Version Clustering: A Top-Down Approach for Process Concept Drift Detection

Bernold Rodrigo Abarca Zúñiga[1,2], Anton Yeshchenko[1], and Han van der Aa[1]

[1] Faculty of Computer Science, University of Vienna, Austria
{bernold.rodrigo.abarca.zuniga,anton.yeshchenko,han.van.der.aa}@univie.ac.at
[2] Doctoral School of Computer Science, University of Vienna, Austria

Abstract. Business processes constantly evolve as organizations adapt to new regulations, technologies, and market conditions. Consequently, models discovered through process mining quickly lose validity unless process concept drifts are detected and managed. The newest and best drift detection approaches rely on supervised learning or complex statistical models, demanding labeled data and high computational effort.
This paper introduces Version Clustering (VC), a novel top-down unsupervised approach for process concept drift detection that is both robust and interpretable. VC's novelty lies in the inversion of the drift detection logic in comparison to existing approaches. Rather than pinpointing drift points outright, VC first clusters windows into coherent process versions via hierarchical density-based clustering, and then marks drift points where execution transitions between versions. This inversion yields three advantages: (i) improved robustness to noise and local behavioral variability, (ii) more accurate predictions, and (iii) a version-level representation that naturally supports visual and analytical interpretation.
Evaluated within a unified experimental framework and tested on established benchmarks, VC achieves high robustness to noise, strong accuracy, practical scalability, coverage of most drift types, and clear interpretability. Under the considered evaluation metrics, it matched or exceeded baseline approaches, offering an efficient and interpretable solution for tracking process evolution in noisy, dynamic environments.

Keywords: Process Mining · Concept Drift Detection · Unsupervised Learning · Process Evolution

1 Introduction

Business processes underpin the operations of modern organizations, and their executions are routinely logged by information systems [19]. Process mining uses such logged data to discover, monitor, and improve processes [4,5]. As they evolve with regulatory, market, and organizational changes, insights from historical logs degrade—a phenomenon known as process concept drift [1]. Detecting such drifts is essential to maintain the validity of obtained insights and support reliable process analysis [6, 15].

A wide range of drift detection techniques has been proposed, spanning statistical window tests, relational comparisons, constraint-based profiling, trace-distribution distances, and, more recently, deep learning [6, 12, 21]. Despite this progress, recent evaluations show that no approach performs consistently well across the crucial criteria of accuracy, latency, robustness, scalability, versatility and parameter sensitivity [1]. Supervised learning methods often achieve the highest accuracy, but require large labeled datasets, complex training pipelines, and full retraining when the behavioral representation changes [12]. Unsupervised, noise robust, and interpretable alternatives remain limited. A possible reason for these limitations is that many existing methods take a bottom-up perspective: they detect drifts by comparing successive windows directly. Since such comparisons are sensitive to local fluctuations, noise and window parametrization, and lack a global view of how behavior changes over the entire log, this bottom-up perspective may impede their accuracy.

Recognizing this, we propose Version Clustering (VC), a top-down, unsupervised approach to process concept drift detection. VC's novelty lies in the inversion of the drift detection logic in comparison to existing approaches. Instead of detecting drift points directly, VC first groups windows into coherent process versions using hierarchical density-based clustering and then detects drifts when execution transitions between versions. This inversion of the detection logic yields three advantages: (i) improved robustness to noise and local behavioral variability, (ii) more accurate predictions, and (iii) a version-level representation that naturally supports visual and analytical interpretation.

We evaluate VC using the unified experimental framework of Adams et al. [1] on established synthetic benchmarks, including the CDLG dataset [10] and the Bose, Ostovar, and Ceravolo logs [4, 8, 17]. VC achieves a solid precision-recall balance (excelling in precision) even at noise levels up to 60%, demonstrating strong robustness. It also offers low detection latency, coverage of most drift types and competitive scalability, outperforming leading supervised and statistical baselines. Finally, we demonstrate the practical applicability of VC through an interactive visualization prototype that integrates version assignments into directly-follows graph exploration, enabling analysts to inspect the concrete behavioral changes underlying detected drifts.

From an Information Systems Engineering perspective, VC can be used as a tool that supports the controlled evolution of process-aware information systems. Rather than treating concept drift detection as an isolated analytical task, we design VC as a reusable and integrable component within the lifecycle of an information system. By providing stable version abstractions instead of isolated drift signals, VC produces outputs that can be directly used for process model revision, compliance checking, and system adaptation.

The remainder of this paper is structured as follows. Section 2 reviews related work and motivates the research gap. Section 3 introduces the Version Clustering approach. Section 4 presents the experimental setup and evaluation results. Section 5 demonstrates how VC supports interactive process analysis. Section 6 discusses key insights, and Section 7 concludes the paper.

2 Background and Related Work

Process drift detection comprises techniques that identify changes in a business process's behavior, structure, or performance over time by analyzing data in event logs. As processes evolve, assuming a stable log leads to inaccurate discovery, conformance checking, and performance assessment [1]. Drift detection therefore aims to segment logs into behaviorally consistent periods.

A broad spectrum of approaches has been proposed. As summarized by Sato et al. [18], existing methods include statistical window-based tests, distance-based comparisons, constraint-oriented profiling, clustering, and learning-based models. These techniques target different drift types (e.g., sudden, gradual, incremental, recurring) across multiple process perspectives. Since control-flow evolution most directly affects discovered model accuracy [2], this work focuses on approaches for behavioral drift detection. Most existing techniques adopt a bottom-up perspective, comparing windows directly to detect local changes. We outline the main families of approaches, their limitations, and motivate the need for a top-down alternative.

Adams et al. [1] group drift detection methods into three stages: *feature extraction*, *process representation*, and *drift assessment*. Feature extraction commonly relies on activity-pair relations such as directly- and eventually-follows counts (used in J-Measure [4], ADWIN-based detectors [5], LCDD [14], and RINV [22]), or more expressive abstractions such as DECLARE constraints (VDD [21]), trace variants (EMD [6]), and partial orders (ProDrift [15]). These features are materialized as feature-population profiles [4], feature-vector time series [21], discovered models [15], or probability distributions over traces [6]. Drift assessment then applies statistical tests [4], distance measures such as EMD [6], model-comparison metrics [15], or general-purpose change-point detection algorithms [1]. To illustrate these perspectives, we next summarize the main families of existing drift detection approaches and their strengths and limitations.

Statistical window-based methods (e.g., J-Measure [4], ADWIN [5]) compare distributions across sliding or adaptive windows using hypothesis tests. While simple and interpretable, they are sensitive to window parameters and degrade under noise.

Distance-based approaches compute similarity between window representations. EMD [6] models each window as a stochastic language and measures differences via Earth Mover's Distance, capturing structural and frequency changes. However, solving minimum-cost flow problems over high-dimensional variant distributions results in high computational cost.

Relation-based approaches track the stability of directly- and eventually-follows relations. RINV [22] uses boolean relation matrices, whereas LCDD [14] applies local completeness checks. Although efficient, they operate at a fine relational granularity and are sensitive to noise and gradual behavioral shifts.

Model-based techniques such as ProDrift [15] and ProGraphs [16] construct process models for each window and compute structural differences. While aligned with process evolution, they inherit the computational cost and noise sensitivity of process discovery.

Visual-analytics and learning-based techniques provide expressive or high-performing detectors. VDD [21] extracts DECLARE-constraint time series and applies clustering to reveal drift patterns. Recent supervised approaches such as CV4CDD-4D [12] detect drifts via object detection on behavioral-similarity images, achieving high accuracy at the cost of labeled data requirements, complex training processes, and retraining when representations change. Comprehensive evaluations [1, 12] confirm that no existing technique consistently excels across accuracy, latency, robustness, scalability, versatility, and parameter sensitivity.

Across these families, most techniques have a common characteristic: they adopt a *bottom-up* perspective, identifying drifts through local comparisons of adjacent windows. This makes them susceptible to short-term fluctuations, noise, and window-boundary effects, while providing limited insight into the global evolution of process behavior. Although clustering has been used for trace-level representation learning [11], it has not been leveraged to group behavioral windows into stable, higher-level process versions.

Therefore, we employ a *top-down clustering perspective*. We first cluster windows into coherent process versions, then detect transitions between versions. This design yields drift detection that is robust to noise, independent of labeled data, and naturally suited for interpretable visualization of process evolution.

3 Approach

This section introduces *Version Clustering* (VC), our top-down approach to process concept drift detection. In contrast to existing techniques that detect changes through direct comparisons of adjacent windows, VC first identifies *stable process versions* by clustering behavioral windows and only then locates the transitions between them. This inversion of the detection logic reduces sensitivity to local noise, reveals global behavioral states, and yields an interpretable representation of process evolution.

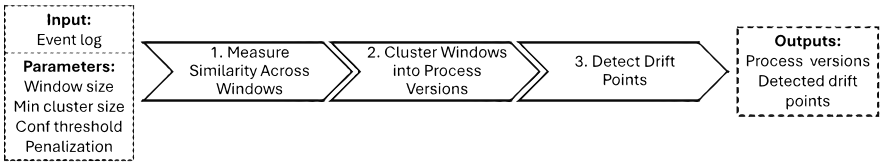

Fig. 1. Overview of the Version Clustering pipeline.

As shown in Figure 1, the approach consists of three steps. (1) The behavioral similarity across different time windows of the event log is computed. (2) The windows are clustered into process versions based on their similarity profiles. (3) A change-point detection algorithm identifies the positions where the execution drifts from one process version to another.

Input. The input is an event log L, containing events e with at least a case identifier, activity label, and timestamp. A *trace* σ is a timestamp-ordered sequence of events from a single case. We use Σ_L to denote the chronologically ordered collection of traces, sorted by the timestamp of their first event.

Parameters. VC requires three parameters: a number of windows N (Step 1), a minimum cluster size for HDBSCAN (Step 2), a cluster membership confidence threshold for HDBSCAN (Step 2), and a penalization parameter for KernelCPD (Step 3). Parameter selection and sensitivity is assessed in Section 4.

Output. Given L, the approach produces (i) a partition of the N windows into *process versions*, and (ii) a set of detected drift points $DP = \{dp_1, \ldots, dp_M\}$, where each dp_i indicates the window at which a drift is detected. These results are visualized jointly to show process evolution over time.

3.1 Step 1: Measure Similarity Across Windows

The first step segments the event log into N windows and computes behavioral similarity between every pair of windows. This produces an $N \times N$ matrix of window-to-window similarities that compactly summarizes process evolution over time. For this step, we follow existing work, such as done in CONDA-PM [9], and CV4CDD-4D [12]. Figure 2 shows the pipeline of this step and Table 1 provides an example for each of the sub-steps using two windows w_i and w_j.

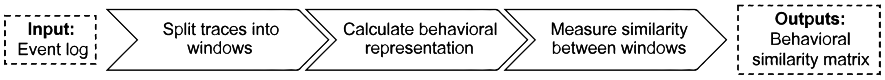

Fig. 2. Behavior representation extraction and similarity computation, adapted from [12].

Split traces into windows. The ordered trace sequence Σ_L is divided into N non-overlapping windows $W = \langle w_1, \ldots, w_N \rangle$, each containing approximately $|\Sigma_L|/N$ traces. The example shows four unique trace variants split across two windows, where $\langle a, b, c \rangle$ appears twice in w_i.

Calculate behavioral representations. For each window w_i, we compute a behavioral representation $B(w_i)$ based on directly-follows frequencies. Each $B(w_i)$ is a vector of frequencies of directly-follows relations observed in the window. Table 1 shows frequency differences for $a \to b$ and $b \to c$, and the absence of $c \to d$ in w_i.

Measure similarity betweeen windows. We compute the similarity between the behavioral representations of each pair of windows via cosine similarity: $S[i, j] = \text{sim}(B(w_i), B(w_j))$, yielding a symmetric similarity matrix S. Table 1 shows the computation of one entry in S, where the frequency differences in the behavioral representations of w_i and w_j lead to a cosine similarity of 0.83.

Table 1. Example of behavioral similarity calculation, adapted from [12].

Windows	Traces	Behavioral representation	Similarity measure
w_i	$\langle a,b,c\rangle^2$ $\langle a,c\rangle$	$B(w_i) = \begin{pmatrix} a \to b : 2 \\ b \to c : 2 \\ a \to c : 1 \\ c \to d : 0 \end{pmatrix}$	$S[i,j] = \text{cosine}\Big(\begin{bmatrix} 2 \\ 2 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \Big) = 0.83$
w_j	$\langle a,b,c\rangle$ $\langle a,c,d\rangle$	$B(w_j) = \begin{pmatrix} a \to b : 1 \\ b \to c : 1 \\ a \to c : 1 \\ c \to d : 1 \end{pmatrix}$	

This step compresses the event log into a fixed-size $N \times N$ matrix, enabling efficient analysis even for large logs, since all subsequent steps depend only on N and not on the number of traces or events. In this matrix, the i-th row corresponds to the similarity vector $S(w_i)$ representing the behavioral similarity between window i and all N windows.

3.2 Step 2: Cluster Windows into Process Versions

The second step performs clustering over windows to identify *process versions*. Clustering windows based on their behavioral similarity is the key novelty of our top-down approach. This step enables the approach to filter out local fluctuations and uncover stable behavioral regions that provide the basis for subsequent drift detection.

To do this, we apply the HDBSCAN algorithm [7], chosen for its robustness to noise and ability to discover clusters of varying density without requiring the number of clusters in advance. Its key parameter is the *minimum cluster size*, defining the smallest number of windows needed to form a *process versions*.

For each window w_i, the similarity vector $S(w_i)$ is input to HDBSCAN, obtaining a probability vector $P(w_i)$ over the discovered clusters. We then assign each window to the version with the highest likelihood unless its membership confidence is below the specified threshold. Table 2 shows five consecutive windows with their similarity vectors and HDBSCAN-derived cluster probabilities across three process versions. In this example, assuming a confidence threshold of 0.75, w_1 would be assigned to process version 1, w_2 and w_3 to version 2, w_4 to version 0 and w_5 to version 3. It classifies w_4 as a noisy window (version 0) because its highest cluster probability (0.70) falls below the 0.75 threshold.

3.3 Step 3: Detect Drift Points

The final step identifies the windows where the execution transitions from one process version to another. While version assignments can reveal major shifts,

Table 2. Example of clustering windows into versions (assigned clusters in bold).

Window	Cosine Similarity Vector	Cluster Probability
w_1	$S(w_1) = [0.80, \ldots, 0.30, 0.25]$	$P(w_1) = [\mathbf{0.90}, 0.05, 0.05]$
w_2	$S(w_2) = [0.80, \ldots, 0.50, 0.50]$	$P(w_2) = [0.10, \mathbf{0.80}, 0.10]$
w_3	$S(w_3) = [0.60, \ldots, 0.50, 0.55]$	$P(w_3) = [0.01, \mathbf{0.98}, 0.01]$
w_4	$S(w_4) = [0.30, \ldots, 0.60, 0.70]$	$P(w_4) = [0.10, 0.20, 0.70]$
w_5	$S(w_5) = [0.40, \ldots, 0.65, 0.80]$	$P(w_5) = [0.08, 0.10, \mathbf{0.82}]$

it would require manually pinpointing the exact drift window, which may be difficult in noisy logs or under gradual drift (exact start and end drift points may be hard to determine). Therefore, to automate drift detection and ensure better predictions, we apply a dedicated change-point detection algorithm.

We use KernelCPD [3], a kernel-based method capable of detecting nonlinear distributional changes. KernelCPD requires a *penalization* parameter that controls the number of detected change points: lower penalties detect more local fluctuations, whereas higher penalties focus on major shifts.

The algorithm operates on the probability sequences $\langle P(w_1), \ldots, P(w_N)\rangle$ and outputs the set of drift-point indices DP. Continuing with the example in Table 2, the KernelCPD algorithm would identify w_2 and w_4 as drift points. These results are already intuitive from Step 2, as w_2 is the first window following a version change and w_4, although classified as noise, indicates the end of version 2 and the transition to version 3.

3.4 Output

VC produces two complementary outputs: (i) a version assignment for each window, and (ii) a list of detected drift points. These are visualized as shown in Figure 3, providing a compact overview of the process evolution, where each window is classified according to its corresponding process version ID and the real and predicted drifts are displayed.

Version ID 0 is reserved for low-confidence windows, indicating noise or intermediate states (e.g., during gradual drifts). The confidence threshold parameter affects only visualization and not drift detection. It allows analysts to choose the confidence needed to visualize a window as a version.

The resulting representation supports not only drift detection but also downstream tasks such as interpreting process changes and identifying period-specific behaviors, as demonstrated in Section 5. Moreover, it could also be extended to classify drift types in future work.

4 Evaluation

This section provides a comprehensive empirical evaluation of Version Clustering (VC) along the six dimensions defined in the unified framework of Adams

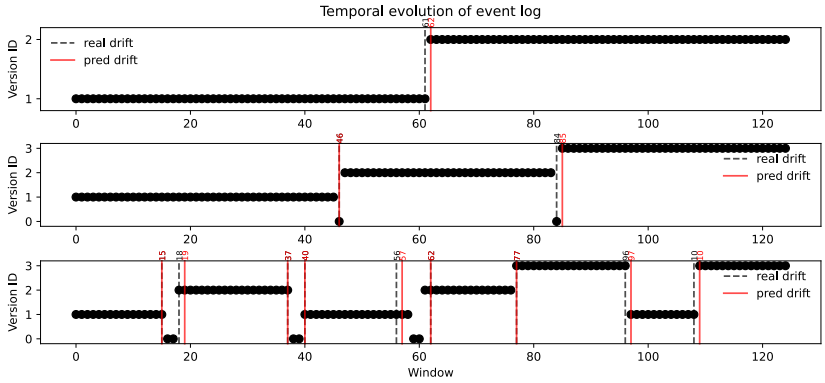

Fig. 3. Example VC outputs for multiple event logs.

et al. [1]. Our aim is not only to quantify detection performance but also to understand VC's behavior across diverse conditions, including noise, latency, drift types, and parameter choices. All experiments follow the protocol and baselines defined by the unified framework, ensuring comparability with prior work. To support reproducibility, we release our full implementation, logs, and raw outputs.[3] We begin by describing the datasets (Section 4.1), then the evaluation metrics and setup (Section 4.2), and finally present the results (Section 4.3).

4.1 Data

To evaluate VC, we use two synthetic datasets with known (gold-standard) drift points, which have been used in prior evaluations of process concept drift detection [1, 12]. A summary of dataset characteristics is shown in Table 3.

CDLG dataset. The CDLG dataset[4] consists of 7,500 event logs generated with the Concept Drift Log Generator [10]. Logs contain multiple drift points and systematically vary in noise level. For comparability, we reuse the exact logs used in the CV4CDD–4D evaluation [12].

CDRIFT dataset. The CDRIFT dataset[5] contains 90 logs generated from three established sources [4, 8, 17]. The logs vary in size, structure, and drift type, and form the benchmark set of the unified evaluation framework [1]

Together, CDLG and CDRIFT cover two complementary evaluation scenarios: CDLG provides large-scale logs with controlled noise and drift placement, while CDRIFT offers heterogeneous logs derived from multiple process generators and drift types. This combination enables both controlled and realistic testing of VC.

[3] `https://gitlab.cs.univie.ac.at/bernolda00cs/version-clustering-cdd`
[4] `https://huggingface.co/datasets/pm-science/cv4cdd_4d`
[5] `https://github.com/cpitsch/cdrift-evaluation/tree/main/EvaluationLogs`

Table 3. Characteristics of the datasets used for evaluation.

Characteristic	CDLG	CDRIFT
Event logs	7,500	90
Logs without drift	1,834	0
Logs with injected noise	3,768	66
Average number of traces	7,200	1,700
Ground-truth change points	22,567	132

4.2 Evaluation Setup

We evaluate VC following the unified framework of Adams et al. [1], ensuring methodological comparability with prior work. The framework defines six complementary evaluation dimensions: accuracy, latency, robustness, scalability, versatility, and parameter sensitivity. Below we describe how each dimension is measured and outline the baselines used for comparison.

Evaluation metrics. We compute the following evaluation metrics:
Accuracy. Measured using precision, recall, and F1-score. A predicted drift is considered correct if it falls within a predefined latency window around a ground-truth drift, allowing for slight temporal misalignment.
Latency. Quantifies how close detected drifts lie to their true positions. Following established practice, we report accuracy under latency thresholds of 1%, 2.5%, and 5% of the log length, capturing both strict and relaxed detection scenarios.
Robustness. Evaluates sensitivity to noise. Using the CDLG dataset, which includes logs with systematically injected noise, we assess detection performance at 0%, 30%, and 60% noise levels.
Scalability. Measures runtime efficiency across heterogeneous log sizes. We report average execution time per log, case, and event.
Versatility. Assesses the ability to detect different types of process change. We follow the taxonomy of sixteen change types introduced by Weber et al. [20].
Parameter sensitivity. To analyze robustness to parameter choices, we conduct a grid search over window size, minimum cluster size, and penalization. For each dataset, we evaluate 125 parameter combinations and report F1-scores at a 5% latency threshold.

Baselines. We compare VC against representative approaches from all major families of process concept drift detection, as introduced in Section 2:
- **statistical:** J-Measure [4], ADWIN [5],
- **model-based:** ProDrift [15], ProGraphs [16],
- **relation-based:** RINV [22], LCDD [14],
- **distance-based:** EMD [6],
- **supervised learning:** CV4CDD–4D [12].

For a fair comparison, we adopt the parameter configurations reported by Kraus et al. [12] for all baselines.

4.3 Results

We now present the results of our evaluation across the six dimensions introduced above. We begin by briefly summarizing the parameter settings used for VC in all experiments. These configurations were obtained from a systematic sensitivity analysis on representative logs. Table 4 summarizes the window size, minimum cluster size, and penalization value used for each data source.

Table 4. Parameter selection

Data	Win.	Min.	Pen.
CDLG	125	10	1.5
Ceravolo	125	5	3.0
Bose	75	5	3.0
Ostovar	75	7	3.0

Accuracy and latency. Table 5 reports precision, recall, and F1-scores for all techniques on the CDLG and CDRIFT datasets under the three latency thresholds. The results highlight the distinct characteristics of the two benchmarks and the conditions under which VC performs particularly well.

Table 5. Overall results for detecting change points (precision, recall, F1) across latency thresholds.

Dataset	Technique	Latency 1%			Latency 2.5%			Latency 5%		
		Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**
CDLG	J-Measure	0.32	0.39	0.25	0.49	0.60	0.54	0.57	0.70	0.63
	Adwin/J	0.43	0.28	0.34	0.58	0.38	0.46	0.63	0.42	0.50
	ProGraphs	0.24	0.26	0.25	0.48	0.52	0.50	0.58	0.64	0.61
	ProDrift	0.55	0.22	0.32	0.74	0.30	0.43	0.76	0.31	0.44
	RINV	0.36	0.44	0.40	0.46	0.56	0.51	0.53	0.64	0.58
	EMD	0.36	0.44	0.39	0.51	0.62	0.56	0.58	0.71	0.64
	LCDD	0.27	0.41	0.32	0.39	0.61	0.48	0.46	0.72	0.56
	CV4CDD–4D	**0.87**	**0.75**	**0.81**	0.90	**0.77**	**0.83**	0.90	**0.77**	0.83
	VC	0.82	0.66	0.73	**0.92**	0.74	0.82	**0.94**	0.75	**0.84**
CDRIFT	J-Measure	0.14	0.10	0.11	0.57	0.69	0.63	0.69	0.81	0.75
	Adwin/J	0.40	0.24	0.30	0.91	0.40	0.56	0.91	0.55	0.69
	ProGraphs	0.67	0.43	0.53	0.84	0.54	0.65	0.92	0.66	0.77
	ProDrift	**0.91**	0.34	0.50	**0.98**	0.36	0.53	**1.00**	0.37	0.54
	RINV	0.02	0.02	0.02	0.65	0.48	0.55	0.68	0.51	0.58
	EMD	0.78	**0.49**	**0.60**	0.80	**0.77**	**0.78**	0.87	0.83	0.85
	LCDD	0.12	0.40	0.18	0.27	0.53	0.35	0.35	0.85	0.50
	CV4CDD–4D	0.17	0.17	0.17	0.52	0.54	0.53	0.79	0.83	0.81
	VC	0.40	0.32	0.36	0.80	0.64	0.71	0.96	**0.96**	**0.96**

On the **CDLG dataset**, which contains many drift points and systematically varied noise levels, performance differences between techniques are particularly pronounced at strict latency thresholds. At the 1% latency level, the supervised CV4CDD–4D approach achieves the highest F1-score (0.81), followed by VC

(0.73). As latency constraints relax, VC improves steadily, reaching an F1-score of 0.84 at the 5% threshold-the best performance across all techniques. Classical statistical and relation-based methods remain consistently behind, particularly at strict thresholds, reflecting the difficulty of CDLG's dense drift scenarios.

On the **CDRIFT dataset**, which includes logs of heterogeneous size and drift patterns, performance is more balanced across approaches. At the 1% threshold, EMD (0.60), ProGraphs (0.53), and ProDrift (0.50) achieve the highest F1-scores, with VC achieving 0.36. At the more realistic 2.5% latency, performance tightens: EMD leads with 0.78, followed by VC at 0.71. At the 5% level, VC reaches an F1-score of 0.96, outperforming all baselines with a near-perfect balance of precision (0.96) and recall (0.96). This demonstrates that VC can capture version transitions reliably across diverse drift types once a small temporal tolerance is permitted.

Robustness. Table 6 reports precision, recall, and F1-scores for all techniques on the CDLG dataset at 5% latency and three noise levels. Across all noise settings, VC maintains consistently high precision (0.94) and strong F1-scores, achieving the highest results at both 30% and 60% noise. CV4CDD–4D performs strongly at zero noise but exhibits a decline in recall as noise increases. ProGraphs shows gradual improvements as noise increases, but remains substantially below the performance level of VC. Overall, VC is substantially less affected by heavy noise than most unsupervised baselines and maintains competitive recall even at 60% noise.

Table 6. Impact of noise on detection accuracy (CDLG dataset, 5% latency).

Technique	**No noise**			**30% noise**			**60% noise**		
	Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**
J-Measure	0.59	0.70	0.64	0.57	0.70	0.63	0.55	0.71	0.62
Adwin/J	0.64	0.43	0.51	0.62	0.41	0.50	0.62	0.40	0.49
ProGraphs	0.55	0.64	0.59	0.60	0.66	0.63	0.64	0.61	0.63
ProDrift	0.76	0.60	0.67	0.67	0.00	0.00	0.33	0.00	0.00
RINV	0.63	**0.83**	0.72	0.39	0.41	0.40	0.40	0.48	0.44
EMD	0.58	0.72	0.64	0.59	0.70	0.64	0.57	0.70	0.62
LCDD	0.63	0.76	0.69	0.35	0.77	0.48	0.36	0.60	0.45
CV4CDD–4D	0.89	0.78	**0.83**	0.89	0.75	0.82	0.89	0.75	0.81
VC	**0.94**	0.71	0.81	**0.94**	**0.82**	**0.88**	**0.94**	**0.78**	**0.85**

Scalability. As shown in Table 7, LCDD is the fastest technique on CDRIFT. VC achieves the fourth-fastest runtime, outperforming ProGraphs by a factor of almost three and vastly outperforming EMD, whose minimum-cost-flow computations lead to high computational overhead. VC's runtime is dominated by feature extraction and clustering over window representations, which both scale linearly with the number of events, explaining its competitive performance.

Table 7. Runtime comparison on the CDRIFT dataset.

Technique\Runtime	per log(s)	per case(ms)	per event(ms)
J-Measure	433.63	225.64	8.67
Adwin/J	154.89	80.60	3.09
ProGraphs	14.92	7.76	0.30
ProDrift	158.53	82.49	3.16
RINV	2.89	1.51	0.06
EMD	516.80	268.91	10.32
LCDD	**0.09**	**0.05**	**0.002**
CV4CDD–4D	4.01	2.08	0.08
VC	6.59	3.43	0.13

Versatility. Versatility results were obtained on the CDRIFT dataset: across sixteen change types, VC detected thirteen with 100% accuracy. It failed to detect three types - duplicate activity, synchronizing two activities, and adding/removing an activity. Nevertheless, the overall results demonstrate strong detection across most change types.

Parameter sensitivity. We assessed VC's sensitivity by evaluating all 125 parameter combinations on the CDLG dataset. Figure 4 visualizes the resulting F1-scores as a grid of heatmaps over window size, minimum cluster size, and penalization. The figure shows a broad high-performing region spanning window sizes between 100 and 150, cluster sizes between 7 and 15, and penalization values of 1.5 and 3.0. Rather than sharp optima, the performance surfaces exhibit smooth plateaus with gradual degradation only at extreme parameter values.

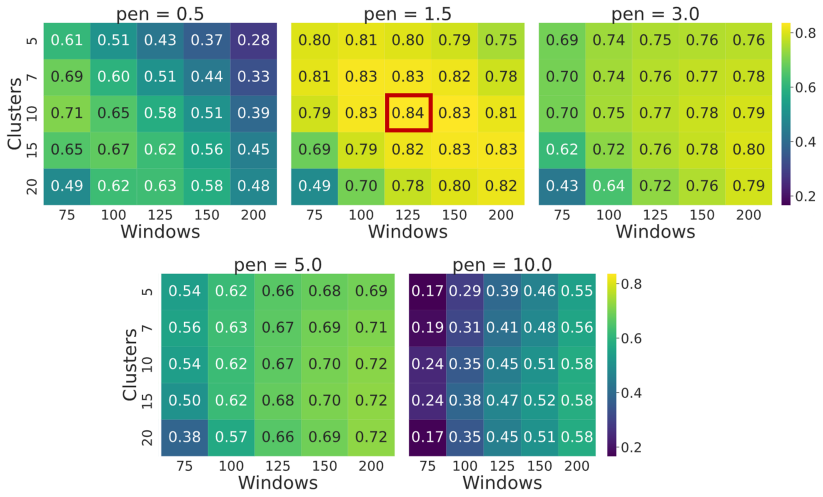

Fig. 4. Sensitivity analysis (F1-scores at 5% latency).

The configurations used in our experiments lie within these stable regions, and the same qualitative pattern holds across all other datasets. This confirms that VC is robust to moderate parameter variation and does not depend on fine-grained hyperparameter tuning.

5 Using Version Clustering in Process Analysis

To demonstrate the practical value of Version Clustering (VC), we developed an interactive visualization prototype: *Process Change Exploration by Version Clustering*[6] *(PCE-VC)* that integrates VC's outputs into a directly-follows graph (DFG)-based analysis system (Figure 5). The prototype shows how VC's top-down drift segmentation can support interactive exploration of process evolution and makes the results of our algorithm directly usable within analysis workflows.

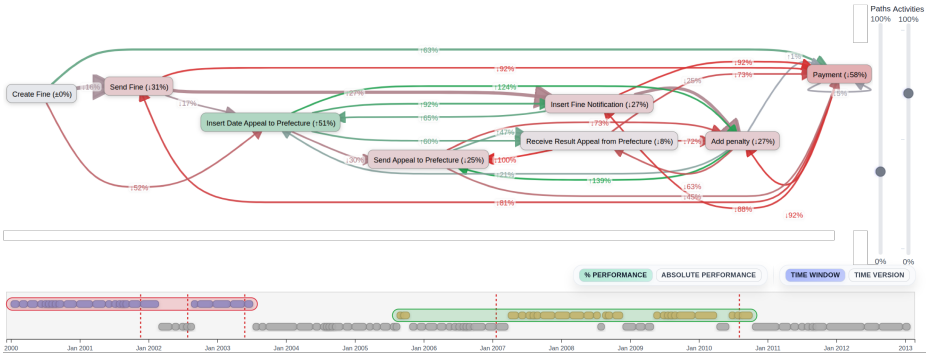

Fig. 5. PCE-VC: exploration of process changes using Version Clustering outputs.

PCE-VC is a lightweight, browser-based visualization system combining (i) a DFG showing activity and path frequencies, and (ii) a coordinated timeline on which users can select individual windows or entire VC-derived process versions. When two periods or versions are selected, the DFG uses a green/red delta encoding to highlight node and edge frequency differences, allowing analysts to quickly see which parts of the process changed and by how much.

Figure 5 illustrates a comparison on the Road Traffic Fine Management event log [13]. The two VC-identified versions correspond to distinct operational phases of the fine-handling procedure. The visualization reveals clear structural shifts: the frequencies of `Send Fine` and `Payment` decrease substantially, while appeal-related activities such as `Insert Date Appeal to Prefecture` become more prominent. Although direct payments (i.e., paths from `Create Fine` to `Payment`) increase, payments associated with appeals (e.g., paths following `Send Appeal to Prefecture`) decrease noticeably. The system also shows an overall decline

[6] `https://github.com/yesanton/Process-Change-Exploration-by-Version-Clustering`

in payment behavior - its relative prominence drops by approximately 58% - which suggests a shift in compliance or internal processing policy. These visual differences align closely with the drift points detected by VC, which separate windows with distinct behavioral clusters.

By pairing VC's version assignments with an interpretable visual encoding, PCE-VC shows that the approach presented in this paper is not only effective in detecting drifts but also readily integrable into practical process-analysis tools. The prototype enables analysts to move seamlessly from identifying a drift to understanding its concrete manifestations in the process model. This demonstrates the broader value of VC: it provides structured, interpretable outputs that support a wide range of analysis tasks, including root-cause investigation, communication of process evolution, and monitoring of ongoing operational changes.

6 Discussion

This section reflects on the insights obtained from the evaluation and discusses the strengths, limitations, and implications of Version Clustering (VC). The results show that VC provides a strong alternative to existing process concept drift detection techniques. By clustering behavioral windows into stable process versions before detecting transitions, it captures process evolution at a higher level of abstraction than bottom-up approaches that compare adjacent windows in isolation. This abstraction allows VC to identify meaningful phase shifts even in the presence of local fluctuations or short-lived deviations that could mislead bottom-up detectors.

A key strength of the method is its robustness to noise. At both 30% and 60% noise levels, it achieves the highest F1-scores among all evaluated techniques. This behavior reflects the strengths of HDBSCAN, which separates unstable windows from stable behavioral patterns, and the use of aggregated similarity profiles that emphasize dominant trends over momentary anomalies.

Accuracy is another area where the approach performs strongly. At practical latency thresholds, it even surpasses supervised methods in some settings, demonstrating that state-of-the-art performance can be achieved without labeled data or computationally expensive model training. Its consistent results across heterogeneous logs further indicate that it generalizes well across diverse drift patterns and log sizes.

From a runtime perspective, VC is efficient and scalable. The fixed-size similarity matrix and window-level compression yield runtimes close to the supervised baseline and substantially faster than model-based or distance-based approaches. These properties make the method suitable for large-scale logs and interactive analysis scenarios.

The outputs are also inherently interpretable: version assignments summarize stable behavioral phases, and transitions highlight where the process changes.

Despite these strengths, several limitations remain. Recall at very strict latency thresholds is lower than for point-level detectors, as clustering can absorb small or transient changes into broader versions. Certain drift types-especially

those producing subtle or highly localized structural changes-also remain difficult to identify. Parameter selection, while manageable, still requires attention, and the current representation based on directly-follows frequencies may miss more nuanced structural or temporal characteristics. These aspects present natural directions for extending the approach.

7 Conclusion

This paper introduced Version Clustering (VC), a top-down approach to process concept drift detection that identifies stable behavioral versions before detecting transitions between them. By shifting from traditional bottom-up drift signals to a holistic representation of process evolution, VC offers a global and structurally robust perspective on how processes change over time.

Experiments on two benchmark datasets show that VC delivers high robustness to noise, strong accuracy, practical scalability, broad drift-type coverage, and interpretability. Despite requiring neither labeled data nor model training, VC performs on par with-and in some cases surpasses state-of-the-art methods, including supervised techniques such as CV4CDD-4D. These characteristics make VC particularly suitable for real-world environments where logs are large, noisy, and continuously evolving.

The design of VC also opens several promising directions for future work. More expressive behavioral representations, such as graph embeddings, constraint-based abstractions, or probabilistic models, could increase sensitivity to subtle structural variations. Exploring alternative clustering strategies or online change-point detection methods could further enhance performance or enable online monitoring. Moreover, the version-based abstraction created by VC provides a natural foundation for automatic drift type classification, including sudden, gradual, and recurring changes.

In summary, VC offers an effective, scalable, and interpretable framework for capturing and understanding process evolution, contributing both theoretical insights and practical tools to the field of process mining.

Acknowledgment. This research was funded in part by the Austrian Science Fund (FWF) 10.55776/PIN7850524.

References

1. Adams, J.N., Pitsch, C., Brockhoff, T., van der Aalst, W.M.P.: An experimental evaluation of process concept drift detection. Proc. VLDB Endow. **16**(8), 1856–1869 (2023). https://doi.org/10.14778/3594512.3594517, https://www.vldb.org/pvldb/vol16/p1856-adams.pdf
2. Adams, J.N., van Zelst, S.J., Rose, T., van der Aalst, W.M.P.: Explainable concept drift in process mining. Information Systems **114**, 102177 (Mar 2023). https://doi.org/10.1016/j.is.2023.102177
3. Arlot, S., Celisse, A., Harchaoui, Z.: A kernel multiple change-point algorithm via model selection. J. Mach. Learn. Res. **20**, 162:1–162:56 (2019), https://jmlr.org/papers/v20/16-155.html

4. Bose, R.P.J.C., van der Aalst, W.M.P., Zliobaite, I., Pechenizkiy, M.: Handling concept drift in process mining. In: Mouratidis, H., Rolland, C. (eds.) Advanced Information Systems Engineering - 23rd International Conference, CAiSE 2011, London, UK, June 20-24, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6741, pp. 391–405. Springer (2011). https://doi.org/10.1007/978-3-642-21640-4_30, https://doi.org/10.1007/978-3-642-21640-4_30
5. Bose, R.P.J.C., van der Aalst, W.M.P., Zliobaite, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE Trans. Neural Networks Learn. Syst. **25**(1), 154–171 (2014). https://doi.org/10.1109/TNNLS.2013.2278313, https://doi.org/10.1109/TNNLS.2013.2278313
6. Brockhoff, T., Uysal, M.S., van der Aalst, W.M.P.: Time-aware concept drift detection using the earth mover's distance. In: van Dongen, B.F., Montali, M., Wynn, M.T. (eds.) 2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, October 4-9, 2020. pp. 33–40. IEEE (2020). https://doi.org/10.1109/ICPM49681.2020.00016, https://doi.org/10.1109/ICPM49681.2020.00016
7. Campello, R.J.G.B., Moulavi, D., Zimek, A., Sander, J.: Hierarchical density estimates for data clustering, visualization, and outlier detection. ACM Trans. Knowl. Discov. Data **10**(1), 5:1–5:51 (2015). https://doi.org/10.1145/2733381, https://doi.org/10.1145/2733381
8. Ceravolo, P., Tavares, G.M., Junior, S.B., Damiani, E.: Evaluation goals for online process mining: A concept drift perspective. IEEE Trans. Serv. Comput. **15**(4), 2473–2489 (2022). https://doi.org/10.1109/TSC.2020.3004532, https://doi.org/10.1109/TSC.2020.3004532
9. El-Khawaga, G., Abu-Elkheir, M., Barakat, S.I., Riad, A.M., Reichert, M.: CONDA-PM - A systematic review and framework for concept drift analysis in process mining. Algorithms **13**(7), 161 (2020). https://doi.org/10.3390/A13070161, https://doi.org/10.3390/a13070161
10. Grimm, J., Kraus, A., van der Aa, H.: CDLG: A tool for the generation of event logs with concept drifts. In: Janiesch, C., Francescomarino, C.D., Grisold, T., Kumar, A., Mendling, J., Pentland, B.T., Reijers, H.A., Weske, M., Winter, R. (eds.) Proceedings of the Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Track at BPM 2022 co-located with 20th International Conference on Business Process Management (BPM 2022), Münster, Germany, September 11th to 16th, 2022. CEUR Workshop Proceedings, vol. 3216, pp. 92–96. CEUR-WS.org (2022), https://ceur-ws.org/Vol-3216/paper_241.pdf
11. Koninck, P.D., vanden Broucke, S., Weerdt, J.D.: act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11080, pp. 305–321. Springer (2018). https://doi.org/10.1007/978-3-319-98648-7_18, https://doi.org/10.1007/978-3-319-98648-7_18
12. Kraus, A., van der Aa, H.: Machine learning-based detection of concept drift in business processes. Process Science **2**(1), 5 (Apr 2025). https://doi.org/10.1007/s44311-025-00012-w
13. de Leoni, M.M., Mannhardt, F.: Road traffic fine management process (2015). https://doi.org/10.4121/UUID:270FD440-1057-4FB9-89A9-B699B47990F5, https://data.4tu.nl/articles/_/12683249/1
14. Lin, L., Wen, L., Lin, L., Pei, J., Yang, H.: LCDD: detecting business process drifts based on local completeness. IEEE Trans. Serv. Comput. **15**(4), 2086–

2099 (2022). https://doi.org/10.1109/TSC.2020.3032787, https://doi.org/10.1109/TSC.2020.3032787
15. Maaradji, A., Dumas, M., Rosa, M.L., Ostovar, A.: Fast and accurate business process drift detection. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9253, pp. 406–422. Springer (2015). https://doi.org/10.1007/978-3-319-23063-4_27, https://doi.org/10.1007/978-3-319-23063-4_27
16. Martjushev, J., Bose, R.P.J.C., van der Aalst, W.M.P.: Change point detection and dealing with gradual and multi-order dynamics in process mining. In: Matulevicius, R., Dumas, M. (eds.) Perspectives in Business Informatics Research - 14th International Conference, BIR 2015, Tartu, Estonia, August 26-28, 2015, Proceedings. Lecture Notes in Business Information Processing, vol. 229, pp. 161–178. Springer (2015). https://doi.org/10.1007/978-3-319-21915-8_11, https://doi.org/10.1007/978-3-319-21915-8_11
17. Ostovar, A., Leemans, S.J.J., Rosa, M.L.: Robust drift characterization from event streams of business processes. ACM Trans. Knowl. Discov. Data **14**(3), 30:1–30:57 (2020). https://doi.org/10.1145/3375398, https://doi.org/10.1145/3375398
18. Sato, D.M.V., Freitas, S.C.D., Barddal, J.P., Scalabrin, E.E.: A survey on concept drift in process mining. ACM Comput. Surv. **54**(9), 189:1–189:38 (2022). https://doi.org/10.1145/3472752, https://doi.org/10.1145/3472752
19. Van Der Aalst, W.: Process Mining. Springer (2016). https://doi.org/10.1007/978-3-662-49851-4, http://link.springer.com/10.1007/978-3-662-49851-4
20. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data Knowl. Eng. **66**(3), 438–466 (2008). https://doi.org/10.1016/J.DATAK.2008.05.001, https://doi.org/10.1016/j.datak.2008.05.001
21. Yeshchenko, A., Ciccio, C.D., Mendling, J., Polyvyanyy, A.: Visual drift detection for event sequence data of business processes. IEEE Trans. Vis. Comput. Graph. **28**(8), 3050–3068 (2022). https://doi.org/10.1109/TVCG.2021.3050071, https://doi.org/10.1109/TVCG.2021.3050071
22. Zheng, C., Wen, L., Wang, J.: Detecting process concept drifts from event logs. In: Panetto, H., Debruyne, C., Gaaloul, W., Papazoglou, M.P., Paschke, A., Ardagna, C.A., Meersman, R. (eds.) On the Move to Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10573, pp. 524–542. Springer (2017). https://doi.org/10.1007/978-3-319-69462-7_33, https://doi.org/10.1007/978-3-319-69462-7_33