Learning Temporal Properties from Event Logs via Sequential Analysis

Francesco Chiariello
IRIT, ANITI, University of Toulouse, France

Abstract

In this work, we present a novel approach to learning Linear Temporal Logic (LTL) formulae from event logs by leveraging statistical techniques from sequential analysis. In particular, we employ the Sequential Probability Ratio Test (SPRT), using Trace Alignment to quantify the discrepancy between a trace and a candidate LTL formula. We then test the proposed approach in a controlled experimental setting and highlight its advantages, which include robustness to noise and data efficiency.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Applied computing → Business process management

Keywords and phrases Process Mining, Declarative Process Discovery, Trace Alignment, Sequential Analysis

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.14

Funding This project has been funded by the French government as part of France 2030 (Grant agreement n°ANR-19-PI3A-0004) and by the European Union - Next Generation EU as part of the France Relance.

1 Introduction

Process Mining (PM) [36] is an interdisciplinary field at the intersection of Data Mining and Business Process Management (BPM) [40]. Its goal is to gain insight into operational processes by analyzing the associated event logs. An operational process, or process for short, corresponds to the series of activities an organization performs to accomplish its routine tasks, such as delivering a particular service or product. In enacting a process, an organization generates sequential data that an information system stores in the form of an event log. Therefore, an event log keeps track of all the activities performed during several task executions, also called traces. A fundamental problem of PM is the one of learning models of processes from event logs, also known as Process Discovery [35]. To achieve this, one may consider different formalisms to model the processes, including UML activity diagrams [16], Business Process Model and Notation [23], and Petri nets [34, 38]. Those formalisms are imperative in that they prescribe, at each step, the activities that can be performed and provide therefore an easy-to-follow recipe for process execution. However, they have known limitations due to the tendency of over-constraining the process [37] and the lack of interpretability. Declarative models [15], by contrast, consist of constraints over process executions. After that, every execution not violating such constraints is admitted. The most widespread declarative process specification language is Declare [28], which consists of a set of templates that allow to specify temporal constraints over the activities of the process. Declare semantics can be grounded into Linear Temporal Logic on process traces (LTL_p) [19], which allow us to exploit efficient automata-theoretic techniques [7]. For this reason, the use of LTL_p for the specification of processes is gaining increasing traction.

An obstacle to learning models of processes from event logs is that they often contain noise. Such noise may be caused by errors in the activities performed. Besides, it may also arise from logging errors, which result in activities being recorded in the wrong order,

duplicated, or omitted entirely (thus creating gaps). Finally, noise can also occur because events are inferred from low-level data using activity recognition techniques, rather than being directly recorded [2]. The presence of noise in the event logs makes it difficult to learn temporal patterns and calls for the use of statistical techniques able to handle such noise. Another challenge is that event logs are streams of data. This means that traces are not all available from the beginning, rather they are continuously collected as the process is executed. Ideally, the log should be processed sequentially, as new traces arrive, allowing conclusions about the satisfaction of temporal properties to be drawn immediately, and only waiting for new evidence if necessary.

In this paper, we address these issues by proposing a method based on sequential analysis to learn temporal properties of a process from its corresponding event log. Sequential analysis [21] consists of performing hypothesis tests where a stopping rule is used to halt the sampling process as soon as the collected evidence is sufficient to accept or reject the hypothesis under examination. In particular, we consider a test known as the Sequential Probability Ratio Test (SPRT) [39], originally developed within the domain of quality control in manufacturing. Given an input *lot*, the general idea of the test is to incrementally sample items from the lot and count the number of defects they have. If, at any step, the total number of defects falls below a specified acceptance threshold, the lot is accepted. Conversely, if it exceeds a rejection threshold (which is set higher than the acceptance threshold), the lot is rejected. Otherwise, sampling continues. Naturally, as the total number of defects increases with the number of items examined, both the acceptance and the rejection threshold progressively increase. In our approach, we put forward trace alignment as a way of quantifying the number of defects of a trace with respect to a given formula, after that SPRT can directly be applied to our learning setting. Trace alignment [12] refers to computing a minimal number of alignments, i.e., of modifications of the trace to make it satisfy the formula. In our approach, we compute alignments by framing the problem as cost-optimal planning [10] and solving it using an off-the-shelf AI planner.

Our approach has two main advantages. First, being based on a statistical approach, makes it robust to the noise naturally occurring in the event logs. Second, its ability to make a decision as soon as enough evidence arises, makes it very data-efficient.

The remainder of the paper is organized as follows: in Section 2 we provide the background and basic notation. Then, in Section 3 we describe the proposed method to discover temporal properties from event logs. In Section 4 we perform and discuss controlled experiments to demonstrate the methods's application. In Section 5 we discuss related work. Finally, Section 6 concludes the paper and points out directions for future work.

2 Background

We start by recalling relevant notions of Linear Temporal Logic over Process Traces (LTL_p) [19]. Then, we describe Declare and show how its templates can be grounded into LTL_p. Finally, we describe Trace Alignment and Sequential Analysis.

2.1 Linear Temporal Logic on Process Traces

Let Σ be a set of propositional symbols, also called *activities*. A *process trace* is a non-empty sequence $\pi \in \Sigma^+$ of activities. An *event* is any occurrence of an activity in the trace. A process trace can be thought of as representing an execution of a process. An *event log* is then a sequence of process traces. Process traces are therefore ordered according to their occurrence (i.e., to the occurrence of their last activity, since multiple process instances may

run in parallel). The same trace may appear multiple times in a log. This is expected since they represent different executions of the same routine task. It is important to note that process traces differ from the traces encountered in Linear Temporal Logic (LTL) [29] in two ways: (i) they are finite, and (ii) exactly one activity occurs per each instant. This last characteristic makes process traces suitable models for logics that reason over actions, rather than over states.

Syntax

The syntax of LTL_p is the same as LTL. An LTL_p formula φ over Σ is defined according to the following grammar:

$$\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U} \varphi_2,$$

with $a \in \Sigma$. The intuitive meaning of the temporal operators "next" X and "until" U is as follows. The formula $\mathsf{X}\varphi$ means that at the next time instant, φ holds. The formula $\varphi_1 \mathsf{U} \varphi_2$ means that at a certain instant φ_2 holds and up to that point φ_1 holds. We assume common propositional and temporal abbreviations. In particular, for temporal operators, we define the "eventually" operator $\mathsf{F}\varphi \equiv \top \mathsf{U} \varphi$, the "always" operator $\mathsf{G}\varphi \equiv \neg\mathsf{F}\neg\phi$, and the "weak until" operator $\varphi \mathsf{W} \varphi' \equiv \mathsf{G}\varphi \vee \varphi \mathsf{U} \varphi'$.

Semantics

The semantics of LTL_p is defined on process traces. Let φ be an LTL_p formula, $\pi = \pi_1 \pi_2 \cdots \pi_{|\pi|}$ a process trace, and $1 \leq i \leq |\pi|$ a time instant. We say that π satisfies φ at time i, and we write $\pi, i \models \varphi$, according to the following definition:

- $\pi, i \models a$ iff $a = \pi_i$;
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$;
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \mathsf{X}\varphi$ iff $i < |\pi|$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \varphi_1 \mathsf{U} \varphi_2$ iff $\exists j$, $i \leq j \leq |\pi|$, s.t. $\pi, j \models \varphi_2$ and for $k = i, i+1, \ldots, j-1$, $\pi, k \models \varphi_1$.

We say that π is a *model* for φ if $\pi, 1 \models \varphi$, denoted as $\pi \models \varphi$.

Note that, since LTL_p works on traces that are finite, and in analogy with the definitions in Linear Temporal Logic on Finite Trace (LTL_f) [17, 11] the "next" operator needs to explicitly require the existence of a next time instant.

Automata-Based Representation

Each LTL_p formula φ over Σ can be associated a finite-state automaton $\mathcal{A}(\varphi)$ over the same alphabet Σ such that for any trace π it holds that $\pi \models \varphi$ iff π, is accepted by $\mathcal{A}(\varphi)$ [7]. Fig 1 shows the minimal automaton associated to the formula $\mathit{Response}(a, b) = \mathsf{G}(a \rightarrow \mathsf{F}b)$, saying that whenever a is performed, then b is performed afterward. Note that transitions are directly labeled with activities in Σ, resulting in an alphabet that is exponentially smaller compared to what we would have if we were interpreting the formula in LTL_f [11]. However, if we identify a process trace with a *simple finite trace* [18], i.e. a finite trace where each propositional interpretation is a singleton, one can use LTL_f to check properties of process traces. Additionally, one can also check whether a trace is a simple trace by suitably modifying the LTL_f formula [8], or by directly modifying the resulting automaton with the addition of a sink state [7].

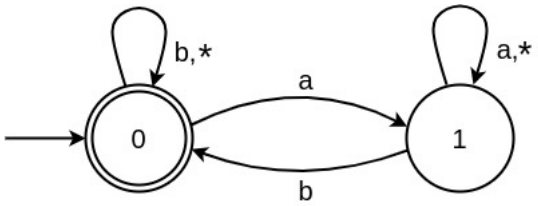

Figure 1 Automaton for the formula $Response(a, b) = \mathsf{G}(a \rightarrow \mathsf{F}b)$. Here the asterisk stands for any activity other than the ones appearing in the formula.

2.2 Declare

Declare [28, 15] is the most common declarative process specification language. It consists of a set of templates that allow to easily specify the temporal constraints of the process. A constraint is any instantiation of the variables in the template with process activities. A process model is then a set of constraints. Table 1 shows how to write some Declare templates (in particular, the ones that will be used later in the experiments) as LTL_p formulae. Be aware that many errors in such encodings are present in the literature. This is due to the difficulty of working with temporal specifications [22]. These formulae have been carefully double-checked by exploiting the automata representation, which sometimes results easier to understand, as well as log generation techniques [5, 6].

Table 1 Some Declare templates and their corresponding LTL_p formula.

Template	**LTL_p**
$Init(a)$	a
$Exactly2(a)$	$\neg a\mathsf{U}(a \wedge \mathsf{X}(\neg a\mathsf{U}(a \wedge \neg\mathsf{X}(\mathsf{F}a))))$
$Response(a, b)$	$\mathsf{G}(a \rightarrow \mathsf{F}b)$
$RespondedExistence(a, b)$	$\mathsf{F}a \rightarrow \mathsf{F}b$
$AlternateResponse(a, b)$	$\mathsf{G}(a \rightarrow\mathsf{X}(\neg a\mathsf{U}b))$
$Precedence(a, b)$	$(\neg b)\mathsf{W}a$
$ChainPrecedence(a, b)$	$\mathsf{G}(\mathsf{X}b \rightarrow a) \wedge \neg b$
$Choice(a, b)$	$\mathsf{F}(a \vee b)$
$ExclusiveChoice(a, b)$	$\mathsf{F}(a \vee b) \wedge \neg(\mathsf{F}a \wedge \mathsf{F}b)$
$CoExistence(a, b)$	$\mathsf{F}a \leftrightarrow \mathsf{F}b$

$Init(a)$ says that any trace of the process must start with the activity a (here a stands for a generic activity and so we talk about templates). $Exactly2(a)$ says that exactly two occurrences of a must be present in the trace. This template can be generalized to consider any number of occurrences. $Init(a)$ and $Exactly2(a)$ are unary templates since they express constraints on one activity only. $Response(a, b)$ says that whenever a is executed, b must be executed afterwards. $RespondedExistence(a, b)$ says that whenever a is executed, then b must be executed (regardless of whether it appears before or after a). $AlternateResponse(a, b)$ says that every execution of a must be followed by b, without any other a in between. For those kinds of binary templates, a and b are sometimes referred to as the *activation* and *target* activity, since the occurrence of a triggers checking the occurrences of b. If no activation is present, then the constraints are automatically (vacuously) satisfied. $Precedence(a, b)$ says that b can be executed only if a has been executed before. $ChainPrecedence(a, b)$ says that a must be executed immediately before any execution of b. A common error in literature while encoding this template in temporal logics with only future operators is to forget that b

cannot be executed in the first instant. $Choice(a, b)$ says that eventually one among a and b must be executed. $Choice(a, b)$ further requires that it is not possible to execute them both. Finally, $CoExistence(a, b)$ says that either a and b are both executed, or none of them is executed.

2.3 Trace Alignment

The trace alignment problem is defined as follows. Given a trace π and an LTL_p formula φ, find a minimum number of alignments that makes π a model of φ [12]. An alignment refers to the removal or addition of an event in the trace. Such a problem can be solved by compiling it into cost-optimal planning [10] and then using any off-the-shelf planner supporting cost optimization. One can visualize the application of the alignments as a text cursor moving from the left to the right of π and that can add an event behind it (as is done by pressing a character key on a keyboard, where the cursor automatically moves forward) or remove one in front of it (as with the delete key). The goal of the planning problem is then to have the cursor reach the right side of the trace (possibly adding some other event at this extreme) and the resulting trace be accepted by $\mathcal{A}(\varphi)$. Note that we have here three kinds of actions since moving forward corresponds to an action of zero cost. This is the reason why we need to resort to cost-optimal planning.

2.4 Sequential Analysis

Sequential analysis [21] (not to be confused with sequence analysis [27]) involves conducting hypothesis tests using a stopping rule to halt sampling once there is sufficient evidence to either accept or reject the hypothesis being tested. One such test is the Sequential Probability Ratio Test (SPRT) [39], originally developed within the domain of quality control in manufacturing. Given a lot, we test the products one by one. For $n = 0, 1, 2, \ldots$, let d_n be the total number of defects found after controlling the first n products (for $n = 0$ we have clearly $d_0 = 0$). Given a strictly increasing sequence $\{A_n\}$ of *acceptance numbers*, and a strictly increasing sequence $\{R_n\}$ of *rejection numbers*, with $A_n < R_n$ for all $n \in \mathbb{N}$, the test is as follows. If at any point n, we have $d_n < A_n$ then the number of errors is sufficiently small and we accept the lot. If $d_n > R_n$ too many errors have already been found in the lot and we reject it. If instead $A_n < d_n < R_n$ we don't have yet enough evidence to accept or reject the lot and we continue sampling. Let p denote the probability of a defect, p_0 a probability generating a tolerable level of noise, i.e., for which we accept the lot, and $p_1 > p_0$ a probability for which we reject the lot. $H_0 : p = p_0$ is then the null hypothesis, while $H_1 : p = p_1$ the alternative hypothesis. The values of A_n and R_n can be defined by:

$$A_n = \frac{\ln\left(\frac{\beta}{1-\alpha}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)} + n\frac{\ln\left(\frac{1-p_0}{1-p_1}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)}, \tag{1}$$

$$R_n = \frac{\ln\left(\frac{1-\beta}{\alpha}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)} + n\frac{\ln\left(\frac{1-p_0}{1-p_1}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)} \tag{2}$$

where the alpha level α is a parameter controlling type I errors (rejecting a true null hypothesis), and the beta level β is a parameter controlling type II errors (accepting a false alternative hypothesis).

3 SPRT-Based Learning of Temporal Formulae

In this section, we shall describe our approach to learning temporal formulae from event logs by applying SPRT. We begin by noting that equations (1),(2) involve four parameters: p_0, $p1$, α, and β. However, they can be rewritten as

$$A_n = mn + c_A, \tag{3}$$

$$R_n = mn + c_R, \tag{4}$$

by posing

$$m = \frac{\ln\left(\frac{1-p_0}{1-p_1}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)},\ c_A = \frac{\ln\left(\frac{\beta}{1-\alpha}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)},\ c_R = \frac{\ln\left(\frac{1-\beta}{\alpha}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)}. \tag{5}$$

In other words, as a function of n, A_n and R_n are lines with the same slope m (i.e. they are parallel) and intercepts c_A and c_R, respectively. Note that for reasonably small significance levels α and β, i.e. such that $\alpha + \beta < 1$, (besides having $0 < p_0 < p_1 < 1$) it follows:

- $m_0 > 0$ (so that A_n and R_n are strictly increasing sequences).
- $c_A < 0 < c_R$ (otherwise one would end up accepting or rejecting a lot even before inspecting it).

Indeed, any value of m, c_A, and c_R satisfying the above inequalities is an admissible choice. Therefore in the following, we will directly work with these parameters, without explicitly modeling and reasoning about the noise. The method is provided in Algorithm 1. It takes as input a log L, a temporal formula φ, and suitable values for the parameters c_A, c_R, and c_R. First, the initial values d_0, A_0, R_0 are assigned (lines 1-3). Then, for each n, we do the following. If $d_n \leq A_n$ we return *Accept* (lines 5-6). If $d_n \geq R_n$ we return *Reject* (lines 7-8). Otherwise, we compute the next values d_{n+1}, A_{n+1}, R_0 (lines 9-12) and proceed. Note that the value of d_{n+1} is computed by first selecting a trace π from the log (line 11), which can be done according to the order of the traces in the log or by random sampling; then computing the number of defects of pi, represented by the number of alignments required to make pi a model of φ, and adding such quantity to the total number of defects (line 12). While the checks $A_0 < d_0$ and $d_0 < R_0$ could be avoided, treating $n = 0$ as all the other values turns out to be mathematically convenient for interpreting the parameters of the algorithm.

It is worth noting that what we have just described is actually a semi-algorithm. In fact, even assuming a new trace is always available, it could be the case that the total number of defects never satisfies any of the inequalities. To obviate this, one can simply return a default value of their choice (be it *Accept*, *Reject*, or, e.g., *Inconclusive*).

4 Experiments

In this section, we illustrate Algorithm 1 by performing some controlled experiments. We begin by detailing the process of constructing an appropriate log in Subsection 4.1. Subsequently, in Subsection 4.2, we employ this log to examine several Declare constraints.

4.1 Log Generation and Description

To perform controlled experiments, we resort to synthetic data. Fixed an alphabet $\Sigma = \{a, b, c, d, e\}$, we define 5 Declare constraints over it:

Algorithm 1 SPRT-based Learning of Temporal Formulae.

Input: L, φ, $m > 0$, $c_A < 0$, and $c_R > 0$
Output: *Accept* or *Reject*

$defects \leftarrow 0$
$A \leftarrow c_A$
$R \leftarrow c_R$
while $True$ **do**
  if $defects \leq A$ **then**
    return *Accept*
  if $defects \geq R$ **then**
    return *Reject*
  $A \leftarrow A + m$
  $R \leftarrow R + m$
  $\pi \leftarrow Next(L)$ `// Extract next trace`
  $defects \leftarrow defects + Align(\varphi, \pi)$ `// add the alignments' cost`

1. $ExclusiveChoice(c, d)$,
2. $Response(a, b)$,
3. $RespondedExistence(a, e)$,
4. $Precedence(e, a)$,
5. $AlternateResponse(b, c)$.

The model $\mathcal{M}$ represented by the conjunction of those five constraints is represented in the intuitive Declare graphical notation in Fig 2.

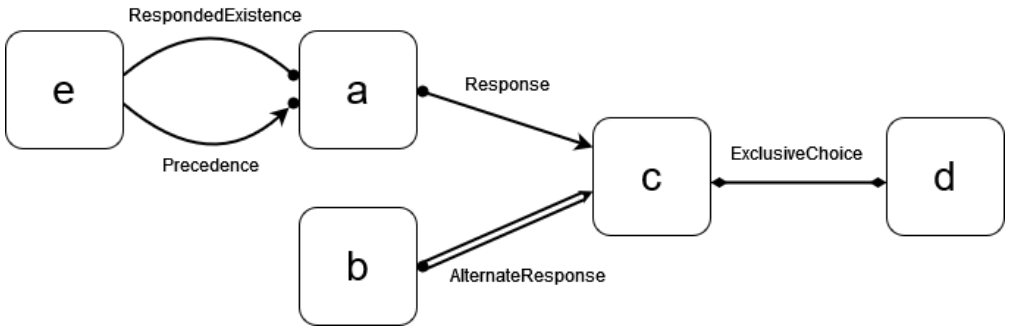

Figure 2 The Declare model used for the experiments.

We then generate a log of 100 traces of length varying from 6 to 15. In particular, for each length, we generate 5 positive traces, i.e. satisfying (all the constraints of) the model, and 5 negative traces, i.e. violating at least one constraint. The generation is performed using ASP Log Generator [5, 6], which converts the constraints into their corresponding automata [7] and use Answer Set Programming [3] to find traces accepted or not by the automata (depending on whether we are looking for a positive or negative one). We generate negative traces by explicitly imposing the violation of the model since injecting some noise on a positive trace, e.g. by randomly adding and removing events, could result in the trace still satisfying the constraints. To improve the quality of the log, avoiding traces that are too repetitive and too similar among each other, we constrain each of the 5 activities to occur

Table 2 Number of traces violating the constraints arranged according to the cost of repairs.

	1	2	3+	#traces	total cost
C1	19	13	6	38	68
C2	14	0	0	14	14
C3	9	0	0	9	9
C4	31	0	0	31	31
C5	16	15	2	33	52

Table 3 Number of alignments per constraint as a function of the trace length.

	6	7	8	9	10	11	12	13	14	15	tot
C1	5	2	3	5	6	8	11	5	12	11	68
C2	2	2	1	1	0	1	2	2	0	3	14
C3	3	0	0	1	1	1	2	0	1	0	9
C4	4	2	4	4	4	2	3	2	2	4	31
C5	2	8	6	9	4	6	6	6	0	5	52
tot	16	14	14	20	15	18	24	15	15	23	174

no more than in 1/2 of the instants. In addition, we modify the search strategy to perform random (rather than heuristic) decisions with probability 0.2 and run the log generator multiple times with different seeds asking each time for 1 solution. Note that how negative traces are generated greatly impacts the cost of the alignments. Table 2 reports, for each constraint, the number of traces violating it, arranged according to the cost of the alignments, together with the total cost of the alignments (traces with a repair cost of 3 or more are merged in one column, however, their different impact is reflected by the total cost). For example, constraint 1 $ExclusiveChoice(c, d)$ (C1 for short in the tables) has 38 traces (out of the 50 negative ones) violating it of which 19 require 1 alignment, 13 require 2 alignments, and 6 require 3 or more alignments, for a total of 68 alignments. The costs are computed by compiling the problem into the Planning Domain Definition Language [9] and solving it with Fast Downward [24]. To better understand the noise of the log, in Table 3 we report, for each trace length, the number of alignments required to make the traces of that length conformant with the constraint. Recall that there are 5 negative traces per length.

It is important to note that templates behave very differently. We can see for example that $ExclusiveChoice(c, d)$ requires a total of 68 alignments over the whole log and that the cost is strongly influenced by the trace length. In fact, in case both c and d are present in the trace, to align it one has to determine which activity has fewer occurrences and remove them. Other templates such as *Response* always require at most 1 alignment, regardless of the trace length, since it is simply sufficient to add the target activity at the end of the trace.

This makes the choice of the values of the parameters m, c_A, and c_R in Algorithm 1 very important, particularly for the slope m. Such a choice, which must be made constraint by constraint, should therefore be undertaken after careful consideration of both the particular formula (where formulae intuitively easier to satisfy correspond to smaller values for m) and the expected noise.

4.2 Execution

In this subsection, we show and discuss the execution of the Algorithm 1 over the generated log. We start by considering the Declare model $\mathcal{M}$. We choose a value of $m = 1$, of $c_R = 8$, and of $c_A = -8$. In Figure 3, we plot the lines A_n and R_n in green and red, respectively.

We then apply the algorithm to each of the five constraints of $\mathcal{M}$. Note that we use here the same parameters across the various constraints just for the sake of simplicity. We want to stress that the choice of such parameters must be done individually for each constraint after carefully pondering the expected noise of the log, the structure of the formula, as well as additional considerations about the risk of a wrong decision. We then plot, for each constraint, the curve representing the total number of defects as a function of the number of traces observed, selecting traces according to their order in the log. We observe that all the constraints are classified as satisfied by the model. The first constraint to be decided is $Response(a, c)$ at step 8. Note that, from $m = 1$ follows that $-c_A$ represents exactly the minimum number of steps before deciding on acceptance. The last constraint to be decided is instead $ExclusiveChoice(c, d)$ at step 44. One could stop plotting a curve after it intersects A_n, indeed the algorithm terminates and the new costs are not computed. In Figure 3 however we continue to show them (this time as dashed lines) to better visualize the behavior of the log with respect to the constraints.

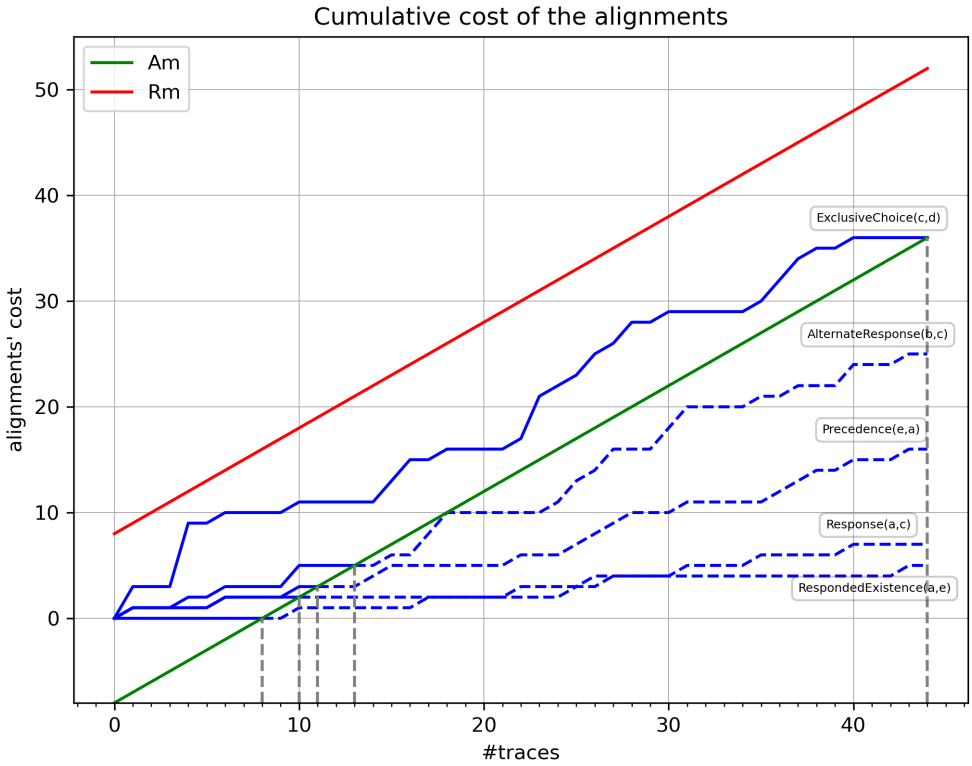

Figure 3 Results of the application of the algorithm to the constraints of the model.

In Figure 4 we consider instead a set of new constraints that were not used during the generation of the log. We again use the same parameters as before. We see in step 8 that the constraint $Choice(c, d)$ is accepted. This constraint is actually implied by one of the constraints used to generate the log, $ExclusiveChoice(c, d)$, that was accepted at a later step. This is a consequence of the fact that any alignment w.r.t. the latter formula is also an alignment w.r.t. to the former one, and holds indeed as a general result, that we state formally as follows:

▶ **Observation 1.** *If a formula φ is accepted in n_φ steps, and ψ is implied by φ then ψ is accepted as well and in a number of steps $n_\psi \leq n_\varphi$. Dual results hold for the rejection.*

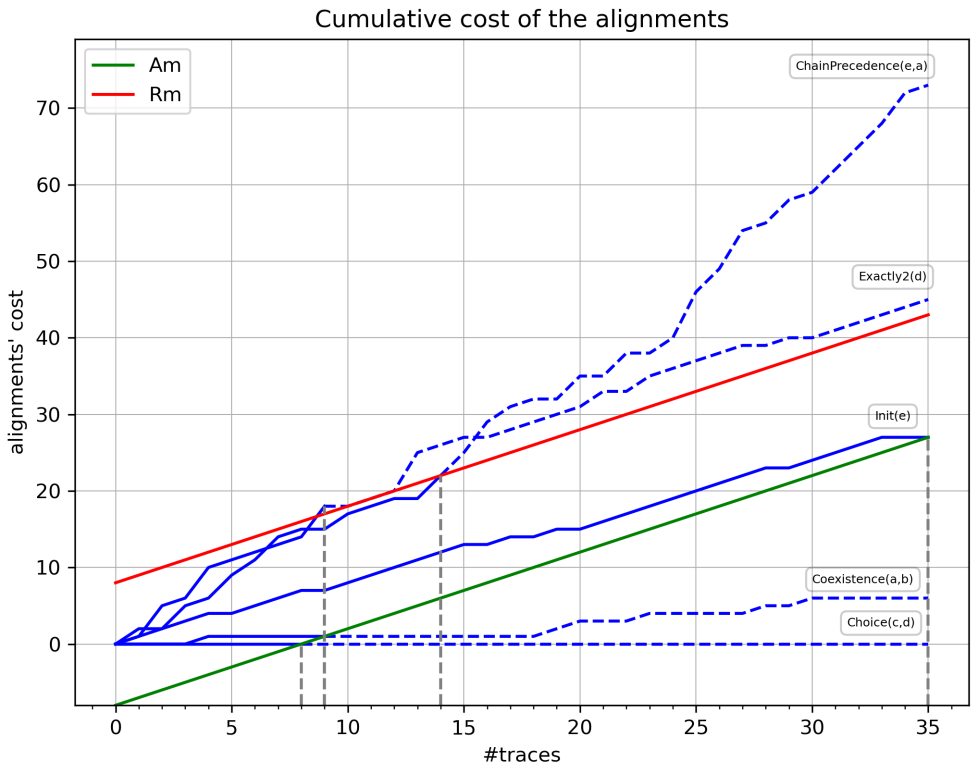

Figure 4 Results of the application of the algorithm to new constraints.

We note from the figure that $Choice(c, d)$ has indeed an alignments' cost of 0 throughout the steps and is therefore decided at step $-c_A = 8$. The fact that the costs are equal to 0 is a consequence of how we generated the log (i.e. by forcing the same activities not to appear too many times in trace). The constraints $CoExistence(a, b)$ and $Init(e)$ are accepted after 9 and 35 steps, respectively. By taking into consideration the meaning of the constraints is clear why the former were accepted: repairing a trace requires at most 1 alignment and therefore $m = 1$ is a high value compared with the expected defects of the traces. We have then the first two cases of rejected constraints: $Exactly2(D)$ rejected after 9 steps, and $ChainPrecedence(e, a)$ rejected after 14 steps. Both of those constraints require acting on the different occurrences of an activity and therefore for them, $m = 1$ may be a too-small value (again, depending on how much noise we expect in the log as well).

Finally, it is worthwhile to point out that, although Algorithm 1 stops after deciding to accept or reject, one could indeed continue to monitor a constraint to be able to change the decision when new evidence emerges. This is important, for example, in cases in which the distribution of the process executions substantially changes over time. In this respect, it is useful to note that, both in Figure 3 and 4 no decision has ever been overturned, not reconsidered. Once a curve enters one of the two half-plane defined by A_m and R_m it tends to stay there. This could however be due to the particular choice of the parameters. Value for m closer to the actual alignment cost, together with a smaller distance $c_R - c_A$ between A_m and R_m, could result in more oscillating situations.

5 Related Works

The problem of learning linear temporal formulae, sometimes known as *LTL (specification) mining* [25], has been approached by the Temporal Logic community mainly with exact methods, rather than relying on statistical ones. Camacho and McIlraith [4] reduce the problem of learning LTL_f formulae to proposition satisfiability. The approach is based on guessing the (alternating) automaton associated with a formula and then checking for conformance with the (positive and negative) example traces. Such automaton can then be converted in linear time into the corresponding formula. Note that no templates are used here; instead, they employ SAT solver to perform a search over the space of all possible formulae. Gaglione et al. [20] later modified the approach to handle noise using MaxSAT solvers. These approaches assume the availability of both positive and negative example traces. Roy et al. [32] consider the problem of learning from positive example only. This setting, which is the standard in Process Mining, is known in the Machine Learning literature as one-class (or unary) classification and is way more challenging than binary classification when considering minimality requirements about the discovered solutions [33, 1]. The approach by Roy et al. [32] is however limited in that it cannot handle noise.

Declarative Process Discovery algorithms [14, 26, 27] have been proposed that exploit statistical analysis, handle noise, and work with positive examples only. However, those algorithms usually consider only Declare constraints. Besides, they assume for simplicity the event log to be a fixed multiset of traces, rather than an ordered collection in continuous evolution, and do not reason about whether the evidence gathered up to some moment is enough to make a decision, as is done with the stopping rules in sequential analysis.

Sequential analysis has also inspired a constraint acquisition algorithm in the context of Constraint Programming [30, 31]. This algorithm is however a coarse simplification of SPRT, using no parameters other than constant acceptance and rejection thresholds, and does not quantify the discrepancy between a constraint and an example. Furthermore, our work differs in that it considers temporal properties of traces rather than constraints over the values of decision variables.

6 Conclusion

In this paper, we have shown how to use statistical techniques from sequential analysis for learning linear temporal formulae from events log. Our approach is based on analyzing the log incrementally and measuring how much a trace deviates from satisfying a given formula. As a measure of such deviation, we use the number of alignments, i.e., of modifications of the trace (in terms of additions and removals of activities) to make the trace a model of the formula. We have then implemented our approach and tested it with controlled experiments. The advantage of a statistical approach is its ability to handle the noise naturally occurring in the event logs. Sequential analysis, in particular, by exploiting only the strictly necessary number of samples, additionally provides early decision-making capabilities. This turns out to be useful both in the case one has to analyze a huge volume of data, alleviating the computational burden, as well as in the case where such data are scarce.

In future work, we intend to study how one can use Algorithm 1 to learn interpretable models, by combining it with a suitable procedure for selecting temporal formulae to be tested, taking into account subsumption-driven hierarchies [13] and their interaction with the learning phase. Furthermore, we intend to study how to suitably select the parameters m, c_A, and c_R of Algorithm 1 by considering both the structure of the formula and the noise

of the log generation process, which need to be explicitly modeled. Finally, we intend to consider other possible measures of the discrepancy between a formula and a trace, study their properties, and the impact they have on applying sequential analysis techniques.

References

1 Simone Agostinelli, Francesco Chiariello, Fabrizio Maria Maggi, Andrea Marrella, and Fabio Patrizi. Process mining meets model learning: Discovering deterministic finite state automata from event logs for business process analysis. *Inf. Syst.*, 114:102180, 2023. doi:10.1016/J.IS.2023.102180.

2 Iris Beerepoot, Claudio Di Ciccio, Hajo A. Reijers, Stefanie Rinderle-Ma, Wasana Bandara, Andrea Burattin, Diego Calvanese, Tianwa Chen, Izack Cohen, Benoît Depaire, Gemma Di Federico, Marlon Dumas, Christopher G. J. van Dun, Tobias Fehrer, Dominik Andreas Fischer, Avigdor Gal, Marta Indulska, Vatche Isahagian, Christopher Klinkmüller, Wolfgang Kratsch, Henrik Leopold, Amy Van Looy, Hugo A. López, Sanja Lukumbuzya, Jan Mendling, Lara Meyers, Linda Moder, Marco Montali, Vinod Muthusamy, Manfred Reichert, Yara Rizk, Michael Rosemann, Maximilian Röglinger, Shazia Sadiq, Ronny Seiger, Tijs Slaats, Mantas Simkus, Ida Asadi Someh, Barbara Weber, Ingo Weber, Mathias Weske, and Francesca Zerbato. The biggest business process management problems to solve before we die. *Comput. Ind.*, 146:103837, 2023. doi:10.1016/J.COMPIND.2022.103837.

3 Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011. doi:10.1145/2043174.2043195.

4 Alberto Camacho and Sheila A. McIlraith. Learning interpretable models expressed in linear temporal logic. In *ICAPS*, pages 621–630. AAAI Press, 2019. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/3529.

5 Francesco Chiariello, Fabrizio Maria Maggi, and Fabio Patrizi. Asp-based declarative process mining. In *AAAI*, pages 5539–5547. AAAI Press, 2022. doi:10.1609/AAAI.V36I5.20493.

6 Francesco Chiariello, Fabrizio Maria Maggi, and Fabio Patrizi. A tool for compiling declarative process mining problems in ASP. *Softw. Impacts*, 14:100435, 2022. doi:10.1016/J.SIMPA.2022.100435.

7 Francesco Chiariello, Fabrizio Maria Maggi, and Fabio Patrizi. From LTL on process traces to finite-state automata. In *BPM (Demos / Resources Forum)*, volume 3469 of *CEUR Workshop Proceedings*, pages 127–131. CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3469/paper-23.pdf.

8 Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *AAAI*, pages 1027–1033. AAAI Press, 2014. doi:10.1609/AAAI.V28I1.8872.

9 Giuseppe De Giacomo, Francesco Fuggitti, Fabrizio Maria Maggi, Andrea Marrella, and Fabio Patrizi. A tool for declarative trace alignment via automated planning. *Softw. Impacts*, 16:100505, 2023. doi:10.1016/J.SIMPA.2023.100505.

10 Giuseppe De Giacomo, Fabrizio Maria Maggi, Andrea Marrella, and Fabio Patrizi. On the disruptive effectiveness of automated planning for ltl$f$-based trace alignment. In *AAAI*, pages 3555–3561. AAAI Press, 2017. doi:10.1609/AAAI.V31I1.11020.

11 Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 854–860. IJCAI/AAAI, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

12 Massimiliano de Leoni, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Inf. Syst.*, 47:258–277, 2015. doi:10.1016/J.IS.2013.12.005.

13 Claudio Di Ciccio, Fabrizio Maria Maggi, Marco Montali, and Jan Mendling. Ensuring model consistency in declarative process discovery. In *BPM*, volume 9253 of *Lecture Notes in Computer Science*, pages 144–159. Springer, 2015. doi:10.1007/978-3-319-23063-4_9.

14 Claudio Di Ciccio and Massimo Mecella. On the discovery of declarative control flows for artful processes. *ACM Trans. Manag. Inf. Syst.*, 5(4):24:1–24:37, 2015. doi:10.1145/2629447.

15 Claudio Di Ciccio and Marco Montali. Declarative process specifications: Reasoning, discovery, monitoring. In *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing*, pages 108–152. Springer, 2022. doi:10.1007/978-3-031-08848-3_4.

16 Marlon Dumas and Arthur H. M. ter Hofstede. UML activity diagrams as a workflow specification language. In *UML*, volume 2185 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2001. doi:10.1007/3-540-45441-1_7.

17 Bernd Finkbeiner and Henny Sipma. Checking finite traces using alternating automata. *Form.Meth.Syst.Des.*, 24(2):101–127, 2004. doi:10.1023/B:FORM.0000017718.28096.48.

18 Valeria Fionda and Gianluigi Greco. The complexity of LTL on finite traces: Hard and easy fragments. In *AAAI*, pages 971–977. AAAI Press, 2016. doi:10.1609/AAAI.V30I1.10104.

19 Valeria Fionda and Gianluigi Greco. LTL on finite and process traces: Complexity results and a practical reasoner. *J. Artif. Intell. Res.*, 63:557–623, 2018. doi:10.1613/JAIR.1.11256.

20 Jean-Raphaël Gaglione, Daniel Neider, Rajarshi Roy, Ufuk Topcu, and Zhe Xu. Learning linear temporal properties from noisy data: A maxsat-based approach. In *ATVA*, volume 12971 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 2021. doi:10.1007/978-3-030-88885-5_6.

21 Bhaskar Kumar Ghosh and Pranab Kumar Sen. *Handbook of sequential analysis*. CRC Press, 1991.

22 Ben Greenman, Sam Saarinen, Tim Nelson, and Shriram Krishnamurthi. Little tricky logic: Misconceptions in the understanding of LTL. *Art Sci. Eng. Program.*, 7(2), 2023. doi:10.22152/PROGRAMMING-JOURNAL.ORG/2023/7/7.

23 Object Management Group. Business process model and notation (bpmn), version 2.0.2, 2014. URL: www.omg.org/spec/BPMN.

24 Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246, 2006. doi:10.1613/JAIR.1705.

25 Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General LTL specification mining (T). In *ASE*, pages 81–92. IEEE Computer Society, 2015. doi:10.1109/ASE.2015.71.

26 Fabrizio Maria Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In *CAiSE*, volume 7328 of *Lecture Notes in Computer Science*, pages 270–285. Springer, 2012. doi:10.1007/978-3-642-31095-9_18.

27 Fabrizio Maria Maggi, Claudio Di Ciccio, Chiara Di Francescomarino, and Taavi Kala. Parallel algorithms for the automated discovery of declarative process models. *Inf. Syst.*, 74(Part):136–152, 2018. doi:10.1016/J.IS.2017.12.002.

28 Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. DECLARE: full support for loosely-structured processes. In *EDOC*, pages 287–300. IEEE Computer Society, 2007. doi:10.1109/EDOC.2007.14.

29 Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

30 Steve Prestwich. Robust constraint acquisition by sequential analysis. In *ECAI*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 355–362. IOS Press, 2020. doi:10.3233/FAIA200113.

31 Steven Prestwich and Nic Wilson. A statistical approach to learning constraints. *International Journal of Approximate Reasoning*, 2024.

32 Rajarshi Roy, Jean-Raphaël Gaglione, Nasim Baharisangari, Daniel Neider, Zhe Xu, and Ufuk Topcu. Learning interpretable temporal properties from positive examples only. In *AAAI*, pages 6507–6515. AAAI Press, 2023. doi:10.1609/AAAI.V37I5.25800.

33 Tijs Slaats, Søren Debois, and Christoffer Olling Back. Weighing the pros and cons: Process discovery with negative examples. In *BPM*, volume 12875 of *Lecture Notes in Computer Science*, pages 47–64. Springer, 2021. doi:10.1007/978-3-030-85469-0_6.

34 Wil M. P. van der Aalst. The application of petri nets to workflow management. *J. Circuits Syst. Comput.*, 8(1):21–66, 1998. doi:10.1142/S0218126698000043.

35 Wil M. P. van der Aalst. Foundations of process discovery. In *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing*, pages 37–75. Springer, 2022. doi:10.1007/978-3-031-08848-3_2.

36 Wil M. P. van der Aalst. Process mining: A 360 degree overview. In *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing*, pages 3–34. Springer, 2022. doi:10.1007/978-3-031-08848-3_1.

37 Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Comput. Sci. Res. Dev.*, 23(2):99–113, 2009. doi:10.1007/S00450-009-0057-9.

38 Wil M. P. van der Aalst and Christian Stahl. *Modeling Business Processes - A Petri Net-Oriented Approach*. Cooperative Information Systems series. MIT Press, 2011.

39 A Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 1945.

40 Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, Third Edition*. Springer, 2019. doi:10.1007/978-3-662-59432-2.