanced integration of visual analytics and statistics," in *IUI*. ACM, 2015, pp. 38–49.
[49] J. Zhao, Z. Liu, M. Dontcheva, A. Hertzmann, and A. Wilson, "Matrixwave: Visual comparison of event sequence data," in *CHI*. ACM, 2015, pp. 259–268.
[50] S. J. J. Leemans, E. Poppe, and M. T. Wynn, "Directly follows-based process mining: Exploration & a case study," in *ICPM*. IEEE, 2019, pp. 25–32.
[51] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Scalable process discovery with guarantees," in *BMMDS/EMMSAD*, ser. Lecture Notes in Business Information Processing, vol. 214. Springer, 2015, pp. 85–101.
[52] A. Berti, S. J. van Zelst, and W. M. P. van der Aalst, "Process mining for python (pm4py): Bridging the gap between process- and data science," *CoRR*, vol. abs/1905.06169, 2019.
[53] W. M. P. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, 2004.
[54] A. Weijters, W. M. van Der Aalst, and A. A. De Medeiros, "Process mining with the heuristics miner-algorithm," *Technische Universiteit Eindhoven, Tech. Rep. WP*, vol. 166, pp. 1–34, 2006.
[55] A. Polyvyanyy, A. Armas-Cervantes, M. Dumas, and L. García-Bañuelos, "On the expressive power of behavioral profiles," *Formal Asp. Comput.*, vol. 28, no. 4, pp. 597–613, 2016.
[56] C. Di Ciccio, F. M. Maggi, M. Montali, and J. Mendling, "Resolving inconsistencies and redundancies in declarative process models," *IS*, vol. 64, pp. 425–446, Mar. 2017.
[57] F. M. Maggi, A. J. Mooij, and W. M. P. van der Aalst, "User-guided discovery of declarative process models," in *CIDM*. IEEE, 2011, pp. 192–199.
[58] A. Cecconi, G. D. Giacomo, C. D. Ciccio, F. M. Maggi, and J. Mendling, "A temporal logic-based measurement framework for process mining," in *ICPM*. IEEE, 2020, pp. 113–120.
[59] F. M. Maggi, C. Di Ciccio, C. Di Francescomarino, and T. Kala, "Parallel algorithms for the automated discovery of declarative process models," *Inf. Syst.*, vol. 74, no. Part 2, pp. 136–152, 2018.
[60] J. D. Smedt, J. D. Weerdt, E. Serral, and J. Vanthienen, "Discovering hidden dependencies in constraint-based declarative process models for improving understandability," *Inf. Syst.*, vol. 74, no. Part 1, pp. 40–52, 2018.
[61] S. R. Aghabozorgi, A. S. Shirkhorshidi, and Y. W. Teh, "Time-series clustering - A decade review," *Inf. Syst.*, vol. 53, pp. 16–38, 2015.
[62] R. Killick, P. Fearnhead, and I. A. Eckley, "Optimal detection of changepoints with a linear computational cost," *Journal of the American Statistical Association*, vol. 107, no. 500, pp. 1590–1598, 2012.
[63] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.
[64] Y.-W. Cheung and K. S. Lai, "Lag order and critical values of the augmented dickey–fuller test," *Journal of Business & Economic Statistics*, vol. 13, no. 3, pp. 277–280, 1995.
[65] M. Meyer, M. Sedlmair, P. S. Quinan, and T. Munzner, "The nested blocks and guidelines model," *Information Visualization*, vol. 14, no. 3, pp. 234–249, 2015.
[66] A. Yeshchenko, J. Mendling, C. D. Ciccio, and A. Polyvyanyy, "VDD: A visual drift detection system for process mining," in *ICPM Doctoral Consortium / Tools*, ser. CEUR Workshop Proceedings, vol. 2703. CEUR-WS.org, 2020, pp. 31–34.
[67] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS quarterly*, pp. 319–340, 1989.
[68] V. Venkatesh, M. G. Morris, G. B. Davis, and F. D. Davis, "User acceptance of information technology: Toward a unified view," *MIS quarterly*, pp. 425–478, 2003.

Anton Yeshchenko is a Ph.D. student and a teaching and research associate at the Institute for Information Business at Wirtschaftsuniversität Wien, Austria. His research focuses on visualization research, data science, and machine learning for temporal event sequence data. He has a theoretical background in applied mathematics and computer science from Taras Shevchenko University (Ukraine) and Tartu University (Estonia). He has worked at research institute FBK in Trento developing new methods in predictive monitoring of business processes. He has published in international A-ranked conferences including BPM, ER, and CoopIS.

Claudio Di Ciccio is an Assistant Professor with the Department of Computer Science at Sapienza University of Rome. He received a PhD in Computer Science and Engineering in 2013 at Sapienza. His research interests include Process Mining, Blockchains, and Declarative Modelling. His work in these fields has been published in renowned international conferences such as BPM, CAiSE, and ICSOC, and top journals, including Decision Support Systems, Information Systems and ACM Trans. on Management Information Systems. He received the best paper award at BPM 2015 and the best user paper award at ECIR 2019. He is a member of the Steering Committee of the IEEE Task Force on Process Mining.

Jan Mendling is a Full Professor with the Institute for Information Business at Wirtschaftsuniversität Wien, Austria. His research interests include business process management and information systems. He has published more than 400 research papers and articles, among others in the Journal of the Association of Information Systems, ACM Transactions on Software Engineering and Methodology, IEEE Transaction on Software Engineering, Information Systems, European Journal of Information Systems, and Decision Support Systems. He is a board member of the Austrian Society for Process Management, one of the founders of the Berliner BPM-Offensive, and member of the IEEE Task Force on Process Mining. He is a co-author of the textbooks Fundamentals of Business Process Management and Wirtschaftsinformatik.

Dr. Artem Polyvyanyy is a senior lecturer at the School of Computing and Information Systems, Melbourne School of Engineering, at the University of Melbourne (Australia). In March 2012, he received a Ph.D. degree in Computer Science from the University of Potsdam (Germany). His research and teaching interests include Information Systems, Process Modeling and Analysis, Process Mining, Process Querying, Data Science, and Algorithms. Artem Polyvyanyy has published over 80 research papers and articles. He has actively contributed to several open-source initiatives that led to a significant impact on research and practice, including jBPT, Oryx, and Apromore.