Online Discovery of Declarative Process Models from Event Streams

Andrea Burattin, *Member, IEEE,* Marta Cimitile,
Fabrizio M. Maggi, Alessandro Sperduti, *Senior Member, IEEE*

Abstract—Today's business processes are often controlled and supported by information systems. These systems record real-time information about business processes during their executions. This enables the analysis at runtime of the process behavior. However, many modern systems produce "big data", i.e., collections of data sets so large and complex that it becomes impossible to store and process all of them. Moreover, few processes are in steady-state but, due to changing circumstances, they evolve and systems need to adapt continuously. In this paper, we present a novel framework for the discovery of LTL-based declarative process models from streaming event data in settings where it is impossible to store all events over an extended period of time or where processes evolve while being analyzed. The framework continuously updates a set of valid business constraints based on the events occurred in the event stream. In addition, our approach is able to provide meaningful information about the most significant concept drifts, i.e., changes occurring in a process during its execution. We report about experimental results obtained using synthetic logs and a real-life event log pertaining to the treatment of patients diagnosed with cancer in a large Dutch academic hospital.

Index Terms—Process Discovery, Event Stream Analysis, Big Data, Declarative Process Models, Concept Drifts, Operational Decision Support.

1 Introduction

Recent years have witnessed an increasing production in business organizations of large size and complex datasets that are hard to store and manage and that, for this reason, are called "big data". In the literature, there are several definitions of what is meant by big data, coming from different and important institutions [1], [2], [3]. Nevertheless, most of them seem to agree on the "three *v*s" characterization: *(i) v*olume; *(ii) v*elocity; and *(iii) v*ariety. The *volume* refers to the "size" of data, which is assumed to be beyond the ability of typical database softwares. The *velocity* at which data is generated is assumed to be extremely high. The *variety* characterization, finally, refers to the heterogeneity of the data that cannot be tackled using existing processing tools. Due to the diffusion of information systems, service-oriented environments and cloud computing, big data are nowadays available also in the form of event logs deriving from the execution of large scale processes and useful to be analyzed using process mining techniques [4].

In this work, we focus on a particular branch of process mining, i.e., process discovery. Process discovery techniques automatically construct a representation of complex business processes based on example executions in an event log, without using any a-priori information. In particular, we focus on online process discovery from event streams as a way to deal with big amounts of data. We process events on-the-fly, as they occur, by storing information only about the most relevant ones in a limited budget of memory. As a result of this analysis, users can monitor and improve, at runtime, the business processes that generated those events.

The variability of the behavior recorded in event logs increases as a consequence of the changeability and high dynamism of the execution environment of the underlying business processes. For this reason, traditional process discovery techniques (mainly based on procedural process modeling languages), applied to event logs coming from unstable environments, often produce models in which too many execution paths are explicitly represented. Therefore, they become completely unreadable for a human business analyst.

To address this issue, recently, several works [5], [6], [7], [8], [9] have been focused on the discovery of declarative process models, proposing algorithms for the off-line discovery of declarative process models. Declarative process models represent a good alternative to procedural process models when the execution environment of the process is turbulent and changeable. However, the work proposed in the context of off-line declarative process discovery cannot be used for dealing with large, growing amounts of data and for online analysis of process models. In fact, most of the existing techniques require several iterations on an event log for discovering a process model and for this reason they are not suitable

- *A. Burattin is with the University of Innsbruck, Austria. Most of his work has been done as part of the Eurostars-Eureka project PROMPT (E!6696), while he was with the University of Padua, Italy. E-mail: andrea.burattin@uibk.ac.at*
- *M. Cimitile is with the Unitelma Sapienza University, Italy. E-mail: marta.cimitile@unitelma.it*
- *F. M. Maggi is with the University of Tartu, Estonia. E-mail: f.m.maggi@ut.ee*
- *A. Sperduti is with University of Padua, Italy. E-mail: sperduti@math.unipd.it*

for online settings.

In this paper, we propose a set of techniques able to mine an event stream that is a real-time, continuous, ordered sequence of events [10]. The discovered process models are represented using Declare [11], a declarative process modeling language that combines a formal semantics grounded in Linear Temporal Logic (LTL) on finite traces,[1] with a graphical notation. Declare is characterized by a set of templates defining parameterized classes of LTL rules, each one equipped with its own graphical representation. Constraints are concrete instantiations of templates and inherit the graphical representation and LTL semantics from the corresponding templates. A Declare model is a set of constraints that should hold in conjunction during the process execution.

The output models are visualized in the form of an animated movie showing how the behavior of the business process that produces the event stream changes over time. To interact with event streams, these techniques have been developed considering that *i)* the complete event stream cannot be stored; *ii)* only one pass over data is allowed and backtracking over the event stream is not feasible; *iii)* the discovered model should be quickly adapted to cope with concept drifts, i.e., with the situation in which the process is changing while being analyzed; *iv)* variable system conditions can be reflected on the event stream (e.g., fluctuating stream rates).

This work extends the study conducted in [12] that presents algorithms for the discovery of declarative process models from event streams. Differently from our previous work, in this paper, some improvements have been made: *i)* a third approximate frequency counting algorithm has been added to the other two described in [12]; *ii)* the discovery algorithms now cover the entire Declare language (in our previous work only a subset of Declare templates could be discovered); *iii)* all the discovery algorithms have been adapted to discover constraints based on two different notions of constraint support (event-based and trace-based, see Section 4); *iv)* to assess the applicability of our approach, we conducted an experimentation with a larger set of synthetic logs and with a real-life event log pertaining to the treatment of patients diagnosed with cancer in a large Dutch academic hospital [13]; *v)* we now provide a graphical visualization of the evolution of the Declare model over time while the event stream is progressing.

The paper is structured as follows. Section 2 introduces the characteristics of event stream mining as well as some basic notions about Declare. Next, Section 3 illustrates the three approaches, based on Sliding Window, Lossy Counting and Lossy Counting with Budget, used in this paper for stream mining. Sections 4 and 5 introduce the algorithms for the online discovery of Declare models underlying the proposed approach. Section 6 briefly describes how the evolution of the Declare model over time is visualized in the process mining tool ProM. In Section 7, the experimentation and the resulting benchmark analysis are discussed. Related work is presented in Section 8 and Section 9 reports conclusions.

1. For compactness, we will use the LTL acronym to denote LTL on finite traces.

TABLE 1
Example of an event log.

Event #	Case Id	Activity Name	Timestamp
1	*Case 1*	Act_1	31-01-2014 00:01
2	*Case 1*	Act_2	31-01-2014 01:02
3	*Case 2*	Act_1	31-01-2014 02:13
4	*Case 2*	Act_2	31-01-2014 13:14
5	*Case 2*	Act_3	31-01-2014 14:25
6	*Case 1*	Act_4	31-01-2014 15:26
7	*Case 2*	Act_4	31-01-2014 16:37

2 PRELIMINARIES

This section provides a general introduction to the basic elements we are going to use throughout the paper. In particular, we introduce the notion of event log and event stream and we give a quick overview of the Declare language.

2.1 From Event Logs to Event Streams

Information systems record information related to business processes during their execution in the form of event logs. Recently, the IEEE Task Force on Process Mining has proposed XES (eXtensible Event Stream) [14], a new standard for event logs, defined starting from the data that most information systems use to store. In an event log, each process instance constitutes a *case*. The sequence of events that belong to the same case is called a *trace* (and different cases can refer to the same trace). In Table 1, we report a simple event log, with 7 events, referring to 4 activities (Act_1, Act_2, Act_3 and Act_4), over two cases (*Case 1* and *Case 2*).

If $\mathcal{A}$ is the set of all activity names, $\mathcal{C}$ is the set of all case identifiers and $\mathcal{T}$ is the set of timestamps, then, it is possible to define:

Definition 1 (Event, Event Universe)**.** *An* event e *is a triple* $e = (c, a, t) \in \mathcal{C} \times \mathcal{A} \times \mathcal{T}$*, which describes the occurrence of activity* a *in case* c *at time* t*. The set of all possible events is called* event universe *and it is indicated as* $\mathcal{E} = \mathcal{C} \times \mathcal{A} \times \mathcal{T}$*.*

In order to identify the single components of an event $e = (c, a, t)$, we define the functions $\#_{\text{case}}(e) = c$, $\#_{\text{activity}}(e) = a$ and $\#_{\text{time}}(e) = t$.

Definition 2 (Sequence)**.** *Given a finite set* $\mathbb{N}_n^+ = \{1, 2, \ldots, n\}$ *and a target set* A*, we call* sequence σ *the function* $\sigma : \mathbb{N}_n^+ \to A$*. It is possible to say that* σ *maps index values to the corresponding elements in* A*. For simplicity, we consider a sequence through its string interpretation:* $\sigma = \langle s_1, \ldots, s_n \rangle$*, where* $s_i = \sigma(i)$ *and* $s_i \in A$*.*

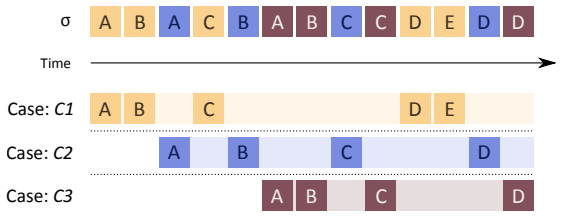


Fig. 1. Visual example of a short portion of an event stream. Square boxes represent events. Color-codes are used for case ids (i.e., different background colors represent different case ids) and labels are used to indicate activity names. The top line reports the entire stream portion, the remaining lines show the single cases contributing to the stream.

In our context, we use timestamps to sort events. Therefore, we can consider a trace as a *sequence* (Definition 2). For example, the event log of Table 1 contains two traces, i.e., $\langle Act_1, Act_2, Act_4 \rangle$ and $\langle Act_1, Act_2, Act_3, Act_4 \rangle$.

Several works in the data mining literature, such as [10], [15], [16], agree in defining a *data stream* as a fast sequence of data items. It is common to assume that: *(i)* data has a small, typically predefined, number of attributes; *(ii)* mining algorithms are able to analyze an infinite number of data items, handling problems related to memory bounds; *(iii)* the amount of memory that the learner can use is finite and much smaller than the memory required to store the data observed in a reasonable span of time; *(iv)* each item is processed within a certain small amount of time (algorithms have to linearly scale with respect to the number of processed items): typically the algorithms work with one pass on the data; *(v)* data models associated to a stream (i.e., the "underlying concepts") can either be stationary or evolving [17].

It is possible to adapt the definition of data streams to streams of events:

Definition 3 (Event Stream)**.** *Given the event universe* $\mathcal{E}$*, a sequence of events* $S : \mathbb{N}^+ \rightarrow \mathcal{E}$ *is called an* event stream*.*

An event stream is a potentially infinite sequence of events. It is possible to assume that sequence indexes comply with the time order of events (i.e., given the sequence S, for all indexes $i \in \mathbb{N}^+$, $\#_{\text{time}}(S(i)) \leq \#_{\text{time}}(S(i+1))$). In Figure 1, we show a small portion of an event stream. This example clarifies that two subsequent events may belong to different cases.

Starting from a general data stream (not necessarily a stream of events), it is possible to perform different analysis. The most common are clustering, classification, frequency counting, time series analysis and change diagnosis (concept drift detection) [18], [17], [15]. More generally, it is possible to distinguish between two types of stream mining approaches: *data-* and *task-based*. Data-based mining algorithms aim at identifying finite datasets in a stream. The main requirement of these datasets is to be representative of the entire stream. The second type of algorithms, the task-based ones, are specifically devised for event streams and are able to minimize time and space complexity for the analysis of infinite sequences of events. All approaches that we are presenting in this paper belong to the latter category.

João Gama, in [19], proposes a characterization of data streams consisting of three different processing models:

1) insert only model: once an item is seen, it cannot be changed;
2) insert-delete model: items can be seen, deleted or updated;
3) additive model: each seen item refers to a numerical variable which is incremented.

All event streams that we are dealing with in this paper belong to the first category (i.e., insert only model), since we observe activity executions after they have been performed and, therefore, it does not make sense to update or delete events.

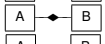
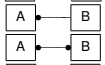
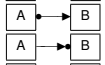
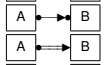
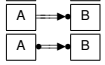
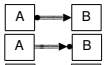
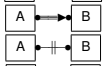
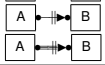
2.2 Declare: Some Basic Notions

In this paper, the process behavior as recorded in an event stream is described using Declare rules. Declare is a declarative process modeling language originally introduced by Pesic and van der Aalst in [20]. Instead of explicitly specifying the flow of the interactions among process events, Declare describes a set of constraints that must be satisfied throughout the process execution. The possible orderings of events are implicitly specified by constraints and anything that does not violate them is possible during execution. In comparison with procedural approaches that produce "closed" models, i.e., all what is not explicitly specified is forbidden, Declare models are "open" and tend to offer more flexibility for the execution.

A Declare model consists of a set of constraints applied to events. Constraints, in turn, are based on templates. Templates are patterns that define parameterized classes of properties and constraints are their concrete instantiations. Templates have a user-friendly graphical representation understandable to the user and their semantics can be formalized using different logics [21], the main one being LTL, making them verifiable and executable. Each constraint inherits the graphical representation and semantics from its templates. The major benefit of using templates is that analysts do not have to be aware of the underlying logic-based formalization to understand the models. They work with the graphical representation of templates, while the underlying formulas remain hidden. TABLE 2 summarizes the Declare templates (we indicate template parameters with capital letters and concrete activities in their instantiations with lower case letters).

We informally introduce here the temporal operators we use to describe the semantics of the Declare templates in TABLE 2. We indicate by φ and ψ LTL formulas. $\bigcirc\varphi$ means that φ has to hold in the next position in

TABLE 2
Graphical notation and LTL formalization of the Declare templates.

Template	Formalization	Notation
Init(A)	A	
Existence(A)	$\Diamond A$	
Existence2(A)	$\Diamond(A \wedge \bigcirc(\Diamond A))$	
Existence3(A)	$\Diamond(A \wedge \bigcirc(\Diamond(A \wedge \bigcirc(\Diamond A))))$	
Absence(A)	$\neg\Diamond A$	
Absence2(A)	$\neg\Diamond(A \wedge \bigcirc(\Diamond A))$	
Absence3(A)	$\neg\Diamond(A \wedge \bigcirc(\Diamond(A \wedge \bigcirc(\Diamond A))))$	
Exactly1(A)	$\Diamond A \wedge \neg\Diamond(A \wedge \bigcirc(\Diamond A))$	
Exactly2(A)	$\Diamond(A \wedge \bigcirc(\Diamond A)) \wedge \neg\Diamond(A \wedge \bigcirc(\Diamond(A \wedge \bigcirc(\Diamond A))))$	
Choice(A,B)	$\Diamond A \vee \Diamond B$	
Exclusive Choice(A,B)	$(\Diamond A \vee \Diamond B) \wedge \neg(\Diamond A \wedge \Diamond B)$	
Responded Existence(A,B)	$\Diamond A \rightarrow \Diamond B$	
Co-Existence(A,B)	$\Diamond A \leftrightarrow \Diamond B$	
Response(A,B)	$\Box(A \rightarrow \Diamond B)$	
Precedence(A,B)	$\neg B \mathcal{W} A$	
Succession(A,B)	$\Box(A \rightarrow \Diamond B) \wedge (\neg B \mathcal{W} A)$	
Alternate Response(A,B)	$\Box(A \rightarrow \bigcirc(\neg A \mathcal{U} B))$	
Alternate Precedence(A,B)	$(\neg B \mathcal{W} A) \wedge \Box(B \rightarrow \bigcirc(\neg B \mathcal{W} A))$	
Alternate Succession(A,B)	$(\neg B \mathcal{W} A) \wedge \Box(B \rightarrow \bigcirc(\neg B \mathcal{W} A)) \wedge \Box(A \rightarrow \bigcirc(\neg A \mathcal{U} B))$	
Chain Response(A,B)	$\Box(A \rightarrow \bigcirc B)$	
Chain Precedence(A,B)	$\Box(\bigcirc B \rightarrow A)$	
Chain Succession(A,B)	$\Box(A \rightarrow \bigcirc B) \wedge \Box(\bigcirc B \rightarrow A)$	
Not Co-Existence(A,B)	$\Diamond A \rightarrow \neg\Diamond B$	
Not Succession(A,B)	$\Box(A \rightarrow \neg\Diamond B)$	
Not Chain Succession(A,B)	$\Box(A \rightarrow \neg\bigcirc B)$	

a trace. $\Box\varphi$ indicates that φ has to hold always in the subsequent positions in a trace. $\Diamond\varphi$ means that φ has to hold eventually (somewhere) in the subsequent positions. $\varphi \mathcal{U} \psi$ requires that φ has to hold in a trace at least until ψ holds. ψ must hold in the current or in a future position. Finally, $\varphi \mathcal{W} \psi$ indicates that φ has to hold in the subsequent positions at least until ψ holds. If ψ never holds, φ must hold everywhere.

Consider, for example, the *response* constraint $\Box(a \rightarrow \Diamond b)$. This constraint indicates that if a *occurs*, b must eventually *follow*. Therefore, this constraint is satisfied for traces such as $\mathbf{t}_1 = \langle a, a, b, c \rangle$, $\mathbf{t}_2 = \langle b, b, c, d \rangle$ and $\mathbf{t}_3 = \langle a, b, c, b \rangle$, but not for $\mathbf{t}_4 = \langle a, b, a, c \rangle$ because, in this case, the second instance of a is not followed by a b. Note that, in $\mathbf{t}_2$, the considered response constraint is satisfied in a trivial way because a never occurs. In this case, we say that the constraint is *vacuously satisfied* [22]. In [23], 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. An *activation* of a constraint 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 the *response* constraint $\Box(a \rightarrow \Diamond b)$, because the execution of a forces b to be executed eventually.

An activation of a constraint can be a *fulfillment* or a *violation* for that constraint. When a trace is perfectly compliant with respect to a constraint, every activation of the constraint in the trace leads to a fulfillment. Consider, again, the response constraint $\Box(a \rightarrow \Diamond b)$. In trace $\mathbf{t}_1$, the constraint is activated and fulfilled twice, whereas, in trace $\mathbf{t}_3$, the same constraint is activated and fulfilled only once. On the other hand, when a trace is not compliant with respect to a constraint, an activation of the constraint in the trace can lead to a fulfillment but also to a violation (at least one activation leads to a violation). In trace $\mathbf{t}_4$, for example, the response constraint $\Box(a \rightarrow \Diamond b)$ is activated twice, but the first activation leads to a fulfillment (eventually b occurs) and the second activation leads to a violation (b does not occur subsequently). An algorithm to discriminate between fulfillments and violations for a constraint in a trace is presented in [23].

3 Approximate Frequency Counting Algorithms

The approaches we present in the rest of the paper are based on three algorithms, already available in the literature. The first algorithm is called *Sliding Window* [19]. The basic idea of this algorithm is to collect events for a certain span of time and then apply a standard "off-line" analysis approach. The other two approaches are called *Lossy Counting* [24] and *Lossy Counting with Budget* [25]. The basic idea of these approaches is to consider aggregated representations of the latest observations, i.e., instead of storing repetitions of the same event, to save space, they store a counter keeping trace of the number of instances of the same observation.

For the sake of clarity, in the following subsections, we present the lossy counting algorithms in their original formulation defined to solve the "frequency counting problem" (i.e., to count the number of times an event is observed in a portion of a stream). We will then explain how these two algorithms can be used for the discovery of declarative process models.

3.1 Lossy Counting

In this section, we describe the Lossy Counting algorithm, first presented in [24]. We have chosen this algorithm instead of the Sticky Sampling algorithm (described in the same paper), since, as the authors state, in practice, the Lossy Counting approach has better performances.

Lossy Counting (LC) is an approximate frequency counting algorithm. The pseudocode of this approach is reported in Algorithm 1. 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

Algorithm 1: Frequency Mining with Lossy Counting

Input: S: data stream; ϵ: maximal approximation error

1 $T \leftarrow \emptyset$ /* Initially empty set */
2 $N \leftarrow 1$ /* Number of observed events */
3 $w \leftarrow \lceil \frac{1}{\epsilon} \rceil$ /* Bucket width */
4 **forever do**
5 $e \leftarrow observe(S)$
6 **if** $analyze(e)$ **then**
7 $b_{curr} \leftarrow \lceil \frac{N}{\epsilon} \rceil$
8 **if** *e is already in* T **then**
9 Increment the frequency of e in T
10 **else**
11 Insert $(e, 1, b_{curr} - 1)$ in T
12 **end**
13 **if** $N \mod w = 0$ **then**
14 **foreach** $(a, f, \Delta) \in T$ *s.t.* $f + \Delta \leq b_{curr}$ **do**
15 Remove (a, f, Δ) from T
16 **end**
17 **end**
18 $N \leftarrow N + 1$
19 **end**
20 **end**

Algorithm 2: Frequency Mining with Lossy Counting with Budget

Input: S: data stream; $\mathcal{B}$: available budget

1 $T \leftarrow \emptyset$ /* Initially empty set */
2 $b_{curr} \leftarrow 0$ /* Initial bucket id */
3 **forever do**
4 $e \leftarrow observe(S)$
5 **if** $analyze(e)$ **then**
6 **if** *e is already in* T **then**
7 Increment the frequency of e in T
8 **else**
9 **if** $|T| = \mathcal{B}$ **then**
10 $b_{curr} \leftarrow b_{curr} + 1$
11 **repeat**
12 **foreach** $(a, f, \Delta) \in T$ *s.t.* $f + \Delta \leq b_{curr}$ **do**
13 Remove (a, f, Δ) from T
14 **end**
15 **if** *no element has been removed* **then**
16 $b_{curr} \leftarrow \min_{(a,f,\Delta) \in T} f + \Delta$
17 **end**
18 **until** $|T| < \mathcal{B}$
19 **end**
20 Insert $(e, 1, b_{curr})$ in T
21 **end**
22 **end**
23 **end**

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 LC 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, if this event must be processed (line 6), 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 $N \mod w = 0$ (i.e., every w events), the algorithm cleans up the data structure by removing entries with maximal approximate frequency (i.e., 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 $f + \Delta \leq b_{curr}$.

Note that the size of the data structure T depends on the stream S. For example, if the stream contains many instances of the same event, the algorithm only updates the f component of the corresponding event in T and, therefore, the space needed to store T is not that large. Moreover, it is worthwhile noting that the only way to control the size of T is indirectly via ϵ and it is not possible to directly control the size of the memory dedicated to store this structure.

3.2 Lossy Counting with Budget

In a recent work by Da San Martino et al. [25], a Lossy Counting with Budget (LCB) algorithm is proposed. Algorithm 2 reports the pseudocode of this procedure. LCB uses the same data structure as LC (T). The fundamental difference between this approach and LC is that, in this new algorithm, we can control the maximum space used to store it.

Differently from LC, LCB dynamically changes the value of ϵ in order to keep the size of T within the available budget $\mathcal{B}$. In LCB, after observing a new event e, if it is already contained in T, then its frequency value is updated as in LC. However, in this case, if e is not in T and T has reached the maximum allowed size, it is necessary to remove old observations from T in order to make space for the new event to be inserted (line 9).

The deletion condition is the same as in LC (i.e., $f + \Delta \leq b_{curr}$) but, in this case, if no event satisfies it (and there is no space for the new observation), b_{curr} is increased until at least one event is removed (line 16). Note that increasing b_{curr} means that the maximal approximation error also increases. Once there is space available for the new event, it is added to the data structure T.

4 Online Discovery of Declare Models

In this section, we describe how to discover, at runtime, a Declare model from a stream of events. We use the algorithms for memory management presented in the previous sections in combination with algorithms for the online discovery of Declare constraints referring to different templates.

Algorithm 3 presents a general overview of our online discovery approach. Here, we keep a set R of *replayers*, one for each template we want to discover (line 3). Note that replayers for different templates implement different discovery algorithms. We will discuss in detail two examples of discovery algorithms in Section 5. Then, when a new event e is observed from the stream S, if it is necessary to analyze it, the algorithm replays e on all the template replayers of R (line 9).

Periodically (the period may depend either on the number of events observed, or on the actual elapsed

Algorithm 3: Online discovery scheme

```
Input: S: event stream, conf: approximate algorithm configuration (either
       LC or LCB, with the corresponding configuration: ε or B), update
       model condition
1  R ← ∅                                  /* Set of template replayers */
2  foreach Declare template t do
3  |  Add a replayer for t (according to conf) to R
4  end
5  forever do
6  |  e ← observe(S)
7  |  if analyze(e) then
8  |  |  foreach r ∈ R do
9  |  |  |  r(e, conf)                   /* Replay e on replayer r */
10 |  |  end
11 |  end
12 |  if update model condition then
13 |  |  model ← initially empty Declare model
14 |  |  foreach r ∈ R do
15 |  |  |  Update model with constraints in r
16 |  |  end
17 |  |  Use model          /* For example, update a graphical
   |  |  representation */
18 |  end
19 end
```

time), it is possible to update the discovered Declare model, by querying each replayer for the set of satisfied constraints (line 15). The Declare model can be used, for example, to update a user interface, or to perform periodical analysis.

In the current implementation of our approach, it is possible to choose the Declare templates to be discovered. In particular, all the standard Declare templates (listed in Table 2) can be discovered. Nevertheless, it is possible to make the approach working with additional user-defined templates by implementing the corresponding replayers and by adding them to R (line 3).

It is worthwhile noting that our approach can easily be distributed over different *computational nodes*. In particular, each replayer is independent of the others and, therefore, all the replay operations (line 9) can be parallelized. The same property holds also for the model update (line 15). This property makes our approach particularly suitable for dealing with high data volumes.

In our previous work [12], we proposed three basic approaches for the discovery of the following six Declare templates: *(i)* Response and Not Response; *(ii)* Precedence and Not Precedence; and *(iii)* Responded Existence and Not Responded Existence. In this work, we extend the number of possible Declare templates to cover the whole Declare language. In particular, our approach is able to discover the following templates: Absence, Absence2 and Absence3; Alternate Precedence; Alternate Response; Alternate Succession; Chain Precedence; Chain Response; Chain Succession and Not Chain Succession; Choice; Co-Existence and Not Co-Existence; Exactly1 and Exactly2; Exclusive Choice; Existence, Existence2 and Existence3; Init; Precedence and Not Precedence; Response and Not Response; Responded Existence and Not Responded Existence; Succession and Not Succession.

Some of these algorithms are very straightforward. For example, the existence templates and the absence templates verify that, in each case in the event stream, an event occurs at least and at most a certain number of times. It is also easy to verify that a case starts with a certain event (Init template). However, as explained in Section 4.1, these algorithms can only be defined if in the event stream there is information about the start and the end event of each case.

All algorithms start from a stream and build a set of candidate constraints, by considering all the possible combinations of activity names (only the ones referring to events detected so far in the event stream). Each algorithm receives as input an event $e = (c, a, t)$ and updates an internal data-structure in order to keep track of the new observed event and of how it affects the satisfaction of the candidates.

The candidate constraints to be included in the discovered Declare model can be selected based on the percentage of activations that leads to a fulfillment (event-based constraint support) or based on the percentage of cases in which the constraint is satisfied (trace-based support). In [12], we implemented our approach using the notion of event-based support. In this paper, we also consider the case in which the satisfaction of a constraint is evaluated in terms of trace-based support. Of course, in this latter case, the event stream should provide not only information about the case to which each event belongs but also an indication on the exact point in which each trace starts and ends. In the following subsections, we explain how our approach can be adapted based on these two notions of constraint support.

4.1 Discovery with Event-Based Support

In [12], we assume that every event in a stream only contains information on the corresponding case id, an activity name and a timestamp. We call this approach "event-based online discovery" because, since we do not have information about the points in the stream in which each case starts and ends, we can only measure the satisfaction of a constraint using the event-based notion of support.

However, the inability to identify start and end events of a case causes two main drawbacks. The first one is connected to the space usage, i.e., it is not possible to clean up the memory from events related to a completed case. The second drawback is related to the semantics of the Declare constraints. Since it is not possible to have a complete view of an entire trace, for some constraints, it is never possible to determine whether they are *permanently* satisfied or violated. For example, the *response* constraint $\Box(a \rightarrow \Diamond b)$ is always *temporarily* violated, if there is an a not (yet) followed by a b, or *temporarily* satisfied, if each a is (currently) followed by a b. In addition, the satisfaction of some constraints cannot be evaluated using the event-based notion of support. It is not natural, for example, to evaluate the percentage of fulfillments for an absence constraint, since such a

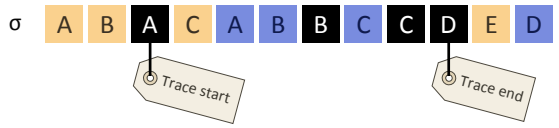


Fig. 2. An example of a stream σ with start and end events of a case tagged.

constraint is only satisfied when the constrained event never occurs. Similarly, it is difficult to count the number of fulfillments for an existence constraint. For example, it is not possible to count the number of fulfillment of a constraint forcing an event to occur "at least three times" or "exactly twice". Without information about the positions in which each case starts and ends, it is also impossible to evaluate constraints like Init, forcing an event to be the first one in a case. Although event-based approaches suffer from these problems, they are more easily applicable, since they require less information to be provided in an event stream.

4.2 Discovery with Trace-Based Support

As mentioned in the previous section, if events of a stream contain no information about the start and the end events of a case, it is not straightforward or even impossible to check the validity of some Declare constraints. On the other hand, we believe that tagging the first and the last event of each case is not particularly costly (the assumption, here, is to have knowledge about the entry and exit points of each case).

Therefore, in the so-called "trace-based" approaches, it is necessary to identify start and end events of each case with tags. Fig. 2 shows a graphical representation of this tagging for the black case (with white labels). In particular, we assume that the first and last event of the case (first and last occurrence of black events) are identifiable. This can be done in different ways. For example, it is possible to include this information in an event attribute, or to specify a list of activity names corresponding to start and end events.

If the start and the end events are identifiable, it is possible to use a modified version of the template replayers for the online discovery using the trace-based notion of constraint support. To this aim, it is necessary to keep track of all the running cases and to evaluate the satisfaction of a constraint only for those cases that are completed. An additional data structure is then needed, which stores information about completed cases.

Trace-based approaches can improve their performance by implementing the *analyze*(e) function (line 7 of Algorithm 3) to check whether a new observed event belongs to a case that is currently running. In particular, it is possible to immediately discard a new event if:

1) it is not the start event of a new case and it belongs to a case that is not yet started or already completed;
2) it is the start event of a case that is already running;
3) it is the end event of a case that is not running.

In all these cases, it is possible to report the invalid behavior to the user for further analysis.

5 Examples of Discovery Algorithms

For space limitations, we cannot describe all the implemented discovery algorithms that cover the entire Declare language. As an example, we show here two of the new algorithms implemented for the discovery of *alternate response* and *chain response* templates. Both of them are defined for the event-based scenario.

In these algorithms, we use the notion of *map*. Given a set of keys K and a set of values V, a map is a set $M \subseteq K \times V$. We use the following operators: *(i)* M.put(k, v), to add value v with key k to M; *(ii)* M.get$(x) \in V$, with $k \in K$, to retrieve from M the value associated with key k; *(iii)* M.keys $\subseteq K$ to obtain the set of keys in M; *(iv)* M.vals $\subseteq V$ to obtain the set of values in M. The keys of these maps are handled as elements in the data structure T when using LC and LCB. Therefore, some keys in the maps can be discarded at a certain point in time during the stream execution. In this case, also the values associated to the key that has been discarded are removed from the memory.

Algorithm 4 can be used for the discovery of *alternate response* constraints, whereas Algorithm 5 is able to discover *chain response* constraints. Based on the template semantics given in TABLE 2, the *alternate response* template indicates that every A has to be eventually followed by a B without another occurrence of A in between. The *chain response* template requires that every A has to be immediately followed by a B. In these algorithms, L is the set of all the activity names observed in the event stream (in all the cases). $activationsCounter_c$ is a map defined for each case c and containing, for each activity name, the number of its occurrences in c. This map can be used to count the number of activations for each constraint (the number of activations can be obtained by counting how many times the corresponding activity name has occurred in each case of the event stream). $pendingActivations_c$, $fulfilledActivations_c$ and $violatedActivations_c$ are maps defined for each case c and containing, respectively, the number of pending activations, the number of fulfillments and the number of violations in c for each constraint activated in c (the keys in the map are pairs of activity names representing the constraint parameters).

The algorithms receive as input an event $e = (c, a, t)$, where c is the case id, a is the activity name and t is the timestamp. This event is processed by updating maps $activationsCounter_c$, $pendingActivations_c$, $fulfilledActivations_c$ and $violatedActivations_c$. Using these maps, it is possible to count the number of activations for every candidate constraint, the number of fulfillments and the number of violations.

The computational complexity of Algorithm 4 is linear in the number of activity names (lines 15-25) and in the number of pending activations (lines 28-39). Similarly,

Algorithm 4: Discovery algorithm for alternate response constraints

Input: $e = (c, a, t)$ the event to be processed (c is the case id, a is the activity name, t is the timestamp)
conf: approximate algorithm configuration (either LC or LCB, with the corresponding configuration: ϵ or $\mathcal{B}$)

```
1  if L is not defined then
2  |  Define the empty map L
3  end
4  if activationsCounter_c is not defined then
5  |  Define the map activationsCounter_c
6  end
7  if pendingActivations_c is not defined then
8  |  Define the map pendingActivations_c
9  end
10 if violatedActivations_c is not defined then
11 |  Define the map violatedActivations_c
12 end
13 if a ∉ activationsCounter_c.keys then
14 |  activationsCounter_c.put(a, 1)
15 |  foreach l ∈ L.vals do
16 |  |  acts ← activationsCounter_c.get(l)
17 |  |  if acts > 1 then
18 |  |  |  violatedActivations_c.put((l, a), acts − 1)
19 |  |  else
20 |  |  |  violatedActivations_c.put((l, a), 0)
21 |  |  end
22 |  |  pendingActivations_c.put((l, a), 0)
23 |  |  pendingActivations_c.put((a, l), 1)
24 |  |  violatedActivations_c.put((a, l), 0)
25 |  end
26 else
27 |  activationsCounter_c.put(a, activationsCounter_c.get(a) + 1)
28 |  foreach (k1, k2) ∈ pendingActivations_c.keys do
29 |  |  if k2 = a then   /* a equals to the second activity
   |  |  name */
30 |  |  |  pendingActivations_c.put((k1, a), 0)
31 |  |  else if k1 = a then  /* a equals to the first activity
   |  |  name */
32 |  |  |  pends ← pendingActivations_c.get((a, k2))
33 |  |  |  pendingActivations_c.put((a, k2), pends + 1)
34 |  |  |  if pends > 0 then
35 |  |  |  |  viols ← violatedActivations_c.get((a, k2))
36 |  |  |  |  violatedActivations_c.put((a, k2), viols + 1)
37 |  |  |  end
38 |  |  end
39 |  end
40 end
41 if (c, a) ∉ L then  L.put((c, a))
```

Algorithm 5: Discovery algorithm for chain response constraints

Input: $e = (c, a, t)$ the event to be processed (c is the case id of the event, a is the activity name, t is the timestamp)
conf: approximate algorithm configuration (either LC or LCB, with the corresponding configuration: ϵ or $\mathcal{B}$)

```
1  if L is not defined then
2  |  Define the empty map L
3  end
4  if activationsCounter_c is not defined then
5  |  Define the map activationsCounter_c
6  end
7  if lastActivityInCase_c is not defined then
8  |  Define the variable lastActivityInCase_c
9  end
10 if fulfilledActivations_c is not defined then
11 |  Define the map fulfilledActivations_c
12 end
13 if a ∉ activationsCounter_c.keys then
14 |  activationsCounter_c.put(a, 1)
15 |  if not lastActivityInCase_c.isEmpty then
16 |  |  foreach l ∈ L.vals do
17 |  |  |  if l = lastActivityInCase_c then   /* l equals to the
   |  |  |  last activity in the case */
18 |  |  |  |  fulfilledActivations_c.put((l, a), 1)
19 |  |  |  else
20 |  |  |  |  fulfilledActivations_c.put((l, a), 0)
21 |  |  |  end
22 |  |  end
23 |  end
24 else
25 |  activationsCounter_c.put(a, activationsCounter_c.get(a) + 1)
26 |  foreach (k1, k2) ∈ fulfilledActivations_c.keys do
27 |  |  if (k1 = lastActivityInCase_c) ∧ (k2 = a) then
28 |  |  |  fulfils ← fulfilledActivations_c.get((k1, a))
29 |  |  |  fulfilledActivations_c.put((k1, a), fulfils + 1)
30 |  |  end
31 |  end
32 end
33 lastActivityInCase_c ← a
34 if (c, a) ∉ L then  L.put((c, a))
```

the complexity of Algorithm 5 is linear in the number of activity names (lines 16-22) and in the number of fulfilled activations (lines 26-31). Therefore, the algorithms are scalable, since the number of activity names is typically finite and rather small and the same assumption also holds for the number of pending and fulfilled activations. In addition, as demonstrated in our experimentation, reported in Section 7, the computational complexity of the discovery algorithms is suitable to guarantee their applicability in real-world scenarios.

6 Implementation

The complete approach has been implemented as a plug-in of the process mining tool ProM,[2] called *StreamDeclare-Discovery*.[3] This plug-in is able to connect to a stream source that emits events (via a TCP connection) and to mine them, in order to keep an updated version of a discovered Declare model. The implemented software is able to show both a graphical representation of the top ten constraints (with the highest support) and a list of all discovered constrains (above a minimum support threshold specified by the user). All the elements in the graphical visualization evolve over time, according to the events observed into the stream. A detailed description of the implementation of this visualizer is reported in [26]. See http://youtu.be/9gbrhkSfRTc for a video demonstration of the visualizer.

7 Experimentation

For our experimental phase, we tested three case studies, two with synthetic data, dealing with the sudden drift case and the gradual drift case respectively and one with real data. The entire experimentation was conducted on a machine with an 8-core Intel i7 processor equipped with 32GB of RAM, Linux 3.10 and Java 7. Each execution was bound to a single core limiting to 16GB the memory available to the Java Virtual Machine.

7.1 Periodical Sudden Drifts

In this section, we discuss the results of a set of experiments carried out using a stream containing periodical sudden drifts [27]. For this case study, we have generated

2. See http://www.processmining.org for information about ProM.
3. The source code of the plug-in is publicly available at https://svn.win.tue.nl/repos/prom/Packages/StreamDeclareDiscovery/Trunk.

two synthetic logs ($\mathcal{L}_1$ and $\mathcal{L}_2$) by modeling two variants of the insurance claim process described in [28] in CPN Tools[4] and by simulating the models. $\mathcal{L}_1$ contains 14,840 events and $\mathcal{L}_2$ contains 16,438 events. We merged the logs (eight alternations of $\mathcal{L}_1$ and $\mathcal{L}_2$) using the *Stream Package* in ProM.[5] The same package has been used to transform the resulting log into an event stream. The event stream contains 250,224 events and has several sudden concept drifts (one for every switch from $\mathcal{L}_1$ to $\mathcal{L}_2$).

Using the generated event stream, we have compared the effectiveness and the efficiency of the online discovery using the SW approach with respect to LC and LCB. In addition, we have also separately evaluated the event-based LC approach (LCe) and the traces-based LC approach (LCt) and, also, the event-based LCB approach (LCBe) and the traces-based LCB approach (LCBt). We discovered a Declare model (using all the standard Declare templates) every 10,000 processed events, i.e., we fixed an *evaluation point* every 10,000 events.

For evaluating the effectiveness of the approaches, we have used the *precision* and *recall* metrics [29]. To compute precision and recall, we assumed that the discovered Declare constraints could be classified into one of four categories, i.e., *i*) true-positive (T_P: correctly discovered); *ii*) false-positive (F_P: incorrectly discovered); *iii*) true-negative (T_N: correctly missed); *iv*) false-negative (F_N: incorrectly missed). Precision and recall are defined as

$$Precision = \frac{T_P}{T_P + F_P} \qquad Recall = \frac{T_P}{T_P + F_N}. \tag{1}$$

The *gold standard* used as reference is the set of all true positive instances. In our experiments, we have used as gold standards two Declare models ($\mathcal{M}_1$ and $\mathcal{M}_2$) discovered from $\mathcal{L}_1$ and $\mathcal{L}_2$ using the Declare Miner (available in ProM) and containing constraints satisfied in all the cases. Precision and recall have been evaluated at every evaluation point. To compute them, we have used either $\mathcal{M}_1$ or $\mathcal{M}_2$ as gold standard based on whether the evaluation point corresponds to an event belonging to $\mathcal{L}_1$ or $\mathcal{L}_2$, respectively. At every evaluation point, we selected the constraints with fulfillment ratio equal to 1 among the candidate constraints generated by the LC and LCB approaches and discovered (with the Declare Miner) the constraints with support 100% in the SW approach. Then, we compared these sets of constraints with the gold standards. A discovered constraint is classified as a true-positive or as a false-positive depending on whether it belongs to the gold standard or not. A constraint that belongs to the gold standard but that has not been discovered is a false-negative. [6]

4. See http://cpntools.org for information about CPN Tools.

5. The source code of the package is publicly available at https://svn.win.tue.nl/repos/prom/Packages/Stream/Trunk.

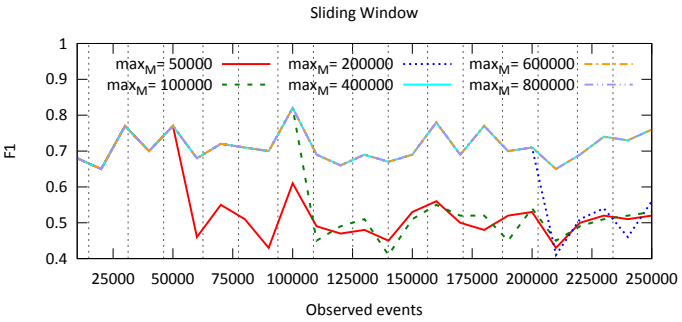


Fig. 3. Trend of F_1 for SW computed for different evaluation points with different configuration of the maximum available memory (max_M). Vertical dotted lines indicate concept drifts.

We have evaluated the efficiency of the proposed approaches in terms of events processed per second and amount of required memory. For LC and LCB, the space used is the number of tuples stored by all the template replayers in the corresponding data structures (maps). For the LCB approaches, the available budget has been partitioned among all the template replayers.

Effectiveness and efficiency have been evaluated for different values of the maximum available memory (max_M) for SW, the available budget ($\mathcal{B}$) for LCBe, LCBt and for different values of the maximal approximation error (ϵ) for LCe, LCt.

7.1.1 Results for Precision and Recall

In Fig. 3, we show the trend of the harmonic mean of precision and recall

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2}$$

for the SW approach for different evaluation points and for different configurations of max_M (50,000; 100,000; 200,000; 400,000; 600,000; 800,000).

The plot shows that the F_1 value lies between 0.41 and 0.82. The lowest values are obtained with the lowest max_M (50,000; 100,000; 200,000). On the other hand, when there is more space available (i.e., max_M equals to 400,000; 600,000 and 800,000) the F_1 values are higher (in particular, the lines referring to the three highest values of max_M are overlapped). This can be explained considering that the log contains 250,000 events and, therefore, when max_M is lower than 250,000, some observations

6. As described in [30], it is possible to divide the metrics for the evaluation of the quality of the mined process models in model-to-log (through the "four competing dimensions") and model-to-model metrics. In our experimentation, since we have a gold standard available, we decided to use the model-to-model approach, i.e., we compare the mined model with the gold standard. This type of metrics provides more reliable results than a comparison based on model-to-log metrics (which are fundamental in case the gold standard is not available). In particular, we are not just stating that the mined model has a good quality according to some general standards, but we are measuring to what extent the mined model is exactly the same as the target one.

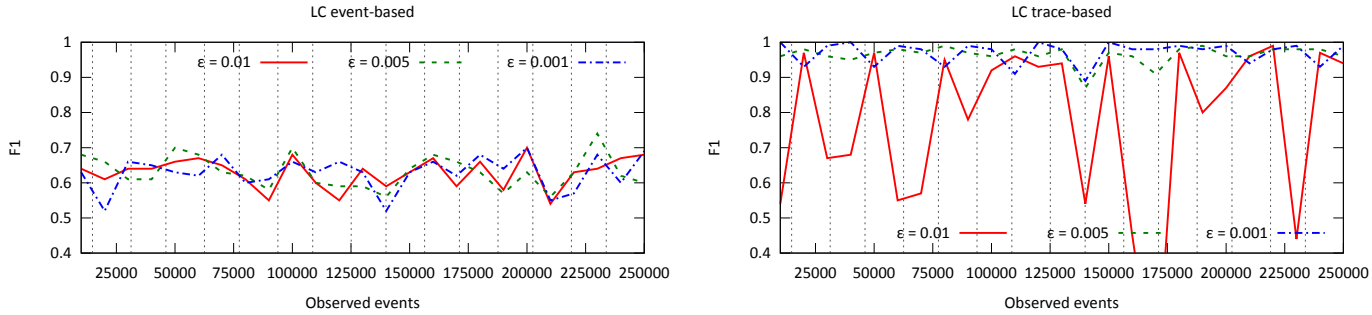


Fig. 4. Trend of F_1 for LC, computed for different evaluation points, with different configuration of the maximal approximation error (ϵ). The left-hand side plot reports the event-based approach; the right-hand side plot reports the trace-based approach. Vertical dotted lines indicate concept drifts.

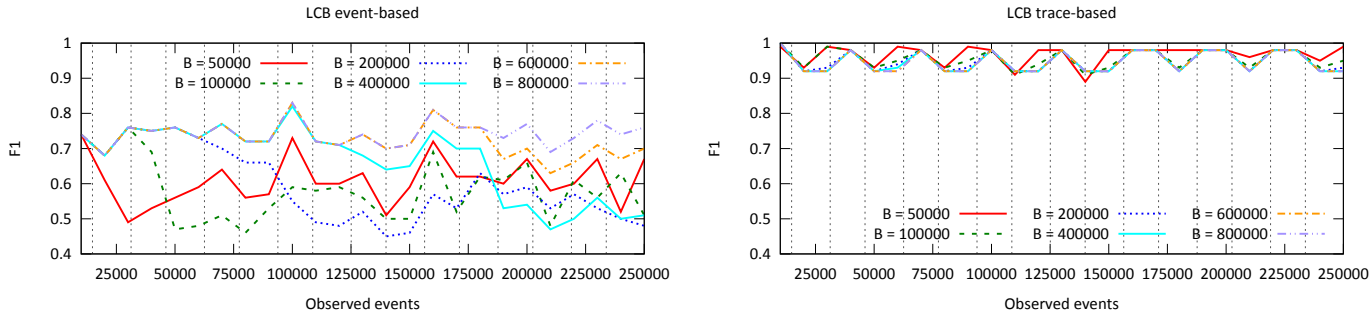


Fig. 5. Trend of F_1 for LCB, computed for different evaluation points, with different configuration of the maximum available memory ($\mathcal{B}$). The left-hand side plot reports the event-based approach; the right-hand side plot reports the trace-based approach. Vertical dotted lines indicate concept drifts.

have to be discarded. Note that, when the evaluation point corresponding to the maximum available memory is reached and old events must be discarded, the F_1 measure decreases.

Fig. 4 reports the trend of the F_1 measure, for the LC approach (both event- and trace-based) for different maximal approximation errors ϵ (0.01, 0.005 and 0.001). As expected, for the trace-based approach, the overall quality of the discovered model is better. This is due to the fact that, in this case, the discovery is performed only on cases that are completed and there is no noise deriving from the analysis of incomplete information. In particular, the F_1 measure of all configurations of the event-based approach lies between 0.5 and 0.7. The trace-based approach has, in general, very high values for F_1 (close to 1). Note that, for $\epsilon = 0.01$, the F_1 values corresponding to concept drifts decrease down to 0. This is due to the fact that, for high values of ϵ, the LC approach becomes more imprecise (this is the reason why F_1 is so unstable after every concept drift). Since for the trace-based approach we take into consideration only finished traces, it is very likely that, after a concept drift, there are only few or even no traces available for the evaluation of the candidate constraints.

Fig. 5 reports the trend of F_1 for LCBe and LCBt, for different values of the budget $\mathcal{B}$. For the LCBe approach, for higher values of the available budget, the F_1 trend goes below the threshold of 0.7 later. For LCBt, it is evident that lower values of the budget allow the approach to adapt more quickly after a concept drift (old behavior is "forgotten" more quickly). As in the LC case, also in this case, the trace-based approach reaches better F_1 values. Note that, differently from the SW approach, in this case, $\mathcal{B}$ does not represent the number of stored *events*. Instead, it refers to the number of *tuples* stored in memory for all the template replayers. For this reason, as shown on the left-hand side plot of Fig. 5, results may change even when the budget is higher than the total number of observed events.

In Fig. 6, we show a comparison of the F_1 trends for all the approaches (with the corresponding optimal configurations, $max_M = \mathcal{B} = 800,000$ and $\epsilon = 0.001$). LCBe is more accurate than LCe and SW. LCt and LCBt clearly outperform all the other approaches. LCt recovers more quickly than LCBt after a concept drift.

7.1.2 Performance Issues

The efficiency of the described approaches has been first evaluated comparing them in terms of number of events processed per second (Fig. 7). The values reported in the figure have been evaluated as the average over three runs.

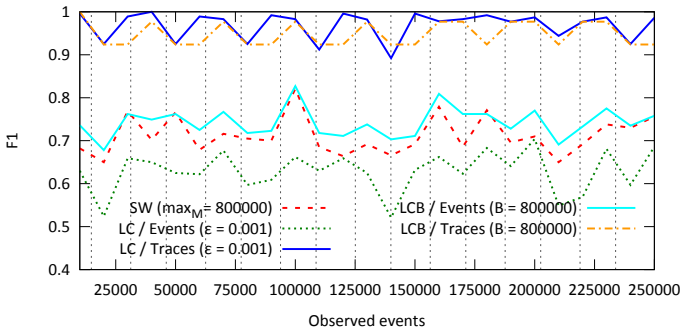


Fig. 6. Trend of F_1 for SW, LCe, LCt, LCBe, LCBt, for different evaluation points, with $max_M = \mathcal{B} = 800,000$ and $\epsilon = 0.001$. Vertical dotted lines indicate concept drifts.

As shown on the left-hand side of Fig. 7, the processing time required by the budget-parametric approaches (i.e., SW and LCB) increases with the budget and, therefore, the efficiency decreases. This phenomenon is due to the higher number of operations that must be performed when handling larger data structures. For higher values of the budget (more than 400,000) the processing time starts to be less sensible to the budget variations. Also note that the LCBt approach is more efficient than the SW approach for low values of the budget but it is less efficient for budget values higher than 200,000. In general, the performances of the LCB approaches decrease more quickly than the ones of the SW approach when the available budget increases.

For the LCe and LCt approaches, reported on the right-hand side of Fig. 7, we evaluated the number of events processed per second for different values of the maximal approximation error. The processing time for LCt is, in general, lower than the one for LCe and, also, the F_1 values for LCt are higher. In general, in terms of efficiency, the LC approaches are the best ones, since they do not impose any limitation on the memory consumption and the memory-related operations (such as iterating through the data structure to remove old items) are very time consuming. The plot in Fig. 8 reports the space usage for the LC approaches (both event- and trace-based). As one may predict, the required memory is higher for lower maximal approximation errors, since, in this case, it is necessary to keep more observations in memory. The traces-based approaches require more memory with respect to the events-based ones for the same ϵ values. This phenomenon is due to the necessity to store the entire set of events referring to the same case.

7.2 Gradual Drifts

The first case study tested only sudden drifts. However, sometimes, it may happen that drifts are gradual and a process moves from one behavior to another through intermediate steps. To test this scenario, we have considered two variants of the insurance claim process described in [28] and designed 6 additional models to represent the intermediate steps between the two models. We have simulated these models generating 8 logs $(\mathcal{L}'_1, \mathcal{L}_a, \ldots, \mathcal{L}_f, \mathcal{L}'_2)$ [27]. $\mathcal{L}'_1$ contains 139,938 events, $\mathcal{L}'_2$ contains 128,696 events and $\mathcal{L}_a, \ldots, \mathcal{L}_f$ contain 77,231 events (altogether). Using the *Stream Package*, we have generated an event stream containing 345,865 events. We have used as gold standards the Declare models $\mathcal{M}'_1$ and $\mathcal{M}'_2$ discovered from $\mathcal{L}'_1$ and $\mathcal{L}'_2$ and $\mathcal{M}_a, \ldots, \mathcal{M}_f$ discovered from $\mathcal{L}_a, \ldots, \mathcal{L}_f$. $\mathcal{M}_a, \ldots, \mathcal{M}_f$ represent the intermediate steps to go from $\mathcal{M}'_1$ to $\mathcal{M}'_2$. Therefore, $\mathcal{M}'_1$ and $\mathcal{M}_a$ are very similar and the same happens for $\mathcal{M}_a$ compared to $\mathcal{M}_b$, for $\mathcal{M}_b$ compared to $\mathcal{M}_c$ and so on.

For the evaluation, we used the F_1 measure as described for the sudden drift case. The results are reported in Fig. 9. In this plot, the gray area indicates the gradual drift. In general, the observations made for the previous case study still hold. In particular, the trace-based approaches (both for LC and LCB) reach the highest values for F_1. The event-based approaches, instead, correspond to lower values for F_1 but perform, in general, better than the SW approach. Note that the drift causes a drop down of the F_1 measure for all the approaches but the event-based approaches are much slower than the trace-based approaches in recovering after the drift.

7.3 Applicability to Real Data

In order to assess the applicability of our approaches to real data, we verified their scalability when the number of activities in the process is extremely high and, therefore, the number of candidate constraints grows. Table 3 and 4 report the time efficiency (i.e., number of processed events per seconds) of LCe, LCt and LCBe and LCBt applied to the BPI Challenge (BPIC) 2011 log [13]. This log, pertaining to the treatment of patients diagnosed with cancer in a large Dutch academic hospital, contains 623 different activities and 150,291 events, distributed over 1,143 cases. This log can be considered as an extreme case, due to the very high number of different activity names and the number of different candidate constraints that can emerge.

The data reported in the two tables shows that the approaches can still be applied, although the overall performance decreases. To improve this aspects, several solutions may be adopted. For example, due to the structure of the online discovery scheme, it is possible to employ several processing units (e.g., one CPU core for each template replayer), or it might be possible to restrict the set of candidate constraints to the most interesting ones. This can be easily done by integrating this approach with approaches to discover frequent item sets like the one proposed in [9].

The analysis of a real-life event log at runtime provides the user with two types of diagnostics. First, the user can "query" the miner about the validity, at a certain point in time during the process execution, of a specific constraint of interest. In addition, the provided list of valid constraints is sorted based on interestingness metrics (see, [9] for more information about these metrics).

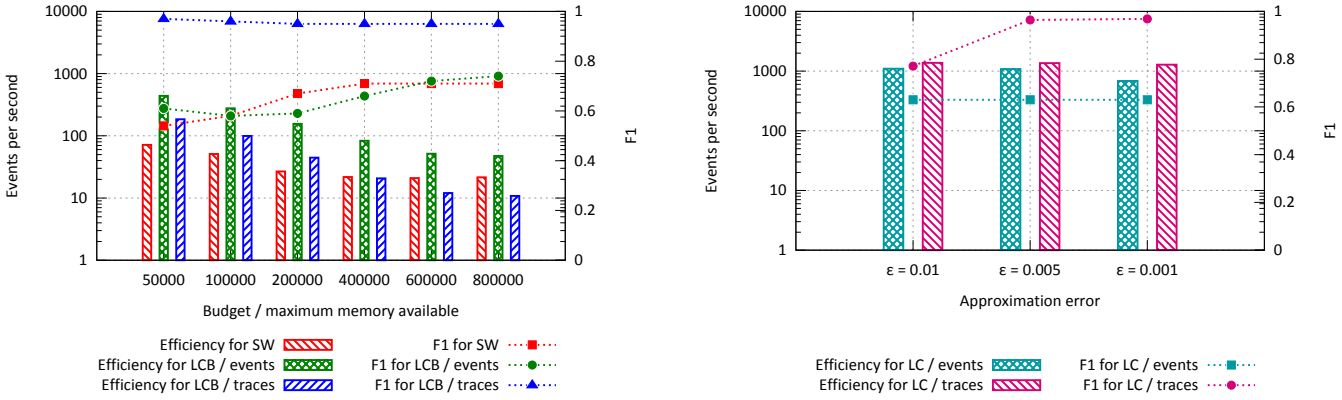


Fig. 7. Efficiency evaluation of all the presented approaches. Both plots report the efficiency in terms of number of events that the miner is able to process per second (logarithmic scale). The secondary y-axis on the right-hand side of each plot reports the F_1 measure of each approach for the given configuration.

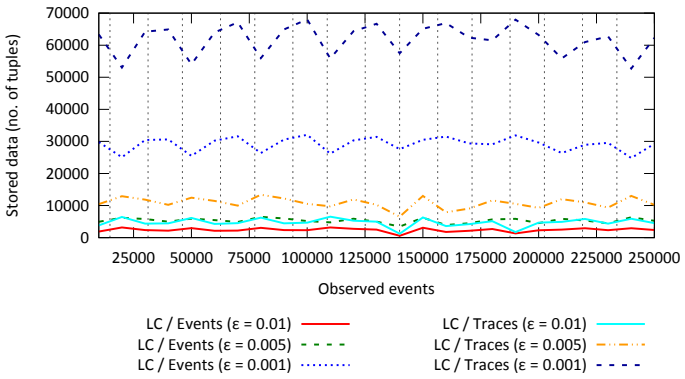


Fig. 8. Space requirements (in terms of Lossy Counting tuples) for LC approaches (both event- and trace-based) with different configuration of maximal approximation error (ϵ). Vertical dotted lines indicate concept drifts.

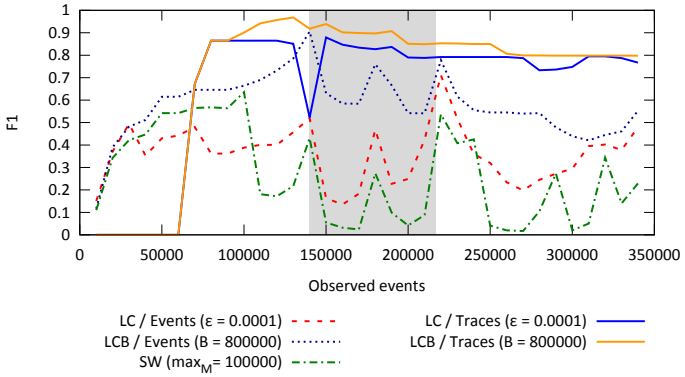


Fig. 9. Trend of F_1 for SW, LCe, LCt, LCBe, LCBt, for different evaluation points, with $\mathcal{B} = 800,000$, $\epsilon = 0.0001$ and $max_M = 100,000$. The gray area indicates where the gradual drift occurs.

Therefore, in practice, only the top-scored constraints according to these metrics are shown to the user so that the discovered models can be quickly read and understood.

TABLE 3
Efficiency (i.e., the number of processed events per second) evaluation of LCe and LCt for the BPIC 2011.

Approximation error (ϵ)	LC / Events	LC / Traces
0.01	240.47	568.00
0.005	147.81	316.25
0.001	59.23	91.47

TABLE 4
Efficiency (i.e., the number of processed events per second) evaluation of LCBe and LCBt for the BPIC 2011.

Budget $\mathcal{B}$	LCB / Events	LCB / Traces
50,000	28.45	69.95
100,000	22.91	27.08
200,000	23.35	20.16
400,000	22.86	16.47
600,000	22.84	18.77
800,000	23.25	19.20

8 Related Work

Process mining consists of a set of techniques allowing for the extraction of structured process descriptions from a set of recorded process executions. Starting from [31], several techniques have been proposed in this field. They can be classified as pure algorithmic, heuristic and genetic [4]. More recently, several works have been published starting from the awareness that techniques based on declarative languages are suitable for the discovery of unpredictable, variable processes working in turbulent environments [6], [7], [8], [9], [32].

Process stream mining consists in 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 [4], these techniques all work on static event logs and not on

streaming data. Only few works in process mining aim at mining event streams. In [33], [34], 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.

Another interesting contribution to the analysis of evolving processes is given in [28]. The approach, based on statistical hypothesis tests, aims at detecting concept drifts and identifying the regions of change in a process. In [35], [36], the authors propose an adaptation of the Heuristics Miner (a well known control-flow discovery algorithm) to data streams. The final aim of these works is to extract a procedural control-flow from an event stream. In [37], an incremental approach for translating transition systems into Petri nets is described. The work reported in this paper inspired our previous work in [32].

In this paper, we use algorithms for data stream mining. As already mentioned, they are divided into two categories, i.e., data- and task-based [18]. The first ones use only a fragment of the entire dataset and mainly consist in sampling [38], load shedding [39], sketching and aggregation techniques [40]. Some of these techniques are based on the idea of randomly selecting items or stream portions. However, since the dataset size is unknown, it becomes hard to define the number of items to collect. Moreover, it may be possible that some of the items that are ignored were actually meaningful. Other approaches, like aggregation, are based on summarization techniques and consider measures such as mean and variance. In these approaches, problems arise when the data distribution contains many fluctuations.

The task-based techniques try to achieve time and space efficient solutions. The most common are approximation algorithms [38], sliding window and algorithm output granularity [18]. The first ones aim at extracting an approximate solution and support for defining error bounds on the procedure to obtain an accuracy measure. Sliding window is based on the idea that users are more interested in most recent data than in the old ones. The algorithm output granularity controls, via three factors that depend on the available memory, the input/output rate of mining taking into account the data stream rate and the computational resource availability.

None of the above works provides a comprehensive online approach for the discovery of declarative process models as proposed in this paper.

9 Conclusion

This paper proposes a framework for the online discovery of declarative process models from streaming event data producing *(i)* a runtime updated picture of the process behavior in terms of LTL-based business constraints; *(ii)* meaningful information about the most significant concept drifts occurring during the process execution. This approach consists of the combination of algorithms for stream mining and algorithms for the online discovery of Declare models.

Different stream mining approaches (SW, LCe, LCt, LCBe and LCBt) have been implemented and experimented. In our experimentation, we have evaluated the effectiveness and the efficiency of the online discovery (using the different approaches) applied to large and complex streams containing drifts of different types. The experiments show that LCBe is more accurate than LCe and SW and that the overall quality of the discovered model is higher for trace-based approaches.

In order to assess the applicability of the proposed approaches to real data, we verified their scalability on the event log provided for the BPI Challenge 2011 pertaining to an application process within a Dutch academic hospital. The results confirm the applicability of LC and LCB approaches also in the extreme case of an event stream including several different activity names in which a large number of candidate constraints must be taken into consideration.

As future work, we will conduct a wider experimentation on several case studies also in real-life scenarios in order to provide a better assessment of the proposed approaches. In addition, it may be useful to take into consideration other information, like data and time in the discovery task to enrich the outcome provided to the user. Finally, more sophisticated mechanisms for the concept drift detection could also be integrated in our framework like those presented in [41], [42].

Acknowledgements

Andrea Burattin and Alessandro Sperduti are supported by the Eurostars-Eureka project PROMPT (E!6696). The research of Fabrizio M. Maggi has received funding from the Estonian Research Council and by ERDF via the Estonian Centre of Excellence in Computer Science. The authors would like to thank Francesca Rossi and Paolo Baldan for their advice.

References

[1] S. Madden, "From Databases to Big Data," *IEEE Internet Computing*, vol. 16, no. 3, pp. 4–6, May 2012.

[2] P. Russom, "Big Data Analytics," *October*, vol. 19, p. 40, 2011. [Online]. Available: http://faculty.ucmerced.edu/frusu/Papers/Conference/2012-sigmod-glade-demo.pdf

[3] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. Hung Byers, "Big Data: The Next Frontier for Innovation, Competition, and Productivity," McKinsey Global Institute, Tech. Rep. June, 2011.

[4] W. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

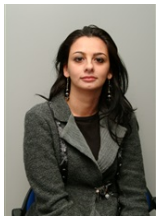
[5] E. Lamma, P. Mello, F. Riguzzi, and S. Storari, "Applying inductive logic programming to process mining," in *Inductive Logic Programming*, 2008, vol. 4894, pp. 132–146.

[6] F. Chesani, E. Lamma, P. Mello, M. Montali, F. Riguzzi, and S. Storari, "Exploiting Inductive Logic Programming Techniques for Declarative Process Mining," *ToPNoC*, vol. 5460, pp. 278–295, 2009.

[7] C. Di Ciccio and M. Mecella, "Mining constraints for artful processes," in *Proc. of BIS*, ser. LNBIP. Springer, 2012, pp. 11–23.

[8] F. Maggi, A. Mooij, and W. van der Aalst, "User-guided discovery of declarative process models," in *Proc. of CIDM*. IEEE, 2011, pp. 192–199.
[9] F. Maggi, J. Bose, and W. van der Aalst, "Efficient discovery of understandable declarative models from event logs," in *CAiSE*, 2012, pp. 270–285.
[10] L. Golab and M. T. Özsu, "Issues in Data Stream Management," *ACM SIGMOD Record*, vol. 32, no. 2, pp. 5–14, Jun. 2003.
[11] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst, "Declare: Full support for loosely-structured processes," in *EDOC*, 2007, pp. 287–300.
[12] F. M. Maggi, A. Burattin, M. Cimitile, and A. Sperduti, "Online process discovery to detect concept drifts in ltl-based declarative process models," in *CoopIS*, 2013, pp. 94–111.
[13] 3TU Data Center, "BPI Challenge 2011 Event Log," 2011, doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54.
[14] C. W. Günther, "XES Standard Definition," www.xes-standard.org, 2009.
[15] C. Aggarwal, *Data Streams: Models and Algorithms*, ser. Advances in Database Systems. Boston, MA: Springer US, 2007, vol. 31.
[16] 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.
[17] G. Widmer and M. Kubat, "Learning in the Presence of Concept Drift and Hidden Contexts," *Machine Learning*, vol. 23, no. 1, pp. 69–101, 1996.
[18] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining Data Streams: a Review," *ACM Sigmod Record*, vol. 34, no. 2, pp. 18–26, Jun. 2005.
[19] J. a. Gama, *Knowledge Discovery from Data Streams*, ser. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. Chapman and Hall/CRC, May 2010, vol. 20103856.
[20] W. van der Aalst, M. Pesic, and H. Schonenberg, "Declarative Workflows: Balancing Between Flexibility and Support," *Computer Science - R&D*, pp. 99–113, 2009.
[21] M. Montali, M. Pesic, W. M. P. van der Aalst, F. Chesani, P. Mello, and S. Storari, "Declarative Specification and Verification of Service Choreographies," *ACM Transactions on the Web*, vol. 4, no. 1, 2010.
[22] O. Kupferman and M. Vardi, "Vacuity Detection in Temporal Model Checking," *Int. Journal on Software Tools for Technology Transfer*, pp. 224–233, 2003.
[23] A. Burattin, F. Maggi, W. van der Aalst, and A. Sperduti, "Techniques for a Posteriori Analysis of Declarative Processes," in *EDOC*, 2012, pp. 41–50.
[24] G. S. Manku and R. Motwani, "Approximate Frequency Counts over Data Streams," in *VLDB*, 2002, pp. 346–357.
[25] G. Da San Martino, N. Navarin, and A. Sperduti, "A Lossy Counting Based Approach for Learning on Streams of Graphs on a Budget," in *IJCAI*. AAAI Press, 2012, pp. 1294–1301.
[26] A. Burattin, F. Maggi, and M. Cimitile, "Lights, Camera, Action! Business Process Movies for Online Process Discovery," in *TAProViz*, 2014.
[27] A. Burattin, "Artificial datasets for online Declare discovery." [Online]. Available: http://dx.doi.org/10.5281/zenodo.19187
[28] R. J. C. Bose, "Process Mining in the Large: Preprocessing, Discovery, and Diagnostics," Ph.D. dissertation, Eindhoven University of Technology, 2012.
[29] A. Rozinat, A. K. A. De Medeiros, C. W. Günther, A. J. M. M. Weijters, and W. M. P. Van Der Aalst, "The need for a process mining evaluation framework in research and practice: position paper," ser. BPM 2007, pp. 84–89.
[30] A. Burattin, *Process Mining Techniques in Business Environments*. Springer International Publishing, 2015.
[31] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining Process Models from Workflow Logs," in *EDBT1*, ser. LNCS, vol. 1377. Springer, 1998, pp. 469–483.
[32] F. Maggi, R. Bose, and W. van der Aalst, "A knowledge-based integrated approach for discovering and repairing declare maps," in *CAiSE*, 2013.
[33] E. Kindler, V. Rubin, and W. Schäfer, "Incremental Workflow Mining Based on Document Versioning Information," in *ISPW*. Springer Verlag, 2005, pp. 287–301.
[34] E. Kindler, V. Rubin, and W. Schafer, "Incremental Workflow Mining for Process Flexibility," in *BPMDS*, 2006, pp. 178–187.
[35] A. Burattin, A. Sperduti, and W. M. P. van der Aalst, "Heuristics Miners for Streaming Event Data," *ArXiv CoRR*, Dec. 2012.
[36] A. Burattin, A. Sperduti, and W. van der Aalst, "Control-flow discovery from event streams," in *CEC*, July 2014, pp. 2420–2427.
[37] A. Sharp and P. McDermott, *Workflow Modeling: Tools for Process Improvement and Application Development*, 2nd ed. Artech House Publishers, 2008.
[38] W. Hu and B. Zhang, "Study of sampling techniques and algorithms in data stream environments," in *FSKD*, 2012, pp. 1028–1034.
[39] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker, "Load shedding in a data stream manager," in *VLDB*. VLDB Endowment, 2003, pp. 309–320.
[40] J. Considine, F. Li, G. Kollios, and J. Byers, "Approximate aggregation techniques for sensor databases," in *ICDE*, 2004, pp. 449–460.
[41] R. Bose, W. van der Aalst, I. Zliobaite, and M. Pechenizkiy, "Dealing With Concept Drifts in Process Mining," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 154–171, Jan 2014.
[42] J. Carmona and R. Gavald, "Online techniques for dealing with concept drift in process mining," in *Advances in Intelligent Data Analysis XI*. Springer Berlin Heidelberg, 2012, pp. 90–102.

Andrea Burattin is working as a post-doc at the University of Innsbruck, Austria. Previously, he has been working as a post-doc at the University of Padua. In 2013, he received his Ph.D. degree in Computer Science from the University of Bologna and the University of Padua. The IEEE Task Force on Process Mining awarded the "Best Process Mining Dissertation Award" for 2012-2013 to his Ph.D. thesis. The thesis has then been published as a Springer monograph entitled "Process Mining Techniques in Business Environments". During his Ph.D., he spent several months at the Technical University of Eindhoven.



Marta Cimitile is an Assistant Professor at the Unitelma Sapienza University in Rome (Italy). She received her Ph.D. degree in Computer Science in 2008 from the University of Bari. She has collaborated with the University of Bari from April to October 2004. Her main research topic is in the field of data and process mining. In the last years, she was involved in several industrial and research projects and was author of several publications about these topics. She serves as a program committee member of several conferences and workshops in the business process management field.

Fabrizio M. Maggi received his Ph.D. degree in Computer Science from the University of Bari in 2010 and after a period at the Architecture of Information Systems (AIS) research group - Department of Mathematics and Computer Science - Eindhoven University of Technology, he is currently a Senior Researcher at the Software Engineering Group - Institute of Computer Science - University of Tartu. His Ph.D. dissertation was entitled "Process Modelling, Implementation and Improvement" and his areas of interest have included in the last years business process management, service-oriented computing and software engineering.

Alessandro Sperduti received his Ph.D. degree in Computer Science from the University of Pisa and since 2002 he is a Full Professor at the University of Padua. He has been chair of the Data Mining and Neural Networks Technical Committees of IEEE CIS and Associate Editor of IEEE TNNLS. He is currently Action Editor for the journals AI Communications, Neural Networks, Theoretical Computer Science (Section C). His research interests include machine learning, neural networks, learning in structured domains, data and process mining.