Uncovering Change: A Streaming Approach for Declarative Processes

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

[1] Technical University of Denmark, Kgs. Lyngby, Denmark
hulo@dtu.dk
[2] Netcompany A/S, Copenhagen, Denmark

Abstract. Process discovery is a family of techniques that helps to comprehend processes from their data footprints. Yet, as processes change over time so should their corresponding models, and failure to do so will lead to models that under- or over-approximate behaviour. We present a discovery algorithm that extracts declarative processes as Dynamic Condition Response (DCR) graphs from event streams. Streams are monitored to generate temporal representations of the process, later processed to create declarative models. We validated the technique by identifying drifts in a publicly available dataset of event streams. The metrics extend the Jaccard similarity measure to account for process change in a declarative setting. The technique and the data used for testing are available online.

Keywords: Streaming process discovery · Declarative processes · DCR graphs

1 Introduction

Process discovery techniques promise that given enough data, it is possible to output a realistic model of the process as is. This evidence-based approach has a caveat: one needs to assume that inputs belong to the same process. Not considering process variance over time might end in under- or over-constrained models that do not represent reality. The second assumption is that it is possible to identify full traces from the event log. This requirement indeed presents considerable obstacles in organizations where processes are constantly 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. For process mining, the simplicity of declarative processes has been demonstrated to fit well with real process executions, and declarative miners are currently the most precise miners in use[1]. However, little research exists regarding how declarative miners are sensitive to process change. The objective of this paper is to study how declarative miners can give accurate and timely views of partial traces (so-called *event streams*). We integrate techniques of streaming process mining to declarative

[1] See https://icpmconference.org/2021/process-discovery-contest/.

Alphabetical order, equal authors contribution.

M. Montali et al. (Eds.): ICPM 2022 Workshops, LNBIP 468, pp. 158–170, 2023.
https://doi.org/10.1007/978-3-031-27815-0_12

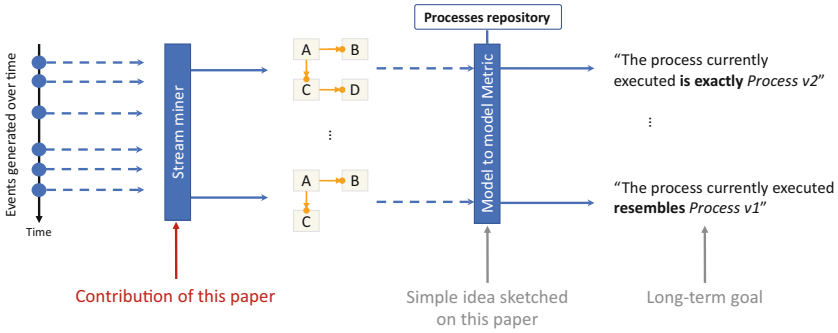


Fig. 1. Contribution of the paper

modelling notations, in particular, DCR graphs [14]. While previous works of streaming conformance checking have addressed other declarative languages (e.g.: Declare [23]), these languages are fundamentally different. Declare provides a predefined set of 18 constraint templates with an underlying semantics based on LTL formulae on finite traces [12]. Instead, DCR is based on a minimal set of 5 constraints, being able to capture regular and omega-regular languages [13]. In comparison with Declare, DCR is a language adopted by the industry: DCR is integrated into KMD Workzone, a case management solution used by 70% of central government institutions in Denmark [22]. Event streams present challenges for discovery. Streams are potentially infinite, making memory and time computation complexities major issues. Our technique optimizes these aspects by relying on intermediate representations that are updated at runtime. Another aspect is *extensibility*: our technique can be extended to more complex workflow patterns via the combination of atomic DCR constraints. Figure 1 illustrates our contribution: a streaming mining component, capable of continuously generating DCR graphs from an event stream (here we use the plural *graphs* to indicate that the DCR model could evolve over time, to accommodate drifts in the model that might occur). Towards the long-term goal of a system capable of spotting changes in a detailed fashion, we will also sketch a simple model-to-model metric for DCR, which can be used to compare the results of stream mining with a catalogue or repository of processes. An implementation of our techniques together with tests and datasets is available in Beamline[2] [6].

The rest of the paper is structured as follows: related works are presented in Sect. 2; theoretical background is covered in Sect. 3. The streaming discovery is presented in Sect. 4 and the approach is validated in Sect. 5. Section 6 concludes.

2 Related Work

This is the first work aiming at discovering DCR graphs from event streams. We find related work in offline discovery of DCR graphs and stream process mining for Declare.

Offline Process Discovery Techniques. The most current discovery technique for DCR graphs is the DisCoveR algorithm [4]. In their paper, the authors claim an accuracy of

[2] See https://github.com/beamline/discovery-dcr.

96,1% with linear time complexity (in PDC 2021 the algorithm achieved 96.2%). The algorithm is an extension of the ParNek algorithm [21] using an efficient implementation of DCR mapping via bit vectors. In its most recent version [24], DisCoveR has been extended with the idea of having both positive and negative examples to produce a more precise process model. Other related works derive from conformance checking [10] and process repair [1] techniques. Both fields aim at understanding whether executions can be replayed on top of an existing processes model. However, in our case, we wanted to separate the identification of the processes (i.e., control-flow discovery) from the calculation of their similarity (i.e., the model-to-model metric) so that these two contributions can be used independently from each other. Conformance checking and process repair, on the other hand, embed the evaluation and the improvement into one "activity".

Online Discovery for Declarative Models. In [7] a framework for the discovery of Declare models from streams was introduced as a way to deal with large collections of datasets that are impossible to store and process altogether. In [20] this work was generalized to handle the mining of data constraints, leveraging the MP-Declare notation [9].

Streaming Process Mining in General. In his PhD thesis [29], van Zelst 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 capture the knowledge observed in the stream before creating the final process model. In [5] the author presents a taxonomy for the classification of streaming process mining techniques. Our techniques constitute a hybrid approach in the categories in [5], 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 the 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 to capture multi-perspective constraints such as time and data [15,26], in this paper we use the classical, set-based formulation first presented in [14] that contains only four most basic behavioural relations: conditions, responses, inclusions and exclusions.

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 $e = (c, a) \in \mathcal{E}$. Given a set $\mathbb{N}_n^+ = 1, 2, \ldots, n$ and a target set A, a sequence $\sigma : \mathbb{N}_n^+ \mapsto A$ 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 $\sigma(i) = a_i \in A$.*

We can now formally characterize an event stream:

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

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

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* α_1 *for node* v *we use the notation* $X(v, \alpha_1)$.

We use the following attributes: avgFO: average index of the first appearance of an activity in a trace; noTraceApp: current number of traces containing the activity; avgIdx: average index of the activity in a trace; and noOccur: number of activity occurrences.

Definition 5 (DCR Graph). *A* DCR graph *is a tuple* $\langle \mathcal{A}, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow+, \rightarrow\%\rangle$*, where* $\mathcal{A}$ *is a set of activities,* $M \subseteq \mathcal{P}(\mathcal{A}) \times \mathcal{P}(\mathcal{A}) \times \mathcal{P}(\mathcal{A})$ *is a* marking*, and* $\phi \subseteq \mathcal{A} \times \mathcal{A}$ *for* $\phi \in \{\rightarrow\bullet, \bullet\rightarrow, \rightarrow+, \rightarrow\%\}$ *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)[3] and the *included* activities (In). DCR relations define what is the effect of executing one activity in the graph. 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. 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[4] P with activities $\mathcal{A}$ and marking $M = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ we write $P_{\bullet\rightarrow}$ for the set of pairs $\{(x, y) \mid x \in \mathcal{A} \wedge y \in \mathcal{A} \wedge (x, y) \in \bullet\rightarrow\}$ (similarly for any of the relations in ϕ) and we write $P_{\mathcal{A}}$ for the set of activities. Definition 5 omits the existence of a set of labels and labelling function present in [14]. This has a consequence in the set of observable traces: Assume a graph $G = \langle \{a, b\}, (\emptyset, \{a, b\}, \{a, b\}), \emptyset, \emptyset, \emptyset, \emptyset \rangle$ as well as a set of labels $L = \{p\}$ and a labelling function $l = \{(a, p), (b, p)\}$. A possible run of G has the shape $\sigma = \langle p, p \rangle$, which can be generated from 1) two executions of a, 2) two executions of b or 3) an interleaved execution of a and b. By removing the labels from the events (or alternatively, assuming an injective surjective labelling function in [14]), we assume that two occurrences of the event in the stream imply event repetition.

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. Definition 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

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

[4] We will use "DCR graph" and "DCR model" interchangeably in this paper.

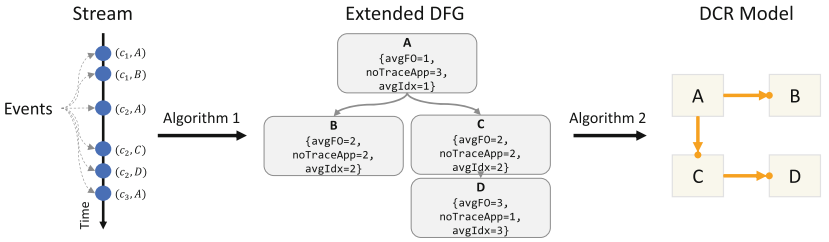


Fig. 2. Conceptual representation of the discovery strategy in this paper.

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; $\langle T, \leq \rangle$: Pattern poset

1 Initialize map obs ▷ Maps case ids to the sequence of activities
2 Initialize map deps ▷ Maps case ids to one activity name
3 Initialize extended DFG $G_X = (V, R, X)$
4 **forever do**
▷ Step 0: Observe new activity a for case c
5 $(c, a) \leftarrow observe(S)$
▷ Step 1: Update of the extended DFG
6 **if** $c \in$ *obs* **then**
7 Refresh the update time of c
8 **if** $|obs(c)| \geq m_e$ **then**
9 Remove oldest (i.e., earliest update time) event from list $obs(c)$
10 Update V and X of G_X to be consistent with the event just removed
11 **else**
12 **if** $|obs| \geq m_t$ **then**
13 Remove the oldest (i.e., earliest update time) trace from obs and all its events
14 Update V and X of G_X to be consistent with the events just removed
15 obs$(c) \leftarrow \langle\rangle$ ▷ Create empty list for obs(c)
16 obs$(c) \leftarrow$ obs$(c) \cdot \langle a \rangle$ ▷ Append a to obs(c)
17 $V \leftarrow V \cup \{a\}$
18 Update frequency and avg appearance index in X component of G_X ▷ The average appearance index is updated considering the new position given by $|obs(c)|$
19 **if** $c \in$ *deps* **then**
20 $R \leftarrow R \cup \{(\text{deps}(c), a)\}$
21 deps$(c) \leftarrow a$
22 **if** *trigger periodic cleanup* **then** ▷ Periodic cleanup of deps
23 Remove the oldest cases from deps
▷ Step 2: Periodic update of the DCR model (enough time/new behaviour)
24 **if** *trigger periodic update of the model* **then**
25 $M \leftarrow$ mine$(\langle T, \leq \rangle, G_X)$ ▷ See Algorithm 2
26 Notify about new model M

into two phases. The former (Algorithm 1) is in charge of extracting the extended DFG, the latter (Algorithms. 2, 3, 4) focuses on the extraction of DCR rules from the extended DFG.

Algorithm 1 takes as input a stream of events S, two parameters referring to the maximum number of traces m_t and events to store m_e and a set of DCR patterns to mine. The algorithm starts by initializing two supporting map data structures obs and deps as well as an empty extended DCR graph G_X (lines 1–3). obs is a map associating

Algorithm 2: Mining of rules starting from the extended DFG

Input: $\langle T, \leq \rangle$: Pattern poset, $G_X = (V, R, X)$: extended DFG

1 $P \leftarrow \langle V, M_{init}, \rightarrow\bullet = \emptyset, \bullet\rightarrow = \emptyset, \rightarrow+ = \emptyset, \rightarrow\% = \emptyset \rangle$ ▷ Initial DCR graph
2 $Rels, CompRels \leftarrow \emptyset, \emptyset$
3 **foreach** $t \in MinimalElements(\langle T, \leq \rangle)$ **do** ▷ Baseline for atomic patterns
4 $\quad Rels \leftarrow Rels \cup MineAtomic(G_X, t)$
5 **foreach** $t \in T \backslash MinimalElements(\langle T \leq \rangle)$ **do** ▷ Composite case
6 $\quad CompRels \leftarrow CompRels \cup MineComposite(G_X, t, Rels)$
7 **if** $CompRels \neq \emptyset$ **then**
8 $\quad P \leftarrow P \oplus CompRels$
9 **else**
10 $\quad P \leftarrow P \oplus Rels$
11 **return** $RemoveRedundancies(P)$ ▷ Apply transitive reduction

case ids to sequences of partial traces; deps is a map associating case ids to activity names. After initialization, the algorithm starts consuming the actual events in a never-ending loop (line 4). The initial step consists of receiving a new event (line 5). Then, two major steps take place: the first step consists of updating the extended DFG; the second consists of transforming the extended DFG into a DCR model. To update the extended DFG the algorithm first updates the set of nodes and extra attributes. 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) and oldest cases (i.e., cases not likely to receive any further events) are removed from deps (line 23). Finally, the algorithm refreshes the DCR model by calling the procedure that transforms (lines 25-26) the extended DFG into a DCR model (cf. Algorithm 2). Updates can be triggered based on some periodicity (line 24) or based on the amount of behaviour seen. The mechanics of such periodicity are beyond the scope of the paper.

Algorithm 2 generates a DCR graph from an extended DFG. First, it (1) defines patterns that describe occurrences of atomic DCR constraints in the extended DFG, and then it (2) defines composite patterns that describe the most common behaviour. Given a set of relation patterns T, $\langle T, \leq \rangle$ denotes a pattern dependency poset with $\leq$ a partial order over T. Similarly $MinimalElements(\langle T, \leq \rangle) = \{x \in T \mid \nexists y \in T.y \leq x\}$. Patterns as posets allow us to reuse and simplify the outputs from the discovery algorithm. Consider a pattern describing a sequential composition from a to b (similar to a flow

Algorithm 3: Atomic miner

Input: $G_X = (V, R, X)$: extended DFG, u: DCR Pattern

```
1  Rels ← ∅ ▷ Empty dictionary of mined relations
2  foreach (s, t) ∈ R do
3      switch u ▷ Pattern match with each atomic pattern
4      do
5          case RESPONSE
6              if X(s, avgIdx) < X(t, avgIdx) then
7                  Rels[u] ← Rels[u] ∪ (s, t, •→)
8          case CONDITION
9              if X(s, avgFO) < X(t, avgFO) ∧ X(s, noTraceApp) ≥ X(t, noTraceApp) then
10                 Rels[u] ← Rels[u] ∪ (s, t, →•)
11         case SELFEXCLUDE
12             if X(s, noOccur) = 1 then
13                 Rels[u] ← Rels[u] ∪ (s, s, →%)
           ▷ Further patterns here...
14     return Rels
```

Algorithm 4: Composite miner

Input: $G_X = (V, R, X)$: extended DFG, u: DCR Pattern, $Rels$: Mined Relations

```
1  switch u do
2      case EXCLUDEINCLUDE
3          return Rels[SELFEXCLUDE] ∪ Rels[PRECEDENCE] ∪ Rels[NOTCHAINSUCCESION] ▷ Removes
             redundant relations
4      ▷ Further patterns here
```

in BPMN). A DCR model that captures a sequential behaviour will need 4 constraints: $\{a{\rightarrow}\bullet b, a{\bullet\!\!\rightarrow}b, a{\rightarrow}\%a, b{\rightarrow}\%b\}$. Consider $T = \{T_1 : \textit{Condition}, T_2 : \textit{Response}, T_3 : \textit{Exclusion}, T_4 : \textit{Sequence}\}$. The pattern poset $\langle T, \{(T_4, T_1), (T_4, T_2), (T_4, T_3)\}\rangle$ defines the dependency relations for a miner capable of mining sequential patterns. Additional patterns (e.g. exclusive choices, escalation patterns, etc.), can be modelled similarly. Pattern posets are finite, thus there exist minimal elements. The generation of a DCR model from an extended DFG is described in Algorithm 2. We illustrate the mining of DCR *conditions*, *responses* and *self-responses*, but more patterns are available in [25]. The algorithm takes as input an extended DFG G_X and a pattern poset. It starts by creating an empty DCR graph P with activities equal to the nodes in G_X and initial marking $M_{init} = \{\emptyset, \emptyset, V\}$, that is, all events are included, not pending and not executed. We then split the processing between atomic patterns (those with no dependencies) and composite patterns. The map *Rel* stores the relations from atomic patterns, that will be used for the composite miner. We use the merge notation $P \oplus \textit{Rels}$ to denote the result of the creation of a DCR graph whose activities and markings are the same as P, and whose relations are the pairwise union of the range of *Rels* and its corresponding relational structure in P. Line 11 applies a transitive reduction strategy [4], reducing the number of relations while maintaining identical reachability properties.

The atomic and composite miners are described in Algorithms 3, and 4. The atomic miner in Algorithm 3 iterates over all node dependencies in the DFG and the pattern matches with the existing set of implemented patterns. Take the case of a response

constraint. We will identify it if the average occurrence of s is before t (line 6). This condition, together with the dependency between s and t in G_X is sufficient to infer a response constraint from s to t. To detect conditions, the algorithm verifies another set of properties: given a dependency between s and t, it checks that the first occurrence of s precedes t and that s and t appeared in the same traces (approximated by counting the number of traces containing both activities, line 9). The composite miner in Algorithm 4 receives the DFG, a pattern, and the list of mined relations from atomic patterns. We provide an example for the case of include and exclude relations. This pattern is built as a combination of self-exclusions, precedence, and not chain successions. As these atomic patterns generate each set of include/exclude relations, the pattern just takes the set union construction.

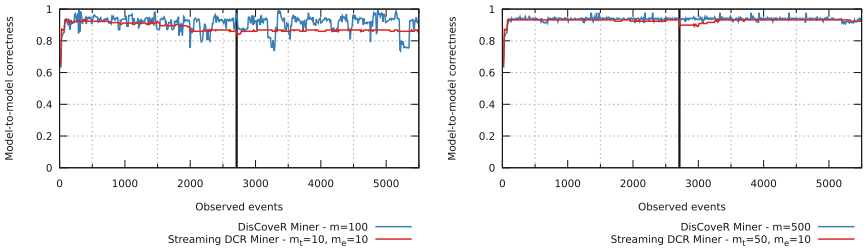
Suitability of the Algorithms for Streaming Settings. Whenever discussing algorithms that can tackle the streaming process mining problem [5], it is important to keep in mind that while a stream is assumed to be infinite, only a finite amount of memory can be used to store all information and that the time complexity for processing each event must be constant. Concerning the memory, an upper bound on the number of stored events in Algorithm 1 is given by $m_t \cdot m_e$ where m_e is the number of unique events and m_t is the number of parallel traces. Moreover, note that the extended DFG is also finite since there is a node for each activity contained in the memory. Concerning the time complexity, Algorithm 1 does not perform any unbounded backtracking. 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 Algorithm 2 as it iterates on the extended DFG which has a size bounded by the provided parameters (and hence, can be considered constant).

5 Experimental Evaluation

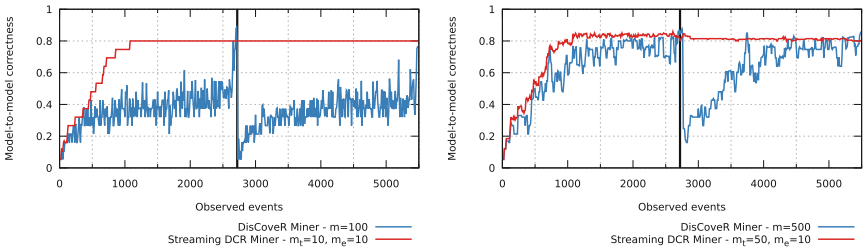
To validate our approach we executed several tests, first to validate quantitatively the streaming discovery on synthetic data, then to qualitatively evaluate the whole approach on a real dataset. Due to lack of space, we only report quantitative tests, while performance and the qualitative evaluation can be found in a separate technical report [8].

5.1 Quantitative Evaluation of Streaming Discovery

Recall from the previous section that time/space complexity are constant for streaming settings. Thus, our analysis will focus on studying how the algorithm behaves when encountering sudden changes in a stream. We compare with other process discovery algorithms for DCR graphs, in this case, the DisCoveR miner [4]. The tests are performed against a publicly available dataset of events streams [11]. This dataset includes (1) a synthetic stream inspired by a loan application process, and (2) perturbations to the original stream using change patterns [28]. Recall that the DisCoveR miner is an *offline* miner, thus it assumes an infinite memory model. To provide a fair evaluation we need to parameterize 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 a



(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 equal storage available (capacity of up to 100 and 500 events).

time) and a complex stream where observations from many instances arrive intertwined. As no initial DCR graph exists for this process, and no streaming DCR miner exists, we used the DisCoveR miner in its original (offline) setting to generate a baseline graph using the entire dataset. This model (the one calculated with offline DisCoveR) 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 future work.

We introduce a metric that quantifies the similarity between two DCR graphs. It can be used, for example, to identify which process is being executed with respect to a model repository, or by quantifying the *change rate* of one 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. Definition 5) with a weight, plus one additional weight for the activities. Then it computes the weighted Jaccard similarity [17] 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 $W : \phi \cup \{\mathsf{act}\} \to \mathbb{R}$ 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 similarity metric compares the relations in each of the two DCR graphs, thus returning a value between 0 and 1, where 1 indicates a perfect match and 0 stands for no match at all. A brief evaluation of the metric is reported in Appendix A.

The results of the quantitative evaluation are reported in Fig. 3. Each figure shows the performance of the incremental version of DisCoveR and the streaming DCR miner against 2 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 more robust to the drift and more stable over time, proving its ability at managing the available memory in a more effective way.

5.2 Discussion

One of the limitations of the approach regards precision with respect to offline miners. A limiting aspect of our work is the choice of the intermediate structure. A DFG representation may report confusing model behaviour as it simplifies the observations using purely a frequency-based threshold [27]. 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 declarative paradigm as a second-class citizen with respect to declarative constraints. We believe that the choice of the DFG as an intermediate data structure carries out a loss of precision with respect to the DisCoveR miner in offline settings. However, in an online setting, the DFG still provides a valid approximation to observations of streams where we do not have complete traces. This is far from an abnormal situation: IoT communication protocols such as MQTT [16] assume that subscriber nodes might connect to the network *after* the communications have started, not being able to identify starting nodes. Specifically, in a streaming setting it is impossible to know exactly when a certain execution is complete and, especially in declarative settings, certain constraints describe liveness behaviours that can only be verified after a whole trace has been completely inspected. While watermarking techniques [3] could be employed to cope with *lateness* issues, we have decided to favour self-contained approaches in this paper, leaving for future work the exploration of watermarking techniques.

6 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 extent two DCR models are the same. An 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 in 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 seman-

tic aspects, such as mentioned in [18]. A possible limitation of the streaming miner algorithm approach followed here relates to the updating mechanism. Currently lines 22–24 of Algorithm 1 perform updates based entirely on periodic updates triggered by time, which will generate notifications even when no potential changes in the model have been identified. A possibility to extend the algorithm will be to integrate the model-to-model similarity as a parameter to the discovery algorithm, so models only get updated after a given change threshold (a similarity value specified by the user) is reached.

A Quantitative Evaluation of Model-to-Model Metric

To validate our metric we used a dataset of 28 DCR process models collected from previous mapping efforts [19]. For each model, we randomly introduced variations such as: adding new activities connected to the existing fragments, adding disconnected activities, deleting existing activities, adding and 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. Figure 4 shows each variation on a scatter plot where the x axis refers to the number of introduced variations 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: $W = \{(\rightarrow\bullet, 0.06), (\bullet\rightarrow, 0.07), (\rightarrow\diamond, 0.06), (\rightarrow+, 0.07), (\rightarrow\%, 0.13), (\mathsf{act}, 0.61)\}$ leading 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 a fast computation.

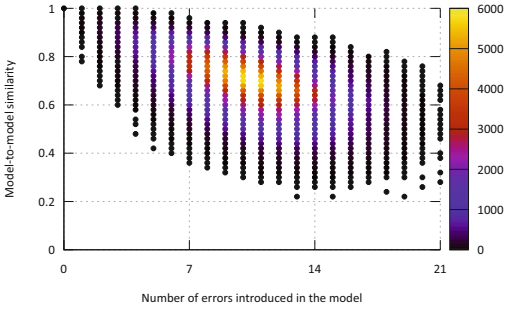


Fig. 4. Correlation between the model-to-model metric and the number of model changes. The colour indicates the density of observations. (Color figure online)

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: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011. LNBIP, vol. 99, pp. 141–146. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28108-2_13
3. Akidau, T., et al.: Watermarks in stream processing systems: semantics and comparative analysis of apache Fink and google cloud dataflow. VLDB (2021)
4. Back, C.O., Slaats, T., Hildebrandt, T.T., Marquard, M.: DisCoveR: accurate and efficient discovery of declarative process models. Int. J. Softw. Tools Technol. Transfer **24**, 563–587 (2021). https://doi.org/10.1007/s10009-021-00616-0
5. Burattin, A.: Streaming process discovery and conformance checking. In: Encyclopedia of Big Data Technologies. Springer, Cham (2019)
6. Burattin, A.: Streaming process mining with beamline. In: ICPM Demos (2022)
7. Burattin, A., Cimitile, M., Maggi, F.M., Sperduti, A.: Online discovery of declarative process models from event streams. IEEE Trans. Serv. Comput. **8**(6), 833–846 (2015)
8. Burattin, A., López, H.A., Starklit, L.: A monitoring and discovery approach for declarative processes based on streams (2022). https://doi.org/10.48550/arXiv.2208.05364
9. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016)
10. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking. Springer, Cham (2018)
11. Ceravolo, P., Tavares, G.M., Junior, S.B., Damiani, E.: Evaluation goals for online process mining: a concept drift perspective. IEEE Trans. Serv. Comput. **15**, 2473–2489 (2020)
12. De Giacomo, G., De Masellis, R., Montali, M.: Reasoning on LTL on finite traces: insensitivity to infiniteness. In: AAAI Conference on Artificial Intelligence (2014)
13. Debois, S., Hildebrandt, T.T., Slaats, T.: Replication, refinement & reachability: complexity in dynamic condition-response graphs. Acta Informatica **55**(6), 489–520 (2018). https://doi.org/10.1007/s00236-017-0303-8
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., Zanitti, F.: Contracts for cross-organizational workflows as timed dynamic condition response graphs. JLAMP **82**(5–7), 164–185 (2013)
16. Hunkeler, U., Truong, H.L., Stanford-Clark, A.: MQTT-S-a publish/subscribe protocol for wireless sensor networks. In: Proceedings of COMSWARE. IEEE (2008)
17. Jaccard, P.: The distribution of the flora of the alpine zone. New Phytol. **11**(2), 37–50 (1912)
18. López, H.A., Debois, S., Slaats, T., Hildebrandt, T.T.: Business process compliance using reference models of law. In: FASE 2020. LNCS, vol. 12076, pp. 378–399. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45234-6_19
19. López, H.A., Strømsted, R., Niyodusenga, J.-M., Marquard, M.: Declarative process discovery: linking process and textual views. In: Nurcan, S., Korthaus, A. (eds.) CAiSE 2021. LNBIP, vol. 424, pp. 109–117. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79108-7_13
20. 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: IJCNN (2020)
21. Nekrasaite, V., Parli, A.T., Back, C.O., Slaats, T.: Discovering responsibilities with dynamic condition response graphs. In: Giorgini, P., Weber, B. (eds.) CAiSE 2019. LNCS, vol. 11483, pp. 595–610. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21290-2_37

22. Norgaard, L.H., et al.: Declarative process models in government centric case and document management. In: BPM (Industry Track). CEUR, vol. 1985, pp. 38–51. CEUR-WS.org (2017)
23. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) BPM 2006. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006). https://doi.org/10.1007/11837862_18
24. Slaats, T., Debois, S., Back, C.O.: Weighing the pros and cons: process discovery with negative examples. In: Polyvyanyy, A., Wynn, M.T., Van Looy, A., Reichert, M. (eds.) BPM 2021. LNCS, vol. 12875, pp. 47–64. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85469-0_6
25. Starklit, L.: Online Discovery and Comparison of DCR models from Event Streams using Beamline. Master's thesis, DTU (2021)
26. Strømsted, R., López, H.A., Debois, S., Marquard, M.: Dynamic evaluation forms using declarative modeling. In: BPM (Demos/Industry), pp. 172–179 (2018)
27. van der Aalst, W.M.: A practitioner's guide to process mining: limitations of the directly-follows graph. Procedia Comput. Sci. **164**, 321–328 (2019)
28. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features-enhancing flexibility in process-aware information systems. Data Knowl. Eng. **66**(3), 438–466 (2008)
29. van Zelst, S.J.: Process mining with streaming data. Ph.D. thesis, Technische Universiteit Eindhoven (2019)