Reinforcement Learning-Based Streaming Process Discovery Under Concept Drift

Rujian Cai, Chao Zheng, Jian Wang(✉), Duantengchuan Li, Chong Wang, and Bing Li(✉)

School of Computer Science, Wuhan University, Wuhan, China
{cai-r-j,chaozheng,jianwang,dtclee1222,cwang,bingli}@whu.edu.cn

Abstract. Streaming process discovery aims to discover a process model that may change over time, coping with the challenges of concept drift in business processes. Existing studies update process models with fixed strategies, neglecting the highly dynamic nature of trace streams. Consequently, they fail to accurately reveal the process evolution caused by concept drift. This paper proposes RLSPD (**R**einforcement **L**earning-based **S**treaming **P**rocess **D**iscovery), a dynamic process discovery approach for constructing an online process model on a trace stream. RLSPD leverages conformance-checking information to characterize trace distribution and employs a reinforcement learning policy to capture fluctuations in the trace stream. Based on the dynamic parameters provided by reinforcement learning, we extract representative trace variants within a memory window using frequency-based sampling and perform concept drift detection. Upon detecting concept drift, the process model is updated by process discovery. Experimental results on real-life event logs demonstrate that our approach effectively adapts to the high dynamics of trace streams, improving the conformance of constructed process models to upcoming traces and reducing erroneous model updates. Additionally, the results highlight the significance of the pre-trained policy in dealing with unknown environments.

Keywords: Process discovery · Concept drift · Trace stream · Reinforcement learning

1 Introduction

Process mining aims to understand the business processes of organizations by analyzing event data recorded in information systems [1]. Process discovery is the primary task of process mining. It explicitly constructs a process model based on the execution records of a business process, providing insights for subsequent process monitoring and improvement. Early process discovery methods [2–4] employ the entire event log to construct a process model by analyzing the execution relationships among activities. These methods operate under the assumption that the business process is in a steady state, and the process model

G. Guizzardi et al. (Eds.): CAiSE 2024, LNCS 14663, pp. 55–70, 2024.
https://doi.org/10.1007/978-3-031-61057-8_4

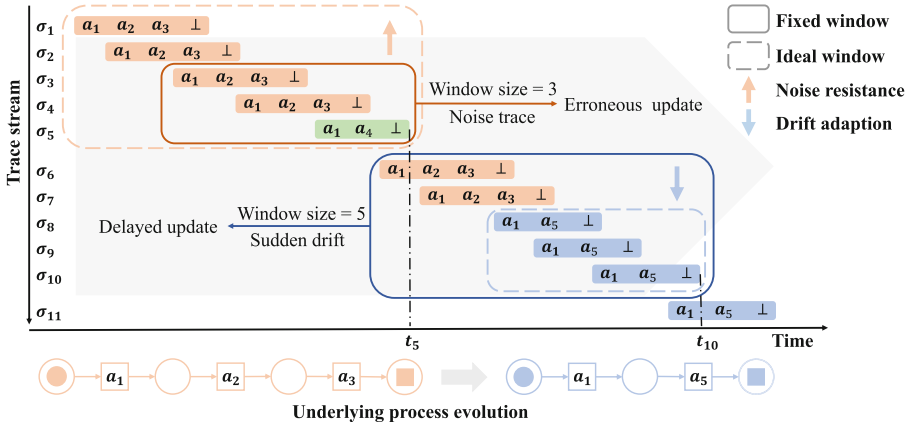


Fig. 1. Illustration of dynamic parameter adjustments in streaming process discovery

extracted from historical event logs can describe the execution of the future process. However, real-life processes may undergo changes over time due to shifts in business environments, such as evolving requirements and policies, herein called concept drift [5]. Organizations must treat the real-time event logs generated by business systems as data streams and handle concept drifts within them [6]. When the underlying business process changes, the distribution of the traces representing different process variants changes. Once a concept drift is detected, the online process model must be updated to ensure high conformance with upcoming traces.

Prior studies [7–12] primarily focus on detecting and locating concept drifts, delving into when they occur, their characteristics, and the root causes. However, the process evolution before and after concept drifts is not explicitly revealed. Several approaches have recently explored dynamic process discovery on event streams. In [13–16], process discovery is performed whenever new data arrives or at regular intervals, without considering potential changes in the process execution. Consequently, the process model is updated solely upon detecting concept drifts in [17]. Nevertheless, it uses the same fixed detection parameters for diverse trace streams or segments within the same trace stream, failing to effectively adapt to the highly dynamic nature of trace streams. This may result in erroneous or delayed updates to the process model.

Figure 1 illustrates a fluctuating trace stream that contains underlying process evolution and noise caused by misoperations. For instance, at time t_5, utilizing a fixed small window (e.g., window size = 3) may mistakenly identify the fluctuation caused by the noise trace σ_5 as a concept drift, leading to an erroneous model update. This issue can be addressed by enlarging the window to filter out noise, thereby enhancing resistance to interference. Conversely, at time t_{10}, the underlying business process has changed. If the fixed window is large (e.g., window size = 5), the discovered model is influenced by distant historical traces, resulting in a delayed detection of the sudden drift. This issue

can be mitigated by reducing the window size to focus exclusively on the most recent traces, facilitating a rapid and accurate update of the process model. In summary, dynamically adjusting parameters is crucial for streaming process discovery.

This paper proposes RLSPD (Reinforcement Learning-based Streaming Process Discovery), a novel dynamic process discovery approach designed to discover an online process model over the trace stream. We utilize the conformance-checking information of the process model for the upcoming traces to characterize the trace distribution. Subsequently, we employ a reinforcement learning policy to dynamically determine the required parameters (i.e., memory window and sampling rate) for identifying the most representative trace variants through frequency-based sampling. A change in the representative variants signifies a significant shift in the trace distribution, indicating the detection of a concept drift. At this point, the process discovery is re-performed based on the new trace variants, and the online process model is promptly updated.

The main contributions of this work are summarized as follows: **1)** We propose a streaming process discovery approach that improves the conformance of the online process model to the upcoming traces under concept drifts, and reduces the number of incorrect model updates. **2)** We characterize trace distribution with conformance-checking information and employ reinforcement learning to capture the dynamic nature of trace streams. This enables the acquisition of robust online process models, even in the presence of totally new trace streams. **3)** We conduct experiments on several real-life event logs treated as trace streams, and the results show the effectiveness of RLSPD.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces preliminary concepts, while Sect. 4 describes the proposed RLSPD approach. The experimental results and discussions are reported in Sect. 5. Finally, Sect. 6 draws conclusions and sketches future work.

2 Related Work

Concept Drift in Process Mining. Traditional process mining algorithms, such as Inductive Miner [2], ILP Miner [3] and Split Miner [4], operate under the assumption that business processes always remain stable. However, real-world business processes often change over time due to factors like customer requirements and market trends.

Concept drift in process mining was initially explored in [7], where three critical issues were identified: drift detection, drift localization, and unraveling process evolution. Most research has primarily focused on the first two problems. Hypothesis testing is a prevalent method for detecting concept drift [7–9]. It involves extracting features at both the log and trace levels and conducting hypothesis tests on consecutive sub-logs to determine whether there is statistical evidence of significant differences in feature sets before and after the drift point. Another category of concept drift detection methods is trace clustering [10–12]. This technique involves mapping traces into a vector space and clustering them

within fixed-size windows. Drift is detected when there are notable differences between two consecutive clusters. Drift localization approaches typically analyze variations in feature sets or trace clusters before and after concept drifts, identifying the drift type and the changed activities or transitions.

These methods focus on detecting and locating concept drifts, but do not reveal the process evolution before and after concept drifts. This necessitates a mechanism for dynamic process discovery within event streams.

Streaming Process Discovery. In recent years, researchers have been dedicated to discovering evolving process models in dynamic business scenarios. In [13–16], various static process discovery algorithms are applied to stream settings. By introducing classic data stream mining techniques like Sliding Window and Lossy Counting, the event stream is managed to continuously update the underlying data structure with the latest observations. The process model is then reconstructed based on this updated representation. Despite addressing the challenge of processing infinite event stream data with limited memory, these approaches do not explicitly detect concept drifts. This leads to restarting process discovery periodically or upon the arrival of each new event, potentially resulting in unchanged process models and unnecessary time consumption.

Combining process discovery with concept drift detection becomes crucial. By capturing the high dynamics of the log stream, it is possible to update the process model only when necessary. STARDUST [17] designs a streamlined drift detection method specifically for streaming process discovery. It monitors the trace stream and detects concept drifts when high-frequency trace variants changes, and then uses these trace variants to discover a new process model. However, its fixed window size and sampling rate cannot effectively adapt to the dynamic changes in trace distribution, potentially resulting in false detections of concept drifts and impacting model fitness and precision for upcoming traces.

Using identical and fixed parameters may result in poor performance when confronted with various trace streams or diverse segments within a single trace stream. While grid search is a prevalent approach for parameter optimization, it is inefficient in highly dynamic real-time scenarios. Hence, we introduce reinforcement learning to adapt to changes in trace streams, dynamically adjusting parameters for concept drift detection and process discovery.

3 Preliminaries

3.1 Event and Trace Stream

A process is a series of interrelated and ordered activities executed according to specific rules and conditions to achieve a particular goal. Each case represents an instance of process execution. Let $\mathcal{A}$ be the set of activities, $\mathcal{C}$ be the set of case identifiers, and $\mathcal{T}$ be the set of timestamps.

Definition 1 (Event). An event e is a triple, $e = (c, a, t) \in \mathcal{C} \times \mathcal{A} \times \mathcal{T}$ representing that in case c, activity a is executed at timestamp t.

To identify the case, activity, and timestamp of an event $e = (c, a, t)$, we introduce the functions: $I_{case}(e) = c$, $I_{activity}(e) = a$, $I_{time}(e) = t$, respectively. Let $\perp$ denote the end activity, an event e with $I_{case}(e) = c$ and $I_{activity}(e) = \perp$ represents the completion of case c.[1] Then, the complete execution trace σ_c of case c can be extracted.

Definition 2 (Trace). A trace σ_c is a sequence of all events within the same case c in time order, $\sigma_c = \langle e_1, e_2, ..., e_n \rangle$, where $I_{case}(e_i) = I_{case}(e_{i+1}) = c, I_{time}(e_i) < I_{time}(e_{i+1}), i \in \{1, 2, \ldots, n-1\}$, and $I_{activity}(e_n) = \perp$.

For trace $\sigma_c = \langle e_1, e_2, ..., e_n \rangle$, let $\sigma_c(i) = e_i$ be the i-th event of trace σ_c, $|\sigma_c| = n$ be the total number of events in completed case c, and the completion time of trace σ_c be $t_c = I_{time}(\sigma_c(|\sigma_c|)) = I_{time}(e_n)$. Different cases are executed as the business process system runs, generating an online stream of traces.

Definition 3 (Trace stream). A trace stream Σ is an infinite sequence of traces in order of trace completion time, i.e., $\Sigma = \sigma_1, \sigma_2, ...$, where for each $c \in \mathcal{C}$, σ_c denotes the complete execution trace of case c, and $t_c \leq t_{c+1}$.

To adapt to the dynamic nature of streaming data, it is necessary to introduce a memory window that only considers the traces from the most recent period.

Definition 4 (Memory window). Let w be the memory size. Given a trace stream Σ and the current trace σ_c, a memory window extracts a trace subset S_c^w that records the most recent w traces, $S_c^w = \{\sigma_{c-w+1}, \sigma_{c-w+2}, ..., \sigma_c\}$.

When analyzing a trace set to discover a process model, the focus is on the execution order of different activities in each trace and the frequency of distinct trace variants.

Definition 5 (Trace variant). A trace variant v is a unique activity execution sequence, $v = \langle a_1, a_2, \ldots, a_n \rangle$, where $a_i \in \mathcal{A}$ and $a_n = \perp$.

Let $v(i)$ be the i-th activity of trace variant v, and $|v|$ be the total number of activities in v. The trace σ_c belongs to the trace variant v only if $|\sigma_c| = |v|$ and $I_{activity}(e_i) = v(i)$ for each $i = 1, 2, ..., |\sigma_c|$.

Definition 6 (Trace variant counter). A trace variant counter V_S is a counter that records distinct trace variants and their frequency in the trace set S.

Taking the trace stream $\Sigma = \sigma_1, \sigma_2, ..., \sigma_{11}, ...$ in Fig. 1 as an example, traces $\sigma_1 - \sigma_4, \sigma_6 - \sigma_7$ belong to trace variant $\langle a_1, a_2, a_3, \perp \rangle$ and traces $\sigma_8 - \sigma_{11}$ belong to trace variant $\langle a_1, a_5, \perp \rangle$. With memory size $w = 10$ and the current trace σ_{11}, the trace set $S_{11}^{10} = \{\sigma_2, \sigma_3, ..., \sigma_{11}\}$ can be transformed into trace variant counter $V_{S_{11}^{10}} = \{\langle a_1, a_2, a_3, \perp \rangle^5, \langle a_1, a_5, \perp \rangle^4, \langle a_1, a_4, \perp \rangle^1\}$.

[1] Business processes usually declare the completion of a case, such as ticket resolution in help desk processes. Business processes that do not explicitly declare the completion of a case cannot be handled in this study and require further investigation in future work.

Before constructing a process model, it is a common practice to filter the input traces to extract essential information and reduce noise interference. An intuitive solution is to sample high-frequency trace variants from the trace variant counter as a preprocessing step for process discovery.

Definition 7 (Frequency-based sampling). Let ρ be the sampling rate. Given a trace variant counter V_S, the frequency-based sampling algorithm is a function $\Phi(\rho, V_S)$ that sorts the trace variants of V_S in descending order of frequency and generates V_S^ρ, which comprises the smallest counter set of the top-frequent trace variants of V_S so that $\frac{\Gamma(V_S^\rho)}{\Gamma(V_S)} \geq \rho$.

The function $\Gamma()$ counts the total occurrences of trace variants recorded in a trace variant counter. For example, given $\rho = 0.8$ and $V_S = \{\langle a_1, a_2, a_3, \perp\rangle^5, \langle a_1, a_5, \perp\rangle^4, \langle a_1, a_4, \perp\rangle^1\}$, $V_S^\rho = \Phi(\rho, V_S) = \{\langle a_1, a_2, a_3, \perp\rangle^5, \langle a_1, a_5, \perp\rangle^4\}$, sampling the top two most frequent trace variant of V_S. In this case, $\frac{\Gamma(V_S^\rho)}{\Gamma(V_S)} = \frac{9}{10} > 0.8$.

3.2 Process Discovery and Conformance Checking

Using a trace variant counter as input, process discovery algorithms abstract the order of activity execution, generating a summarized representation to construct a process model.

Definition 8 (Process model discovery). A process discovery algorithm $\Omega : V_S \mapsto M$ is a function that constructs a process model M from a trace variant counter V_S.

Process models come in various forms (e.g., Petri nets, process trees, and BPMN). Still, they exhibit similar behaviors in executing and generating a set of traces based on the represented processes. For a given process model M, its behavior B_M refers to the set of traces that can be generated through its execution.

To evaluate a model, the typical practice involves checking its conformance with a given trace variant counter, examining the model's behavior against the counter to measure its appropriateness.

Definition 9 (Conformance checking). Given a process model M and a trace variant counter V_S, a conformance-checking algorithm $\Psi(M, V_S)$ measures the appropriateness of M for V_S, denoted as $\alpha = \{\alpha^f, \alpha^p\}$. α^f represents fitness, quantifying how much of V_S can be reproduced in M, while α^p represents precision, quantifying the proportion of B_M can be observed in V_S.

Alignment [18] and Token Replay [19] are two widely used conformance-checking algorithms. The former identifies inconsistencies by comparing traces and executable paths of the model. Although intuitive and accurate, it exhibits high computational complexity for large-scale models and traces. Conversely, the latter employs tokens in a Petri net to simulate the execution of traces, identifying errors or rule violations during the replay of traces in the model. It achieves approximate alignment results with higher computational efficiency but only applies to process models that can be converted into standard Petri nets.

Table 1. Example of concept drift detected between t_{10} and t_{11} in Σ of Fig. 1

Timestamp	V_S	V_S^ρ with $\rho = 0.8$	$\Delta(V_S^\rho)$
t_9	$\{\langle a_1, a_2, a_3, \perp\rangle^2, \langle a_1, a_5, \perp\rangle^2, \langle a_1, a_4, \perp\rangle^1\}$	$\{\langle a_1, a_2, a_3, \perp\rangle^3, \langle a_1, a_5, \perp\rangle^2\}$	$\{\langle a_1, a_2, a_3, \perp\rangle, \langle a_1, a_5, \perp\rangle\}$
t_{10}	$\{\langle a_1, a_2, a_3, \perp\rangle^2, \langle a_1, a_5, \perp\rangle^3\}$	$\{\langle a_1, a_2, a_3, \perp\rangle^2, \langle a_1, a_5, \perp\rangle^3\}$	$\{\langle a_1, a_2, a_3, \perp\rangle, \langle a_1, a_5, \perp\rangle\}$
t_{11}	$\{\langle a_1, a_2, a_3, \perp\rangle^1, \langle a_1, a_5, \perp\rangle^4\}$	$\{\langle a_1, a_5, \perp\rangle^4\}$	$\{\langle a_1, a_5, \perp\rangle\}$

3.3 Streaming Process Discovery

When conducting process discovery tasks on a trace stream, the expectation is that the process model discovered based on existing historical traces exhibits high conformance with upcoming traces, thereby providing insights for downstream tasks such as process monitoring, prediction, and enhancement. However, in the presence of concept drifts, the existing model may fail to effectively depict the ongoing business process. In such cases, it becomes necessary to dynamically restart the process discovery for model updates.

We introduce function $\Delta(V_S)$ to identify distinct trace variants recorded in a trace variant counter V_S, e.g., $\Delta(\{\langle a_1, a_2, a_3, \perp\rangle^5, \langle a_1, a_5, \perp\rangle^4\}) = \{\langle a_1, a_2, a_3, \perp\rangle, \langle a_1, a_5, \perp\rangle\}$. When the high-frequency trace variants $\Delta(V_S^\rho)$ differ between two memory windows, we consider that there is a significant change in the trace distribution between these two memory windows, that is, a concept drift occurs.

Definition 10 (Concept drift). Considering two trace sets S_i^w and S_j^w recorded by a memory window in a trace stream Σ at timestamps t_i and t_j, if $\Delta(V_{S_i^w}^\rho) \neq \Delta(V_{S_j^w}^\rho)$, a concept drift is detected in Σ between t_i and t_j.

According to the definition, a concept drift is detected when a particular trace variant in the trace stream experiences a substantial increase or decrease in frequency. Table 1 illustrates a scenario of a concept drift detected between t_{10} and t_{11} in the trace stream Σ of Fig. 1, considering a memory size $w = 5$. This drift is attributed to the emergence of a newly frequent trace variant $\langle a_1, a_5, \perp\rangle$.

4 Methodology

This section introduces RLSPD, as illustrated in Fig. 2, with four modules: Process Discovery, Conformance Checking, RL-based Parameter Selector, and Concept Drift Detection. Detailed descriptions of each module are provided below.

4.1 Process Discovery

In the initialization phase, we observe the trace stream Σ to obtain the initial set $S_0 = \{\sigma_1, \sigma_2, ..., \sigma_w\}$ consisting of the first w traces. We then extract the most representative trace variants $V_{S_0}^\rho = \Phi(\rho, V_{S_0})$ by frequency-based sampling. In this way, the online process model is initialized to $M = \Omega(V_{S_0}^\rho)$.

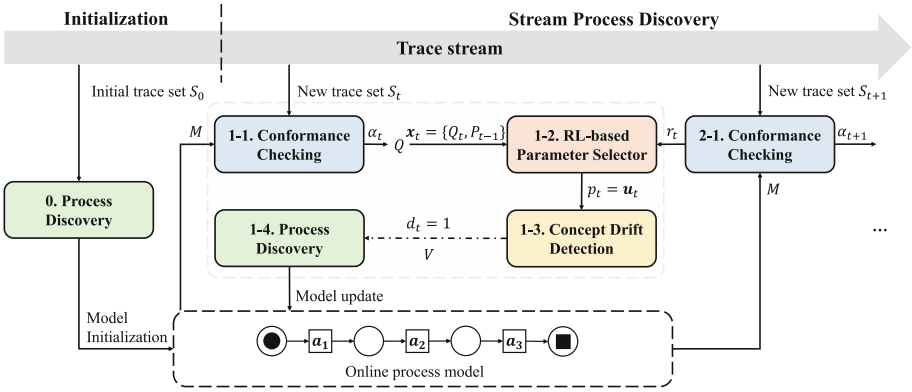


Fig. 2. The framework of RLSPD

After completing the initialization step, we continuously monitor the trace stream. Once a concept drift is detected, we re-run the process discovery algorithm and update the online process model $M = \Omega(V)$ according to the latest representative trace variants V provided by the concept drift detection module. Otherwise, the model M remains unchanged.

4.2 Conformance Checking

After initialization, we evaluate the online process model by checking its conformance with upcoming traces. Considering computational efficiency, we choose Token Replay [19] as the conformance-checking algorithm Ψ. To maintain evaluation stability, we calculate the appropriateness with the upcoming trace set rather than a single trace, avoiding drastic fluctuations cause by noise traces.

Let o be the observation window size. At each time step t, we observe the trace stream Σ and record the next o arriving traces, obtaining the set $S_t = \{\sigma_{i+1}, \sigma_{i+2}, ..., \sigma_{i+o}\}$, where i represents the count of traces already seen in the trace stream. Then, we transform S_t to a trace variant counter V_{S_t}, perform a conformance-checking algorithm $\Psi(M, V_{S_t})$ to calculate the appropriateness $\alpha_t = \{\alpha_t^f, \alpha_t^p\}$ and insert it into the historical queue Q.

4.3 RL-Based Parameter Selector

We aim to use the appropriateness queue Q as a feature to capture the fluctuations in the trace stream, dynamically providing parameters p_t to extract the currently most representative trace variants for streaming process discovery. This can be considered as an agent in reinforcement learning making real-time adaptive adjustments to the dynamic environment.

In reinforcement learning, an agent learns how to make optimal decisions in interacting with the environment. At each time step $t \in [1, T]$, the agent takes an action $\mathbf{u}_t$ according to its current policy $\pi(\mathbf{u}_t|\mathbf{x}_t)$ and state $\mathbf{x}_t$ of the

environment. The environment interacts with this action, gives a feedback reward $r_t = \mathcal{R}(\mathbf{x}_t, \mathbf{u}_t)$, and proceeds to the next state. According to this new state, the agent moves into the next time step and decides a new action. The goal of reinforcement learning is to train an agent with policy π to maximize the expected sum of rewards.

For streaming process discovery, the key is to find the optimal parameters to select the most representative trace variants in the trace stream, thus properly characterizing the trace distribution and detecting concept drift. The trace stream and the online model can be viewed as the environment in reinforcement learning, and parameter adjusting can be viewed as the action. Then, parameter optimization can be considered as training an adjusting agent to learn a policy for finding the optimal action. In streaming process discovery, we evaluate the appropriateness of the online process model with the upcoming traces in the trace stream, which shows how the trace stream fluctuates and hence offers rich information for adjusting the parameters. Having observed this, we apply the appropriateness queue Q at time step t (denoted as Q_t) to construct the state $\mathbf{x}_t$, denoted as

$$\mathbf{x}_t = \{Q_t, P_{t-1}\}, \tag{1}$$

where $Q_t = \langle \alpha_{t-h+1}, \alpha_{t-h+2}, ..., \alpha_t \rangle$ and $P_{t-1} = \langle p_{t-h}, p_{t-h+1}, ..., p_{t-1} \rangle$ are queues consisting of appropriateness and parameters of the history window with length h, respectively. Note that one time step of reinforcement learning corresponds to o new arrival traces observed in trace stream Σ.

Given the state $\mathbf{x}_t$, the agent will take an action, that is, choose a parameter vector $\mathbf{u}_t \in \mathbb{R}^{N_u}$, where N_u is the number of parameters to be adjusted. This is a user-defined variable that varies with different streaming process discovery methods. We choose two key parameters, including memory size w_t and sampling rate ρ_t. Thus, the action is defined as

$$\mathbf{u}_t = \{w_t, \rho_t\}. \tag{2}$$

The memory size determines how many recent traces in the trace stream should be considered, while the sampling rate determines how frequently a trace variant should appear to be considered representative. After taking the action, the parameters are updated with $\mathbf{u}_t$, formulated as $p_t = \{w_t, \rho_t\}$.

Based on the updated parameters p_t, we detect whether a concept drift has occurred, indicated by a binary variable d_t. If $d_t = 1$, we perform process discovery and update the online process model. Then, we continue to observe the trace stream and perform conformance checking on the model and the set of traces within the next observation window, calculating the appropriateness α_{t+1}. After that, the reward function $\mathcal{R}(\mathbf{x}_t, \mathbf{u}_t)$ is defined as

$$\mathcal{R}(\mathbf{x}_t, \mathbf{u}_t) = \alpha_{t+1} - \beta \cdot d_t - \gamma \cdot w_t, \tag{3}$$

where β and γ denote penalty coefficients. The purpose of doing this is to maintain high appropriateness while reducing the occurrence of concept drift to save

computational resources and decrease time consumption, and reducing the memory window size to minimize memory consumption. Finally, the environment proceeds to the next time step and updates the state to $\mathbf{x}_{t+1} = \{Q_{t+1}, P_t\}$.

We train the agent with the Twin Delayed Deep Deterministic policy gradient algorithm (TD3) [20]. Each episode corresponds to a trace stream transformed from a historical event log. Both the actor and critic networks adopt Gate Recurrent Unit (GRU) [21] to capture time-series features of the state.

During the online execution phase of streaming process discovery, the trained actor network is used for inference. At each time step t, it takes an environmental state $\mathbf{x}_t$ as input and outputs the considered optimal parameter $p_t = \mathbf{u}_t$.

4.4 Concept Drift Detection

As defined in **Definition** 10, we believe that a concept drift has occurred when the representative trace variants change in the trace stream.

At each time step t, after calculating the appropriateness α_t of the online process model with the newly arrived trace set $S = \{\sigma_{i+1}, \sigma_{i+2}, ..., \sigma_{i+o}\}$, the RL agent gives parameter $p_t = \{w_t, \rho_t\}$. Then, the trace set to be considered in the memory window w_t can be extracted as $S_{i+o}^{w_t} = \{\sigma_{i+o-w_t+1}, \sigma_{i+o-w_t+2}, ..., \sigma_{i+o}\}$, and the current representative trace variant counter is $V_{S_{i+o}^{w_t}}^{\rho_t} = \Phi(\rho_t, V_{S_{i+o}^{w_t}})$. Similarly, the trace set in the previous memory window w_{t-1} is $S_i^{w_{t-1}} = \{\sigma_{i-w_{t-1}+1}, \sigma_{i-w_{t-1}+2}, ..., \sigma_i\}$ and the previous representative trace variant counter is $V_{S_i^{w_{t-1}}}^{\rho_{t-1}} = \Phi(\rho_{t-1}, V_{S_i^{w_{t-1}}})$. If these two are not the same, we believe that the trace distribution has changed significantly, indicating a concept drift has occurred. We use a binary variable d_t to denote whether a concept drift is detected at time step t, formulated as

$$d_t = \begin{cases} 1, \Delta\left(V_{S_i^{w_{t-1}}}^{\rho_{t-1}}\right) \neq \Delta\left(V_{S_{i+o}^{w_t}}^{\rho_t}\right) \\ 0, \Delta\left(V_{S_i^{w_{t-1}}}^{\rho_{t-1}}\right) = \Delta\left(V_{S_{i+o}^{w_t}}^{\rho_t}\right) \end{cases} \tag{4}$$

If a concept drift occurs(i.e., $d_t = 1$), the process discovery module is triggered, updating the online process model; otherwise (i.e., $d_t = 0$), the model remains unchanged. After that, we go to the next time step $t + 1$ and continue monitoring the trace stream.

5 Experiment

The proposed RLSPD is implemented in Python, and the source code is available in the GitHub repository[2]. We evaluate the effectiveness of RLSPD on several benchmark event logs.

[2] https://github.com/WHU-Process-Mining/RLSPD.

Table 2. Statistics of event logs: number of activities, events, traces and trace variants

Event log	#Activities	#Events	#Traces	#Variants
BPIC2013Incidents(BPIC13I)	4	65533	7554	1511
BPIC2020DomesticDeclarations(BPIC20DD)	17	56437	10500	99
BPIC2020InternationalDeclarations(BPIC20ID)	34	72151	6449	753
BPIC2020PermitLog(BPIC20PL)	51	86581	7065	1478
BPIC2020PrepaidTravelCost(BPIC20PC)	29	18246	2099	202
BPIC2020RequestForPayment(BPIC20RP)	19	36796	6886	89

5.1 Experimental Setting

Datasets. We generate trace streams from six real-life event logs available in 4TU. ResearchData[3] for experiments. These logs record the execution of business processes related to travel reimbursement and traffic management. Table 2 provides the characteristics of these logs.

Hyperparameters. For each trace stream, an initial model is constructed using the first 200 traces (i.e., $w_0 = 200$). According to [17], the initial sampling rate ρ_0 is set to 0.8. Subsequently, continuously observing the trace stream to obtain the next trace, conformance checks are performed between the model and the next ten traces ($o = 10$). Then, parameters are given based on ten historical evaluations ($h = 10$) by the reinforcement learning agent, and the online process model is updated through process discovery if a concept drift is detected. The process discovery algorithms we employed are Inductive Miner (IND) [2] and ILP Miner (ILP) [3], both conveniently imported from PM4PY[4]. The hyperparameters w_0, o, h are chosen based on experience and preliminary experimental results as better values.

Comparison Methods. To validate the effectiveness of RLSPD, we compare it with STARDUST [17], the state-of-the-art method that employs fixed parameters for concept drift detection and process discovery. Additionally, we use the STATIC process discovery as a baseline, which keeps the process model discovered in the initialization step without updates upon detecting concept drifts.

Evaluation Metrics. To evaluate the performance of different methods, we use the average F-measure of fitness and precision, denoted as

$$F = \frac{1}{T}\sum_{t=1}^{T} F_t = \frac{1}{T}\sum_{t=1}^{T} \frac{2\alpha_t^f \alpha_t^p}{\alpha_t^f + \alpha_t^p}, \tag{5}$$

where F_t is the F1-score of α_t^f and α_t^p, and it is calculated together with appropriateness in the online process discovery phase. In addition, we record the number of model updates (i.e., detected concept drifts) and the computation time

[3] https://data.4tu.nl/.
[4] https://github.com/pm4py/pm4py-core.

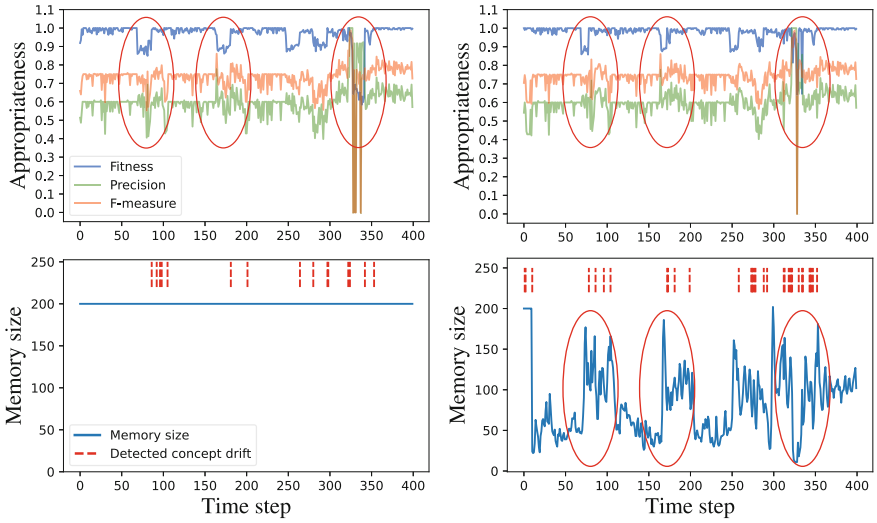
Table 3. Overall performance

Trace stream	Method	F-measure		Update count		Time(s)	
		IND	ILP	IND	ILP	IND	ILP
BPIC13I	STATIC	0.73	0.65	0	0	**10**	**9**
	STARDUST	0.68	0.62	329	329	14	12
	RLSPD	**0.75**	**0.67**	67	345	**10**	11
BPIC20DD	STATIC	0.72	0.72	0	0	**2**	**2**
	STARDUST	0.93	0.89	8	8	**2**	**2**
	RLSPD	**0.95**	**0.95**	65	141	**2**	5
BPIC20ID	STATIC	0.46	0.61	0	0	10	**8**
	STARDUST	0.50	0.63	487	487	25	140
	RLSPD	**0.90**	**0.90**	35	45	**6**	**8**
BPIC20PL	STATIC	0.30	0.59	0	0	25	15
	STARDUST	0.16	0.32	630	630	442	1895
	RLSPD	**0.83**	**0.84**	50	55	**10**	**12**
BPIC20PC	STATIC	0.48	0.50	0	0	2	**2**
	STARDUST	0.57	0.77	134	134	3	14
	RLSPD	**0.88**	**0.88**	12	15	**1**	**2**
BPIC20RP	STATIC	0.69	0.69	0	0	**1**	**1**
	STARDUST	0.92	0.89	12	12	**1**	2
	RLSPD	**0.94**	**0.94**	6	4	**1**	2

spent in seconds when processing the entire trace stream. The computation time reflects the duration of a full episode during the testing phase of reinforcement learning. This encompasses model initialization and the online phase (including conformance checking, adaptive parameter adjustment by the trained RL agent, concept drift detection, and potential restart of process discovery). The computation time is collected through experiments conducted on a workstation with Intel(R) Core(TM) i7-12700 CPU, NVIDIA GeForce GTX 1080 Ti GPU and 125 GB RAM Memory, operating on Ubuntu 20.04.4 LTS.

5.2 Results and Discussions

Overall Performance Evaluation. To evaluate RLSPD's overall performance, we train the RL agent and conduct testing on the same dataset. We investigate the impact of updating process models when concept drifts are detected through a comparison with STATIC. Additionally, we explore the effectiveness of leveraging reinforcement learning for dynamically adjusting parameters in streaming process discovery by comparing RLSPD with STARDUST. Table 3 reports the average F-measure, as well as the update counts and computational time for the online process model handling the entire trace stream.



(a) Fixed parameters (b) Dynamic parameters

Fig. 3. The model's appropriateness using fixed and dynamic parameters on BPIC13I

The results highlight a substantial performance boost of RLSPD compared to the baselines. Continuous updates to the online process model, as evidenced by the F-measure, prove to be more effective in adapting to the trace stream and aligning with upcoming traces than maintaining the initial model unchanged. Notably, on BPIC13I and BPIC20PL, STARDUST's fixed parameter strategy results in delayed or erroneous model updates due to numerous trace variants, leading to adverse effects. Regarding computation time, STATIC generally requires less time due to its non-updating nature. However, in cases involving a complex initial model followed by subsequent process simplification, STATIC consumes more time due to conformance checking. In general, RLSPD excels in capturing trace distribution fluctuations through dynamic parameter adjustments, which not only enhances conformance with upcoming traces but also reduces unnecessary model updates and time consumption.

Dynamic Parameter Analysis. To further explore the effectiveness of dynamic parameters, we analyze the adjustment of the memory window in RLSPD while processing the trace stream BPIC13I as an illustrative example. We record the appropriateness of the online model at each time step (as shown in Fig. 3(b)) and compare it with the fixed parameter strategy ($w_0 = 200$) depicted in Fig. 3(a).

The observations indicate that RLSPD tends to use smaller memory windows, enabling it to promptly adapt to the numerous trace variants and significant fluctuations in BPIC13I. Around time steps 100 and 200, a slight decline in the appropriateness is noticeable. RLSPD swiftly detects and mitigates this

Table 4. Zero-shot performance

Trace stream	Method	Memory size	Sampling rate	F-measure		Update count	
				IND	ILP	IND	ILP
BPIC13I	STARDUST	800	0.8	0.68	0.62	329	329
	Grid Search	10	0.1	**0.75**	**0.67**	341	341
	RLSPD-Z	200	0.8	**0.75**	**0.67**	**107**	**226**
BPIC20DD	STARDUST	1000	0.8	0.93	0.89	8	8
	Grid Search	200	0.2	**0.95**	**0.95**	**5**	**5**
	RLSPD-Z	200	0.8	**0.95**	**0.95**	12	15
BPIC20ID	STARDUST	600	0.8	0.50	0.63	487	487
	Grid Search	50	0.1	**0.89**	0.89	66	66
	RLSPD-Z	200	0.8	**0.89**	**0.90**	**28**	**47**
BPIC20PL	STARDUST	700	0.8	0.16	0.32	630	630
	Grid Search	50	0.1	**0.83**	**0.84**	105	105
	RLSPD-Z	200	0.8	0.82	**0.84**	**79**	**15**
BPIC20PC	STARDUST	200	0.8	0.57	0.77	134	134
	Grid Search	700	0.1	**0.89**	**0.89**	**0**	**0**
	RLSPD-Z	200	0.8	0.88	0.88	15	17
BPIC20RP	STARDUST	700	0.8	0.92	0.89	12	12
	Grid Search	200	0.2	**0.94**	**0.94**	**3**	**3**
	RLSPD-Z	200	0.8	**0.94**	**0.94**	6	4

change by increasing the memory window, thereby reducing noise interference and preventing erroneous model updates. Around time step 300, a sharp appropriateness decline prompts RLSPD to infer a sudden concept drift, reducing the memory window for rapid adaptation and timely model updates. This suggests that RLSPD adeptly captures trace stream fluctuations through evaluation metrics, effectively enhancing the model's conformance with future traces.

Zero-shot Performance Assessment. To assess RLSPD's zero-shot capability with entirely new trace streams, we conduct transfer reinforcement learning on RLSPD-Z. The RL agent is trained on five trace streams and tested on the remaining new ones. In contrast, STARDUST employs a fixed strategy when handling different trace streams, setting the memory window to 10% of the trace stream size and the sampling rate to 0.8. In addition, we perform a grid search on STARDUST's parameters, varying them in the grid $w_0 = \{10, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000\}$ and $\rho_0 = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ to find the best fixed parameters (labeled as Grid Search). Table 4 presents the initial parameters for different methods, along with the process model's average F-measure and update counts.

The results indicate that RLSPD-Z achieves comparable performance to Grid Search regarding the F-measure and effectively reduces the number of model updates. The optimal parameters identified through Grid Search suggest varying memory window requirements for different trace streams, with sampling rates

favoring lower values such as 0.1 or 0.2. This implies diverse fluctuation patterns among trace streams, where a small number of high-frequency trace variants can represent the overall trace distribution. In practical and various business scenarios, STARDUST exhibits poor performance with a fixed parameter strategy. Additionally, Grid Search requires partial data collection before searching and becomes time-consuming as the search space expands. In contrast, RLSPD, based on reinforcement learning with offline pre-training, demonstrates the ability to achieve satisfactory zero-shot results on entirely new trace streams.

6 Conclusion

In this paper, we propose RLSPD, a streaming process discovery approach for dynamically adapting to evolving business processes. It leverages feedback from conformance checking by reinforcement learning and updates the online process model upon detecting concept drifts. Experimental results demonstrate that RLSPD, through dynamic parameter adjustments, effectively improves model conformance with upcoming traces and reduces unnecessary model updates.

Currently, our approach relies on event stream data partitioning into complete traces, which is only suitable for business processes with declared completion activities. We plan to introduce predictive process monitoring to predict completion marks for ongoing cases. Additionally, our method only detects whether concept drifts occur, lacking an analysis of drift characteristics and root causes. Future work involves integrating relevant concept drift analysis methods. Furthermore, our future research will explore how activity sequence information in traces can be exploited in reinforcement learning, providing a more comprehensive understanding of trace stream dynamics.

Acknowledgments. This work is supported by the National Key Research and Development Program of China (No. 2022YFF0902701) and the National Natural Science Foundation of China (No. 62032016).

References

1. Van Der Aalst, W.: Process mining: overview and opportunities. ACM Trans. Manag. Inf. Syst. **3**(2), 1–17 (2012)
2. Leemans, S.J., Fahland, D., Van Der Aalst, W.M.: Discovering block-structured process models from event logs-a constructive approach. In: Colom, J.M., Desel, J. (eds.) Application and Theory of Petri Nets and Concurrency. Lecture Notes in Computer Science, vol. 7927, pp. 311–329. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-38697-8_17
3. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M., Verbeek, H.: Discovering workflow nets using integer linear programming. Computing **100**, 529–556 (2018)
4. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowl. Inf. Syst. **59**, 251–284 (2019)
5. Sato, D.M.V., De Freitas, S.C., Barddal, J.P., Scalabrin, E.E.: A survey on concept drift in process mining. ACM Comput. Surv. **54**(9), 1–38 (2021)

6. Burattin, A.: Streaming process mining. In: Process Mining Handbook, vol. 349 (2022)
7. Bose, R.J.C., van der Aalst, W.M., Žliobaitė, I., Pechenizkiy, M.: Handling concept drift in process mining. In: Mouratidis, H., Rolland, C. (eds.) Advanced Information Systems Engineering. Lecture Notes in Computer Science, vol. 6741, pp. 391–405. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-21640-4_30
8. Maaradji, A., Dumas, M., La Rosa, M., Ostovar, A.: Detecting sudden and gradual drifts in business processes from execution traces. IEEE Trans. Knowl. Data Eng. **29**(10), 2140–2154 (2017)
9. Adams, J.N., van Zelst, S.J., Rose, T., van der Aalst, W.M.: Explainable concept drift in process mining. Inf. Syst. **114**, 102177 (2023)
10. Barbon Junior, S., Tavares, G.M., da Costa, V.G.T., Ceravolo, P., Damiani, E.: A framework for human-in-the-loop monitoring of concept-drift detection in event log stream. In: Companion Proceedings of the the Web Conference 2018, pp. 319–326 (2018)
11. de Sousa, R.G., Peres, S.M., Fantinato, M., Reijers, H.A.: Concept drift detection and localization in process mining: an integrated and efficient approach enabled by trace clustering. In: Proceedings of the 36th Annual ACM Symposium on Applied Computing, pp. 364–373 (2021)
12. Zellner, L., Richter, F., Sontheim, J., Maldonado, A., Seidl, T.: Concept drift detection on streaming data with dynamic outlier aggregation. In: Leemans, S., Leopold, H. (eds.) Process Mining Workshops. Lecture Notes in Business Information Processing, vol. 406, pp. 206–217. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72693-5_16
13. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.: Event stream-based process discovery using abstract representations. Knowl. Inf. Syst. **54**, 407–435 (2018)
14. Batyuk, A., Voityshyn, V.: Streaming process discovery for lambda architecture-based process monitoring platform. In: 2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies, vol. 1, pp. 298–301. IEEE (2018)
15. Leno, V., Armas-Cervantes, A., Dumas, M., La Rosa, M., Maggi, F.M.: Discovering process maps from event streams. In: Proceedings of the 2018 International Conference on Software and System Process, pp. 86–95 (2018)
16. Batyuk, A., Voityshyn, V.: Streaming process discovery method for semi-structured business processes. In: 2020 IEEE Third International Conference on Data Stream Mining & Processing, pp. 444–448. IEEE (2020)
17. Pasquadibisceglie, V., Appice, A., Castellano, G., Fiorentino, N., Malerba, D.: Stardust: a novel process mining approach to discover evolving models from trace streams. IEEE Trans. Serv. Comput. **16**(4), 2970–2984 (2023)
18. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.: Alignment based precision checking. In: La Rosa, M., Soffer, P. (eds.) Business Process Management Workshops. Lecture Notes in Business Information Processing, vol. 132, pp. 137–149. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-36285-9_15
19. Berti, A., van der Aalst, W.M.: Reviving token-based replay: increasing speed while improving diagnostics. ATAED@ Petri Nets/ACSD **2371**, 87–103 (2019)
20. Fujimoto, S., Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: International Conference on Machine Learning, pp. 1587–1596. PMLR (2018)
21. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint: arXiv:1412.3555 (2014)