Computing Trace Alignment against Declarative Process Models through Planning

Giuseppe De Giacomo
Sapienza - Universitá di Roma, Italy
degiacomo@dis.uniroma1.it

Fabrizio Maria Maggi
University of Tartu, Estonia
f.m.maggi@ut.ee

Andrea Marrella
Sapienza - Universitá di Roma, Italy
marrella@dis.uniroma1.it

Sebastian Sardina
RMIT University, Melbourne, Australia
sebastian.sardina@rmit.edu.au

Abstract

Process mining techniques aim at extracting non-trivial knowledge from event traces, which record the concrete execution of business processes. Typically, traces are "dirty" and contain spurious events or miss relevant events. Trace alignment is the problem of cleaning such traces against a process specification. There has recently been a growing use of *declarative* process models, e.g., DECLARE (based on LTL over finite traces) to capture constraints on the allowed task flows. We demonstrate here how state-of-the-art classical planning technologies can be used for trace alignment by presenting a suitable encoding. We report experimental results using a real log from a financial domain.

1 Introduction

This paper is concerned with *process mining* and *trace alignment* in *Business Process Management* (BPM) (van der Aalst 2013a), an active area of research that is highly relevant from a practical perspective while offering many technical challenges to computer scientists. BPM is based on the observation that each product/service that an organization provides is the outcome of a number of activities performed. *Business processes* are the key instruments for organizing such activities and improving the understanding of their interrelationships. Examples of processes include insurance claim processing, order handling, and hospital procedures.

The fast pace of change in markets is more and more imposing the definition and use of flexible information systems for supporting business processes of companies. In fact, such dynamic markets make the specification of a business process obsolete in a relatively short span of time, due to the need of bringing frequent modifications and updates to the process in order to accommodate for new market conditions. The availability of event data recorded by information systems makes *process mining* a valuable instrument to improve and support business processes (van der Aalst 2011).

The starting point for process mining is an *event log*. XES (eXtensible Event Stream) (Task Force on Process Mining 2013; Verbeek et al. 2010) has been developed as the standard for storing, exchanging, and analyzing event logs. Each event in a log refers to an *activity* (i.e., a well-defined step in some process) and is related to a particular *case* (i.e., a *process instance* generated by an underlying process model). The events belonging to a case are *ordered* and form a so-called *trace*, which can be seen as one "run" of the process. Event logs may store additional information about events, such as the *resource* executing or initiating the activity, the *timestamp* of the event, or *data elements* recorded with the event. Ideally, every event in a trace corresponds to the execution of an activity in the underlying process model.

In the BPM literature, three types of process mining can be distinguished (van der Aalst 2011): *(i) process discovery* (learning a process model from a set of traces in an event log), *(ii) conformance checking* (comparing the observed behavior in the event log with the modeled behavior), and *(iii) model enhancement* (extending models based on additional information in the event logs). In this paper, we focus on *conformance checking*, and specifically on *trace alignment*.

Conformance checking starts from the assumption that, since process models are typically not enforced by information systems (see human behavior is often involved), traces can be "dirty", and possibly containing spurious or missing events. This implies a discrepancy between the modeled behavior and the observed one. In the context of conformance checking, *trace alignment* is the problem of "cleaning" such dirty traces on the basis of expected process specifications to the aim of understanding the root cause of each deviation.

In this paper, we show how to exploit state-of-the-art classical planning techniques and technologies (Ghallab, Nau, and Traverso 2004; Bonet and Geffner 2001) for trace alignment with respect to a predefined *declarative process model*. In comparison to imperative approaches, which *explicitly* specify all possible sequences of activities in a process, declarative languages *implicitly* specify the allowed behavior of the process with a set of (temporal) *constraints* that must be met. Declarative models offer flexibility (everything that is not specified is allowed), and are appropriate to describe dynamic environments, where several execution paths are allowed. Specifically, we will focus on the well-established DECLARE process modeling language (van der Aalst, Pesic, and Schonenberg 2009), which enjoys formal semantics grounded in LTL (Linear Temporal Logic) (Pnueli 1977) on finite traces.

In a nutshell, given a set of LTL-like DECLARE constraints modeling a process, and an actual trace of the process (obtained from an event log), we build a planning domain and problem instance, correspondingly, such that a solution plan for the planning problem amounts to the set of interventions to align the trace with the constraints, if required. Of course, an empty plan implies that no alignment is required: the trace already conforms to the DECLARE model.

Before providing our reduction to a planning task (Section 4) and reporting on experiment results (Section 5) on a real case study from the financial domain (Section 3), we first review the relevant background required (Section 2).

2 Preliminaries

2.1 The DECLARE Modeling Language

DECLARE is a declarative process modeling language originally introduced by (van der Aalst, Pesic, and Schonenberg 2009). Instead of explicitly specifying the flow of the interactions among process activities, DECLARE describes a set of (temporally extended) constraints that must be satisfied throughout the process execution. The orderings of activities are implicitly specified by constraints and anything that does not violate them is possible during execution.

Technically, a DECLARE model $\mathcal{D} = (\mathcal{A}, \pi_{\mathcal{D}})$ consists of a set of possible activities $\mathcal{A}$ involved in a process and a collection of constraints $\pi_{\mathcal{D}}$ defined over such activities. DECLARE constraints are instantiations of templates, i.e., patterns that define parameterized classes of properties. Templates have a graphical representation understandable to the user and analyst, but they also enjoy precise semantics in different logics (Montali et al. 2010) (e.g., LTL over finite traces), making them verifiable and executable.

Table 1 summarizes some DECLARE templates (the reader can refer to (van der Aalst, Pesic, and Schonenberg 2009) for a full description of the language). Templates *existence(A)* and *absence(A)* require that *A* occurs at least once and never occurs in every process instance, respectively. Template *init(A)* specifies that *A* must occur in the first position of the process instance. Template *co-existence(A,B)* requires that if one of the activities *A* or *B* occur, the other one must also occur. Templates *choice* and *exclusive choice(A,B)* indicate that *A* or *B* occur eventually in each process instance. The exclusive choice template is more restrictive because it forbids *A* and *B* to occur both in the same process instance. The *responded existence(A,B)* template states that if *A* occurs, then *B* should also occur (either before or after *A*). The *response(A,B)* template specifies that when *A* occurs, then *B* should eventually occur after *A*. The *precedence(A,B)* template indicates that *B* can occur only if *A* has occurred before. The *succession(A,B)* template states that both response and precedence relations hold between *A* and *B*.

Templates *alternate response(A,B)* and *alternate precedence(A,B)* strengthen the response and precedence templates respectively by specifying that activities must alternate without repetitions in between. The *alternate succession(A,B)* template is the combination of the alternate response and alternate precedence templates. Even stronger ordering relations are specified by templates *chain response(A,B)* and *chain precedence(A,B)*. These templates require that the occurrences of *A* and *B* are next to each other. The *chain succession(A,B)* template is the combination of the chain response and chain precedence templates.

DECLARE also includes some negative constraints to explicitly forbid the execution of activities. The *not coexistence(A,B)* template indicates that *A* and *B* can not occur together in the same process instance. According to the *not succession(A,B)* template any occurrence of *A* can not be eventually followed by *B*. Finally, the *not chain succession(A,B)* template states that *A* and *B* can not occur one immediately after the other.

2.2 Conformance Checking and Trace Alignment

Lying under the umbrella of process mining, *conformance checking* techniques focus on comparing the observed behavior in an event log with the modeled behavior reflected by a process model. Typically, conformance requirements on business processes stem from different sources such as laws, regulations, or guidelines that are often available as textual descriptions. An example constraint from the medical domain would be *"The patient has to be informed about the risks of a surgery before the surgery takes place."* In practice, conformance checks are often conducted manually and hence perceived as a burden (Sadiq, Governatori, and Naimiri 2007), although their importance is undoubted.

The need to check for conformance of business processes based on a set of constraints may emerge in different phases of the process lifecycle (Ly et al. 2012; Sadiq 2011). During design time, the conformance of a process model with a set of constraints is checked. At runtime, the progress of a potentially large number of process instances is monitored to detect or even predict violations. Finally, processes can be diagnosed for conformance violations in a *post mortem* or *off-line* manner, i.e., after the process execution has completed. In this paper, we focus on this last type of analysis and in particular on *trace alignment*.

The starting points for trace alignment are an *event log* and a *compliance model*. An event log contains evidence of process executions stored like *traces*. Traces consist of *events*. Each event is represented using a unique identifier (such that it appears at most once in the entire log) and is associated to the execution of a specific activity. Therefore, for the sake of simplicity, we shall assume in this work that a trace $\mathcal{T}$ amounts to a sequence of activities, describing one run of the process. A log is just a set of such traces.

Being based on constraints, DECLARE is very suitable for specifying compliance models that are used to check if the behavior of a system complies with desired regulations. The compliance model defines the constraints related to a single process instance, and the overall expectation is that all instances comply with the model. 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 by traces $\mathcal{T}_1 = \langle a, a, b, c\rangle$, $\mathcal{T}_2 = \langle b, b, c, d\rangle$, and $\mathcal{T}_3 = \langle a, b, c, b\rangle$, but not by $\mathcal{T}_4 = \langle a, b, a, c\rangle$ (no b response exists for the second instance of activity a).

In this context, *trace alignment* is the task of verifying if the actual flow of work of completed process instances recorded in an event log is compliant with the intended declarative process model. If a discrepancy (i.e., violations of one or more DECLARE constraints) between a specific trace of the log and the model is identified, the goal of trace alignment is to repair such a discrepancy by *aligning* the trace (i.e., the observed behavior) with the model (i.e., the modeled behavior).

Specifically, given a DECLARE model $\mathcal{D} = (\mathcal{A}, \pi_{\mathcal{D}})$ and a trace $\mathcal{T}$ over activities in $\mathcal{A}$ (possibly not compliant with $\mathcal{D}$), the *trace alignment task* amounts to finding a trace $\hat{\mathcal{T}}$ over $\mathcal{A}$ that is compliant with $\mathcal{D}$. Here, trace $\hat{\mathcal{T}}$ is called an *alignment* of $\mathcal{T}$ over $\mathcal{D}$. Of course, not all alignments are judged equally and one strives for those deviating less, or with less associated cost, from the original trace. So, in the extreme case, if trace $\mathcal{T}$ already meets the DECLARE model $\mathcal{D}$, one would prefer $\mathcal{T}$ itself as the (trivial) alignment over any other alternative alignment.

3 Case Study

Since 2011, the BPI Workshop features an initiative called International Business Process Intelligence Challenge (BPIC). The idea is that an event log is provided with some background information and points of interest. Researchers and practitioners participate in a competition in which they are asked to test, apply or validate whatever technique or tool they developed using this log. In 2010, the three universities of technology in The Netherlands joined forces in erecting the 3TU Datacenter. This initiative aimed at publicly sharing datasets such that other researchers can benefit from whatever data can be collected.

The constraints used for our evaluation were extracted from the log provided for the BPIC 2012 (3TU Data Center 2012). The event log pertains to an application process for personal loans or overdrafts in a Dutch financial institute. It merges three intertwined sub-processes. Therefore, in each case, events belonging to different sub-processes can occur. The log contains 262,200 events distributed across 36 activities and includes 13,087 cases.

Figure 1 shows a DECLARE model representing the expected behavior of the application process. It contains 15 activities that can be associated to different lifecycle states indicating that an activity can be scheduled, started, or completed. The model contains 16 constraints.

In the figure, application activities (denoted with “A_”) refer to the management of the application. A_SUBMITTED and A_PARTLYSUBMITTED indicate the initial application submission. A_ACCEPTED indicates that the application has been accepted. A_APPROVED, A_REGISTERED, and A_ACTIVATED specify the end states of successful applications. A_CANCELLED and A_DECLINED specify the end states of unsuccessful applications.

Offer activities (denoted with “O_”) refer to offers communicated to the customer. O_SELECTED indicates that the applicant asked to receive an offer. O_CREATED and O_SENT specify that the offer has been prepared and transmitted to the applicant.

Work item activities (denoted with “W_”) refer to work items that occur during the approval process. These events mainly refer to manual tasks executed by the resources of the financial institute during the application approval process. W_Afhandelen leads indicates a follow up on incomplete initial submissions. W_Completeren aanvraag specifies that accepted applications are completed. W_Beoordelen fraude refers to the investigation of suspect fraud cases. W_Wijzigen contractgegevens indicates the modification of approved contracts.

The case study described in this section has been used to conduct our experimentation. See Section 5 for a detailed description of the experimental results.

4 Encoding Trace Alignment in PDDL

Our planning-based approach to perform trace alignment requires to define a single PDDL (McDermott et al. 1998) planning domain encoding all the constraints of a DECLARE model, the predicates and the planning actions to keep track and modify a generic trace, and as many planning problems as are the traces of the event log to be checked for alignment.

Once a DECLARE model $\mathcal{D}$ and a trace $\mathcal{T}$ have been properly encoded in PDDL, we feed a state-of-the-art classical planner with such inputs. The plan synthesized will consist of a sequence of alignment steps that modify $\mathcal{T}$ by removing/adding existing/new activity instances from/in $\mathcal{T}$, in order to obtain an aligned trace $\hat{\mathcal{T}}$ compliant with $\mathcal{D}$. If $\mathcal{T}$ is already aligned with respect to $\mathcal{D}$, the plan will be empty.

In order to quantify the severity of a deviation between a trace $\mathcal{T}$ and the model $\mathcal{D}$, we exploit a cost function on the alignment steps. The costs depend on the specific characteristics of the process, e.g., it may be more costly to skip an insurance check for high claims than for low claims.

In this work, we represent planning domains and planning problems using PDDL 2.2 (Edelkamp and Hoffmann 2004), which includes derived predicates plus the `:action-costs` requirement, used to keep track of the costs of planning actions. Next, we show how the problem of aligning a trace with a DECLARE model can be modeled in PDDL.

4.1 The Planning Domain

In the planning domain, we provide a unique abstract type `activity`, which captures every possible activity instance involved in some DECLARE constraint or included in some trace of the event log. Several sub-types of `activity` are defined to express the specific activities of interest. For example, if we suppose that the event log is composed of a single trace $\mathcal{T}_0 = \langle a, b, c, a, d \rangle$, and that the DECLARE model includes the single constraint ***not succession(a,e)***, five activities subtypes would be required in the planning domain:

```
(:types a b c d e - activity)
```

Every activity instance possibly involved in the alignment procedure has certain properties. An activity instance may be included (or not) in the original trace, and if this is the case, it may be in the first or in the last position of the trace, and it may have a successor. To capture such properties, we introduce a predicate `(traced ?a - activity)` expressing that instance `a` of type `activity` is currently included in the trace to be aligned. Then, two predicates `(first ?a - activity)` and `(last ?a - activity)` hold if activity instance `a` is in the first/last position of the trace. Notice that if a trace is composed by a single activity instance `a`, predicates `traced(a)`, `first(a)` and `last(a)` hold together.

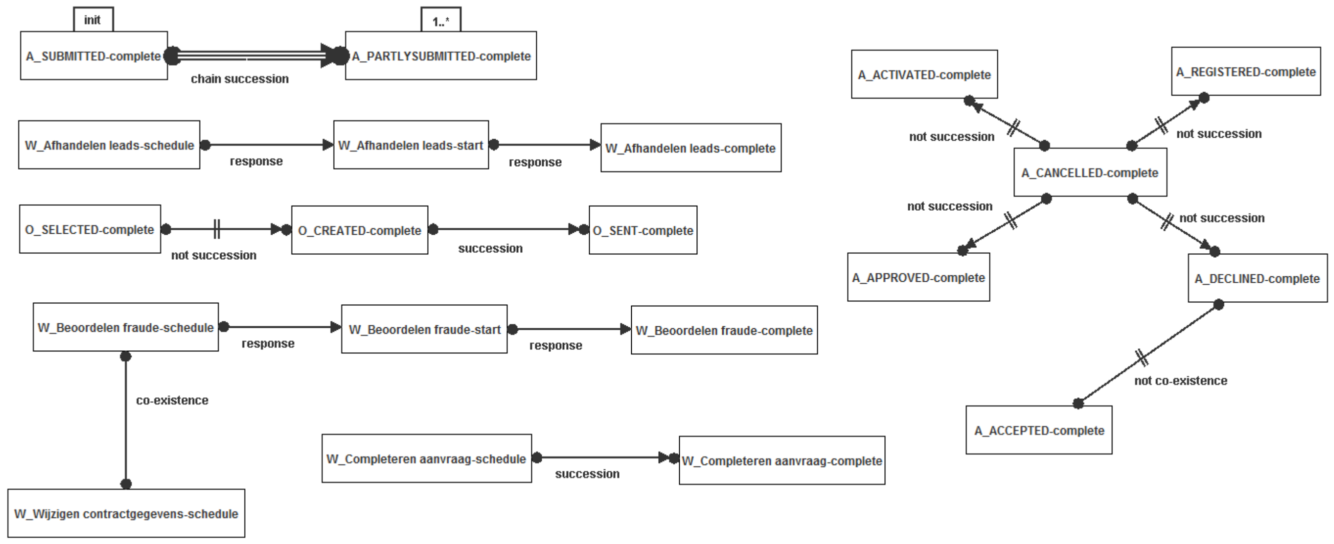


Figure 1: An example DECLARE model.

Finally, to indicate that `a` is followed in the trace by another activity instance `b`, the following predicate is defined: `(succ ?a - activity ?b - activity)`.

Planning actions are the means to add/remove activity instances in/from the trace. We specify 8 basic planning actions to *(i)* add an activity instance in an empty trace, *(ii)* remove an activity instance from a trace of length 1, *(iii)* add/remove an activity instance in/from the first or the last position of a non-empty trace, *(iv)* add/remove an activity instance between/from other existing activity instances already included and ordered in the trace.

Specifically, we modeled the `add` actions in a way that they can only be applied to those activity instances that are not yet included in the trace. For example, the PDDL specification of the action `addBetween` is as follows:

```
(:action addBetween
 :parameters (?x1 - activity ?x2 - activity
              ?x3 - activity)
 :precondition (and (succ ?x1 ?x3)
                    (not (traced ?x2)))
 :effect (and (traced ?x2)
              (succ ?x1 ?x2) (succ ?x2 ?x3)
              (not (succ ?x1 ?x3))
              (increase (total-cost)
                        (adding-cost ?x2))))
```

This action is in charge of inserting activity `x2` (that does not appear in the trace) between activities `x1` and `x3` (that are already included and "connected" in the trace).

Conversely, `remove` actions can be applied only to activity instances that are already in the trace. For example, we define the PDDL action `deleteBetween` as follows:

```
(:action deleteBetween
 :parameters (?x1 - activity ?x2 - activity
              ?x3 - activity)
 :precondition (and (succ ?x1 ?x2)
                    (succ ?x2 ?x3))
 :effect (and (not (traced ?x2))
              (succ ?x1 ?x3)
              (not (succ ?x1 ?x2))
              (not (succ ?x2 ?x3))
              (increase (total-cost)
                        (removing-cost ?x2))))
```

This action removes activity `x2` from the trace. Before the deletion, `x2` is preceded by an activity `x1` and followed by an activity `x3` in the trace. After the deletion, `x1` and `x3` will be connected. Notice that both `add` and `remove` actions have as effect the reduction or the increasing of the length of the trace, as they modify the values of the four predicates defined above.

Notice also that the adding or the deletion of an activity instance has a well defined cost. To specify the cost of a planning action applied to a specific activity instance, we define three PDDL numeric fluents. The first two, `(adding-cost ?a - activity)` and `(removing-cost ?a - activity)`, are used to record the cost of adding/removing an activity instance in/from a trace. The third one, `(total-cost)`, keeps track of the total cost of the alignment.

The constraints belonging to a DECLARE model have their representation as PDDL derived predicates. Unlike basic predicates, they are not directly affected by planning actions, but by means of a formula defined over the basic and derived predicates of the planning domain. For example, the DECLARE constraint *existence(a)* is represented as follows:

```
(:derived (existence-a)
  (exists (?act - a) (traced ?act)))
```

The predicate states that the constraint is fulfilled if in the trace under analysis there exists at least an activity instance of type '`a`' for which the predicate `traced` holds.

Some DECLARE constraints require a more complex specification with nested derived predicates, such in the case of the *response(a,b)* constraint:

```
(:derived (response-a-b)
 (forall (?ta - a) (exists (?tb - b)
                   (response ?ta ?tb))))

(:derived (response ?t1 - activity
                    ?t2 - activity)
 (or (not (traced ?t1)) (succ ?t1 ?t2)
     (and (succ ?t1 ?t3)
          (response ?t3 ?t2))))
```

The predicate is applied on activities of type 'a' and on activities of type 'b', and is satisfied if when an activity instance of type 'a' occurs in the trace, then at least an instance of type 'b' occurs in the trace after the instance of a.

Each DECLARE constraint listed in Table 1 has its own independent formalization as a PDDL derived predicate.

4.2 The Planning Problem

The planning problem involves defining the initial state, the goal condition, and a finite set of objects (or *constants*), required to properly ground all the predicates defined in the planning domain. In our case, constants will correspond to the activity instances involved in the trace to be aligned or included in the DECLARE model.

Number of activity constants. A key problem is to determine how many instances (and constants) for each activity are needed to guarantee completeness, i.e., to guarantee that if there is a way to align the trace, it will be found.

At the very minimum, we need as many constants per activity as the number of times the activity appears in the trace of concern. However, if the trace does *not* conform to the DECLARE constraints, additional instances of activities may be required for alignment, *but how many exactly?*

To answer this question, we define, given a DECLARE model $\mathcal{D}$ and a trace $\mathcal{T}$, the constraint satisfaction problem $\mathcal{C}_{\mathcal{D}}^{\mathcal{T}} = (V, C_{\mathcal{D}} \cup C_{\mathcal{T}})$ where:

- $V = \{\#A_I, \#A_E, \#A_T \mid A \text{ is an activity}\}$. Variables $\#A_I, \#A_E$, and $\#A_T$ denote the number of instances of activity A that appear in the trace of concern, the number of additional instances that may be needed in the worst case for the trace to be "aligned," and the total number of instances (the sum of initial and additional instances).
- $C_{\mathcal{D}}$ is the set of inequality constraints as per Table 1 using the set of DECLARE constraints in $\mathcal{D}$ union the following set of constraints defining variable $\#A_T$:

$$\{(\#A_T = \#A_I + \#A_E) \mid A \text{ is an activity}\}.$$

- $C_{\mathcal{T}} = \{(\#A_I = n) \mid A \text{ is mentioned } n \text{ times in } \mathcal{T}\}$.

Observe that as the DECLARE negative constraints state that certain activity should *not* arise, they never impose the need for extra activity constants. Also, since there are no strict inequalities, inconsistency is ruled out trivially.

Proposition 1. *For any trace $\mathcal{T}$ and set of* DECLARE *constraints in $\mathcal{D}$, the constraint problem $\mathcal{C}_{\mathcal{D}}^{\mathcal{T}}$ has a solution.*

Of course, while any solution to $\mathcal{C}_{\mathcal{D}}^{\mathcal{T}}$ will guarantee completeness of the approaches (see below), we should look at the smallest solutions for $\mathcal{C}_{\mathcal{D}}^{\mathcal{T}}$: the less constants are used, the more efficient the planner will in general be.

As an example, let us consider a DECLARE model $\mathcal{D}_1$ that consists of the constraints *existence(a)* and *chain response(a,b)*, and a trace $\mathcal{T}_1 = \langle a, b, a, c \rangle$, being *a, b, c* activity instances. To find the exact number of activity constants required in the planning problem, we can build a constraint problem as described above, whose solution will be to have 2 constants of type 'a' (a0 and a1), 2 constants of type 'b' (b0 and b1) and 1 constant of type 'c' (c1). Finally, the planning problem will include the following sets of objects:

```
(:objects
 a0 a1 - a
 b0 b1 - b
 c0 - c)
```

Initial State and Goal Condition. The initial state captures the original composition of the trace to be aligned. Basically, it consists of the conjunction of the basic predicates that hold before any alignment step is performed, and of the cost of adding/removing activity constants to/from the trace. For example, if we consider trace $\mathcal{T}_1$ defined before, the initial state would be as follows:

```
(:init
 (traced a0) (traced a1) (traced b0)
 (traced c0) (first a0) (last c0)
 (succ a0 b0) (succ b0 a1) (succ a1 c0)
 (= (adding-cost a0) 1)
 (= (removing-cost a0) 5)
 (= (adding-cost a1) 1)
 (= (removing-cost a1) 5)
 (= (adding-cost b0) 1)
 (= (removing-cost b0) 5))
 (= (adding-cost b1) 1)
 (= (removing-cost b1) 5))
 (= (adding-cost c0) 1)
 (= (removing-cost c0) 5))
```

The goal condition is a conjunction of derived predicates, i.e., of the DECLARE constraints to which the trace must comply. If we consider, for example, the DECLARE model $\mathcal{D}_1$ specified above, the goal condition would be as follows:

```
(:goal (and (existence-a)
            (chain_response-a-b))
```

Finally, as our purpose is to minimize the total cost of the alignment, the planning problem contains the following specification: `(:metric minimize (total-cost))`.

To complete our discussion, let us show an example of trace alignment starting from trace $\mathcal{T}_1$ and model $\mathcal{D}_1$. It is evident that $\mathcal{T}_1$ is not compliant with $\mathcal{D}_1$, since the constraint *chain response(a,b)* does not hold (there is no instance of *b* in the trace that comes immediately after the second instance of *a*). There are two possible alignments in this case: *(i)* remove the second instance of *a* from the trace, or *(ii)* add a second instance of *b* after the second instance of *a*. As the cost of removing an instance of *a* is greater than the cost of adding a second instance of *b*, the second strategy will be preferred by the planner. The plan will therefore consist of a single alignment step, i.e., an Add action to put the second instance of *b* between the second instance of *a* and the first instance of *c*. The aligned trace will be: $\hat{\mathcal{T}}_1 = \langle a, b, a, b, c \rangle$.

This concludes the definition of the planning domain and problem with respect to a given DECLARE model and a trace

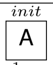
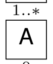
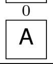
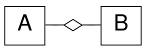
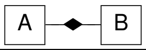
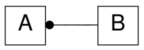
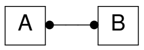
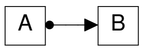
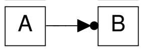
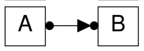
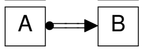
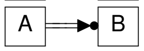
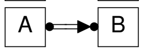
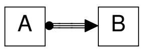
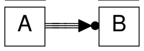
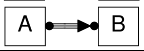
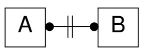
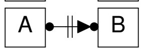
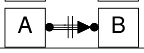
TEMPLATE	NOTATION	LTL FORMALIZATION	INEQUALITY CONSTRAINT
init(A)		A	$(\#A_T \geq 1)$
existence(A)		$\Diamond A$	$(\#A_T \geq 1)$
absence(A)		$\neg\Diamond A$	
choice(A, B)		$\Diamond A \vee \Diamond B$	$(\#A_T \geq 1 \wedge \#B_T \geq 1)$
exclusive_choice(A, B)		$(\Diamond A \vee \Diamond B) \wedge \neg(\Diamond A \wedge \Diamond B)$	$(\#A_T \geq 1 \wedge \#B_T \geq 1)$
responded_existence(A, B)		$\Diamond A \rightarrow \Diamond B$	$(\#A_T = 0 \vee \#B_T \geq 1)$
co_existence(A, B)		$(\Diamond A \rightarrow \Diamond B) \wedge (\Diamond B \rightarrow \Diamond A)$	$(\#A_T \geq 1 \wedge \#B_T \geq 1)$
response(A, B)		$\Box(A \rightarrow \Diamond B)$	$(\#B_E \geq \#A_T)$
precedence(A, B)		$\neg B \mathcal{W} A$	$(\#A_E \geq \#B_T)$
succession(A, B)		$\Box(A \rightarrow \Diamond B) \wedge (\neg B \mathcal{W} A)$	$(\#A_E = \#B_T)$
alternate_response(A, B)		$\Box(A \rightarrow \bigcirc(\neg A \mathcal{U} B))$	$(\#B_E \geq \#A_T)$
alternate_precedence(A, B)		$(\neg B \mathcal{W} A) \wedge \Box(B \rightarrow (\neg B \mathcal{W} A)))$	$(\#A_E \geq \#B_T)$
alternate_succession(A, B)		$\Box(A \rightarrow \bigcirc(\neg A \mathcal{U} B)) \wedge (\neg B \mathcal{W} A) \wedge$ $\Box(B \rightarrow (\neg B \mathcal{W} A)))$	$(\#B_E = \#A_T)$
chain_response(A, B)		$\Box(A \rightarrow \bigcirc B)$	$(\#B_E \geq \#A_T)$
chain_precedence(A, B)		$\Box(\bigcirc B \rightarrow A)$	$(\#A_E \geq \#B_T)$
chain_succession(A, B)		$\Box(A \rightarrow \bigcirc B) \wedge \Box(\bigcirc A \rightarrow B)$	$(\#B_E = \#A_T)$
not_co_existence(A, B)		$(\Diamond A \rightarrow \neg\Diamond B) \wedge (\Diamond B \rightarrow \neg\Diamond A)$	
not_succession(A, B)		$\Box(A \rightarrow \neg\Diamond B)$	
not_chain_succession(A, B)		$\Box(A \rightarrow \neg \bigcirc B)$	

Table 1: DECLARE templates with their corresponding notation, logical formalization, and associated inequality constraints.

to be checked and aligned. It turns out that by solving such planning problem one can solve the trace alignment task.

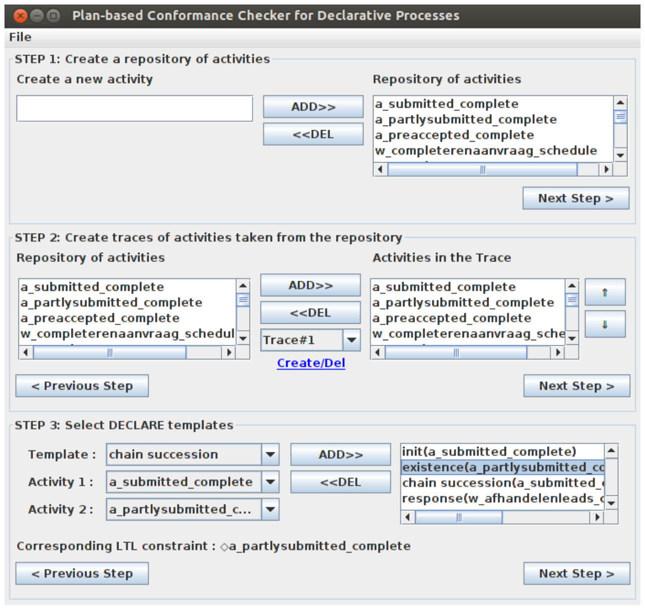
Proposition 2 (Correctness). *Let $\mathcal{P}$ be the planning problem constructed as above for a given set of* DECLARE *constraints $\mathcal{D}$ and a trace $\mathcal{T}$. Then, there exists an alignment $\hat{\mathcal{T}}$ with $\mathcal{D}$ if and only if $\mathcal{P}$ has a solution.*

Intuitively, this holds because each inequality represents the largest "worst" possible number of additional task instances that may be needed. For example, for *response(a,b)* we may need to add one instance of b for each instance of a. It is clear to see that from the plan provided by the planner, one can easily reconstruct the aligned trace. In addition, if we use optimal planning systems, we will achieve alignments with the lowest cost possible.

5 Alignment Tool and Experiments

Following the approach depicted in Section 4, we developed a planning-based alignment tool that is able to find the minimum cost trace alignment against a pre-specified DECLARE process model. The tool, which has been developed as a standard Java application, can be ran in batch mode (to perform large experiments) or interactively using a GUI interface (see Figure 2). The fact that the tool uses a classical planner to perform trace alignment is completely transparent to the user. Specifically, our tool makes use of the LAMA planner (Richter and Westphal 2010) from the FAST-DOWNWARD planning framework.[1] If used via its GUI interface, the process designer needs to *(i)* specify an alphabet $\mathcal{A}$ of activities that will be used for the construction of the traces and of the DECLARE constraints (cf. step 1 in Figure 2); *(ii)* define the structure of every trace to be analyzed (cf. step 2 in Figure 2); and *(iii)* build a DECLARE model (cf. step 3 in Figure 2). The tool also allows us to load existing event logs formatted with the XES standard and to import DECLARE models (saved as XML files) previously designed through the official DECLARE design tool (Westergaard and Maggi 2011). In order to calculate the number of instances required to align traces (cf. Section 4), the tool

[1]`http://www.fast-downward.org/`



Repository of activities: a_submitted_complete, a_partlysubmitted_complete, a_preaccepted_complete, w_completerenaanvraag_schedule

Activities in the Trace (Trace#1): a_submitted_complete, a_partlysubmitted_complete, a_preaccepted_complete, w_completerenaanvraag_sche

Template : chain succession

Activity 1 : a_submitted_complete

Activity 2 : a_partlysubmitted_c...

init(a_submitted_complete)
existence(a_partlysubmitted_co
chain succession(a_submitted_
response(w_afhandelenleads_c

Corresponding LTL constraint : ◇a_partlysubmitted_complete

Figure 2: A screenshot of the alignment tool.

relies on the LP_SOLVE[2] integer linear programming solver. The final step for the user consists of selecting the type of heuristic used to search for a valid plan alignment (the tool allows us to select between a Blind A* search strategy or a Lazy greedy best-first search with preferred operators[3]), the cost of adding/removing a specific activity constant in/from the trace, and the subset of traces of the event log that need to be analyzed.

We tested our planning-based approach in our tool with a *real* event log from the financial domain case study described in Section 3. The log tested comes from real process executions and contains 200 traces of various lengths, between 3 and 58 events. In terms of the model, the original case study includes 16 constraints (see Figure 1), which is arguably a large number of constraints. To have a sense of scalability with respect to the size of the model, though, we have also tested the log against sets of 10 and 20 meaningful subset/superset of such constraints. We performed our tests by using a machine consisting of an Intel Core i7-4770S CPU 3.10GHz Quad Core and 4GB RAM. For the plan synthesis, we selected the Lazy Greedy search strategy, and we assigned the same unitary cost to add/remove activity constants in/from a trace.

The results of our experiments can be seen in Table 2 and Figure 3. For a better understanding of the performance, we separated the time taken for pre-processing from the actual planning time involved. We also aimed at understating how performance scales up with longer traces and more DECLARE constraints. We point out that real traces involve most often less than 25 events, and process models with 16 DECLARE constraints are considered large. So, one should consider our test not only as a practical one based on real settings, but also one that is complex. Given this, some conclusions to be drawn are:

1. Planners are able to align even the most complicated traces in no more than 10 minutes overall (for the largest traces). However, in our logs, 50% of the traces are not longer than 15 events and 70% of the traces contain no more than 25 events. Thus, one would expect to align such traces in less than one minute and a half. This all makes the approach acceptable for offline reasoning.
2. Importantly, the majority of the time taken happens at pre-processing time, mostly in the translation and grounding phase. We note that such pre-planning phase depends on the DECLARE constraints only and the number of activity instances we expect to use. This suggests that, rather than doing the pre-processing for every trace, one could potentially factor that phase out and do it once for the same DECLARE model. As one can see from the results, this would mean that large traces on large models can be aligned in half a minute.
3. The tool performance degrades as traces become longer, though not in a dramatic way to preclude from practical applicability. More interesting, the approach does not seem to degrade on performance as the process model increases on size (i.e., more DECLARE constraints).

We notice that the above remarks should all be taken with care, as they represent preliminary experimentation with only one domain. Nonetheless, the fact that we tested our tool in real logs, with large DECLARE models, and with the planning system out-of-the-box (i.e., no optimizations) suggests that the approach is both feasible and promising.

6 Discussion

Business process conformance has recently become an important research topic. Substantial work on conformance checking has focused on procedural process modeling languages (e.g., (Cook and Wolf 1999; Rozinat and van der Aalst 2008; Adriansyah, van Dongen, and van der Aalst 2011; Mannhardt et al. 2015; van der Aalst 2013b; de Leoni et al. 2014; Munoz-Gama, Carmona, and van der Aalst 2014). In recent years, though, an increasing number of researchers are focusing on the conformance checking with respect to declarative models. For example, (Chesani et al. 2009) proposed an approach based on Abductive Logic Programming for compliance checking with respect to *reactive business rules*; (Montali et al. 2010) extended it by mapping constraints to LTL and evaluating them using automata. In turn, (Burattin et al. 2012) reported an approach to evaluate the conformance and “healthiness” of a log with respect to a DECLARE model, by converting a DECLARE constraint into an automaton and, using a so-called “activation tree” to compute, for each trace, whether a DECLARE constraint is violated or fulfilled. All those works do not aim at aligning the trace of concern but at calculating its “fitness” value (i.e., how much it adheres to a DECLARE model).

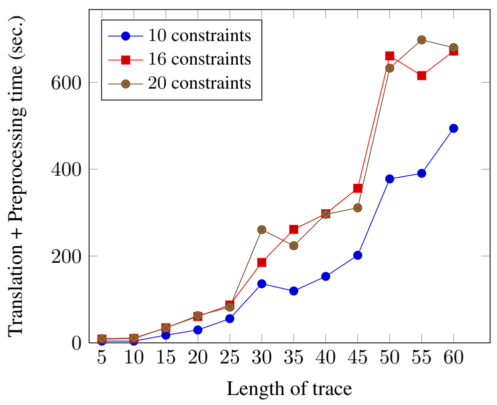
The work described in (de Leoni, Maggi, and van der Aalst 2012; 2015) is closer to ours as it performs conformance checking analysis of a log by an A*-like specialized

[2]http://lpsolve.sourceforge.net/5.5/

[3]http://www.fast-downward.org/Doc/SearchEngine

		10 DECLARE constraints			16 DECLARE constraints			20 DECLARE constraints		
trace length	no. traces	search time	total time	alignment actions	search time	total time	alignment actions	search time	total time	alignment actions
5	51	0.73	4.63	0.2	1.31	9.77	0	1.42	10.86	1
10	42	0.74	4.59	0.1	1.48	11.41	0	1.59	12.55	1.13
15	13	2.21	19.93	0	3.78	38.52	0	3.75	37.97	1
20	18	3.33	32.96	0.56	5.94	66.56	0.53	5.64	68.24	1.06
25	19	5.05	60.75	0.89	7.07	94.14	0.85	7.74	90	3.78
30	9	10.11	146.19	1.33	12.39	197.55	1.14	14.41	275.06	2
35	14	9.26	128.7	0.86	15.38	276.77	1	13.61	237.32	1.67
40	8	12.26	165.31	1.86	19.13	316.55	2	18.61	314.65	3.57
45	10	16.44	218.11	3.3	20.91	377.12	2.33	29.08	340.17	8.44
50	7	22.74	400.45	1.71	31.44	692.58	1.5	36.4	669.37	7.57
55	2	25.25	415.74	3.5	32.39	648.06	4	35.68	733.73	5.5
60	7	30.47	524.69	1.86	33.56	705.91	1.67	35.41	715.58	4.67

Table 2: Experimental results. Numbers reported are averages over all traces of length less or equal than n and longer than $n-5$, where n is the value under column "trace length".



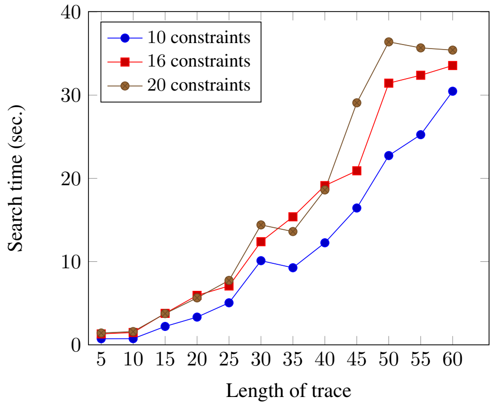


Figure 3: Performance of planning-based alignment: translation, pre-processing, and total time (left) and actual search time (right). Each point in the x-axis represents the interval $(x-5, x]$ on length of traces.

search on an automaton generated from a DECLARE model. As a result of such analysis, a trace is converted into the most "similar" trace that the model accepts. The approach has been implemented as a plug-in in the process mining framework PROM (Maggi 2013).

In this work, we have presented a different perspective to trace alignment via automated planning techniques. Planning provides a mature and "elaboration tolerant" technology and, in fact, in the BPM literature, there exists a number of works utilising planning in the various stages of a process life cycle, e.g., to create process models from declarative activity specifications (Marrella and Lespérance 2013) or to run, monitor and adapt processes at run-time (Marrella, Russo, and Mecella 2012; Marrella, Mecella, and Sardiña 2014). However, the application of automated planning techniques to perform trace alignment is novel. While preliminary, the results obtained in an event log from a real case study are promising and demonstrate that the planning approach can cope with realistic complex settings.

More importantly, and in contrast with the search-based technique in (Maggi 2013; de Leoni, Maggi, and van der Aalst 2012; 2015), our approach based on planning is well suited for advanced types of conformance checking and trace alignment in which *data* (Montali et al. 2013) and *time* (Westergaard and Maggi 2012) are taken into account, something that is being recognized as a necessary though challenging problem in the BPM community. Being an area rooted in knowledge representation, planning provides convenient ways for reasoning about rich type of knowledge around processes. For example, there are planning systems coping with time that can be potentially exploited to solve this issue (Eyerich, Mattmüller, and Röger 2012).

We note that the encoding proposed may not be the only one for the problem being tackled. In fact, we have so far not paid much attention to knowledge engineering aspects, since our objective was to show that a reduction to planning is possible, even with the PDDL limitation of having a finite number of objects (for task instances). Hence, we do not rule out that better, more efficient, encodings are possible. For example, our encoding does not handle symmetries (which are many) and hence we rely on the planner to find and handle them.

Before concluding, we notice that another approach worth investigating to tackle the alignment task is Constraint Satisfaction Problems (Tsang 1993). However, it is not clear how to represent the DECLARE constraints and alignment steps without requiring a combinatorial explosions in the number of variables.

Finally, we are also interested in extending our study from DECLARE to whole LTL on finite traces.

Acknowledgments. This work has been partly supported by the Italian projects RoMA and NEPTIS, the Italian cluster SM&ST and the Sapienza Grant ICE. Sebastian Sardina acknowledges the support of the Australian Research Council under Discovery Project DP120100332.

References

3TU Data Center. 2012. BPI Challenge 2012 Event Log.

Adriansyah, A.; van Dongen, B.; and van der Aalst, W. 2011. Conformance Checking Using Cost-Based Fitness Analysis. In *15th Int. Enterprise Distributed Object Computing Conference (EDOC)*.

Bonet, B., and Geffner, H. 2001. Planning and Control in Artificial Intelligence: A Unifying Perspective. *Applied Intelligence* 14(3).

Burattin, A.; Maggi, F. M.; van der Aalst, W.; and Sperduti, A. 2012. Techniques for a Posteriori Analysis of Declarative Processes. In *16th Int. Enterprise Distributed Object Computing Conference (EDOC)*.

Chesani, F.; Mello, P.; Montali, M.; Riguzzi, F.; Sebastianis, M.; and Storari, S. 2009. Checking Compliance of Execution Traces to Business Rules. In *Business Process Management Workshops*. Springer.

Cook, J. E., and Wolf, A. L. 1999. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Trans. Softw. Eng. Methodol.* 8(2).

de Leoni, M.; Munoz-Gama, J.; Carmona, J.; and van der Aalst, W. 2014. Decomposing Alignment-Based Conformance Checking of Data-Aware Process Models. In *OTM Conf. Int. Conferences*.

de Leoni, M.; Maggi, F. M.; and van der Aalst, W. 2012. Aligning Event Logs and Declarative Process Models for Conformance Checking. In *Business Process Management*. Springer.

de Leoni, M.; Maggi, F. M.; and van der Aalst, W. 2015. An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Inf. Syst.* 47.

Edelkamp, S., and Hoffmann, J. 2004. PDDL 2.2: The Language for the Classical Part of the 4th Int. Planning Competition. Technical report, Albert Ludwigs Universität Institüt fur Informatik, Freiburg.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2012. *Using the context-enhanced additive heuristic for temporal and numeric planning*. Springer.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Elsevier.

Ly, L. T.; Rinderle-Ma, S.; Göser, K.; and Dadam, P. 2012. On enabling integrated process compliance with semantic constraints in process management systems. *Inf. Syst. Frontiers* 14(2).

Maggi, F. M. 2013. Declarative Process Mining with the Declare Component of ProM. In *Business Process Management (Demos)*.

Mannhardt, F.; de Leoni, M.; Reijers, H.; and van der Aalst, W. 2015. Balanced multi-perspective checking of process conformance. *Computing*.

Marrella, A., and Lespérance, Y. 2013. Synthesizing a Library of Process Templates through Partial-Order Planning Algorithms. In *Business-Process and Information Systems Modeling (BPMDS)*.

Marrella, A.; Mecella, M.; and Sardiña, S. 2014. SmartPM: An Adaptive Process Management System through Situation Calculus, IndiGolog, and Classical Planning. In *Knowledge Representation and Reasoning (KR)*.

Marrella, A.; Russo, A.; and Mecella, M. 2012. Planlets: Automatically Recovering Dynamic Processes in YAWL. In *OTM Conf. Int. Conferences*.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C. A.; Ram, A.; Veloso, M.; Weld, D. S.; and Wilkins, D. E. 1998. PDDL - The Planning Domain Definition Language. Technical Report DCS TR-1165, Yale Center for Computational Vision and Control.

Montali, M.; Pesic, M.; van der Aalst, W.; Chesani, F.; Mello, P.; and Storari, S. 2010. Declarative Specification and Verification of Service Choreographies. *ACM Trans. on the Web* 4(1).

Montali, M.; Chesani, F.; Mello, P.; and Maggi, F. M. 2013. Towards data-aware constraints in DECLARE. In *ACM Symp. on Applied Computing (SAC)*.

Munoz-Gama, J.; Carmona, J.; and van der Aalst, W. 2014. Single-Entry Single-Exit decomposed conformance checking. *Inf. Syst.* 46.

Pnueli, A. 1977. The Temporal Logic of Programs. In *18th Annual Symp. on Foundations of Computer Science (SFCS)*. IEEE Computer Society.

Richter, S., and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research (JAIR)* 39.

Rozinat, A., and van der Aalst, W. 2008. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.* 33(1).

Sadiq, S.; Governatori, G.; and Naimiri, K. 2007. Modeling Control Objectives for Business Process Compliance. In *Business Process Management*. Springer.

Sadiq, S. 2011. A Roadmap for Research in Business Process Compliance. In *Business Inf. Systems Workshops*. Springer.

Task Force on Process Mining. 2013. *XES Standard Definition*.

Tsang, E. 1993. *Foundations of Constraint Satisfaction*. Academic Press.

van der Aalst, W.; Pesic, M.; and Schonenberg, H. 2009. Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science - R&D*.

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

van der Aalst, W. 2013a. Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*.

van der Aalst, W. 2013b. Decomposing Petri nets for process mining: A generic approach. *Distributed and Parallel Databases* 31(4).

Verbeek, H.; Buijs, J.; van Dongen, B.; and van der Aalst, W. 2010. XES, XESame, and ProM 6. In *Information Systems Evolution - CAiSE Forum*.

Westergaard, M., and Maggi, F. M. 2011. Declare: A Tool Suite for Declarative Workflow Modeling and Enactment. In *Business Process Management (Demos)*.

Westergaard, M., and Maggi, F. M. 2012. Looking into the Future. Using Timed Automata to Provide a Priori Advice about Timed Declarative Process Models. In *OTM Conf. Int. Conferences*.