Towards Online Discovery of Data-Aware Declarative Process Models from Event Streams

Nicolò Navarin, Member IEEE
Department of Mathematics "Tullio Levi-Civita"
University of Padua
Padova, Italy
nnavarin@math.unipd.it

Matteo Cambiaso
DIBRIS
University of Genoa
Genova, Italy
cambiaso.matteo@gmail.com

Andrea Burattin, Member IEEE
Software and Process Engineering
Technical University of Denmark
Lyngby, Denmark
andbur@dtu.dk

Fabrizio M. Maggi
Faculty of Computer Science
Free University of Bozen-Bolzano
Bolzano, Italy
maggi@inf.unibz.it

Luca Oneto, Member IEEE
DIBRIS
University of Genoa
Genova, Italy
luca.oneto@unige.it

Alessandro Sperduti, Senior Member IEEE
Department of Mathematics "Tullio Levi-Civita"
University of Padua
Padova, Italy
sperduti@math.unipd.it

***Abstract*—In recent years, several techniques have been made available to automatically discover declarative process models from event logs. These techniques are useful to provide a comprehensible picture of the process as opposed to full specifications of process behavior provided by procedural modeling languages. Since many modern systems produce "big data" from business process executions, in previous work, a framework for the discovery of LTL-based declarative process 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. However, the proposed framework does not take into account data attributes associated with events in the log, which can otherwise provide valuable insights into the rules that govern the process. This paper makes the first proposal to close this gap by presenting a technique for discovering declarative process models from event streams that incorporates both control-flow dependencies and data conditions. Specifically, we use Hoeffding trees to incrementally discover data-aware declarative process models, which are represented as conjunctions of first-order temporal logic expressions. The proposed technique has been validated on a synthetic event log, and on a real-life log of a cancer treatment process.**

***Index Terms*—online process discovery, data-aware process model, declarative process models**

I. Introduction

Nowadays, in most companies, information systems generate high volumes of data every day, which is eventually stored in event logs. Process mining techniques [1] allow for the extraction of relevant business information from these logs for supporting different types of process analysis such as process discovery, conformance checking, bottleneck detection, or performance monitoring.

However, analyzing event logs offline might become, in some cases, very challenging due to the high volume and velocity of the data generated. To address this problem, event streams coming from process executions can be processed online as they occur.

Therefore, dealing with streaming data is recognized as a relevant challenge for process mining in the Process Mining Manifesto [2]. Our contribution focuses on the main branch of techniques of the process mining family, i.e., *process discovery*. Standard process discovery algorithms are functions mapping an event log to a process model such that the model is representative of the behavior seen in the event log [1]. In addition, we discover declarative process models. A declarative model consists of a set of business rules, i.e., constraints defining the boundaries of the allowed behaviors of the process. In such a paradigm, an open-world assumption is typically employed: everything that is not constrained by the model is allowed. These models can be very effective to express process behaviors characterized by high variability and multiple alternatives [3]. Indeed, in these scenarios, the open-world assumption allows process models to represent all possible process behaviors in a compact way.

In [4], [5], an approach to automatically discover declarative processes models from streams of data is presented. This approach, however, does not take into account data attributes attached to events. Hence, the resulting models lack insights into the role of data in the execution of the process. However, the role that data plays in business processes, particularly the ones involving several alternative paths, is crucial as it is often data that drives the decisions that participants make. In these processes, the fact that tasks are executed in a certain order often tells us little about the future behavior of the process. It is only when considering the data produced by the tasks that we can state that something must or must not happen later.

This paper starts to address the above gap by presenting a technique for the online discovery of data-aware declarative process models, represented using an extension of the *Declare* notation [6]. Declare is a declarative process modeling language that combines a formal semantics grounded in Linear Temporal Logic on finite traces (LTL) with a graphical representation. Declare itself is not designed to capture data

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

aspects of a process. Therefore, we use an extension of the language with the ability to define data conditions presented in [7]. The extended (data-aware) Declare notation is defined in terms of LTL-FO (First-Order LTL) rules, each one capturing an association between a task, a condition and another task. An example of such rules is that if a task is executed and a certain data condition holds on the data produced by the task after its execution, some other task must eventually be performed afterward.

The proposed approach relies on the notion of *constraint activation* [8]. For example, for the constraint "every request is eventually acknowledged" each request is an activation. This activation becomes a fulfillment or a violation depending on whether the request is followed by an acknowledgement or not. In our approach, we apply a set of algorithms to analyze streaming data and classify activations (with their data snapshots) into fulfillments and violations. Given the resulting classification problem, we use Hoeffding trees to incrementally identify the data conditions that should hold for a constraint activation to be fulfilled. The approximate frequency counting algorithm *Lossy Counting* is integrated into the discovery algorithms so that outdated information can be "forgotten" thus keeping the data structures needed for the discovery compact.

The remainder of this paper is structured as follows: Section II introduces the background definitions needed through the rest of the text. Section III reports the formal problem we solve in this paper. Section IV presents the solution we propose and Section V reports the experimental sessions we conducted to validate it. Section VI describes the state of the art of the approaches tackling similar problems. Finally, Section VII concludes the paper and sketches some possible future work and research directions.

TABLE I
LTL-FO SEMANTICS AND GRAPHICAL REPRESENTATION FOR SOME DECLARE CONSTRAINTS EXTENDED WITH DATA CONDITIONS.

Template	Description	Formalization	Notation
responded existence (A,B,Cond)	if A occurs and Cond holds, B must occur before or after A	$\Diamond(A \wedge Cond) \rightarrow \Diamond B$	A $Cond$ B
response (A,B,Cond)	if A occurs and Cond holds, B must occur afterwards	$\Box((A \wedge Cond) \rightarrow \Diamond B)$	A $Cond$ B
precedence (A,B,Cond)	if B occurs and Cond holds, A must have occurred before	$(\neg(B \wedge Cond) \sqcup A) \vee \Box(\neg(B \wedge Cond))$	A $Cond$ B
alternate response (A,B,Cond)	if A occurs and Cond holds, B must occur afterwards, without further As in between	$\Box((A \wedge Cond) \rightarrow \bigcirc(\neg A \sqcup B))$	A $Cond$ B
alternate precedence (A,B,Cond)	if B occurs and Cond holds, A must have occurred before, without other Bs in between	$((\neg(B \wedge Cond) \sqcup A) \vee \Box(\neg(B \wedge Cond))) \wedge\Box((B \wedge Cond) \rightarrow \bigcirc(\neg B \sqcup A))$	A $Cond$ B
chain response (A,B,Cond)	if A occurs and Cond holds, B must occur next	$\Box((A \wedge Cond) \rightarrow \bigcirc B)$	A $Cond$ B
chain precedence (A,B,Cond)	if B occurs and Cond holds, A must have occurred immediately before	$\Box(\bigcirc(B \wedge Cond) \rightarrow A)$	A $Cond$ B
not resp. existence(A,B,Cond)	if A occurs and Cond holds, B can never occur	$\Diamond(A \wedge Cond) \rightarrow \neg\Diamond B$	A $Cond$ B
not response (A,B,Cond)	if A occurs and Cond holds, B cannot occur afterwards	$\Box((A \wedge Cond) \rightarrow \neg\Diamond B)$	A $Cond$ B
not precedence (A,B,Cond)	if B occurs and Cond holds, A cannot have occurred before	$\Box(A \rightarrow \neg\Diamond(B \wedge Cond))$	A $Cond$ B
not chain response (A,B,Cond)	if A occurs and Cond holds, B cannot be executed next	$\Box((A \wedge Cond) \rightarrow \neg \bigcirc B)$	A $Cond$ B
not chain precedence (A,B,Cond)	if B occurs and Cond holds, A cannot have occurred immediately before	$\Box(\bigcirc(B \wedge Cond) \rightarrow \neg A)$	A $Cond$ B

II. BACKGROUND

This section contains the preliminary notions required to properly understand the rest of the paper.

A. Declare

Declare is a declarative process modeling language first introduced by Pesic and van der Aalst in [9]. A Declare model is a set of constraints that must hold in conjunction during the process execution. A constraint is an instantiation of a parameterized constraint type (a.k.a. template) on a set of real activities. Declare constraints are equipped with a graphical notation and an LTL semantics. An example of Declare constraint is $response(A, B)$ (formally: $\Box(A \rightarrow \Diamond B)$). We refer the reader to [10] for a complete overview of the language. Constraint $response(A, B)$ indicates that if A occurs, B must eventually follow. Therefore, this constraint is satisfied for traces such as $t_1 = \langle A, A, B, C\rangle$, $t_2 = \langle B, B, C, D\rangle$ and $t_3 = \langle A, B, C, B\rangle$, but not for $t_4 = \langle A, B, A, C\rangle$ because, in this case, the second A is not followed by a B. Note that, in t_2, $response(A, B)$ is satisfied in a trivial way because A never occurs. In this case, we say that the constraint is *vacuously satisfied* [11]. In [8], the authors introduce the notion of *behavioral vacuity detection* according to which a constraint is non-vacuously satisfied in a trace when it is activated in that trace. A *constraint activation* in a trace is an event whose occurrence imposes, because of that constraint, some obligations on other events in the same trace. For example, A is an activation for $response(A, B)$ because the execution of A forces B to be executed eventually.

A constraint activation can be classified as a *fulfillment* or a *violation*. When a trace is perfectly compliant with respect to a constraint, every constraint activation in the trace leads to a fulfillment. Consider, again, constraint $response(A, B)$. In trace t_1, the constraint is activated and fulfilled twice, whereas, in trace t_3, the same constraint is activated and fulfilled only once. On the other hand, when a trace is not compliant with respect to a constraint, a constraint activation in the trace can lead to a fulfillment but also to a violation (and at least one activation leads to a violation). In trace t_4, for example, $response(A, B)$ is activated twice, but the first activation leads to a fulfillment (eventually B occurs) and the second activation leads to a violation (the target B does not occur eventually). In [8], the authors define two metrics to measure the conformance of an event log with respect to a constraint in terms of violations and fulfillments, called *violation ratio* and *fulfillment ratio* of the constraint in the log, defined as the percentage of violations and fulfillments of the constraint over the total number of activations.

In [7], the authors define a semantics to extend the standard Declare constraints with data conditions. To do this, they

use First-Order Linear Temporal Logic (LTL-FO), which is the first-order extension of propositional LTL. While many reasoning tasks are clearly undecidable for LTL-FO, this logic is appropriate to unambiguously describe the semantics of the data-aware Declare constraints we can generate by using our algorithms. The defined semantics (shown in TABLE I) is quite straightforward. In particular, the original LTL semantics of a Declare constraint is extended by requiring an additional condition on data, $Cond$, to hold when the constraint is activated. $Cond$ is a closed first-order formula with the following structure: $\exists x_1, \ldots, x_n. curState(x_1, \ldots, x_n) \wedge \Phi(x_1, \ldots, x_n)$, where $curState/n$ is a relation storing the current values of the n data attributes available in the system (modified by events) and Φ/n is a first-order formula constraining such data by means of conjunctions, disjunctions and relational operators. For example, $response(A, B, Cond)$ specifies that whenever A occurs and condition $Cond$ holds true after the occurrence of A, then a corresponding occurrence of B is expected to eventually happen. Constraint $precedence(A, B, Cond)$ indicates that whenever B occurs and $Cond$, then an occurrence of A must have been executed beforehand. The semantics for negative relations is also very intuitive. For example, $not\ responded\ existence(A, B, Cond)$ indicates that if an instance of A occurs and $Cond$ holds, then no occurrence of B can happen before or after A. Based on this semantics, the notion of constraint activation changes. Activations of data-aware Declare constraints are all those constraint activations (according to the standard definition) for which $Cond$ is true. For example, $response(A, B, Cond)$ is activated when A occurs and, also, $Cond$ is valid. On the other hand, $precedence(A, B, Cond)$ is activated when B occurs and $Cond$ is valid. The definitions of fulfillments and violations are also adapted accordingly.

B. Event Streams

In the data mining literature, there are few definitions of an event stream. In this work, we consider an event stream as an unbounded, sequence of data items, observed at a high speed [12], [13]. Stream mining approaches can be divided into two main categories: *data-based* and *task-based* [14]. Data-based mining algorithms reduce the stream into finite datasets, which are supposed to be representatives of the complete stream; task-based algorithms are modified (or new) approaches, specifically designed for streams, in order to minimize time and space complexity. The algorithms presented here fall into this latter category.

As typically reported in the literature, we assume that: *(i)* the data that constitutes the stream (i.e., the *events*) have a small and fixed amount of attributes; *(ii)* a mining approach should be able to analyze an infinite amount of data; *(iii)* a mining approach should use a finite amount of memory that is considerably smaller with respect to the data observed in a reasonable span of time; *(iv)* there is an upper bound on the time allowed to analyze an event, typically the mining approach is required to linearly scale with the number of processed items (e.g., the algorithm works with one pass of the data [15]); *(v)* the "concepts" generating the event stream may be *stationary* or *evolving* [16], [17].

We assume that each event in an event stream is associated with an activity (i.e., a well-defined step in the process), is related to a particular case (i.e., a process instance), is executed at a certain point in time specified through a timestamp and is associated with a set of attribute-value pairs. In particular, an event ev is a 5-tuple: $ev = (C, c, t, A, te)$ where: C is the name of the activity that ev is referring to (i.e., which task was executed); c is the "case-id": an identifier that groups event referring to the same process instance; t is a timestamp, which indicates the execution time of the specific action; A is a set of attribute-value pairs associated with the execution of event ev. te is a boolean variable that indicates if an event is the last one of its trace. We use a projection operator π to extract specific elements of an event. Specifically, given an event $ev = (C, c, t, A)$, we define $\pi_C(ev) = C$, $\pi_c(ev) = c$, $\pi_t(ev) = t$, $\pi_A(ev) = A$, and $\pi_{te}(ev) = te$. It is reasonable to assume that events in the stream comply with the time order of events, i.e., $\forall i \in \mathbb{N}^+, ev_i, ev_{i+1} \in S$ we assume that $\pi_t(ev_i) \leq \pi_t(ev_{i+1})$.

C. Lossy Counting

Lossy Counting [18] is an algorithm for computing the approximated frequency counts of events in a data stream, with guarantees on the approximation error. The idea behind this approach is to conceptually divide the stream into *buckets* of width $w = \lceil \frac{1}{\epsilon} \rceil$, where $\epsilon \in (0, 1)$ is an *error parameter*. The *current* bucket (i.e., the bucket of the latest seen event) is identified as $b_{curr} = \lceil \frac{N}{w} \rceil$, where N is a progressive event counter.

The basic data structure that Lossy Counting requires is a set T of entries of the form (e, f, Δ) where:

- e is an event of the stream;
- f is the estimated frequency of event e; and
- Δ is the maximum number of times e can occur.

Every time a new event e is observed, the algorithm verifies if the data structure already contains an entry for it. If such an entry exists, then its frequency is incremented by one, otherwise a new tuple $(e, 1, b_{curr} - 1)$ is added. In this latter case, the new tuple has a frequency value set to 1. Every time $N \bmod w = 0$ (i.e., every w events), the algorithm cleans up the data structure by removing entries with maximal approximate frequency (i.e., the sum of frequency and maximum number of occurrences) less than the current bucket id, i.e., the algorithm removes the entries that satisfy the inequality $f + \Delta \leq b_{curr}$.

In this paper, we will exploit the principles behind the Lossy Counting algorithm in order to count the events in an event stream, and also to keep the most frequent rules that are mined from the stream. Note that Lossy Counting does not give guarantees on the maximum memory occupation (even if in real cases this is relatively small). An implementation of our algorithm adopting the Lossy Counting version with fixed budget [19] will be future work.

D. Hoeffding Trees

The Hoeffding tree [20] algorithm introduces an efficient method for incremental decision tree generation from a data stream. Hoeffding trees generate the decision tree incrementally. The main differences with respect to standard decision tree learning algorithms (e.g., C4.5) is that learning in Hoeffding tree is constant time per example (instance) making it suitable for mining data streaming. Moreover, the resulting trees are nearly identical with trees built by conventional batch learners, if enough training examples are available. Hoeffding trees employ the Hoeffding (or additive Chernoff [21]) bound to select the attribute for the split. The Hoeffding bound states that, with probability $1-\delta$, the true mean of a random variable of range R (the difference between its largest and smallest values), will not differ from the estimated mean after n independent observations, by more than $\epsilon = \sqrt{\frac{R^2 \ln(\frac{1}{\delta})}{2n}}$. The Hoeffding bound is used to estimate the **information gain** for each attribute. Then, the split is performed on the attribute with higher information gain. In order to compute the bound, it is not necessary to store all the examples seen so far, but just some statistics over their attributes.

III. Problem Statement

In this paper, we address the problem of extracting data-aware declarative rules [7] in the setting where events come from a data stream [22], i.e., there exists an event source which emits a possibly infinite sequence of events at a certain (possibly high) rate.

This setting is more constrained with respect to a batch one since: *(i)* the processing has to be very fast because, if an event arrives and the system is still processing the last one, the new event cannot be memorized and it is lost; for this reason, typically, a constant time complexity should be allowed for processing each event; *(ii)* the memory occupation has to be bounded, even in the case of an infinite stream, otherwise the physical available memory at a certain time would fill up; *(iii)* since the stream is supposed to be infinite, sooner or later concept drifts will occur, i.e., the model generating the logs may change (slowly or suddenly) over time.

Each event in the data stream belongs to a process instance, namely a trace. Each trace, in turn, represents the sequence of events characterizing a specific execution of the process. Additionally, we need to consider data attributes associated with such a process instance. For this purpose, we consider data attributes to be *attached* to the trace under examination and their values to be manipulated by the corresponding events. In particular, we follow the classical common-sense law of *inertia*: given a data attribute, its value remains constant until it is explicitly overridden by an event which provides a new value for it. Once a trace is over, the corresponding attributes are assumed to be removed. In this work, we assume to have knowledge regarding the termination of a trace. Although in many contexts this is a reasonable assumption, we plan to drop it in a future extension of this work.

IV. Proposed Solution

In this section, we detail how we tackled the problem presented in Section III. The intuition is to maintain a sketch, in the form of a Lossy Counting, of the *Declare* rules extracted from the stream so far. For each of these rules, following [7], we associate a Hoeffding tree structure, which is fed with the events (and the data attributes associated with them) that satisfy the Declare rule (*Fulfillment*) or violate it (*Violation*). The aim of the classifier is to learn conditions on the data attributes able to discriminate between fulfillments and violations.

From the learned tree, it is easy to extract data conditions: each path starting from the root and ending in a node labeled as *Fulfillment* is a condition on the data. All those paths can be merged with *or* clauses, obtaining a single condition over the considered *Declare* constraint.

To illustrate our idea, assume we have collected these three traces from the data stream:

$$t_1 = \langle (A,1,1,\{x{=}1,y{=}1\},0),(B,1,5,\{x{=}2,y{=}2\},0),(C,1,8,\{x{=}3,y{=}3\},1)\rangle;$$

$$t_2 = \langle (A,2,1,\{x{=}1,y{=}2\},0),(B,2,3,\{x{=}1,y{=}2\},0)\rangle;$$

$$t_3 = \langle (A,3,1,\{x{=}2,y{=}1\},0),(C,3,7,\{x{=}2,y{=}4\},0)\rangle.$$

These traces report three activities: A, B and C. All possible pairs of these activities need to be considered to create a list of candidate constraints (that can be discovered) using each template. For example, considering the response template, the candidates are:

$A \bullet\!\!\longrightarrow B$, $A \bullet\!\!\longrightarrow C$, $B \bullet\!\!\longrightarrow A$, $B \bullet\!\!\longrightarrow C$, $C \bullet\!\!\longrightarrow A$, $C \bullet\!\!\longrightarrow B$.

Starting from this list, only the "frequent" constraints are kept. In order to find the frequent constraints, we count the number of activations, which, in the case of response, corresponds to the number of occurrences of the source activity [7]. Note that, differently from [4], [5], we propose to adopt Lossy Counting also as an approximated frequency counter on the frequency of the extracted constraints. In particular, a constraint is added to the model when we see its first activation. Then, the model follows the standard Lossy Counting rule, i.e., every $\frac{1}{\epsilon}$ insertions, rare constraints (i.e., constraints with a low number of activations) are removed from the model. Note that, whereas in [7] a user-defined support threshold was needed in order to filter out irrelevant constraints, in this paper, we adopt Lossy Counting as an implicit thresholder, i.e., only those constraints that are stored in the Lossy Counting structure are considered.

Algorithm 1 reports the pseudo-code of the main procedure of our stream discovery algorithm. We apply, when necessary, relational algebra to the sets and, in particular, the *projection* operator π (already introduced in Section II-B) and the *selection* operator σ that selects from a set of tuples the ones that satisfy a selection condition.

In this algorithm, $\mathcal{D}_E = \{(c, ev, f, \Delta)\}$ is a set of events (ev) grouped by trace (c), where f is the observed frequency

Algorithm 1: Main discovery procedure

Data: ϵ: Lossy Counting approximation parameter
$stream$: a stream of events.
Result: $\mathcal{D}_M$: set of discovered data-aware constraints.

1 Initialize set $\mathcal{D}_E = \{(c, ev, f, \Delta)\}$;
2 Initialize set $\mathcal{S} = \{(a, v)\}$;
3 Initialize set $\mathcal{D}_M = \{(r, C_1, C_2, HT, f, \Delta)\}$;
4 $N \leftarrow 1$, $N_r \leftarrow 1$, $w \leftarrow \lceil \frac{1}{\epsilon} \rceil$;
5 **for** e ***in*** *stream* **do**
6     $b_{curr} = \lceil \frac{N}{w} \rceil$, $\mathcal{S} \leftarrow \mathcal{S} \cup \pi_A(e)$;
7   **if** $(\pi_c(e), e) \notin \pi_{c,ev}(\mathcal{D}_E)$ **then**
8         $\mathcal{D}_E \leftarrow \mathcal{D}_E \cup \{(\pi_c(e), e, 1, b_{curr} - 1)\}$;
9   **else**
10         $f' \leftarrow \pi_f \sigma_{c=\pi_c(e), ev=e}(\mathcal{D}_E)$;
11         $\Delta' \leftarrow \pi_\Delta \sigma_{c=\pi_c(e), ev=e}(\mathcal{D}_E)$;
12         $\mathcal{D}_E \leftarrow (\mathcal{D}_E \setminus \{(\pi_c(e), e, f', \Delta')\}) \cup \{(\pi_c(e), e, f' + 1, \Delta')\}$;
13     $\mathcal{D}_F$= Discover($\mathcal{D}_M$, $\sigma_{c=\pi_c(e)}(\mathcal{D}_E)$, e, $\mathcal{S}$);
14   **forall** $(r', C_1', C_2', \mathcal{S}') \in \mathcal{D}_F$ **do**
15         $b_r = \lceil \frac{N_r}{w} \rceil$
16         $f' \leftarrow \max(0, \pi_f(\sigma_{r=r', C_1=C_1', C_2=C_2'}(\mathcal{D}_M)))$;
17         $\Delta' \leftarrow \min(b_r, \pi_\Delta(\sigma_{r=r', C_1=C_1', C_2=C_2'}(\mathcal{D}_M)))$;
18         $HT \leftarrow \pi_{HT}(\sigma_{r=r', C_1=C_1', C_2=C_2'}(\mathcal{D}_M))$;
19   HT.addObservation($\mathcal{S}$',"fulfillment");
20         $\mathcal{D}_M \leftarrow (\mathcal{D}_M \setminus \{(r', C_1', C_2', HT, f', \Delta')\}) \cup \{(r', C_1', C_2', HT, f' + 1, \Delta')\}$;
        /* Lossy Counting cleaning procedure */
21   **if** $N_r = 0 \mod w$ **then**
22   **forall** $(r, C_1, C_2, HT, f, \Delta) \in \mathcal{D}_M$ ***s.t.*** $f + \Delta \leq b_r$ **do**
23                 $\mathcal{D}_E \leftarrow \mathcal{D}_E \setminus \{(r, C_1, C_2, HT, f, \Delta)\}$
24         $N_r \leftarrow N_r + 1$;
25     $\mathcal{D}_V$=DiscoverViolations($\mathcal{D}_M$, $\sigma_{c=\pi_c(e)}(\mathcal{D}_E)$, e, $\mathcal{S}$);
26   **forall** $(r', C_1', C_2', \mathcal{S}') \in \mathcal{D}_V$ **do**
27         $b_r = \lceil \frac{N_r}{w} \rceil$
28         $f' \leftarrow \max(0, \pi_f(\sigma_{r=r', C_1=C_1', C_2=C_2'}(\mathcal{D}_M)))$;
29         $\Delta' \leftarrow \min(b_r, \pi_\Delta(\sigma_{r=r', C_1=C_1', C_2=C_2'}(\mathcal{D}_M)))$;
30         $HT \leftarrow \pi_{HT}(\sigma_{r=r', C_1=C_1', C_2=C_2'}(\mathcal{D}_M))$;
31   HT.addObservation($\mathcal{S}$',"violation");
32         $\mathcal{D}_M \leftarrow (\mathcal{D}_M \setminus \{(r', C_1', C_2', HT, f', \Delta')\}) \cup \{(r', C_1', C_2', HT, f' + 1, \Delta')\}$;
        /* Lossy Counting cleaning */
33   **if** $N_r = 0 \mod w$ **then**
34   **forall** $(r, C_1, C_2, HT, f, \Delta) \in \mathcal{D}_M$ ***s.t.*** $f + \Delta \leq b_r$ **do**
35                 $\mathcal{D}_E \leftarrow \mathcal{D}_E \setminus \{(r, C_1, C_2, HT, f, \Delta)\}$
36         $N_r \leftarrow N_r + 1$;
    /* Lossy Counting cleaning */
37   **if** $N = 0 \mod w$ **then**
38   **forall** $(a, f, \Delta) \in \mathcal{D}_E$ ***s.t.*** $f + \Delta \leq b_{curr}$ **do**
39             $\mathcal{D}_E \leftarrow \mathcal{D}_E \setminus \{(a, f, \Delta)\}$
40     $N \leftarrow N + 1$

Algorithm 2: Auxiliary functions (dispatchers)

1 **Function** *Discover*
    Data: $\mathcal{D}_M$: the model
    $\mathcal{D}_E$: the set of events in the trace $\pi_c(e)$
    e: an event
    $\mathcal{S}$: the system state.
    Result: $\mathcal{D}_F$: a set of discovered fulfillments.
2   Initialize data structure $\mathcal{D}_F = (r, C_1, C_2, \mathcal{S})$;
3   **forall** *RULE* $\in$ *implemented Declare discovery procedures* **do**
4         $fulfillments \leftarrow Discover_RULE(\mathcal{D}_M, \mathcal{D}_E, e, \mathcal{S})$;
5         $\mathcal{D}_F \leftarrow \mathcal{D}_F \cup fulfillments$;

6 **Function** *DiscoverViolations*
    Data: $\mathcal{D}_M$: the model
    $\mathcal{D}_E$: the set of events in the trace $\pi_c(e)$
    e: an event
    $\mathcal{S}$: the system state.
    Result: $\mathcal{D}_V$: set of discovered violations
7   Initialize data structure $\mathcal{D}_F = (r, C_1, C_2, \mathcal{S})$;
8   **forall** *RULE* $\in$ *Declare discovery procedures* **do**
9         $violations \leftarrow DiscoverViolations_RULE(\mathcal{D}_M, \mathcal{D}_E, e, \mathcal{S})$;
10         $\mathcal{D}_V \leftarrow \mathcal{D}_V \cup violations$;

of the event and Δ its frequency bound (according to the Lossy Counting rule). $\mathcal{D}_M = \{(r, C_1, C_2, HT, f, \Delta)\}$ is the set of discovered constraints (the *model*), where r is a string representing the template (e.g., "response"), C_1 and C_2 are the first and second activity, HT is a Hoeffding tree structure (used for the generation of the data-aware constraints), f is the observed frequency of the constraint, and Δ its frequency bound (according to the Lossy Counting rule). $\mathcal{S} = \{(a, v)\}$ is the set of all the attributes seen so far in the stream (system state), where a is the identifier of an attribute and v its value. These sets implement the Lossy Counting policy, according to which the less frequent items are deleted if necessary (lines 21-23, 33-35, and 37-39 of Algorithm 1).

The logic of Algorithm 1 is the following. It analyzes the events from the stream, one at a time. It updates the data structures and variables (lines 1-12). Then, it calls the *Discover* procedure (see Algorithm 2), which is a dispatcher invoking all the implemented *Discover* procedures, one for each considered Declare template. This procedure is used to identify the fulfillments triggered by the processed event. The fulfillments are finally injected into the Hoeffding trees (lines 14-24). The same process is repeated for violations (lines 25-36).

A. Online discovery of Declare constraints

In the current version of our implementation, we are only able to discover precedence and response constraints. Note that, as stated in Section II-B, we assume the last event

Algorithm 3: Auxiliary functions for Precedence constraint

```
1  Function Discover_precedence
       Data: D_M: the model
       D_E: the set of events in the trace π_c(e)
       e: an event
       S: the system state.
       Result: D_F: a set of discovered fulfillments for
               some precedence rules.
2      Initialize set D_F = {(r, C_1, C_2, S)};
3      for C'_1 in D_E do
4          D_F ← D_F ∪ {("precedence", C'_1, e, S)};

5  Function DiscoverViolations_precedence
       Data: D_M: the model
       D_E: the set of events in the trace π_c(e)
       e: an event
       S: the system state.
       Result: D_V: a set of discovered violations for
               some precedence rules.
6      Initialize set D_V = {(r, C_1, C_2, S)};
7      forall C'_1 ∈ π_{C_1}(σ_{r="precedence", C_2=e}(D_M)) do
8          if C'_1 ∉ D_E then
9              D_V ← D_V ∪ {("precedence", C'_1, e, S)};
```

in every trace to be tagged. This assumption simplifies the discovery algorithms and is not costly to implement in real-world systems.

Algorithm 3 reports the pseudo-code we developed for the online discovery of data-aware *precedence* constraints. The logic for discovering the fulfillments of *precedence* constraints is the following. For each event e that belongs to a certain trace c, a precedence constraint is satisfied for the activity $\pi_C(e)$ and all the activities of the events that occurred in c before e. As for the discovery of violations, we check the model for all the precedence rules that have as second element the activity of the current event $\pi_C(e)$. If the corresponding first activity is not present in any event of the trace $\pi_c(e)$ (i.e., the trace e belongs to), a violation for that rule just happened.

The logic for implementing the discovery of *response* constraints is more complex and we omit it for lack of space. The main difference with respect to the discovery of *precedence* constraints is that, here, two more data structures are needed. $\mathcal{D}_P = \{(c, C_1, C_2, f)\}$, where c is the trace, C_1 and C_2 are two activities, and f is the frequency, is a structure containing *pending* constraints, i.e., constraints that will eventually become fulfillments or violations, depending on the continuation of the trace c. The second data structure is $\mathcal{D}_S = \{(c, C, \mathcal{S}, t)\}$ represents a (multi-) set of *snapshots* ($\mathcal{S}$), each one with the associated trace c, activity C, and a timestamp t (or an increasing integer) that keeps track of the order in which the snapshots are inserted in the set. A prototype implementing both algorithms is publicly available at *https://github.com/MCambiaso/ODADD*.

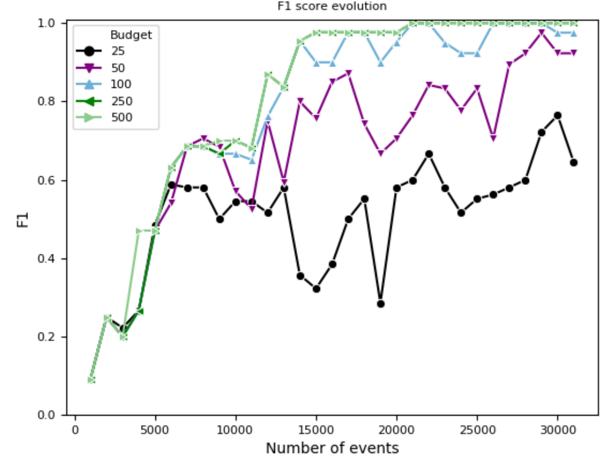

Fig. 1. Evolution of F1 score of our proposed algorithm on a synthetic data stream.

V. Experimental evaluation

We evaluated our approach in two experimental settings. In our first experiment, we validated our approach on a toy data stream, to study its behavior in a controlled setting. The stream is composed of $5,000$ traces, generated starting from a relatively simple BPMN model for a total of approximately $31,000$ events.

The *gold standard* (manually extracted) is composed of 21 response constraints, 6 of which are associated with a data condition. The batch data-aware Declare miner algorithm [7] (as expected) was able to correctly extract all the rules (F1 score 1.0). As for our online approach, in Figure 1, we report the evolution of the F1-score over the stream (we compute the F1 every $1,000$ events), with respect to the *gold standard* rules. We report the results for different budget values. For low budget values (25 and 50), some important rules get deleted from the model, so that the F1 scores are low. Note, however, that the quality of the model tends to increase with the number of events. When the budget is large enough (250 and 500) we see that the F1 score achieves the value of 1.0, thus extracting the same set of rules as the batch data-aware Declare miner.

In our second experiment, we compare the fulfillment ratio of the rules extracted with our approach from a real-life log with the fulfillment ratio of the same rules without considering the data conditions. We validated the approach with the event log used in the BPI challenge 2011 [23] that records the treatment of patients diagnosed with cancer from a large Dutch hospital. The event log contains 1143 cases and 150,291 events distributed across 623 activities. Moreover, the event log contains a total of 13 domain-specific attributes, e.g., *Age*, *Diagnosis Code*, *Treatment code*, in addition to the standard XES attributes, i.e., *concept:name*, *lifecycle:transition*, *time:timestamp* and *org:group*.

We report in Tables II and III some examples of *precedence* and *response* rules discovered by the proposed algorithms.

TABLE II
EXAMPLE OF 8 DISCOVERED *precedence* CONSTRAINTS.

Constr. ID	Activations no data	Ful. ratio no data	Activations data-aware	Ful. ratio data-aware
1	15298	0.778	13381	0.860
2	15298	0.744	12614	0.860
3	15296	0.830	14110	0.882
4	15257	0.758	13388	0.834
5	15243	0.764	13114	0.856
6	15243	0.779	13518	0.853
7	15243	0.787	13671	0.853
8	15243	0.774	13478	0.852

TABLE III
EXAMPLE OF 8 DISCOVERED *response* CONSTRAINTS.

Constr. ID	Activations no data	Ful. ratio no data	Activations data-aware	Ful. ratio data-aware
1	4476	0.723	3241	0.999
2	3444	0.999	3444	0.999
3	1802	0.699	1227	0.931
4	1004	0.389	351	1
5	873	0.176	187	0.759
6	749	0.166	101	1
7	627	0.575	292	1
8	578	0.349	266	0.751

As expected, the fulfillment ratio of the data-aware rules discovered by the proposed algorithms is grater or equal compared to that of the corresponding standard Declare rules (i.e., the rules obtained by dropping the data conditions).

VI. RELATED WORK

Several approaches have been proposed in the literature for the discovery of declarative process models. In [24], the authors present the first approach for the discovery of Declare rules from event logs. Several approaches to improve the performance of the discovery task and the richness of its outcome are presented in [25]–[31]. Additionally, there are post-processing approaches that aim at simplifying the resulting Declare models in terms of inconsistency/redundancy elimination [32]–[34] and disambiguation [35].

In [36], the authors introduce for the first time a data-aware semantics for Declare and [7] first covered the data perspective in declarative process discovery. In [37], the authors propose a mining technique for discovering Multi-Perspective Declare (MP-Declare) [38] models that support the definition of complex constraints that integrate activation, correlation, and temporal dependencies. The approach is implemented starting from the SQL-based process mining approach described in [30], relying on RXES, a standardized architecture for storing event log data in relational databases [39]. Differently from [37] where the models are identified based on user-specified queries, in [40], [41], the discovery of MP-Declare models is fully automated.

Process stream mining consists of the extraction of process structures from continuous and rapid process data records. Even if, in the last years, dozens of process discovery techniques have been proposed [1], these techniques all work on static event logs and not on streaming data. Only a few works in process mining aim at analyzing event streams. In [42], [43], the authors focus on incremental workflow mining and task mining. The basic idea is to mine cases as soon as they are observed; each new model is then merged with the previous one to refine the global process representation. The approach described is thought to deal with the incremental process refinement based on logs generated from version management systems.

In [44], [45], the authors propose an adaptation of the Heuristics Miner (a control-flow discovery algorithm) to data streams. The aim of these works is to extract a procedural control-flow from an event stream. In [46], an incremental approach for translating transition systems into Petri nets is described. In [47], an approach to automatically discover directly-follows graphs from streams of data is presented. In [48] declarative rules are used in an online analysis of streams of events targeted at a security-oriented classification. None of the above works provides an online approach for the discovery of data-aware declarative process models as proposed in this paper.

VII. CONCLUSION

In this paper, we proposed the first prototype of an online approach to extract data-aware Declare constraints from possibly unbounded streams of events, with a limited memory consumption and fast processing time. We implemented two sample algorithms for the discovery of *precedence* and *response* constraints.

As future work, we plan to implement other Declare rules in our software. Moreover, we plan to use the alternative definition of Lossy Counting fixing the budget instead of the error, as proposed in [19]. Finally, the approach will be integrated into an operational support system [49], [50] in the same vein as the one implemented in [51].

REFERENCES

[1] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[2] W. M. P. van der Aalst, A. Adriansyah, A. K. A. Medeeiros, F. Arcieri, T. Baier, T. Blickle, R. P. J. C. Bose, P. Brand, R. Brandtjen, and J. C. A. M. Buijs, "Process mining manifesto," in *Business process management workshops*, 2012.

[3] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg, "Declarative workflows: Balancing between flexibility and support," *Computer Science-Research and Development*, vol. 23, no. 2, pp. 99–113, 2009.

[4] F. M. Maggi, A. Burattin, M. Cimitile, and A. Sperduti, "Online process discovery to detect concept drifts in ltl-based declarative process models," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, 2013.

[5] A. Burattin, M. Cimitile, F. M. Maggi, and A. Sperduti, "Online discovery of declarative process models from event streams," *IEEE Transactions on Services Computing*, vol. 8, no. 6, pp. 833–846, 2015.

[6] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst, "Declare: Full support for loosely-structured processes," in *IEEE International Enterprise Distributed Object Computing Conference*, 2007.

[7] F. M. Maggi, M. Dumas, L. García-Bañuelos, and M. Montali, "Discovering data-aware declarative process models from event logs," in *Business Process Management*, 2013, pp. 81–96.

[8] A. Burattin, F. M. Maggi, W. M. P. van der Aalst, and A. Sperduti, "Techniques for a Posteriori Analysis of Declarative Processes," in *IEEE International Enterprise Distributed Object Computing Conference*, 2012.
[9] M. Pesic and W. M. P. van der Aalst, "A declarative approach for flexible business processes management," in *Business process management workshops*, 2006.
[10] M. Pesic, "Constraint-Based Workflow Management Systems: Shifting Controls to Users," Ph.D. dissertation, Beta Research School for Operations Management and Logistics, Eindhoven, 2008.
[11] O. Kupferman and M. Y. Vardi, "Vacuity detection in temporal model checking," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 4, no. 2, pp. 224–233, 2003.
[12] C. C. Aggarwal, *Data streams: models and algorithms*. Springer Science & Business Media, 2007.
[13] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive Online Analysis Learning Examples," *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.
[14] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: a review," *ACM Sigmod Record*, vol. 34, no. 2, pp. 18–26, 2005.
[15] N. Schweikardt, "Short-Entry on One-Pass Algorithms," in *Encyclopedia of Database Systems*, 2009.
[16] M. van Leeuwen and A. Siebes, "StreamKrimp: Detecting Change in Data Streams." in *Machine Learning and Knowledge Discovery in Databases*, 2008.
[17] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Mach. Learn.*, vol. 23, no. 1, pp. 69–101, 1996.
[18] G. Manku and R. Motwani, "Approximate frequency counts over data streams," in *international conference on Very Large Data Bases*, 2002.
[19] G. Da San Martino, N. Navarin, and A. Sperduti, "A Lossy Counting Based Approach for Learning on Streams of Graphs on a Budget," in *International Joint Conference on Artificial Intelligence*, 2013.
[20] P. Domingos and G. Hulten, "Mining High-Speed Data Streams," in *International Conference on Knowledge Disovery and Data Mining*, 2000.
[21] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American statistical association*, vol. 58, no. 301, pp. 13–30, 1963.
[22] A. Burattin, *Process Mining Techniques in Business Environments*. Springer, 2015.
[23] B. F. van Dongen, "Real-life event logs - Hospital log," 2011. [Online]. Available: https://doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b- 0d1120ffcf54
[24] F. M. Maggi, A. Mooij, and W. M. P. van der Aalst, "User-Guided Discovery of Declarative Process Models," in *IEEE Symposium on Computational Intelligence and Data Mining*, 2011.
[25] F. M. Maggi, R. P. J. C. Bose, and W. M. P. van der Aalst, "Efficient discovery of understandable declarative process models from event logs," in *International Conference on Advanced Information Systems Engineering*, 2012.
[26] C. Di Ciccio, M. H. M. Schouten, M. de Leoni, and J. Mendling, "Declarative process discovery with MINERful in ProM." in *BPM (Demos)*, 2015, pp. 60–64.
[27] M. Westergaard, C. Stahl, and H. A. Reijers, "UnconstrainedMiner: Efficient Discovery of Generalized Declarative Process Models," *BPM Center Report BPM-13-28, BPMcenter.org*, 2013.
[28] C. Di Ciccio, F. M. Maggi, and J. Mendling, "Discovering target-branched Declare constraints," in *International Conference on Business Process Management*, 2014.
[29] ——, "Efficient discovery of target-branched Declare constraints," *Information Systems*, vol. 56, pp. 258–283, 2016.
[30] S. Schönig, A. Rogge-Solti, C. Cabanillas, S. Jablonski, and J. Mendling, "Efficient and customisable declarative process mining with SQL," in *International Conference on Advanced Information Systems Engineering*, 2016.
[31] F. M. Maggi, C. Di Ciccio, C. Di Francescomarino, and T. Kala, "Parallel algorithms for the automated discovery of declarative process models," *Inf. Syst.*, vol. 74, no. Part, pp. 136–152, 2018.
[32] F. M. Maggi, R. P. J. C. Bose, and W. M. P. van der Aalst, "A knowledge-based integrated approach for discovering and repairing Declare maps," in *International Conference on Advanced Information Systems Engineering*, 2013.
[33] C. Di Ciccio, F. M. Maggi, M. Montali, and J. Mendling, "Ensuring model consistency in declarative process discovery," in *International Conference on Business Process Management*, 2015.
[34] ——, "Resolving inconsistencies and redundancies in declarative process models," *Information Systems*, vol. 64, pp. 425–446, 2017.
[35] R. P. J. C. Bose, F. M. Maggi, and W. M. P. van der Aalst, "Enhancing Declare Maps Based on Event Correlations," in *Business Process Management*, 2013.
[36] M. Montali, F. Chesani, P. Mello, and F. M. Maggi, "Towards data-aware constraints in Declare," in *Annual ACM Symposium on Applied Computing*, 2013.
[37] S. Schönig, C. Di Ciccio, F. M. Maggi, and J. Mendling, "Discovery of multi-perspective declarative process models," in *International Conference on Service-Oriented Computing*, 2016.
[38] A. Burattin, F. M. Maggi, and A. Sperduti, "Conformance checking based on multi-perspective declarative process models," *Expert Syst. Appl.*, vol. 65, pp. 194–211, 2016.
[39] B. F. van Dongen and S. Shabani, "Relational xes: Data management for process mining." in *CAiSE Forum*, 2015.
[40] V. Leno, M. Dumas, and F. M. Maggi, "Correlating activation and target conditions in data-aware declarative process discovery," in *Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings*, 2018, pp. 176–193.
[41] V. Leno, M. Dumas, F. M. Maggi, M. La Rosa, and A. Polyvyanyy, "Automated discovery of declarative process models with correlated data conditions," *Inf. Syst.*, vol. 89, p. 101482, 2020.
[42] E. Kindler, V. Rubin, and W. Schäfer, "Incremental workflow mining based on document versioning information," in *Software Process Workshop*, 2005.
[43] ——, "Incremental workflow mining for process flexibility." in *BPMDS*, 2006.
[44] A. Burattin, A. Sperduti, and W. M. P. van der Aalst, "Heuristics miners for streaming event data," *arXiv preprint arXiv:1212.6383*, 2012.
[45] ——, "Control-flow discovery from event streams," in *IEEE Congress on Evolutionary Computation*, 2014.
[46] A. Sharp and P. McDermott, *Workflow modeling: tools for process improvement and applications development*. Artech House, 2009.
[47] V. Leno, A. Armas-Cervantes, M. Dumas, M. La Rosa, and F. M. Maggi, "Discovering process maps from event streams," in *Proceedings of the 2018 International Conference on Software and System Process, ICSSP 2018, Gothenburg, Sweden, May 26-27, 2018*, 2018, pp. 86–95.
[48] B. Fazzinga, S. Flesca, F. Furfaro, and L. Pontieri, "Online and offline classification of traces of event logs on the basis of security risks," *Journal of Intelligent Information Systems*, vol. 50, no. 1, pp. 195–230, 2018.
[49] M. Westergaard and F. M. Maggi, "Modeling and verification of a protocol for operational support using coloured Petri nets," in *Applications and Theory of Petri Nets - 32nd International Conference, PETRI NETS 2011, Newcastle, UK, June 20-24, 2011. Proceedings*, 2011, pp. 169–188.
[50] F. M. Maggi and M. Westergaard, "Designing software for operational decision support through coloured Petri nets," *Enterprise IS*, vol. 11, no. 5, pp. 576–596, 2017.
[51] A. Burattin, M. Cimitile, and F. M. Maggi, "Lights, camera, action! business process movies for online process discovery," in *Business Process Management Workshops - BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers*, 2014, pp. 408–419.