# On the Use of Steady-State Detection for Process Mining: Achieving More Accurate Insights

Alexander Kraus[1], Keyvan Amiri Elyasi[1], and Han van der Aa[2]

[1] Data and Web Science Group, University of Mannheim, Germany
{alexander.kraus,keyvan}@uni-mannheim.de
[2] Faculty of Computer Science, University of Vienna, Austria
han.van.der.aa@univie.ac.at

**Abstract.** Steady-state detection (SSD) is a critical task in the analysis of dynamic systems, as it enables the reliable evaluation of system behavior by differentiating between stable and unstable states. While SSD techniques have been developed and tested in domains such as signal processing and industrial systems, their application in the information systems domain, particularly in process mining, has been largely overlooked. Specifically, event logs that record the executed behavior of a business process often contain data from both steady and non-steady states, which can distort process mining results, such as performance analysis and remaining time prediction. This paper highlights the importance of SSD in the process mining domain and investigates the applicability of existing SSD solutions. To operationalize this, we propose a two-step framework for detecting steady states in business processes. The framework extracts relevant process characteristics from an event log and applies established SSD techniques to identify periods during which a business process operated in a steady state. We evaluate the framework through experiments that assess its accuracy within a controlled environment using simulated event logs and that demonstrate the benefits of SSD for a downstream process mining task: remaining time prediction. The findings emphasize the potential of SSD for obtaining more accurate process mining insights.

**Keywords:** Process mining · Business Process · Steady-state detection

## 1 Introduction

Business processes are often supported by information systems that record execution data in event logs, which are then used in process mining to extract data-driven insights [1]. However, these event logs often capture business processes executed in both steady and non-steady states. *Steady states* refer to periods when process behavior remains stable and consistent over time [10], while *non-steady states* are marked by fluctuations and irregularities due to the dynamic environments in which processes operate. These non-steady states can arise from factors such as increased case arrivals during peak seasons or reduced resource availability during holidays, causing the process to deviate from its usual operations and performance levels.

The distinction between steady and non-steady states of processes is crucial for various process mining tasks. As shown later in this work, failing to distinguish between such states can, for instance, distort performance insights obtained through lead-time analysis or hurt the accuracy of predictive process monitoring models. Recognizing the impact that state fluctuations have in dynamic environments, the task of *steady-state detection* (SSD) aims to identify periods when a system operates in a steady state (or when it does not). Various techniques for this task have already been developed and tested in different application contexts, such as industrial systems [14] and signal processing [6]. However, their application in process mining has so far been largely overlooked, despite the potential of SSD to improve the accuracy of process mining insights.

Therefore, this paper highlights the importance of SSD in process mining and investigates the applicability of existing SSD solutions within this domain. To operationalize this, we propose a framework designed to identify steady states in business processes based on event data. The framework consists of two steps: (1) extracting time series from an event log that capture the progression of relevant process characteristics and (2) applying an established SSD technique to detect steady and non-steady states per process characteristic and aggregating these results to detect steady states at the process level. The effectiveness of our framework is evaluated in two experiments: one assessing its accuracy in a controlled environment based on simulated event logs and the other demonstrating its practical benefits in a downstream process mining task, specifically for remaining time prediction. Our findings showcase that our framework indeed enables the use of SSD for process mining and highlight the potential of SSD to provide more accurate insights into the operations of organizations.

The remainder of this paper is organized as follows. Section 2 provides background and illustrates the importance of SSD in process mining. Section 3 introduces our proposed framework for SSD for business processes. In Section 4, we present the results of our evaluation experiments, demonstrating the framework's accuracy and usefulness. Finally, Section 5 discusses the relationship between SSD and other related problems in process mining, while Section 6 summarizes our findings and suggests potential directions for future work.

## 2 Background and Problem Illustration

In this section, we provide background information on steady states and demonstrate the importance of their consideration in process mining.

**Steady states and the SSD problem.** A steady state refers to a condition in which the behavior of a system remains constant over time [10], making its behavior predictable and allowing for more precise and meaningful analysis. The study of steady states has a long history and has proven to be important in various fields, including mechanics [4], biology [11], ecology [22], and economics [5].

*Steady state of a business process.* In process mining, we define a business process to be in a steady state if its system-level behavior remains stable and consistent over time. As shown in Figure 1, a business process can be represented as a

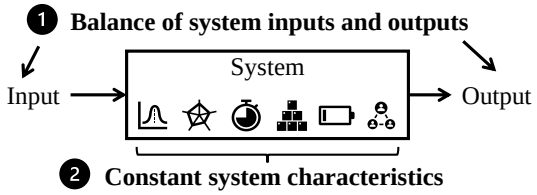



Fig. 1: Two key properties of a steady state.

system with its inputs, internal characteristics, and outputs. The steady state of a business process can then be characterized by the following two properties:

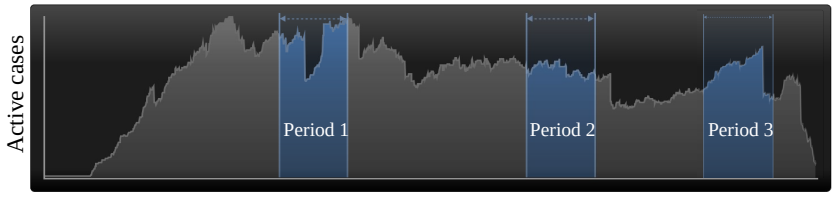
1. *Balance of system inputs and outputs*: A system in a steady state maintains a balance between input and output, ensuring that no significant fluctuation occurs over time. In the context of a business process, this means, e.g., that the number of incoming and completed cases remains consistent over time.
2. *Constant system characteristics*: The characteristics of a system in a steady state remain consistent. For a business process, this could mean that, e.g., the number of active cases and available resources remain stable.

It is important to note that when examining the system-level behavior of a process, we focus on process characteristics that provide a holistic description of its behavior that evolves over time, exhibiting notable fluctuation.

*The SSD problem.* In the context of process mining, we define the SSD problem as the task of detecting periods when system-level process behavior, derived based on information recorded in an event log, remains in a steady state.

**Importance of SSD in process mining.** To illustrate the importance of SSD in process mining, we examine how the performance of a business process, measured by average and median lead times, can differ between steady and non-steady periods, and the implications this may have on a downstream process mining task. For this purpose, we use a real-life event log describing a permit application process at a municipality (BPIC2015-2) [7] as a running example. The event log contains 44,354 events, capturing the execution of 832 cases over a period of approximately 5 years. During this period, the process exhibits an average lead time of 22.9 weeks, with a median lead time of 15.5 weeks. For simplicity, we focus on a single system-level process characteristic, namely the number of active cases, when examining the steady state of a business process.

The number of active cases, shown in Figure 2, indicates that the process was not steady throughout the recorded timeframe, with both stable and unstable periods. For instance, in Period 1, spanning 5 months and involving 23 cases, the process shows instability, marked by a significant drop in active cases. The average lead time is 10.7 weeks, with a median of 11.3 weeks. Period 2, also 5 months long with 23 cases, is more stable, with fewer fluctuations in active cases. The average lead time is 5.6 weeks, and the median is 3.6 weeks. Lastly, Period 3 exhibits a rise and fall in active cases, indicating a non-steady state. It has an average lead time of 12.1 weeks and a median lead time of 13.9 weeks.




| | | | Steady State | |
|---|---|---|---|---|
| **Measure** | **Event log** | **Period 1** | **Period 2** | **Period 3** |
| Number of traces | 832 | 23 | 23 | 25 |
| Duration | 56 months | 5 months | 5 months | 5 months |
| Average lead time | 22.9 weeks | 10.7 weeks | 5.6 weeks | 12.1 weeks |
| Median lead time | 15.5 weeks | 11.3 weeks | 3.6 weeks | 13.9 weeks |

Fig. 2: Comparison of process performance between different periods.

As observed, performance in the steady state (Period 2) is nearly twice as good as in the other periods and about four times better than the overall average across all recorded cases. Such differences are particularly relevant for process mining tasks such as remaining time prediction, as demonstrated in our evaluation. Specifically, when significant performance differences exist between steady and non-steady states, it may be beneficial to use SSD as a bucketing method to split the event log into sublogs representing steady and non-steady states. Separate models can then be trained for each sublog, allowing the appropriate model to be applied based on whether the process is currently in a steady or non-steady state, improving the accuracy of predictions for ongoing cases.

## 3 Steady-State Detection Framework for Process Mining

This section describes our proposed SSD framework. As illustrated in Figure 3, the framework takes an event log as input and then extracts time series that represent the progression of relevant process characteristics over time. These time series are then analyzed using existing SSD techniques to identify periods when a process is in a steady state. As output, the framework provides the detected steady-state periods along with a sublog of traces corresponding to them. In the following, we describe these two main steps.

### 3.1 Time Series Extraction

In Step 1, we generate time series from an event log to capture the evolution of process characteristics relevant to SSD. Such transformations are widely used in process mining, for purposes including business process simulation [19], assessing process resilience [13], and evaluating process complexity [26]. Below, we outline the specifics of this step.

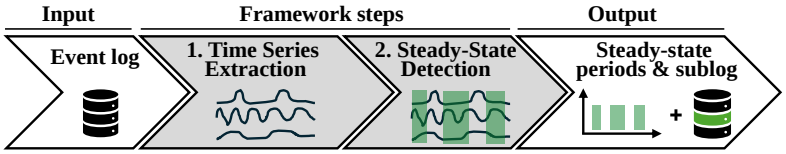



Fig. 3: Overview of the main steps of our framework.

**Input.** Our approach takes an *event log* $L$, which we define as a collection of events recorded by a process-aware information system. Each *event* $e \in L$ is represented as a tuple with at least three attributes $e = (\text{caseID}, \text{activity}, \text{timestamp})$, where *caseID* is the unique identifier for the executed case, *activity* indicates the executed process activity, and *timestamp* denotes the event moment. A *trace* $\sigma$ is a sequence of events from $L$ with the same caseID, ordered by their timestamps. We denote $\Sigma_L$ as the ordered collection of all traces from $L$, arranged according to the timestamp of their first event.

**Windowing.** We divide the entire timeframe of an event log $L$ into $n \in \mathbb{N}$ equally spaced *time windows* $W_l = \langle w_1, \ldots, w_n \rangle$, each with a fixed length $l$ (e.g., a day or a week). Consequently, each event $e \in L$ is assigned to exactly one time window $w_t$, where $t \in \{1, \ldots, n\}$.

**Time series construction.** Next, we construct time series over $w_t \in W_l$ for different process characteristics. In our framework, we consider 3 process characteristics that are relevant for SSD and can be derived from a standard event log $L$: *the number of active cases* ($ac$), *the number of completed cases* ($cc$), and *the average lead time* ($alt$) of completed cases during a time window $w_t$. If the event log includes further information, such as resource details, additional process characteristics can be considered to enrich the process representation.

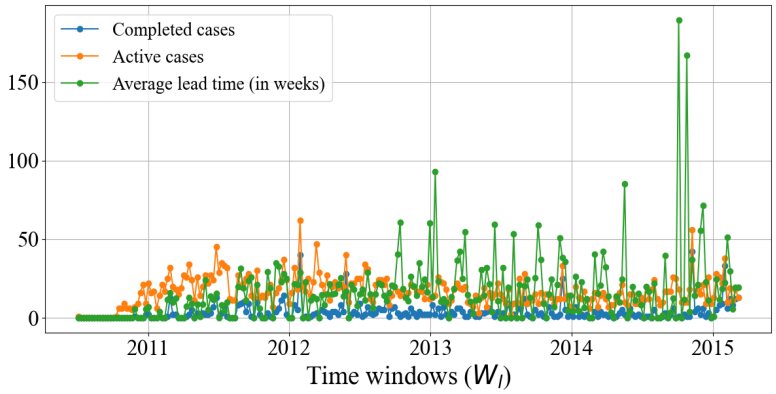



Fig. 4: Outcome of the first framework step.

We use $y^f_{w_t} \in \mathbb{R}$ to denote the value of a characteristic (or *feature)* $f \in F = \{ac, cc, alt\}$ during a time window $w_t$. For each feature, we concatenate these values into a *time series* $\{y^f_{w_t}\}^n_{t=1}$, which captures the evolution of $f$ over the time windows in $W_l$. Figure 4 shows the outcome of this step with weekly windowing for the BPIC2015-2 event log, serving as a running example.

### 3.2 Steady-State Detection

After extracting time series, the next step is to identify time windows when a process is in a steady state. This involves performing SSD at the time series level (i.e., per characteristic) and then at the process level.

**SSD at time series level.** For each time series $\{y^f_{w_t}\}^n_{t=1}$, we derive a corresponding *binary time series* $\{p^f_{w_t}\}^n_{t=1}$, with $p^f_{w_t} \in \{0, 1\}$ for each time window $w_t$ using an existing SSD technique. This binary time series indicates whether the corresponding process characteristic is in a steady state during $w_t$, where $p^f_{w_t} = 1$ signifies a steady state and $p^f_{w_t} = 0$ indicates a non-steady state.

To obtain $\{p^f_{w_t}\}^n_{t=1}$, we can use an SSD technique from a range of existing ones. Our framework's implementation currently supports the following options:

- *Rolling Window (RW) [28]*: The RW technique detects steady states in a time series by comparing the short-term and long-term rolling averages of its values. It identifies a drift when the deviation between the short-term and long-term averages exceeds a threshold that is scaled by the standard deviation of the long-term average.
- *Cumulative Sum (CS) [8]*: The CS algorithm monitors cumulative increases and decreases in the data and flags a change when these values exceed a predefined threshold. Once a change is identified, the cumulative calculation resets to ensure continued monitoring.
- *Variance Filter (VR) [23]*: The VR method proposed by Rhinehart uses a variance filter to distinguish between steady and non-steady states based on statistical analysis. It applies a filter that evaluates the ratio of the variance of the signal, with thresholds used to identify steady states.
- *ED Pelt with Transitions (EDP) [9]*: The EDP technique identifies steady states in a time series by splitting the time series into "statistically homogeneous" segments using the pruned exact linear time (Pelt) change point detection algorithm. The Pelt method guarantees optimal segmentation while maintaining a linear computational complexity.

Beyond these techniques, our framework is compatible with any SSD method that accepts a real-valued time series and generates a binary time series.

**SSD at process level.** After performing SSD per process characteristic, we next aggregate the information from the binary time series to determine if indeed the entire process can be considered to be in a steady state during a given time window. Our framework supports several aggregation techniques for this:

*Kernel-based aggregation* computes a *steady-state probability curve* as a time series $\{P_{w_t}\}^n_{t=1}$ with $P_{w_t} \in [0, 1], \forall w_t$ that represents the likelihood of a time window $w_t$ to record a steady state of a process. To do this, we first aggregate

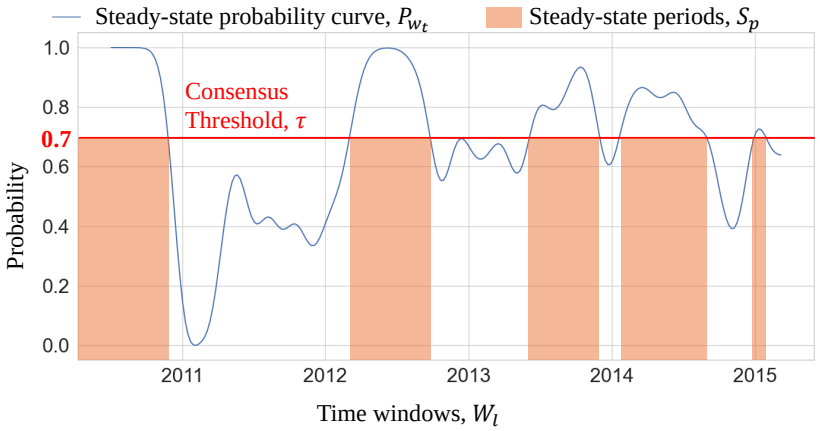



Fig. 5: SSD using the probability curve and consensus threshold.

insights from different process characteristics by calculating the average value across all binary time series $\{p^f_{w_t}\}^n_{t=1}$ for each time window. We then apply a Gaussian filter [17] with a kernel of 4 standard deviations to smooth the curve and reduce fluctuations. After smoothing, the time series is rescaled using Min-Max normalization to ensure that the values lie between 0 and 1. Finally, to identify the steady states of a business process, we compare the values of the steady-state probability curve with a *consensus threshold* $\tau \in [0, 1]$. If $P_{w_t} \geq \tau$, the time window $w_t$ is considered to be part of a steady state; otherwise, as non-steady. Figure 5 illustrates the outcome of this transformation for the time series shown in Figure 4, assuming a consensus threshold $\tau = 0.7$.

In addition to the kernel-based aggregation technique, our framework also supports more straightforward aggregation techniques. *Consensus-based aggregation* considers a time window $w_t$ as a steady state if $p^f_{w_t} = 1$ for all process characteristics. *Majority-based aggregation* deems a time window $w_t$ as a steady state if $p^f_{w_t} = 1$ holds for at least 50% of the process characteristics. Finally, *single-source aggregation* classifies a time window $w_t$ as a steady state if $p^f_{w_t} = 1$ holds for at least one process characteristic $f$.

As output, we obtain the time windows $W'_l = \langle w_{i_1}, \ldots, w_{i_m} \rangle$ as a subsequence of $W_l$ with $1 \leq i_1 < i_m \leq n$, where the process is in a steady state.

**Detection of steady-state periods and a sublog.** Finally, after identifying time windows that characterize a process in a steady state, we define steady-state periods $S_p = \langle s_1, \ldots, s_d \rangle$ by merging consecutive time windows from $W'_l$ into continuous intervals. In Figure 5, we detect a total of 5 steady-state periods.

In addition, to enable downstream process mining task, we identify traces that correspond to the detected steady-state periods. To do this, we analyze each trace $\sigma \in \Sigma_L$ and check whether the timestamps of its events fall within the identified periods in $S_p$. If the proportion of such events relative to the total number of events in $\sigma$ exceeds a predefined *trace acceptance threshold* $\theta \in$

$[0, 1]$, the trace is classified as a trace that belongs to a steady-state sublog. For example, if 3 out of 5 events in a trace fall within one or more periods from $S_p$ and $\theta = 0.5$, the trace is classified as belonging to a steady state since $3/5 > \theta$. Otherwise, the trace does not belong to a steady state. This results in a sublog $\Sigma_L^S \subseteq \Sigma_L$, containing traces associated with the process in a steady state.

Depending on the downstream process mining task, the assignments of traces to steady-state periods can also be done at the event or sub-trace level if more fine-granular information is desired.

## 4 Evaluation

This section presents two conducted evaluation experiments. In the first experiment, detailed in Section 4.1, we evaluate the accuracy of our framework in detecting steady states using synthetic data. The second experiment, explained in Section 4.2, demonstrates the usefulness of the framework using real-life event logs and a concrete process mining task, i.e., the prediction of the remaining time for ongoing cases. To ensure reproducibility, we have provided the data, implementation, configurations, and raw results in a publicly accessible repository[3].

### 4.1 Experiment 1: Accuracy

In the first experiment, we assess the ability of our framework to identify steady states in event logs. In the following, we discuss the data collection, setup, evaluation measure, and obtained results.

**Data collection.** In this experiment, we generate data by simulating an order-to-cash process for a medium-sized company, as described in the work by Zahoransky et al. [27]. The simulation model is built using the CIW library [18], an open-source tool for discrete event simulation.[4] To introduce steady and non-steady states, we vary the number of incoming cases during the simulation, ensuring a balanced distribution between steady and non-steady periods. Specifically, we create non-steady states by applying periods of linear increases and decreases in the arrival rate, followed by periods of constant arrival rate to establish steady states. We implement up to 5 changes in the arrival rates, starting with either increases or decreases, resulting in 10 distinct scenarios, as shown in Figure 6. To ensure robust evaluation, we generate 10 event logs for each scenario, producing a total of 100 event logs.

**Setup.** In Step 1 of our framework, we apply weekly windowing and consider 3 process characteristics: the number of active cases, the number of completed cases, and the average lead time of completed cases, i.e., $f \in \{\text{ac}, \text{cc}, \text{alt}\}$.

In Step 2, we evaluate all four implemented techniques for SSD: Rolling Window (RW), Cumulative Sum (CS), Variance Filter (VR), and ED Pelt with Transitions (EDP). For each technique, we test a variety of parameter combinations,[5]

[3] Project repository: https://gitlab.uni-mannheim.de/processanalytics/ssd.
[4] Available online: https://ciw.readthedocs.io/en/latest/index.html
[5] The exact parameters tested for each technique are specified in our repository.

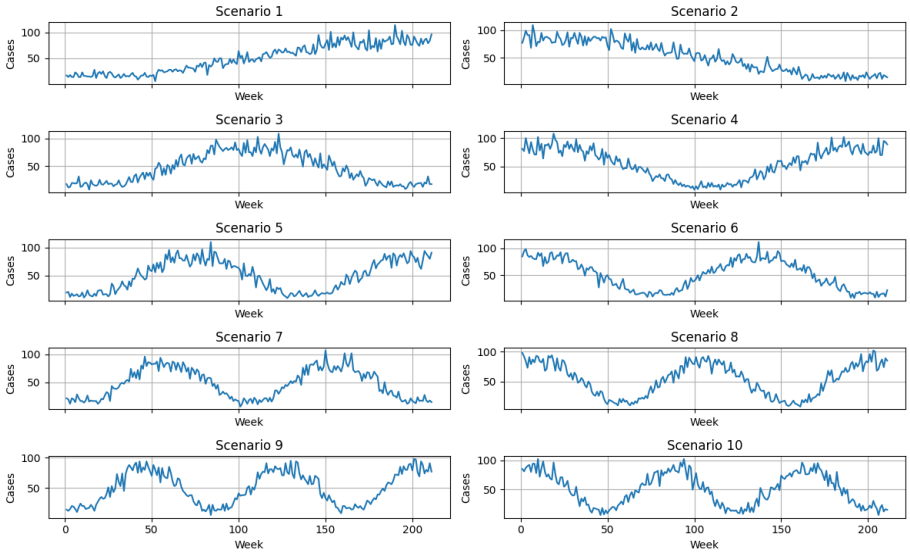



Fig. 6: Simulated number of arrived cases for each scenario.

resulting in a total of 564 evaluations per event log. To detect steady states of the process, we evaluate 4 aggregation techniques (i.e., aggregation-based SSD) with a trace acceptance threshold of $\theta = 0.8$: kernel-based, consensus-based, majority-based, and single-source. For the kernel-based approach, we set the consensus threshold to $\tau = 0.7$. Additionally, we compare the results of our framework when the decision about steady states is made based solely on a single process characteristic (i.e., feature-based SSD).

**Evaluation measure.** To measure our framework's accuracy in classifying each time window as a steady or non-steady state, we use the $\phi$ coefficient [15], a widely used binary classification metric for assessing the strength of observed associations. This metric offers a balanced evaluation by considering all components of the confusion matrix. It is defined as follows::

$$\phi = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}},$$

where TP, FP, TN, and FN represent true positives (correctly predicted steady-state windows), false positives (incorrectly predicted steady-state windows), true negatives (correctly predicted non-steady state windows), and false negatives (incorrectly predicted non-steady state windows), respectively. The $\phi$ coefficient ranges from -1 to +1, where +1 indicates perfect classification, 0 indicates random guessing, and -1 indicates complete disagreement.

**Results.** Table 1 presents the results obtained on our data collection, showing the average and standard deviations of the $\phi$ coefficient for both feature-based

Table 1: Results of Experiment 1: The average $\phi$ coefficient along with its standard deviation.

| SSD configuration | SSD technique | | | |
|---|---|---|---|---|
| | **RW** | **CS** | **VF** | **EDP** |
| **Feature-based** | | | | |
| Active cases | $0.35 \pm 0.03$ | $0.27 \pm 0.08$ | $0.19 \pm 0.06$ | $\mathbf{0.43 \pm 0.12}$ |
| Avg. lead time | $0.21 \pm 0.06$ | $\mathbf{0.38 \pm 0.12}$ | $0.37 \pm 0.10$ | $0.04 \pm 0.07$ |
| Case completions | $0.36 \pm 0.03$ | $0.26 \pm 0.05$ | $0.25 \pm 0.06$ | $\mathbf{0.43 \pm 0.08}$ |
| **Aggregation-based** | | | | |
| Kernel-based | $\mathbf{0.47 \pm 0.04}$ | $0.33 \pm 0.06$ | $0.32 \pm 0.10$ | $0.35 \pm 0.09$ |
| Consensus-based | $0.32 \pm 0.03$ | $0.25 \pm 0.05$ | $0.32 \pm 0.13$ | $\mathbf{0.35 \pm 0.09}$ |
| Majority-based | $0.34 \pm 0.03$ | $0.30 \pm 0.06$ | $0.18 \pm 0.05$ | $\mathbf{0.42 \pm 0.08}$ |
| Single-source | $0.25 \pm 0.04$ | $\mathbf{0.40 \pm 0.10}$ | $0.20 \pm 0.07$ | $0.07 \pm 0.09$ |

Note: The highlighted values show the best results in each row.

and aggregation-based configurations. The table shows that the SSD technique using rolling windows (RW) achieves the highest $\phi$ coefficient of 0.47 with kernel-based aggregation, indicating a moderate positive association between the predicted steady and non-steady states. The EDP technique produces similar outcomes, with a $\phi$ coefficient around 0.43 when using either the number of active cases or the number of case completions. In contrast, the VF technique demonstrates the lowest performance, consistently underperforming relative to other techniques across all configurations.

When comparing these results with the results observed in other domains [25], we see that the accuracy is slightly lower. The main reason for this is the specific and more complex interrelations between different process characteristics in a business process, compared to other domains where relationships typically follow well-defined laws or equations. For example, in a business process, an increase in the arrival rate does not automatically lead to an increase in the average lead time, since the process may have additional capacities that allow it to handle the increased workload without significant changes in system behavior.

Overall, existing SSD techniques can be applied in process mining, but the evaluation shows room for improvement due to the unique properties and interrelations of process characteristics, requiring domain-specific adjustments to SSD for better accuracy and applicability.

### 4.2 Experiment 2: Usefulness

In this experiment, we demonstrate the usefulness of our SSD framework by considering a well-known task in process mining, namely the remaining time prediction problem. Specifically, we compare the prediction accuracy of various state-of-the-art approaches applied to entire event logs with their accuracy

when using only data from steady states. In the following, we discuss the data collection, setup, and obtained results.

Table 2: Characteristics of the employed event logs.

| Event log | Number of | | | | Case length | | Duration (days) | |
|---|---|---|---|---|---|---|---|---|
| | Cases | Variants | Events | Classes | Avg | Max | Avg | Max |
| **Steady and non-steady states ($\Sigma_L$)** | | | | | | | | |
| Hospital | 100 000 | 1020 | 451 359 | 18 | 4.5 | 217 | 127.2 | 1035 |
| Sepsis | 1050 | 846 | 15 214 | 16 | 14.5 | 185 | 28.5 | 422 |
| Helpdesk | 4580 | 226 | 21 348 | 14 | 4.7 | 15 | 40.9 | 60 |
| BPIC12 | 13 087 | 4366 | 262 200 | 36 | 20.0 | 175 | 8.6 | 137 |
| BPIC15-1 | 1199 | 1170 | 52 217 | 398 | 43.6 | 101 | 95.9 | 1486 |
| BPIC15-2 | 832 | 828 | 44 354 | 410 | 53.3 | 131 | 160.3 | 1326 |
| BPIC15-3 | 1409 | 1349 | 59 681 | 383 | 42.4 | 123 | 62.2 | 1512 |
| BPIC15-4 | 1053 | 1049 | 47 293 | 356 | 44.9 | 115 | 116.9 | 927 |
| BPIC15-5 | 1156 | 1153 | 59 083 | 389 | 51.1 | 153 | 98.0 | 1344 |
| **Steady states ($\Sigma_L^S$)** | | | | | | | | |
| Hospital | 8315 | 176 | 27 117 | 15 | 3.3 | 217 | 54.3 | 773 |
| Sepsis | 439 | 378 | 6242 | 16 | 14.2 | 170 | 35.8 | 422 |
| Helpdesk | 745 | 92 | 3742 | 10 | 5.0 | 14 | 40.4 | 60 |
| BPIC12 | 5692 | 1417 | 81 125 | 36 | 14.2 | 142 | 5.0 | 67 |
| BPIC15-1 | 682 | 667 | 29 956 | 377 | 43.9 | 93 | 99.8 | 1486 |
| BPIC15-2 | 311 | 310 | 17 823 | 341 | 57.3 | 132 | 152.9 | 1171 |
| BPIC15-3 | 521 | 505 | 22 363 | 303 | 42.9 | 101 | 58.3 | 1261 |
| BPIC15-4 | 677 | 674 | 30 813 | 321 | 45.5 | 116 | 104.9 | 831 |
| BPIC15-5 | 520 | 519 | 27 462 | 329 | 52.8 | 108 | 86.9 | 812 |

**Data collection.** Our data collection consists of 9 publicly available real-life event logs that are commonly used for predicting the remaining runtime of ongoing cases.[6] As summarized in Table 2, these logs represent the execution of various processes and display diverse characteristics across multiple dimensions, including the number of cases, variants (i.e., unique traces), recorded events, event classes (i.e., unique activities), average case lengths and durations. In addition to the characteristics of the original event logs that include all traces ($\Sigma_L$), we include the characteristics of the sublogs with traces that correspond to steady states ($\Sigma_L^S$), as identified using our framework.

**Setup.** Next, we discuss the framework configurations, employed data split, and used remaining time prediction approaches.

[6] We excluded event logs from the BPI Challenge 2013 and 2020 due to long periods of process inactivity, the Traffic Fine log for its strong batching behavior, and the Environment Permit log for having too few events, making further segmentation unsuitable for training a prediction model.

*Configurations.* In Step 1 of our framework, we apply weekly windowing for all event logs, except for the BPIC12 event log, which covers a relatively short time period. For this log, we use daily windowing instead. We again consider 3 process characteristics, i.e., $f \in \{\text{ac}, \text{cc}, \text{alt}\}$. In Step 2, we use the configuration that yielded the best results in Experiment 1, specifically the rolling window (RW) and kernel-based aggregation with a consensus threshold of $\tau = 0.7$ and a trace acceptance threshold of $\theta = 0.8$.

*Data split and prefix generation.* We use a 64%-16%-20% chronological holdout split that divides data into training, validation, and testing sets while preserving the natural chronological order. This method mitigates data leakage and simulates real-world scenarios where predictions are made based on historical data [24]. For each trace $\sigma$ in a split, we extract all prefixes between lengths 2 and $|\sigma| - 1$ to establish prediction problems.

*Approaches.* We consider 3 remaining time prediction approaches that estimate the remaining time of an ongoing case based on the sequence of already executed activities (and possibly other available attributes):

- DUMMY: A simple baseline that predicts the remaining time of an ongoing case by averaging the remaining time of training cases that share the same sequence of executed activities.
- DALSTM: This deep learning model, based on the LSTM architecture, outperforms other LSTM-based approaches in remaining time prediction [21].
- PGTNet: This approach employs graph transformers to balance learning from the local contexts with capturing long-range dependencies [2], demonstrating state-of-the-art results.

For DALSTM and PGTNet, we use the settings reported in the original papers.

**Evaluation measures.** To evaluate the impact of SSD on prediction accuracy, we consider three evaluation measures:

- *Mean Absolute Error* (MAE) quantifies the average magnitude of absolute errors between predicted and actual remaining time. It is formally defined as: $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$, where $n$ is the number of predictions, $y_i$ represents the actual observed values, and $\hat{y}_i$ denotes the predicted values. Lower MAE values indicate higher predictive accuracy.
- *Average Performance Change* (APC) measures the average change in MAI across all approaches when comparing values obtained for all traces in an event log, denoted as $\Sigma_L$, with those obtained for traces belonging to steady-state periods, denoted as $\Sigma_L^S$. It is defined as: $APC = \frac{1}{3}\sum_{i=1}^{3}\frac{MAE_i^{\Sigma_L^S}}{MAE_i^{\Sigma_L}} - 1$, where the index $i$ iterates over the three remaining time prediction approaches considered in this experiment. An APC value closer to zero indicates smaller differences in prediction accuracy between $\Sigma_L$ and $\Sigma_L^S$.
- *Steady-State Ratio* (SSR) represents the proportion of steady-state traces relative to the total number of traces in each event log. It is calculated as: $SSR = \frac{|\Sigma_L^S|}{|\Sigma_L|}$. This measure provides context for the APC by showing the prevalence of steady-state traces within the event log.

**Results.** Table 3 presents the MAE values for all traces in an event log ($\Sigma_L$) and those associated with steady-state periods ($\Sigma_L^S$), along with the corresponding APC and SSR values for each event log. To enhance interpretability, the rows are sorted in ascending order based on the APC values.

First, we observe that for most event logs, the APC is negative, ranging from -27.1% to -5.7%. This indicates that the MAE for traces executed in a steady state (columns $\Sigma_L^S$) is, on average, lower than when predictions are made using the entire dataset (columns $\Sigma_L$). This is expected, as many processes in steady states have shorter lead times with less fluctuation, allowing for more accurate predictions of the remaining time for ongoing cases compared to non-steady states[7]. In some event logs, such as BPIC15 Municipalities 2, 3, and 5, this trend holds consistently across different approaches. For the remaining time prediction task, this finding highlights the importance of training separate models: one tailored for steady states and another optimized for non-steady states. This strategy is likely to provide more accurate predictions in terms of MAE compared to using a single model that is trained on the entire event log.

Table 3: Mean Absolute Error for remaining time prediction.

| **Event log** | **DUMMY** | | **DALSTM** | | **PGTNet** | | **APC** | **SSR** |
|---|---|---|---|---|---|---|---|---|
| | $\Sigma_L$ | $\Sigma_L^S$ | $\Sigma_L$ | $\Sigma_L^S$ | $\Sigma_L$ | $\Sigma_L^S$ | in % | in % |
| BPIC15-4 | 77.2 | 86.5 | 72.7 | 45.4 | 82.7 | 36.6 | −27.1 | 64.3 |
| BPIC15-3 | 24.2 | 22.5 | 15.1 | 10.6 | 15.0 | 10.3 | −22.6 | 37.0 |
| BPIC15-5 | 48.3 | 45.4 | 43.6 | 32.9 | 36.3 | 32.6 | −13.6 | 44.0 |
| BPIC15-2 | 72.9 | 61.4 | 47.0 | 43.8 | 68.8 | 59.2 | −12.1 | 37.4 |
| BPIC12 | 7.6 | 7.5 | 8.0 | 4.8 | 5.5 | 5.9 | −11.3 | 43.5 |
| Helpdesk | 12.3 | 10.6 | 12.9 | 10.9 | 5.4 | 6.1 | −5.7 | 16.3 |
| BPIC15-1 | 38.2 | 40.9 | 29.3 | 37.9 | 20.4 | 27.3 | 23.6 | 56.9 |
| Sepsis | 32.7 | 43.4 | 15.7 | 22.2 | 16.4 | 24.9 | 41.8 | 41.8 |
| Hospital | 47.9 | 66.8 | 36.7 | 35.9 | 24.2 | 59.0 | 60.4 | 8.3 |

However, in some event logs, the APC is positive, meaning the MAE has increased. This can be attributed to the specific characteristics of the recorded processes. In the Sepsis event log, the APC is 40%, due to the process's long warm-up and cool-down phases, which together account for over 50% of the total recorded time. As a result, steady states are detected too early, misclassifying the warm-up period and causing inaccurate detection. This highlights a challenge in business process mining, where SSD techniques from other fields may struggle to accurately detect and differentiate steady states from warm-up and cool-down phases. In the Hospital event log, the APC is also positive. However, the SSR

[7] SSD remains useful when traces in a non-steady state have shorter lead times. For example, an emergency call center may receive a surge of calls during a disaster, prompting faster processing to assist more people, reducing average lead time.

is only 8%, indicating that the proportion of traces belonging to a steady state is very low. Consequently, the steady-state sublog may be too small to yield reliable results. Finally, for the BPIC15-1 event log, the APC is approximately 24%. A closer analysis of the detected steady states reveals that the event log may contain multiple steady states with different properties. The detected steady state at the beginning of the event log occurs when the number of active cases is high, while the second part features a lower number of active cases, leading to a qualitatively different steady state. In this case, a more appropriate approach would be to consider these two steady states independently.

Overall, this experiment demonstrates that our SSD framework can notably impact the insights for a downstream process mining task, making it a valuable preprocessing step, such as bucketing in the case of remaining time prediction. While the applicability of our framework may be influenced by certain specific characteristics of the recorded process behavior, it remains a highly effective approach for many business processes.

## 5 Related Work

In this section, we relate the SSD problem to other problems in process mining.

**Concept drift detection.** The problem of SSD is related to concept drift detection in process mining, but they address different aspects. Concept drift detection identifies changes in the process that lead to a new process version [3], which operates for a certain period. In contrast, SSD focuses on the system-level behavior of the process, identifying periods where key process characteristics remain stable over time. These aspects are not necessarily correlated. For example, if a new activity (drift in the control flow) creates a bottleneck due to limited resource capacity, it is likely to impact system-level characteristics such as the average lead time. This would disrupt the steady state, potentially leading to a non-steady state or another steady state. However, if the new activity does not create a bottleneck, the system may remain in the same steady state despite transitioning to a new process version. Conversely, a business process can transition from a steady state to a non-steady state without changing its process version, for example, due to fluctuations in the arrival rate.

**Business process simulation.** In business process simulation, SSD can be used to address the initialization bias (or startup issue) of simulation models [20]. Many simulations begin from an empty state, causing early fluctuations that distort results and limit analysis. The primary goal of SSD in process simulation is to identify when a process reaches a steady state, which is essential for predicting reliable long-term insights. Despite the similar terminology, the SSD problem discussed in this paper is distinct, focusing on detecting the steady state of a business process based on past recorded behavior in an event log, and serving as a crucial preprocessing step for various offline process mining techniques. We believe that the SSD problem discussed in this paper could also impact future applications in business process simulation, particularly in the automated extraction of business process simulation models from event logs.

**Anomaly detection.** Anomaly detection seeks to identify outliers or unusual patterns at the case level that deviate from expected process behavior [12]. In contrast, SSD focuses on identifying periods of stable, consistent process behavior across all active cases for a given period. However, SSD can provide a baseline for anomaly detection, making it easier to identify and explain unexpected behaviors. Once a steady state is reached, significant deviations can signal potential irregularities, while anomalies during non-steady states can often be explained by the process's inherent instability during that period.

**Statistical quality control.** The problem of SSD is closely related to statistical quality control (SQC) [16], with both aiming to monitor process stability over time. However, SSD focuses on identifying when a process has reached a steady state, where its characteristics remain relatively stable. In contrast, SQC emphasizes detecting deviations from a desired range, typically defined by specific process characteristics that reflect the process's quality or efficiency. Moreover, it is important to note that reaching a steady state does not necessarily mean the process is operating within the optimal performance range that SQC seeks to maintain. A process can be stable but still fall outside the desired limits.

## 6 Conclusion

This paper addresses the problem of steady-state detection (SSD) in business processes, emphasizing its importance in process mining and examining the applicability of existing SSD solutions within this domain. We propose a framework for identifying when a process is in a steady state in a data-driven manner using information recorded in event logs. The framework first generates time series to represent key process characteristics and applies established SSD techniques to identify steady states in the time series and process levels, producing a sublog that captures the process behavior during these periods. The evaluation demonstrates that the framework effectively detects steady states in many real-life business processes and can enhance the accuracy and reliability of insights derived from a downstream process mining task.

In future work, we plan to pursue two key directions: enhancing the proposed framework and further investigating how SSD affects process mining tasks. To strengthen the SSD framework, we aim to develop a technique specifically tailored to the unique characteristics of business processes. Our evaluation has demonstrated that existing generic SSD techniques from other domains are not fully effective, highlighting the need for a specialized approach. This tailored SSD technique would be applicable to any event log and account for atypical behaviors, such as extended warm-up periods or periods of inactivity that may occur within event logs. To develop a more comprehensive understanding of the impact of SSD on process mining, we plan to investigate its effects across a wider range of tasks, including process discovery, conformance checking, concept drift detection, and more. This investigation will further highlight the importance of SSD and demonstrate its value in process mining research and practice.

## References


1. van der Aalst, W.: Process Mining: Data Science in Action. Springer (2016)
2. Amiri Elyasi, K., van der Aa, H., Stuckenschmidt, H.: PGTNet: A process graph transformer network for remaining time prediction of business process instances. In: International Conference on Advanced Information Systems Engineering. pp. 124–140. Springer (2024)
3. Bose, R., van der Aalst, W., Žliobaitė, I., Pechenizkiy, M.: Handling concept drift in process mining. In: International Conference on Advanced Information Systems Engineering. pp. 391–405. Springer (2011)
4. Capecchi, D., Vestroni, F.: Steady-state dynamic analysis of hysteretic systems. Journal of engineering mechanics **111**(12), 1515–1531 (1985)
5. Daly, H.E.: The economics of the steady state. The American Economic Review **64**(2), 15–21 (1974)
6. Derzsi, Z.: Optimal approach for signal detection in steady-state visual evoked potentials in humans using single - channel EEG and stereoscopic stimuli. Frontiers in Neuroscience **15**, 600543 (2021)
7. van Dongen, B.B.: BPI challenge 2015. 4TU ResearchData collection (2015)
8. Gustafsson, F., Gustafsson, F.: Adaptive filtering and change detection, vol. 1. Wiley New York (2000)
9. Haynes, K., Fearnhead, P., Eckley, I.A.: A computationally efficient nonparametric approach for changepoint detection. Statistics and computing **27**, 1293–1305 (2017)
10. Kast, F.E., Rosenzweig, J.E.: General systems theory: Applications for organization and management. Academy of management journal **15**(4), 447–465 (1972)
11. Kitano, H.: Systems biology: A brief overview. Science **295**(5560), 1662–1664 (2002)
12. Ko, J., Comuzzi, M.: A Systematic Review of Anomaly Detection for Business Process Event Logs. Business & Information Systems Engineering **65**(4), 441–462 (Aug 2023)
13. Kraus, A., Rehse, J.R., van der Aa, H.: Data-driven assessment of business process resilience. Process Science **1**(1), 4 (2024)
14. Lipo, T.A., Cornell, E.P.: State-variable steady-state analysis of a controlled current induction motor drive. IEEE Transactions on Industry Applications **IA-11**(6), 704–712 (1975)
15. Matthews, B.W.: Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochimica et Biophysica Acta (BBA)-Protein Structure **405**(2), 442–451 (1975)
16. Montgomery, D.C.: Introduction to statistical quality control. John Wiley & Sons, Hoboken, NJ, 8th edn. (2019)
17. Oppenheim, A.V.: Discrete-time signal processing. Pearson Education India (1999)
18. Palmer, G.I., Knight, V.A., Harper, P.R., Hawa, A.L.: CIW: An open-source discrete event simulation library. Journal of Simulation **13**(1), 68–82 (2019)
19. Pourbafrani, M., van der Aalst, W.: Discovering system dynamics simulation models using process mining. IEEE Access **10**, 78527–78547 (2022)
20. Pourbafrani, M., Lücking, N., Lucke, M., van der Aalst, W.: Steady state estimation for business process simulations. In: International Conference on Business Process Management. pp. 178–195. Springer (2023)
21. Rama-Maneiro, E., Vidal, J.C., Lama, M.: Deep Learning for Predictive Business Process Monitoring: Review and Benchmark. IEEE Transactions on Services Computing **16**(1), 739–756 (2023)

22. Ranta, E., Lundberg, P., Kaitala, V.: Ecology of populations. Cambridge University Press (2005)
23. Rhinehart, R.R.: Automated steady and transient state identification in noisy processes. In: 2013 American Control Conference. pp. 4477–4493. IEEE (2013)
24. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive Business Process Monitoring with LSTM Neural Networks. In: International Conference on Advanced Information Systems Engineering. pp. 477–492. Springer International Publishing, Cham (2017)
25. Turan, E.M., Jäschke, J.: A simple two-parameter steady-state detection algorithm: Concept and experimental validation. In: Computer Aided Chemical Engineering, vol. 52, pp. 1765–1770. Elsevier (2023)
26. Vidgof, M., Wurm, B., Mendling, J.: The impact of process complexity on process performance: A study using event log data. In: International Conference on Business Process Management. pp. 413–429. Springer (2023)
27. Zahoransky, R.M., Brenig, C., Koslowski, T.: Towards a process-centered resilience framework. In: ARES. pp. 266–273. IEEE (2015)
28. Zivot, E., Wang, J.: Modeling financial time series with S-PLUS, vol. 2. Springer (2006)