Uncovering and Quantifying Change: A Streaming Approach for Declarative Processes

Lasse Starklit[1], Hugo A. López[2], and Andrea Burattin[1]

[1] Technical University of Denmark, Kgs. Lyngby, Denmark
[2] University of Copenhagen, Copenhagen, Denmark

Abstract. Modern information systems have become vital and an integral part of most modern organizations. But everything changes over time, including processes. Process mining is a crucial tool to comprehend and elicit processes models. But as processes certainly adapt over time so should corresponding models. Therefore, the ability to understand which process is currently being executed represents a fundamental step in environments that continuously evolve (e.g., due to new regulations, policies, practices, seasons). In this paper we present a novel process discovery algorithm that extracts declarative processes as DCR graphs, starting from an event stream. Additionally, we present a new model-to-model metric capable of comparing DCR graphs, thus allowing the identification of similarities between the process observed in real-time and the stream and different candidate versions of the process. The technique has been validated both quantitatively and qualitatively on synthetic as well as real data, thus showing the benefit of the approach in a real setting.

Keywords: Streaming process discovery · DCR graphs · Model-to-model metric

1 Introduction

The only constant aspect of processes is their change. Either because of internal organization restructuring or because of variables external to the organization, processes are required to adapt quickly to achieve required outcomes. The COVID-19 pandemic showed us how organizations needed to move from physical work to hybrid or remote production facilities, forcing them to abandon optimized routes towards new flows. In administrative processes, each regulation change will require municipal governments to adapt their processes to preserve compliance. In Denmark, the laws determining the guidelines for case management in the social sector had 4,686 changes between 2009 and 2020 [13].

Process mining approaches promise that given enough data, a discovery technique will generate a model that is as close to reality as possible. This evidence-based approach has a caveat: one needs to assume that the observations that were used as inputs correspond to the same process. Not taking into consideration change might end in under- or over-constrained processes that do not represent the reality of the process. The second assumption is that it is possible to identify complete traces from the event log. This requirement indeed presents considerable obstacles in the organizations in which processes are constantly happening and evolving, either because the starting events are located in legacy systems no longer in use, or because current traces have not finished yet.

Accounting for change is particularly important in declarative processes. Based on a "outside-in" approach, declarative processes describe the minimal set of rules that generate accepting traces. To achieve this goal, declarative models place constraints between activities such that they restrict or enforce only compliant behaviour. For process mining, the simplicity of declarative processes have been demonstrated to fit well with real process executions, and declarative miners are at the moment the most precise miners in use[3]. However, little research exists regarding how declarative miners are sensitive to process change.

The objective is to study how declarative miners can give accurate and timely views of incomplete traces (so-called *event streams*) as well as measuring how similar these models are to each other. We integrate techniques of streaming process mining to declarative modelling notations, in particular, DCR graphs [14]. While previous techniques of streaming conformance checking have been applied to other declarative languages (e.g.: Declare [24]), these languages are fundamentally different. Declare provides a predefined set of 18 constraint templates taking inspiration from [12] with an underlying semantics based in LTL formulae on finite traces [8]. Instead, DCR is based on a minimal set of 5 constraints, being able to capture regular and omega-regular languages [10]. The choice of DCR is not fortuitous: DCR is an industrial process language integrated into KMD Workzone, a case management solution used by 70% of central government institutions in Denmark [23]. For the metrics, we present a fast & syntax-driven model-to-model metric and show its suitability in quantifying model changes.

Event streams presents challenges for process discovery. Streams are potentially infinite, making memory and processing power a major consideration. Our technique optimizes these aspects by relying on intermediate representations that will be updated at runtime. Another aspect considered is *extensibility*: our techniques not only rely on the minimal set of DCR constraints but it can be extended to more complex workflow patterns that combine atomic constraints.

Fig. 1 shows the paper's contributions: the first is a streaming mining component, capable of continuously generating DCR graphs from an event stream, the second is a model-to-model metric for DCR, which can be used to compare the results of the stream mining with a catalogue or repository of processes and thus indicate which of processes in the catalogue is currently being executed. An implementation of our techniques is implemented in Java and it is publicly available, together with all the tests and datasets[4].

The rest of the paper is structured as follows: related works are presented in Sec. 2; background on streaming process discovery and DCR graphs is in Sec. 3. The streaming discovery is presented in Sec. 4 and the model-to-model metric in Sec. 5. The approach is validated in Sec. 6 and Sec. 7 concludes the paper.

[3] See `https://icpmconference.org/2021/process-discovery-contest/`.
[4] See `https://github.com/beamline/dcr-miners`.

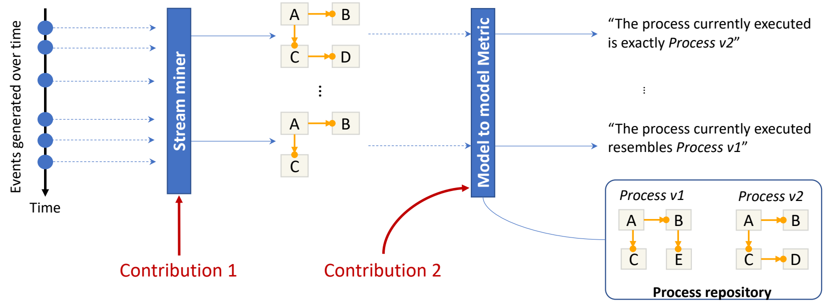


Fig. 1: Contributions of the paper

2 Related Work

This paper is the first work aiming at the discovery of DCR graphs from an event stream. In the literature, it is possible to find work referring to either offline discovery for DCR graphs or online discovery for Declare models. In the rest of the section, we will also discuss streaming process mining in general terms.

Offline process discovery techniques. The state of the art in the offline discovery of DCR is represented by the DisCoveR algorithm [3]. In their paper, the authors claim an accuracy of 96,2% with linear time complexity. The algorithm is an extension of the ParNek algorithm [22] but it uses a highly efficient implementation of DCR, mapping to bit vectors, where each activity corresponds to a particular index of the vector. A more recent approach, described in [25], presents the Rejection miner which exploits the idea of having both positive and negative examples to produce a better process model.

Online Discovery for Declarative Models. In [5] a framework for the discovery of Declare models from streaming event data has been proposed. This framework can be used to process events online, as they occur, as a way to deal with large and complex collections of datasets that are impossible to store and process altogether. In [21] the work has been generalized to handle also the mining of data constraints, leveraging the MP-Declare notation [6].

Streaming Process Mining in General. Another important research conducted within the area of online process mining is the work done by van Zelst, in his PhD thesis [30]. Throughout the thesis, the author proposes process mining techniques applicable to process discovery, conformance checking and process enhancement from event streams. An important conclusion from his research consists of the idea of building intermediate models that can capture the knowledge observed in the stream before creating the final process model.

In [4] the author presents a taxonomy for the classification of streaming process mining techniques. Our techniques constitute a hybrid approach in the categories in [4], mixing a smart window-based model which is used to construct and maintain an intermediate structure updated, and a problem reduction technique used to transform such structure into a DCR graph.

3 Background

In the following section, we recall basic notions of Directly Follows Graphs [1] and the Dynamic Condition Response (DCR) graphs [14]. While in general, DCR is expressive enough to capture multi-perspective constraints such as time [16], data [27], sub-processes [9] and message-passing constraints [17], in this paper we use the classical, set-based formulation first presented in [14].

Definition 1 (Sets, Events and Sequences). *Let $\mathcal{C}$ denote the set of possible case identifiers and let $\mathcal{A}$ denote the set of possible activity names. The event universe is the set of all possible events $\mathcal{E} = \mathcal{C} \times \mathcal{A}$ and an event is an element $e = (c, a) \in \mathcal{E}$. The set of activity labels is denoted by L. Given a set $\mathbb{N}_n^+ = 1, 2, \ldots, n$ and a target set A, a sequence $\sigma : \mathbb{N}_n^+ \to A$ maps index values to elements in A. For simplicity we can consider sequences using a string interpretation: $\sigma = \langle a_1, \ldots, a_n \rangle$ where $\sigma(i) = a_i \in A$.*

With this definition, we can now formally characterize an event stream:

Definition 2 (Event stream). *An event stream is an unbounded sequence mapping indexes to events: $\mathcal{S} : \mathbb{N}^+ \to \mathcal{E}$.*

Our approach for extracting DCR graphs leverages the notion of Extended Directly Follows Grap, which is an extension of DFG:

Definition 3 (Directly Follows Graph (DFG)). *A directly follows graph is a graph $G = (V, R)$ where nodes represent activities (i.e., $V \subseteq \mathcal{A}$), and edges indicate a directly follows relation from the source activity to the target activity (i.e., $(a_s, a_t) \in V$ with $a_s, a_t \in R$, so $V \subseteq V \times V$).*

Definition 4 (Extended DFG). *An extended DFG is a graph $G_x = (V, R, X)$ where (V, R) is a DFG and X contains additional numerical attributes referring to the nodes: $X : V \times Attrs \to \mathbb{R}$, where $Attrs$ is the set of all attribute names. To access attribute a_1 for node v we use the notation $X(v, a_1)$.*

In the rest of the paper we will consider the following attributes $Attrs = \{\mathsf{avgFO}, \mathsf{noTraceApp}, \mathsf{avgIdx}\}$:

- avgFO: this attribute indicates the average first occurrence of the activity among the traces seen so far;
- $\mathsf{noTraceApp}$: this attribute indicate the number of traces that the activity has appeared in so far;
- avgIdx: this attribute indicate the average occurrence index of the activity.

A DCR graph consists of a multi-directed graph and a marking.

Definition 5 (DCR Graph). *A* DCR graph *is a tuple* $\langle \mathcal{A}, M, L, \ell, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\% \rangle$*, where* $M \in \mathcal{P}(\mathcal{A}) \times \mathcal{P}(\mathcal{A}) \times \mathcal{P}(\mathcal{A})$ *is a* marking*,* $\ell : \mathcal{A} \rightarrow L$ *is a* labelling map*, and* $\phi \subseteq \mathcal{A} \times \mathcal{A}$ *for* $\phi \in \{\rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%\}$ *are* relations *between activities.*

A DCR graph defines processes whose executions are finite and infinite sequences of activities. An activity may be executed several times. The three sets of activities in the marking $M = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ define the state of a process, and they are referred to as the *executed* activities (Ex), the *pending* response (Re)[5] and the *included* activities (In). DCR relations define what is the effect of executing one activity in the graph for its context. Briefly:

- Condition relations $a \rightarrow\!\bullet a'$ say that the execution of a is a prerequisite for a', i.e. if a is included, then a must have been executed for a' to be enabled for execution.
- Response relations $a \bullet\!\rightarrow a'$ say that whenever a is executed, a' becomes pending. In a run, a pending event must eventually be executed or be excluded. We refer to a' as a response to a.
- A milestone relation $a \rightarrow\!\diamond a'$ means that if a is included it must not be pending for a' to be enabled for execution. We refer to a as a milestone for a'. Milestones are typically used in cyclic behaviour, when some earlier executed event a may be required to be executed again, i.e. it becomes pending before the process can proceed to execute event a'.
- An inclusion (respectively exclusion) relation $a \rightarrow\!+ a'$ (respectively $a \rightarrow\!\% a'$) means that if a is executed, then a' is included (respectively excluded).

For a DCR graph P with activities $\mathcal{A}$ and marking $M = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ we write $P_{\bullet\rightarrow}$ for the set of pairs $\{(x, y) \mid \{x, y\} \in \mathcal{A} \wedge (x, y) \in \bullet\!\rightarrow\}$ (similarly for any of the relations in ϕ) and we write $P_{\mathcal{A}}$ for the set of activities. Please note that in the paper we will use DCR graph and DCR model interchangeably.

4 Streaming DCR Miner

This section presents the general structure of the stream mining algorithm for DCR graphs. The general idea of the approach presented in this paper is depicted in Fig. 2: constructing and maintaining an extended DFG structure (cf. Def. 4) starting from the stream and then, periodically, a new DCR graph is extracted from the most recent version of the extended DFG available. The extraction of the different DCR rules starts from the same extended DFG instance. For readability purposes, we split the approach into two algorithms. The former (Alg. 1) is in charge of extracting the extended DFG, the latter (Alg. 2) focuses only on the extraction of DCR rules from the extended DFG.

Algorithm 1 takes as input a stream of events S, a set of DCR rules T to mine and two parameters referring to the maximum number of traces m_t and events to store m_e. The algorithm starts by initializing two supporting map data

[5] We might simply say pending when it is clear from the context.

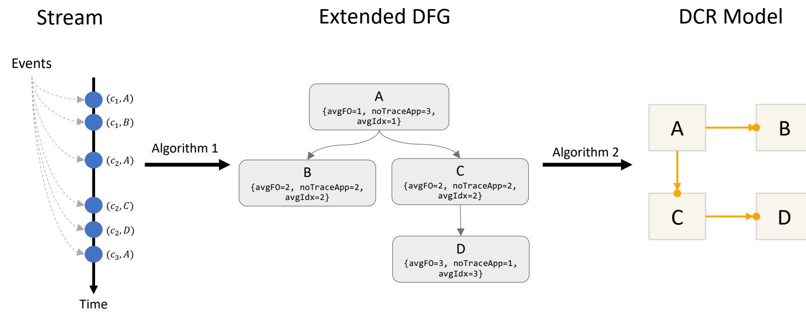


Fig. 2: Conceptual representation of the discovery strategy in this paper.

structures obs and deps as well as an empty extended DCR graph G_X (lines 1-3). obs is a map associating case ids to sequences of partial traces; deps is a map associating case ids to activity names. After this initialization phase, the algorithm starts consuming the actual events in a never-ending loop (line 4). The initial step of such loop consists of receiving the new event (line 5) which consists of a case id c and activity name a. Then, two major steps take place: the first step consists of updating the extended DFG graph; the second consists of transforming the extended DFG into a DCR model.

To update the extended DFG we must update 3 components: the set of nodes V, the set of relations R and the extra attributes X. The algorithm first updates V and X: if the case id c of the new event has been seen before (line 6), then the algorithm refreshes the update time of the case id (line 7, useful to keep track of which cases are the most recent ones) and checks whether the maximum length of the partial trace for that case id has been reached (line 8). If that is the case, then the oldest event is removed and the G_X is updated to incorporate the removal of the event. If this is the first time this case id is seen (line 11), then it is first necessary to verify that the new case can be accommodated (line 12) and, if there is no room, then first some space needs to be created by removing oldest cases and propagating corresponding changes (lines 13-14) and then a new empty list can be created to host the partial trace (line 15). In either situation, the new event is added to the partial trace (line 16) and, if needed, a new node is added to the set of vertices V (line 17). The X data structure can be refreshed by leveraging the properties of the partial trace seen so far (line 18).

To update the relations in the extended DFG (i.e., the R component of G_X), the algorithm checks whether an activity was seen previously for the given case id c and, if that is the case, the relation from such activity (i.e., deps(c)) to the new activity just seen (i.e., a) is added (lines 19-20). In any case, the activity just observed is now the latest activity for case id c (line 21).

Finally, according to some periodicity (line 22), the algorithm refreshes the DCR model by calling the procedure that transforms the extended DFG into a DCR model (cf. Alg. 2) (lines 23-24).

Algorithm 1: General structure of Streaming DCR Miner

Input: S: stream of events
m_t: maximum number of traces to store
m_e: maximum number of events per trace to store
T: set of DCR rules to mine

Initialize map obs// Maps case ids to sequence of activities
Initialize map deps// Maps case ids to one activity name
Initialize extended DFG $G_X = (V, R, X)$
forever do
    // Step 0: Observe new activity a for case c
    $(c, a) \leftarrow observe(S)$
    // Step 1: Update of the extended DFG
    if $c \in$ *obs* **then**
        Refresh the update time of c
        if $|obs(c)| \geq m_e$ **then**
            Remove oldest event from list $obs(c)$
            Update V and X of G_X to be consistent with the event just removed
    else
        if $|obs| \geq m_t$ **then**
            Remove the oldest trace from obs and all its events
            Update V and X of G_X to be consistent with the events just removed
        $\mathsf{obs}(c) \leftarrow \langle\rangle$ // Create empty list for obs(c)
    $\mathsf{obs}(c) \leftarrow \mathsf{obs}(c) \cdot \langle a \rangle$ // Append a to obs(c)
    $V \leftarrow V \cup \{a\}$
    Update frequency and avg appearance index in X component of G_X // The average appearance index is updated considering the new position given by $|obs(c)|$
    if $c \in$ *deps* **then**
        $R \leftarrow R \cup \{(\mathsf{deps}(c), a)\}$
    $\mathsf{deps}(c) \leftarrow a$
    // Step 2: Periodic update of the DCR model
    if *trigger periodic update of the model* **then**
        $M \leftarrow \mathsf{mine}(T, G_X)$ // See Algorithm 2
        Notify about new model M

The generation of a DCR model from an extended DFG is described in Algorithm 2. We illustrate the mining of DCR *conditions* and *responses* but a more complete algorithm comprising *inclusions* and *exclusions*, is available in [26]. The algorithm takes as input an extended DFG G_X and a set of rule patterns. It starts by creating an empty DCR graph P, where the set of activities are the same as the nodes in G_X, with an initial marking $M_{init} = \{\emptyset, \emptyset, V\}$, that is, all events are included, not pending and not executed. The identity function I maps elements in V to itself. We can now iterate over all edges of G_X. For each edge between activity s and t, the algorithm checks if, on average, s appears before t (line 4). This condition, together with the fact that there is a dependency between s and t (derived from the fact that this is an edge in the DFG), allows us to infer that there is a DCR response constraint from s to t which means that whenever s executes, t becomes pending, i.e., t should be executed afterwards. Hence, the rule is added to the model (line 5). To detect conditions, the algorithm verifies a different set of properties: given a dependency in the DFG between s and t, it checks that s occurs for the first time before t, to conclude that indeed s is a necessary condition for t (using the avgFO attribute, in the first half of line 7); and that every time t appeared in a trace, s was there too (approximated by

Algorithm 2: Mining of rules starting from the extended DFG

Input: T: set of DCR rules to mine
$G_X = (V, R, X)$: extended DFG

1 $P \leftarrow \langle V, M_{init}, V, I, \rightarrow\!\bullet = \emptyset, \bullet\!\rightarrow = \emptyset, \rightarrow\!\diamond = \emptyset, \rightarrow\!+ = \emptyset, \rightarrow\!\% = \emptyset \rangle$ // Initial DCR graph
2 **foreach** $(s, t) \in R$ **do** // R is the set of relations in G_X
3   **if** *RESPONSE* $\in T$ **then**
4   **if** $X(s, \mathit{avgIdx}) < X(t, \mathit{avgIdx})$ **then**
5             $P \leftarrow \langle V, M_{init}, V, I, P_{\rightarrow\bullet}, P_{\bullet\rightarrow} \cup (s, t), P_{\rightarrow\diamond}, P_{\rightarrow+}, P_{\rightarrow\%} \rangle$ // Response from s to t
6   **if** *CONDITION* $\in T$ **then**
7   **if** $X(s, \mathit{avgFO}) < X(t, \mathit{avgFO}) \wedge X(s, \mathit{noTraceApp}) \geq X(t, \mathit{noTraceApp})$ **then**
8             $P \leftarrow \langle V, M_{init}, V, I, P_{\rightarrow\bullet} \cup (s, t), P_{\bullet\rightarrow}, P_{\rightarrow\diamond}, P_{\rightarrow+}, P_{\rightarrow\%} \rangle$ // Condition from s to t
    // Further patterns here
9     $P \leftarrow \mathit{RemoveRedundancies}(P)$ // Apply rule reduction from transitive closure of rules
10 **return** P

counting in how many traces contain the two activities, in the second half of line 7). If these properties are fulfilled, then a condition from s to t can be added. The algorithm finishes by applying a transitive reduction strategy [3], removing the number of rules while maintaining identical reachability properties.

One of the aspects to consider in this algorithm was modularization. Behaviours such as exclusive choices or sequential composition are not modelled by a single DCR constraint, but by a range of them. For example, the (very restricted) sequential composition a to b in BPMN will require 4 constraints in DCR: $\{a\rightarrow\!\bullet b, a\bullet\!\rightarrow b, a\rightarrow\!\% a, b\rightarrow\!\% b\}$. Behavioural patterns in DCR are defined in composition: atomic patterns (single constraints) serve as sink nodes of a dependency graph of patterns. Pattern application requires the exploration of pattern dependencies, which can be resolved via standard post-order graph traversal.

Suitability of the Algorithms for Streaming Settings Whenever discussing algorithms that can tackle the streaming process mining problem [4], it is important to keep in mind that while a stream is assumed to be infinite, only a finite amount of memory can be used to store all information and that the time complexity for processing each event must be constant. Concerning the memory, an upper bound on the number of stored events in Alg. 1 is given by $m_t \cdot m_e$ which happens when each trace contains more than m_e events and at least m_t traces are seen in parallel. Additionally, please note that the extended DFG is also finite since there is a node for each activity contained in the memory. Concerning the time complexity, Alg. 1 does not perform any unbounded backtracking but, instead, for each event, it operates using just maps that have amortized constant complexity or on the extended DFG (which has finite, controlled size). The same observation holds for Alg. 2 as it iterates on the extended DFG which has a size bounded by the provided parameters (and hence, can be considered constant).

5 Model-to-Model Metric for DCR Models

Our second contribution is a model-to-model metric capable of quantifying to which extent two DCR graphs are similar. This metric can be used, for example, to identify which process is currently being executed with respect to a repository of candidate processes (cf. Contribution 2 in Fig. 1), or by quantifying the *change rate* of the same process over time. The metric takes as input two DCR graphs P and Q as well as a weight relation W that associates each DCR relation in ϕ (cf. Def. 5) with a weight, plus one additional weight for the activities. Then it computes the weighted Jaccard similarity [18] of the sets of relations and the set of activities, similarly to what happens in [2] imperative models:

Definition 6 (DCR Model-to-Model metric). *Given P and Q two DCR graphs, and $W : \phi \cup \{\mathsf{act}\} \rightarrow \mathbb{R}$ a weight function in the range $[0, 1]$ such that $\sum_{r \in \phi \cup \{\mathsf{act}\}} W(r) = 1$. The model-to-model similarity metric is defined as:*

$$S(P, Q, W) = W(\mathsf{act}) \cdot \frac{|P_{\mathcal{A}} \cap Q_{\mathcal{A}}|}{|P_{\mathcal{A}} \cup Q_{\mathcal{A}}|} + \sum_{r \in \phi} W(r) \cdot \frac{|P_r \cap Q_r|}{|P_r \cup Q_r|} \tag{1}$$

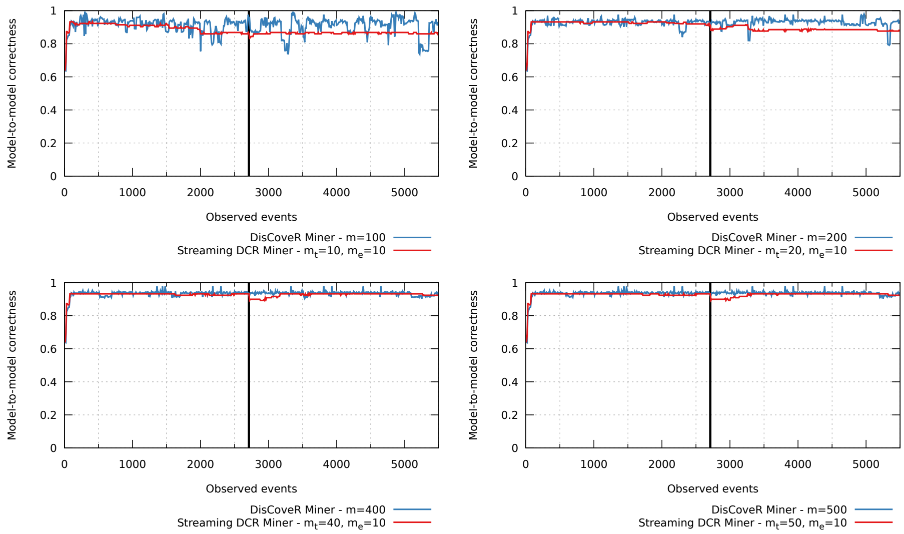
The model-to-model metric is a similarity metric comparing the relations in each of the two DCR graphs, thus returning a value between 0 and 1, where 1 indicates perfect match and 0 stands for no match at all.

6 Experimental Evaluation

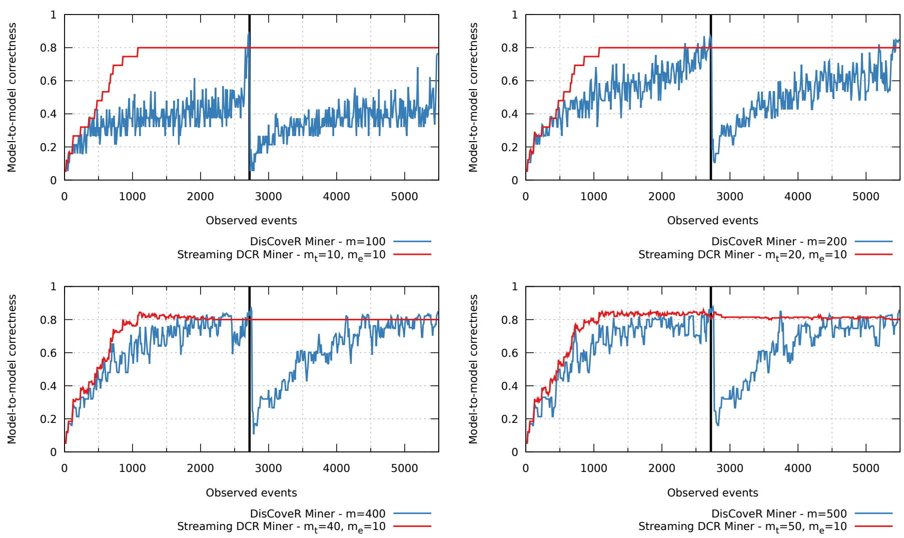
To validate the results of the two approaches presented in the paper we executed several tests, first to validate quantitatively each contribution (i.e., the streaming discovery and the model-to-model metric) on synthetic data, then to qualitatively evaluate the whole approach on a real dataset.

6.1 Quantitative Evaluation of Streaming Discovery

Our goal with the first quantitative evaluation is to compare the stability of the streaming DCR miner against sudden changes. We compare with other process discovery algorithms for DCR graphs, in this case, the DisCoveR miner [3]. The tests are performed against a publicly available dataset of events streams [7]. This dataset includes (1) a synthetic stream inspired in a loan application process, and (2) perturbations to the original stream using change patterns [29]. Recall that the DisCoveR miner is an *offline* miner, thus it assumes an infinite memory model. To provide a fairer evaluation we need to parameterise DisCoveR with the same amount of available memory. We divided the experiment into two parts: a simple stream where the observations of each process instance arrive in an ordered manner (i.e., one complete process instance at the time) and a complex stream where observations from many instances arrive intertwined. As no initial DCR graph exists for this process, we used the DisCoveR miner in its original (offline) setting to generate a baseline graph using the entire dataset. This model was used to calculate the model-to-model similarity between the DCR stream miner and the DisCoveR miner with memory limits. For the sake of simplicity, in this paper, we considered only the case of sudden drifts, while we discuss other types of drift in the future work section.



(a) Performance comparison on a simple stream.



(b) Performance comparison on a complex stream.

Fig. 3: Performance comparison between the offline DisCoveR miner and the streaming DCR Miner with the same amount of storage available (with a capacity of up to 100, 200, 400 and 500 events). A vertical black bar indicates a drift in the model generating the stream.

The results are reported in Fig. 3a and 3b (for simple and complex stream resp.). Each figure shows the performance of the incremental version of DisCoveR and the streaming DCR miner against four different configurations over time. The vertical black bars indicate where a sudden drift occurred in the stream. While the performance for the simple stream is very good for both the DisCoveR and the streaming DCR miners, when the stream becomes more complicated (i.e., Fig. 3b), DisCoveR becomes less effective and, though its average performance increases over time, the presence of the drift completely disrupt the accuracy. In contrast, our approach is much more robust to the drift and much more stable over time, proving its ability at managing the available memory in a much more effective way.

6.2 Quantitative Evaluation of Model-to-model Metric

For the second experiment, we investigated the quality of our model-to-model metric. To accomplish this goal, we used a dataset of 28 DCR process models collected from previous mapping efforts [20] and, for each model, we randomly introduced variations such as: adding new activities connected to the existing fragments, adding disconnected activities, deleting existing activities (with corresponding constraints), adding constraints, removing constraints, and swapping activity labels in the process. By systematically applying all possible combinations of variations in a different amount (e.g., adding 1/2/3 activities and nothing else; adding 1/2/3 activities and removing 1/2/3 constraints) we ended up with a total of 455,826 process models with a quantifiable amount of variation from the 28 starting processes.

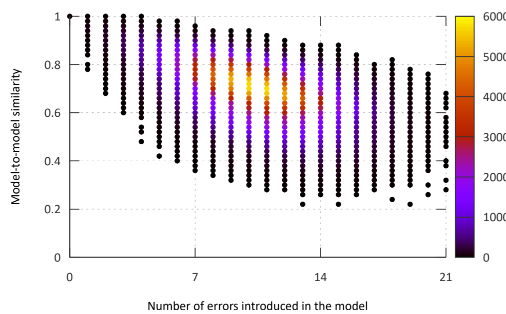


Fig. 4: Scatter plot showing the correlation between the model-to-model metric and the number of changes introduced in the model. The colour indicates the density of observations.

Fig. 4 shows each variation on a scatter plot where the x axis refers to the number of variations introduced in the model and the y axis refers to the model-to-model similarity. The colour indicates the number of models in the proximity of each point (since multiple processes have very close similarity scores). For identifying the optimal weights we solve an optimization problem, aiming at finding the highest correlation between the points, ending up with: $W = \{(\rightarrow\bullet, 0.06), (\bullet\rightarrow, 0.07), (\rightarrow\diamond, 0.06), (\rightarrow+, 0.07), (\rightarrow\%, 0.13), (\mathsf{act}, 0.61)\}$ which leads to a Pearson's correlation of -0.56 and a Spearman's correlation of -0.55. These values indicate that our metric is indeed capable of capturing the changes. As the metric is very compact (value in $[0, 1]$) and operates just on the topological structure of the model, it cannot identify all details. However, the metric benefits from the fast speed of computation.

6.3 Qualitative Evaluation of the Entire Approach

For the final evaluation, we run a qualitative analysis on a real stream. We compared the results of the streaming miner against a real process model from one of our partner companies: the Dreyers Fond case [11]. The stream contains activities for a grant application system from December 2013 until May 2015, with 11,044 events, some of which belong to partial (never finished) traces. The data was generated from a DCR graph[6]. We used this data to explore whether the streaming miner is capable to identify actual drifts and validate such drifts with the process owner. We finally studied the computational overhead on data from a real scenario and the number of constraints identified in a realistic case.

Fig. 5a presents the model-to-model similarity against the normative model. It is important to point out that the normative model includes language features that decrease the number of constraints in the model (e.g. nesting [15] and data events), resulting in a low absolute similarity score. However, two important drifts in the model were detected: one before March 2013, and another in May 2015 (highlighted in the figure as well). We inquired the process owner regarding these changes, and they were both were confirmed. In the first case, a testing phase, before the process entered into production (cit. *"The Dreyers Fond went live in December 2013 but in reality, did not process any applications until March 2014"*), took place. The second drift uncovered a system malfunction (cit. *"In May 2015 I recall we had a server crash where we manually had to fix things"*).

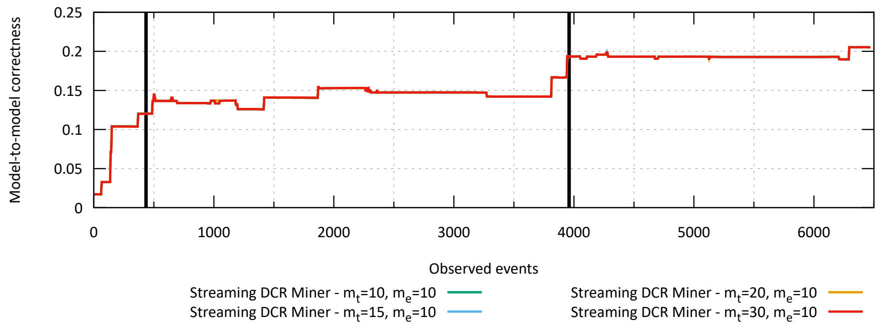
In the second analysis, depicted in Fig. 5b, we reported the time required to process each event. As the graph shows, processing each event on a consumer laptop (Macbook) requires about 1 millisecond, thus showing the applicability of our technique in real settings.

Finally, Fig. 5c shows the number of discovered constraints over time. It is worth noticing that initially, the number of constraints grow consistently for all different configurations, however, after a while, the configuration with the larger memory extracts a smaller number of constraints. This is because more memory available means having more observations and thus more potential counterexamples to the requirements for having a constraint (cf. Alg. 2).

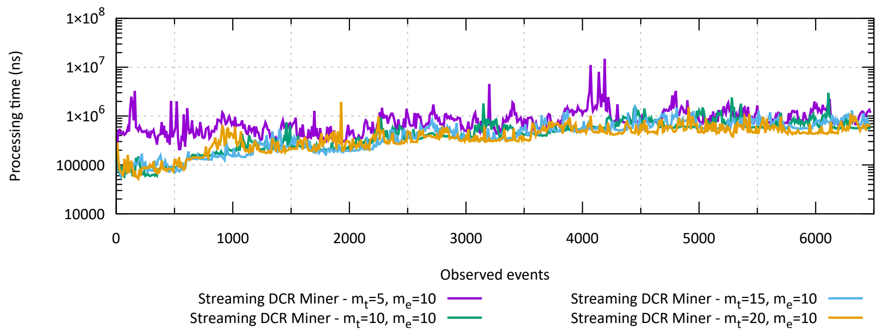
6.4 Discussion

A limiting aspect of the approach relies on the choice of the intermediate structure. As mentioned in [28], a DFG representation may report confusing model behaviour as it simplifies the observations using purely a frequency-based threshold. Second, a DFG is in essence an imperative data structure that captures the most common flows that appear in a stream. This, in a sense, goes against the

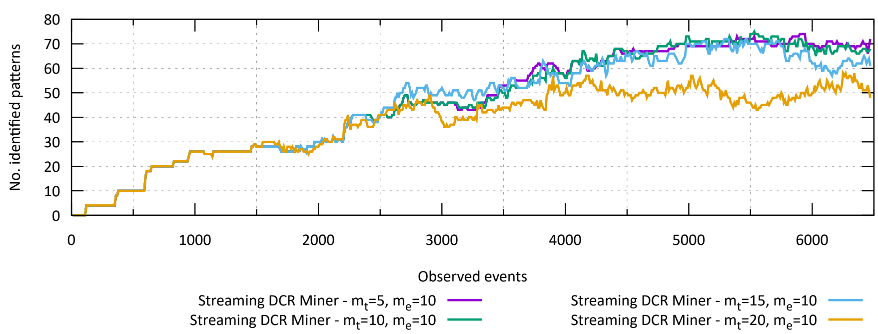
[6] The latest version is available at `https://www.dcrgraphs.net/tool/main/Graph?id=59a932f8-1011-4232-bbc5-9b39efb1fc18`.



(a) Model-to-model similarity calculated between the model mined at each point in time and the normative model, with different memory configurations. The vertical black bars are positioned according to actual drifts in the process.



(b) Time required to process each event (expressed in nanoseconds, log scale) over time on the real stream data, with four different memory configurations.



(c) Number of constraints extracted over time on the real stream data, with four different memory configurations.

Fig. 5: Analyses on the real stream referring to the Dreyers Fond.

declarative paradigm as a second-class citizen with respect to declarative constraints. However, the DFG still provides a valid approximation to observations of streams where we do not have complete traces.

7 Conclusion and Future Work

This paper presented a novel streaming discovery technique capable of extracting declarative models, expressed using the DCR language, from event streams. Additionally, a model-to-model metric is reported which allows understanding if and to what extend two DCR models are the same. A thorough experimental evaluation, comprising both synthetic and real data, validated the two contributions separately as well as their combination in a qualitative fashion, which included interviews with the process owner.

We plan to explore several directions as future work. Regarding the miner, we plan to extend its capabilities to the identification of sub-processes, nesting and data constraints. Regarding the model-to-model similarity, we would like to embed more semantic aspects, such as mentioned in [19].

Acknowledgments We would like to thank Morten Marquard from DCR Solutions for providing valuable information regarding the Dreyers Fond case.

References

1. van der Aalst, W.: Process Mining. Springer Berlin Heidelberg (2016)
2. Aiolli, F., Burattin, A., Sperduti, A.: A business process metric based on the alpha algorithm relations. In: BPM Workshop. pp. 141–146. Springer (2011)
3. Back, C.O., Slaats, T., Hildebrandt, T.T., Marquard, M.: Discover: accurate and efficient discovery of declarative process models. International Journal on Software Tools for Technology Transfer (Jun 2021). https://doi.org/10.1007/s10009-021-00616-0, `https://doi.org/10.1007/s10009-021-00616-0`
4. Burattin, A.: Streaming process discovery and conformance checking. In: Encyclopedia of Big Data Technologies. Springer (2019)
5. Burattin, A., Cimitile, M., Maggi, F.M., Sperduti, A.: Online discovery of declarative process models from event streams. IEEE Trans. Serv. Comput. **8**(6) (2015)
6. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016)
7. Ceravolo, P., Tavares, G.M., Junior, S.B., Damiani, E.: Evaluation goals for online process mining: a concept drift perspective. IEEE Trans. Serv. Comput. (2020)
8. De Giacomo, G., De Masellis, R., Montali, M.: Reasoning on ltl on finite traces: Insensitivity to infiniteness. In: AAAI Conference on Artificial Intelligence (2014)
9. Debois, S., Hildebrandt, T., Slaats, T.: Safety, liveness and run-time refinement for modular process-aware information systems with dynamic sub processes. In: FM 2015. pp. 143–160. No. 9109 in LNCS, Springer (2015)
10. Debois, S., Hildebrandt, T.T., Slaats, T.: Replication, refinement & reachability: complexity in dynamic condition-response graphs. Acta Informatica **55**(6) (2018)
11. Debois, S., Slaats, T.: The analysis of a real life declarative process. In: 2015 IEEE Symposium Series on Computational Intelligence. pp. 1374–1382. IEEE (2015)

12. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of ICSE. pp. 411–420 (1999)
13. Eiriksson, B.A., Nordland, I.: Analyse: Regelkompleksitet på det socialretlige område. Tech. rep., Justitia - Danmarks uafhængige juridiske tænketank (2020), http://justitia-int.org/wp-content/uploads/2020/03/Analyse-Regelkompleksitet_Endelig.pdf
14. Hildebrandt, T., Mukkamala, R.R.: Declarative Event-Based Workflow as Distributed Dynamic Condition Response Graphs. In: PLACES. vol. 69 (2010)
15. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T.: Nested Dynamic Condition Response Graphs. In: FSEN. vol. 7141, pp. 343–350. Springer (2011)
16. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T., Zanitti, F.: Contracts for cross-organizational workflows as timed dynamic condition response graphs. JLAMP **82**(5-7), 164–185 (2013)
17. Hildebrandt, T.T., Slaats, T., López, H.A., Debois, S., Carbone, M.: Declarative choreographies and liveness. In: FORTE. LNCS, Springer (February 2019)
18. Jaccard, P.: The Distribution of the Flora of the Alpine Zone. New Phytologist **11**(2), 37–50 (1912)
19. López, H.A., Debois, S., Slaats, T., Hildebrandt, T.T.: Business process compliance using reference models of law. In: FASE. pp. 378–399. Springer (2020)
20. López, H.A., Strømsted, R., Niyodusenga, J., Marquard, M.: Declarative process discovery: Linking process and textual views. In: CAiSE Forum. Lecture Notes in Business Information Processing, vol. 424, pp. 109–117. Springer (2021)
21. Navarin, N., Cambiaso, M., Burattin, A., Maggi, F.M., Oneto, L., Sperduti, A.: Towards Online Discovery of Data-Aware Declarative Process Models from Event Streams. In: Proceedings of IJCNN (2020)
22. Nekrasaite, V., Parli, A.T., Back, C.O., Slaats, T.: Discovering responsibilities with dynamic condition response graphs. In: CAiSE. pp. 595–610. Springer (2019)
23. Norgaard, L.H., Andreasen, J.B., Marquard, M., Debois, S., Larsen, F.S., Jeppesen, V.: Declarative process models in government centric case and document management. In: BPM (Industry Track). CEUR Workshop Proceedings, vol. 1985, pp. 38–51. CEUR-WS.org (2017)
24. Pesic, M., Van der Aalst, W.M.: A declarative approach for flexible business processes management. In: Proc. of BPM. pp. 169–180. Springer (2006)
25. Slaats, T., Debois, S., Back, C.O.: Weighing the pros and cons: Process discovery with negative examples. In: Proc. of BPM. pp. 47–64. Springer (2021)
26. Starklit, L.: Online Discovery and Comparison of DCR models from Event Streams using Beamline. Master's thesis, DTU (2021), https://findit.dtu.dk/en/catalog/2692726044
27. Strømsted, R., López, H.A., Debois, S., Marquard, M.: Dynamic evaluation forms using declarative modeling. In: BPM (Dissertation/Demos/Industry). vol. 2196, pp. 172–179. CEUR-WS.org (2018)
28. van der Aalst, W.M.: A practitioner's guide to process mining: Limitations of the directly-follows graph. Procedia Computer Science **164**, 321–328 (2019)
29. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features–enhancing flexibility in process-aware information systems. Data & knowledge engineering **66**(3), 438–466 (2008)
30. van Zelst, S.J.: Process mining with streaming data. Ph.D. thesis, Technische Universiteit Eindhoven (2019)