Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

On the Disruptive Effectiveness of Automated Planning for LTL$_f$-Based Trace Alignment

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

Fabio Patrizi
Sapienza - Universitá di Roma, Italy
patrizi@dis.uniroma1.it

Abstract

One major task in business process management is that of aligning real process execution traces to a process model by (minimally) introducing and eliminating steps. Here, we look at declarative process specifications expressed in Linear Temporal Logic on finite traces (LTL$_f$). We provide a sound and complete technique to synthesize the alignment instructions relying on finite automata theoretic manipulations. Such a technique can be effectively implemented by using planning technology. Notably, the resulting planning-based alignment system significantly outperforms all current state-of-the-art ad-hoc alignment systems. We report an in-depth experimental study that supports this claim.

1 Introduction

In this paper, we introduce a planning approach to solve the problem of *trace alignment* for business processes (BPs). A BP defines the temporal (partial) ordering of some activities of interest. Some examples include insurance claim processing, order handling, and hospital procedures. BPs can be specified either procedurally or declaratively, depending on the purpose of the specification: the former approach is well suited for actual execution; the latter, which is the one we deal with, is typically used to provide a description of the process amenable to various forms of analysis.

BPs are supported by information systems that drive process executions (e.g., guarantee that activity executions take place in the expected order) and store the event data related to the activities involved in each execution – e.g., in an insurance claim process, the system might record, for each claim, the sequence of activities executed. The resulting log consists of a set of *traces*, each related to a distinct process execution, and consisting of a sequence of *events*, which, in turn, contain information about the executed activities.

Often, BP activities are human-based. For instance, in an insurance claim process, a human operator might be responsible for collecting all the documents related to the claim, checking the information they contain and, if correct, start the claim process (by initiating some activity, possibly demanded to another operator). As a result, the log traces can be inconsistent with the expected process behavior. Thus, the need arises for identifying and analyzing such traces with the aim of preventing errors from occurring again. This is the goal of *trace alignment*.

Trace alignment is the problem of: *(i)* checking whether an actual trace stemmed from a BP execution conforms to the expected process behavior and, if not, *(ii)* finding a "minimal set" of changes that "align" the trace to the process. Such changes consist in adding or removing activities at some points in the trace. In this paper, we focus on trace alignment against declarative specifications. The input trace is a sequence of activity names, while the process behavior is specified in LTL$_f$ (Linear Temporal Logic on finite traces (De Giacomo and Vardi 2013)). The goal of the problem is to make the trace satisfy the LTL$_f$ formula.

We address this problem by resorting to *cost-optimal planning*, a form of deterministic planning where actions have costs, and where a successful plan of minimal cost (defined as the sum of the costs of the component actions) has to be found. The intuition behind our solution is that actions capture additions and deletions to the input trace (having non-zero costs), and the goal is to make the input trace conform with the process behavior at a minimal cost. We reduce trace alignment to cost-optimal planning in two steps. First, we provide a sound and complete technique, based on automata-theoretic manipulations, to synthesize the alignment instructions. Then, we show how the technique can be implemented with planning technology.

We validate our approach through an extensive experimentation showing that it impressively outperforms a technique included in the ProM toolkit for BP analysis (promtools.org) and based on an implementation of A* specifically tailored for the alignment problem (de Leoni, Maggi, and van der Aalst 2012; 2015), as well as previous approaches based on classical planning (De Giacomo et al. 2016). We stress that process behaviors are typically spec-

ified in DECLARE (van der Aalst, Pesic, and Schonenberg 2009), a well-established declarative process modeling language, which can be seen as a dialect of LTL_f, while our technique deals with the whole LTL_f. Thus, our approach is more general than any other previous one.

2 The logic LTL_f

Let $\mathcal{L}_{Prop}$ be the set of propositional formulas over a finite set $Prop$ of propositional symbols. The logic LTL_f (LTL interpreted on finite traces) is defined as follows :

$$\varphi ::= \phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2,$$

where $\phi \in \mathcal{L}_{Prop}$, and $\bigcirc$ and $\mathcal{U}$ are, respectively, the *next* and *until* operators. For syntactical convenience, we also use the standard abbreviations $\vee$, $\bullet$ (*weak next*), $\Diamond$ (*eventually*), $\Box$ (*always*), and $\mathcal{W}$ (*weak until*), defined as follows: $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$; $\bullet\varphi \equiv \neg\bigcirc\neg\varphi$; $\Diamond\varphi \equiv \mathit{true}\,\mathcal{U}\,\varphi$; $\Box\varphi \equiv \neg\Diamond\neg\varphi$; $\varphi_1 \,\mathcal{W}\, \varphi_2 \equiv (\varphi_1 \,\mathcal{U}\, \varphi_2) \vee \Box\varphi_1$.

Formulas of LTL_f are interpreted on finite nonempty words t from $(2^{Prop})^+$, which we call *traces*. Given a trace t, $\mathit{length}(t)$ denotes its length and $t(i)$, with $1 \leq i \leq \mathit{length}(t)$, the propositional interpretation of t at the i-th position (represented, as standard, by the set of propositions that are true in the interpretation). We inductively define when an LTL_f formula φ *is true* at step i of t, written $t, i \models \varphi$, as follows:

- $t, i \models \phi$ iff $t(i) \models \phi$ (ϕ propositional);
- $t, i \models \neg\varphi$ iff $t, i \not\models \varphi$;
- $t, i \models \varphi_1 \wedge \varphi_2$ iff $t, i \models \varphi_1$ and $t, i \models \varphi_2$;
- $t, i \models \bigcirc\varphi$ iff $i < \mathit{length}(t)$ and $t, i{+}1 \models \varphi$;
- $t, i \models \varphi_1 \,\mathcal{U}\, \varphi_2$ iff for some j s.t. $i \leq j \leq \mathit{length}(t)$, we have $t, j \models \varphi_2$, and for all k s.t. $i \leq k < j$, we have $t, k \models \varphi_1$.

We say that t *satisfies* φ, written $t \models \varphi$, if $t, 1 \models \varphi$.

Every LTL_f formula φ can be associated with a nondeterministic finite-state automaton (NFA) $\mathcal{A}$ that accepts exactly all traces satisfying φ (De Giacomo and Vardi 2015). Formally, such NFA is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \rho, F \rangle$, where: 1. $\Sigma = Prop$ is the *input alphabet*; 2. Q is the finite set of *automaton states*; 3. $q_0 \in Q$ is the *initial state*; 4. $\rho \subseteq Q \times \mathcal{L}_{Prop} \times Q$ is the *transition relation*; and 5. $F \subseteq Q$ is the set of *final states*.

Let $t = e_1 \cdots e_n$ be a trace and $\mathcal{A}$ the NFA associated with an LTL_f formula φ. A *computation* of $\mathcal{A}$ on t is a sequence $\delta = q_0 \xrightarrow{e_1} q_1 \cdots q_{n-1} \xrightarrow{e_n} q_n$ s.t., for $i = 0, \ldots, n-1$, there exists a transition $q_i \xrightarrow{\psi_i} q_{i+1} \in \rho$ s.t. $e_i \models \psi_i$. Since $\mathcal{A}$ is nondeterministic, there exist, in general, many computations of $\mathcal{A}$ on a trace t. We say that $\mathcal{A}$ *accepts* t if there exists a computation δ on t s.t. the last state is final, i.e., belongs to F.

3 The Trace Alignment Problem

A *log trace* is a trace such that the propositional interpretation associated with each position contains only one proposition (i.e., is a singleton). In this paper, we deal only with log traces. This is not a restriction, as log traces represent the natural input of the problem addressed here. For notational convenience, we use single propositions (and not singletons). Thus, we write $t = a\ b\ c$, instead of $t = \{a\}\{b\}\{c\}$.

Consider a trace t and an LTL_f formula φ s.t. $t \not\models \varphi$. We are interested in "repairing" t, i.e., to transform it into a new trace $\hat{t}$ s.t. $\hat{t} \models \varphi$. We consider two types of *repair*: *addition* and *deletion*. Given a trace $t = e_1 \cdots e_k \cdots e_n$, a proposition p can be added to t at position k only if $1 \leq k \leq n$. After the addition, the resulting trace is $\hat{t} = e_1 \cdots e_{k-1}\ p\ e_k \cdots e_n$. A proposition e_k, $1 \leq k \leq n$, can also be deleted from t, with resulting trace $\hat{t}' = e_1 \cdots e_{k-1}\ e_{k+1} \cdots e_n$. For traces t and $\hat{t}$, we define the cost function cost s.t. $\mathit{cost}(t, \hat{t}) = c$ iff c is the minimal number of repairs needed to obtain $\hat{t}$ from t.

The *trace alignment problem* is defined as follows: given a trace t and an LTL_f formula φ s.t. $t \not\models \varphi$, find a trace $\hat{t}$ s.t. $\hat{t} \models \varphi$ and $\mathit{cost}(t, \hat{t})$ is minimal. It is immediate to see that if φ is satisfiable, a solution to the problem always exists, as the repairs allow one to obtain any (log) trace, no matter what the original trace is.

Trace alignment can be addressed by resorting to automata. To see this, let $t = e_1 \cdots e_n$ be the log trace, φ the constraint to check t against, and $\mathcal{A} = \langle \Sigma, Q, q_0, \rho, F \rangle$ the corresponding NFA, which we call the *constraint automaton*. From t, we define a further automaton, called the *trace automaton*, $\mathcal{T} = \langle \Sigma_t, Q_t, q_0^t, \rho_t, F_t \rangle$, where: 1. $\Sigma_t = \{e_1, \ldots, e_n\}$; 2. $Q_t = \{q_0^t, \ldots, q_n^t\}$ is a set of $n{+}1$ arbitrary states; 3. $\rho^t = \bigcup_{i=0,\ldots,n-1} \langle q_i^t, e_{i+1}, q_{i+1}^t \rangle$; 4. $F^t = \{q_n^t\}$. By construction, $\mathcal{T}$ is deterministic and accepts only t.

Next, we augment $\mathcal{T}$ and $\mathcal{A}$ to make them suitable for trace alignment. From $\mathcal{T}$, we generate the automaton $\mathcal{T}^+ = \langle \Sigma_t^+, Q_t, q_0^t, \rho_t^+, F_t \rangle$, where:

- Σ_t^+ contains all the propositions in Σ_t, plus: one fresh proposition del_p, for all propositions $p \in \Sigma$; and one fresh proposition add_p, for all propositions $p \in \Sigma \cup \Sigma_t$;
- ρ_t^+ contains all the transitions in ρ_t, plus: a new transition $\langle q, del\_p, q' \rangle$, for all transitions $\langle q, p, q' \rangle \in \rho_t$; and, for all propositions p in Σ_t and states $q \in Q_t$, a new transition $\langle q, add\_p, q \rangle$, if there is no transition $\langle q, p, q' \rangle \in \rho_t$ (for all $q' \in Q_t$).

We call the newly introduced propositions *repair propositions*. Notice that, by construction, $\mathcal{T}^+$ is deterministic. Intuitively, $\mathcal{T}^+$ accepts all the traces over Σ obtained by repairing t, with the repairs "marked" by repair propositions. For instance, the automaton $\mathcal{T}^+$ for $t = a\ b$ accepts t but also $t' = del\_a\ b$ but not $t'' = b$, as in the latter, despite being obtained by repairing t, repairs are not marked (by repair propositions).

Similarly from $\mathcal{A}$, we obtain an automaton $\mathcal{A}^+ = \langle \Sigma^+, Q, q_0, \rho^+, F \rangle$, s.t.:

- $\Sigma^+ = \Sigma_t^+$; and
- ρ^+ contains all the transitions in ρ, plus: one fresh transition $\langle q, del\_p, q \rangle$ for all $q \in Q$ and $p \in \Sigma_t$; and one fresh transition $\langle q, add\_p, q' \rangle$ for all transitions $\langle q, \psi, q' \rangle \in \rho$ s.t. $p \models \psi$.

Intuitively, $\mathcal{A}^+$ accepts all traces $\hat{t}$ that satisfy φ and have been obtained by repairing t, with the repairs explicitly

marked. For instance, let $t = a\ b$, let $\mathcal{A}$ be the automaton for $\varphi = \Box(b \rightarrow \Diamond c)$, and let $\mathcal{A}^+$ its augmented version. Obviously, $t \not\models \varphi$ (because c does not occur after b). Therefore, $\mathcal{A}$ and $\mathcal{A}^+$ do not accept t. However, if we repair t by adding c at the end, and we explicitly mark the repair with add_c, then $\mathcal{A}^+$ accepts the new trace $\hat{t} = a\ b\ add\_c$.

The following result relates the trace alignment problem to the automata defined above.

Theorem 1 *Consider a log trace t and an LTL_f formula φ, both over $Prop$, s.t. $t \not\models \varphi$. Let $\mathcal{T}^+$ and $\mathcal{A}^+$ be the automata obtained from t and φ, as described above. If t^+ is a trace accepted by both $\mathcal{A}^+$ and $\mathcal{T}^+$ containing a minimal number of repair propositions (with respect to all other traces accepted by $\mathcal{A}^+$ and $\mathcal{T}^+$), then a trace $\hat{t}$ with minimal cost $cost(t, \hat{t})$ s.t. $\hat{t} \models \varphi$ can be obtained from t^+ by removing all propositions of the form del_p and replacing all propositions of the form add_p with p.*

Thus, given t and φ, trace alignment is equivalent to searching for a trace accepted by both $\mathcal{A}^+$ and $\mathcal{T}^+$ with a minimal number of repair propositions.

We close the section by observing that Th. 1 can be easily extended to the case of many LTL_f constraints $\varphi_1, \ldots, \varphi_n$. One way to do so consists in taking the conjunction of such constraints and then proceeding as shown above. Another way, which is the one we use later on, consists in computing the augmented constraint automata for all constraints, i.e., $\mathcal{A}_1^+, \ldots, \mathcal{A}_n^+$, and then searching for a trace that is accepted by $\mathcal{T}^+$ and $\mathcal{A}_1^+, \ldots, \mathcal{A}_n^+$. It is immediate to see that such approaches are equivalent.

We next show how to take advantage of the planning technology to efficiently search for the desired repaired trace.

4 Trace Alignment as Planning

A *(deterministic) planning domain with action costs* (over a set of propositions $Prop$) is a tuple $\mathcal{D} = \langle \mathcal{S}, A, \mathcal{C}, \tau \rangle$, where: 1. $\mathcal{S} \subseteq 2^{Prop}$ is the finite set of *domain states*, seen as propositional interpretations; 2. A is the finite set of *domain actions*; 3. $\mathcal{C} : A \mapsto \mathbb{N}^+$ is a *cost function*; 4. $\tau : \mathcal{S} \times A \mapsto \mathcal{S}$ is the *transition function*. A *plan for $\mathcal{D}$* is a finite sequence $\pi \in A^*$ of actions. A plan $\pi = a_1 \cdots a_n$ is said to be *executable* from a state $s_0 \in \mathcal{S}$, if there exists a sequence of states $\sigma = s_0 \cdots s_n$ s.t., for $i = 0, \ldots, n-1$, $s_{i+1} = \tau(s_i, a_{i+1})$. If it exists, σ (which is unique) is said to be the *(domain) trace induced by π*. Finally, the *cost* of π (on $\mathcal{D}$) is $\mathcal{C}(\pi) \doteq \sum_{i=1,\ldots,n} \mathcal{C}(a_i)$.

A *cost-optimal planning problem* is a tuple $\mathcal{P} = \langle \mathcal{D}, s_0, G \rangle$, where: 1. $\mathcal{D}$ is a planning domain with action costs; 2. $s_0 \in \mathcal{S}$ is the *initial state* of the problem; 3. G, the *problem goal*, is a propositional formula over $Prop$. A plan π is a *solution* to $\mathcal{P}$ if the last state s_n of the trace induced by π is such that $s_n \models G$. A solution π to $\mathcal{P}$ is said to be *optimal* if for all other solutions π', we have $\mathcal{C}(\pi) \leq \mathcal{C}(\pi')$.

Using Theorem 1, we can reduce trace alignment to cost-optimal planning. To see this, consider an instance of trace alignment, i.e., a trace t and an LTL_f formula φ. To build the planning problem, we first compute the automata $\mathcal{T}^+ = \langle \Sigma_t^+, Q_t, q_0^t, \rho_t^+, F_t \rangle$ and $\mathcal{A}^+ = \langle \Sigma^+, Q, q_0, \rho^+, F \rangle$, and then define the planning domain $\mathcal{D} = \langle \mathcal{S}, A, \mathcal{C}, \tau \rangle$, s.t.:

- $\mathcal{S} \subseteq 2^{Q_t \cup Q}$ (automata states are seen as propositions);
- $A = \{sync\_e, del\_e, add\_e \mid e \in \Sigma \cup \Sigma_t\}$; (repair propositions are seen as actions);
- for all $e \in \Sigma \cup \Sigma_t$, $\mathcal{C}(sync\_e) = 0$ and $\mathcal{C}(del\_e) = \mathcal{C}(add\_e) = 1$;
- τ is defined as follows, for all $e \in \Sigma \cup \Sigma_t$, $q_t, q_t' \in Q_t$, and $R, R' \subseteq Q$:
  - $\tau(\{q_t\} \cup R, sync\_e) = \{q_t'\} \cup R'$ iff $q_t \xrightarrow{e} q_t' \in \rho_t^+$ and, for all $q \in R$ and $q' \in R'$, there exists ψ s.t. $e \models \psi$ and $q \xrightarrow{\psi} q' \in \rho^+$;
  - $\tau(\{q_t\} \cup R, del\_e) = \{q_t'\} \cup R'$ iff $q_t \xrightarrow{del\_e} q_t' \in \rho_t^+$ and, for all $q \in R$ and $q' \in R'$, $q \xrightarrow{del\_e} q' \in \rho^+$;
  - $\tau(\{q_t\} \cup R, add\_e) = \{q_t'\} \cup R'$ iff $q_t \xrightarrow{add\_e} q_t' \in \rho_t^+$ and, for all $q \in R$ and $q' \in R'$, $q \xrightarrow{add\_e} q' \in \rho^+$;

The domain is intended to represent the synchronous product of $\mathcal{T}^+$ and $\mathcal{A}^+$, that is, intuitively, the synchronous execution of $\mathcal{T}^+$ and $\mathcal{A}^+$ over the same input. Propositions represent the states that each automaton is in. Since $\mathcal{T}^+$ is deterministic, every state of the planning domain contains exactly one state from Q_t; instead, $\mathcal{A}^+$ being nondeterministic, many states from Q are included in each domain state. Also, the transitions triggered by the actions take $\mathcal{T}^+$ to exactly one successor state but, in general, $\mathcal{A}^+$ to many ones. Actions capture the repairs on the original trace t: $sync$ (synchronization) actions stand for no change, add for addition and del for deletion. Thus, executable plans represent sequences of changes applied to the original trace t (based on which both $\mathcal{T}^+$ and $\mathcal{A}^+$ are generated). $sync$ actions have cost 0 while add and del have cost 1 thus, when searching for a plan, $sync$ actions are preferred, as well as, consequently, repaired traces "closer" to t.

Observe that, as actions take place, the (augmented) trace automaton and the constraint automaton are progressed. The intuition is that when a plan takes both automata to a final state, the repaired trace resulting from the plan execution – i.e., the trace obtained from the obtained plan by replacing $sync\_e$ and add_e with e, and by removing del_e – satisfies the original constraint φ. Of course, for the constraint automaton it is enough that any of its current states is final. Finally, if the plan has minimal cost, the obtained trace is an optimal solution to the original trace alignment problem. The above observations suggest to define the cost-optimal planning problem $\mathcal{P} = \langle \mathcal{D}, s_0, G \rangle$ as follows: $s_0 = \{q_0, q_0^t, \}$ and $G = q_t^f \wedge \bigvee_{q \in F} q$, for $q_t^f \in F_t$ (remember that $\mathcal{T}^+$ has a single final state).

The construction of the cost-optimal planning problem defined above can be easily generalized to many constraints. To this end, it is enough to include, in the domain states, new propositions used to capture the states of all automata and then generalize the actions so that their execution progresses all the automata at once. This approach is adopted in Section 6, where we deal with multiple constraints.

5 Encoding the Alignment Problem in PDDL

In this section, we show how, given a set of *augmented constraint automata* $\mathcal{A}_1^+, .., \mathcal{A}_n^+$ obtained from n LTL$_f$ formulas $\varphi_1, .., \varphi_n$, and an *augmented trace automaton* $\mathcal{T}^+$ obtained from a trace t, we build a cost-optimal planning domain $\mathcal{D}$ and a problem instance $\mathcal{P}$ in the standard *Planning Domain Definition Language* (PDDL (McDermott et al. 1998)). $\mathcal{D}$ and $\mathcal{P}$ can be used to feed any state-of-the-art planning technology. In particular, we represent them making use of PDDL 2.1 (Fox and Long 2003), which provides numeric features to keep track of the costs of planning actions. A solution plan for $\mathcal{P}$ amounts to the set of interventions of minimal cost to align the trace with respect to the LTL$_f$ formulas.

Planning Domain. In $\mathcal{D}$, we provide two abstract types: `activity` and `state`. The first captures the activities involved in a transition between two different states of a constraint/trace automaton. The second is used to uniquely identify the states of any constraint automaton (through the sub-type `automaton_state`) and of the trace automaton (through the sub-type `trace_state`). To capture the structure of the automata and to monitor their evolution, we defined four *domain propositions* as boolean predicates in $\mathcal{D}$:

- `(trace ?t1 - trace_state ?e - activity ?t2 - trace_state)` holds if there exists a transition in the trace automaton from two different states `t1` and `t2`, being `e` the activity involved in the transition.
- `(automaton ?s1 - automaton_state ?e - activity ?s2 - automaton_state)` holds if there exists a transition from two different states `s1` to `s2` of a constraint automaton, being `e` the activity involved in the transition.
- `(cur_state ?s - state)` holds if `s` is the current state of a constraint/trace automaton.
- `(final_state ?s - state)` holds if `s` is a final accepting state of a constraint/trace automaton.

Furthermore, we define a *numeric fluent* `total-cost` to keep track of the cost of the alignment. In the remainder of the paper, we remain consistent with PDDL terminology, which allows for both the values of predicates and fluents to change as result of the execution of actions.

Planning actions are used to express the *alignments* on the original trace t. Each action is characterized by its *preconditions* and *effects*, stated in terms of the domain propositions. In our encoding, we have defined three actions to perform *synchronous moves* in the trace automaton and in the constraint automata, or to *add/remove* activities in/from the trace automaton.

```
(:action sync
 :parameters (?t1 - trace_state ?e - activity
              ?t2 - trace_state)
 :precondition (and (cur_state ?t1) (trace ?t1 ?e ?t2))
 :effect(and (not (cur_state ?t1)) (cur_state ?t2)
         (forall (?s1 ?s2 - automaton_state)
                 (when (and (cur_state ?s1)
                            (automaton ?s1 ?e ?s2))
                       (and (not (cur_state ?s1))
                            (cur_state ?s2))))))

(:action add
 :parameters (?e - activity)
 :effect (and (increase (total-cost) 1)
          (forall (?s1 ?s2 - automaton_state)
            (when (and (cur_state ?s1)
                       (automaton ?s1 ?e ?s2))
                  (and (not (cur_state ?s1))
                       (cur_state ?s2))))))

(:action del
 :parameters (?t1 - trace_state ?e - activity
              ?t2 - trace_state)
 :precondition (and (cur_state ?t1) (trace ?t1 ?e ?t2))
 :effect(and (increase (total-cost) 1)
             (not (cur_state ?t1)) (cur_state ?t2)))
```

We modeled `sync` and `del` in such a way that they can be applied only if there exists a transition from the current state `t1` of the trace automaton to a subsequent state `t2`, being `e` the activity involved in the transition. Differently from the abstract encoding presented in Section 4, here states occur as action parameters. This is a technical convenience adopted to avoid existential quantifiers in preconditions and effects.

Notice that, while the `del` action yields a *single* move in the trace automaton, the `sync` action yields, in addition, one move per constraint automaton (all to be performed synchronously). In particular, a synchronous move is performed in each constraint automaton for which there exists a transition involving the activity `e` that connects `s1` – the current state of the automaton – with a different state `s2`. Finally, the `add` action is performed only for transitions involving the activity `e` between two different states of any constraint automaton, with the current state of the trace automaton that remains the same after the execution of the action. It is worthy observing how the execution of a `del` or `add` action makes total cost (of the alignment) increasing of a predefined value (here we are assuming a unitary cost); conversely, `sync` action has no cost, as it stands for *no change* in the trace.

Planning Problem. In $\mathcal{P}$, we first define a finite set of constants required to properly ground all the domain propositions defined in $\mathcal{D}$. In our case, constants will correspond to the state and activity instances involved in the trace automaton and in any constraint automaton.

Secondly, we define the *initial state* of $\mathcal{P}$ to capture the exact structure of the trace automaton and of every constraint automaton. This includes the specification of all the existing transitions that connect two different states of the automata. The current state and the accepting states of any trace/constraint automaton are identified as well.

Thirdly, to encode the goal condition, we first preprocess each constraint automaton by: 1. adding a fresh dummy state with no outgoing transitions; 2. adding a fresh special action, executable only in the accepting states of the original automaton, which makes the automaton move to the dummy state; 3. including in the set of accepting states only the dummy state. Then, we define the goal condition as the conjunction of the accepting states of the trace automaton and of all the accepting states of the constraint automata. In this

TEMPLATE	FORMALIZATION	TEMPLATE	FORMALIZATION
existence(A)	$\Diamond A$	absence(A)	$\neg\Diamond A$
resp. existence(A,B)	$\Diamond A \rightarrow \Diamond B$	not resp. existence(A,B)	$\Diamond A \rightarrow \neg\Diamond B$
response(A,B)	$\Box(A \rightarrow \Diamond B)$	not response(A,B)	$\Box(A \rightarrow \neg\Diamond B)$
chain response(A,B)	$\Box(A \rightarrow \bigcirc B)$	not chain response(A,B)	$\Box(A \rightarrow \neg\bigcirc B)$

Table 1: LTL_f formalization of some DECLARE templates.

trace length	no. traces	Fast-Downward	SymBA*-2	de Giacomo et al.	de Leoni et al.	Alignment Cost
Real-life log		16 constraints				
3-50	607	2.47	2.92	11.02	0.15	0.63
51-75	38	2.59	4.13	35.69	0.45	1.02
76-100	5	2.65	4.99	72.43	2.78	2.4
101-128	4	2.66	5.61	123.45	5.88	2.5

Table 2: Experimental results of the *real-life* case study. The time (in *seconds*) refers to the average per trace.

way, we avoid using disjunctions in goal formulas, as not supported by all planners.

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

6 Experiments

We have developed a planning-based alignment tool as a standard Java application that implements the approach discussed in Sections 4 and 5. The tool can be ran interactively using a GUI interface, and allows us to load existing logs formatted with the XES (eXtensible Event Stream) standard and to import DECLARE models previously designed through the DECLARE design tool (Westergaard and Maggi 2011). A DECLARE model consists of a set of *constraints*, i.e., *rule templates* applied to activities. Their semantics can be formalized using LTL_f, 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.

In order to find the minimum cost trace alignment against a pre-specified DECLARE model, our tool makes use of the FAST-DOWNWARD (Helmert 2006) and of the SYMBA*-2 (Torralba et al. 2014) planning systems. To produce optimal alignments, FAST-DOWNWARD uses a best-first search in the first iteration to find a plan and a weighted A* search to iteratively decreasing weights of plans, while SYMBA*-2 (winner of the sequential optimizing track at the 2014 Int. Planning Competition) performs a bidirectional A* search.

We tested our approach on the grounded version of the problem presented in Section 5. We used both a *real-life* log and *synthetic* logs. We performed our experiments with a machine consisting of an Intel Core i7-4770S CPU 3.10GHz Quad Core and 4GB RAM. We used a standard cost function with unit costs for any alignment step that adds/removes activities in/from the input trace, and cost 0 for synchronous moves.

Real-life Log. The real-life log comes from real process executions and refers to an application process for personal loans in a Dutch financial institute. The original log contains 262,200 events distributed across 36 activities and includes 13,087 traces, out of which we randomly extracted 654 traces of various lengths (between 3 and 128 events) for the experiments. We employed the same DECLARE model depicted in Fig. 1 of (De Giacomo et al. 2016), which contains 16 constraints. Notice that the real-life log tested is of average complexity, as the optimal alignment for any log trace has a cost varying from 0 to 2 for short traces (less than 50 events) and from 0 to 3 for longer traces.

Synthetic Logs. To have a sense of the scalability with respect to the "size" of the model and the "noise" in the traces, we have also tested the approach on synthetic logs of different complexity. Specifically, we generated syntectic logs using the log generator presented in (Di Ciccio et al. 2015). We defined 3 DECLARE models having the same alphabet of activities and containing 10, 15, and 20 DECLARE constraints respectively. Then, to create logs containing noise, i.e., behaviors non-compliant with the original DECLARE models, we changed some of the constraints in these models and generated logs from them. In particular, we modified the original DECLARE models by replacing 3, 4, and 6 constraints in each model with their negative counterparts (see Table 1). Each modified model was used to generate 4 logs of 100 traces containing traces of different lengths, i.e., from 1 to 50 events, from 51 to 100 events, from 101 to 150 events, and from 151 to 200 events, respectively.

Results. The results of the experiments can be seen in Tables 2, 3 and in Fig. 1. They include the results obtained by testing the approaches of De Giacomo et al. (De Giacomo et al. 2016) and de Leoni et al. (de Leoni, Maggi, and van der Aalst 2012). The results show that both in the real-life and in the synthetic logs the approach of de Leoni et al. is faster for short traces with a small amount of noise. Conversely, the approach of de Giacomo et al. (which has been tested on the real-life log only) is the slowest one in all tests. Such a poor performance depends on the fact that it needs to determine, for each trace, a bound on the maximum number of instances of each activity needed to align the trace. However, such a bound is not minimal, i.e., more activity instances than those needed for the alignment are incorporated in the planning problem. This dramatically increases the search space.

When the noise increases and/or the model becomes larger, our planning-based approach outperforms the existing ones by several orders of magnitude. For example, using the synthetic log generated by the DECLARE model with 20 constraints and 3 constraints modified, containing traces of lengths varying from 151 to 200 events, our approach requires on average around 27.49 seconds (with FAST-DOWNWARD) and 28.97 seconds (with SYMBA*-2) per trace to compute an optimal alignment, while the approach of de Leoni et al. takes 223.47 seconds. This can be explained with the observation that the heuristics adopted by planners are able to efficiently cope with the size of the state space, which is exponential with respect to the size of the model, the amount of noise and the trace length.

Finally, in order to study the "boundaries" of our approach and to understand *how much noise a log needs to contain to make our approach ineffective*, we performed a third assessment by modifying 4 and 6 constraints in each of the DECLARE models. The results are shown in Table 3 and in

Trace length	Fast-Downward	SymBA*-2	de Leoni et al.	Alignment Cost	Fast-Downward	SymBA*-2	de Leoni et al.	Alignment Cost	Fast-Downward	SymBA*-2	de Leoni et al.	Alignment Cost
3 const. modified	10 constraints				15 constraints				20 constraints			
1-50	0.62	1.95	0.34	1.77	1.97	3.49	1.08	1.71	17.63	12.42	3.99	1.87
51-100	0.85	3.63	1.37	2.11	2.79	5.3	6.64	2.23	19.05	15.02	34.91	2.61
101-150	1.15	6.4	5.9	3.03	3.61	8.26	24.05	3.07	23.23	20.45	87.89	3.35
151-200	1.46	10.75	12.98	3.79	5.12	13.63	91.39	4.2	27.49	28.97	223.47	4.2
4 const. modified	10 constraints				15 constraints				20 constraints			
1-50	0.59	1.86	-	2.74	2.09	3.49	-	3.21	18.21	12.37	-	3.89
51-100	0.87	3.35	-	5.86	3.04	5.12	-	6.12	31.53	14.73	-	6.92
101-150	1.26	5.72	-	9.68	4.9	8.06	-	10.35	52.21	18.89	-	10.87
151-200	1.7	8.87	-	13.42	6.94	12.2	-	14.2	64.99	24.62	-	15.1
6 const. modified	10 constraints				15 constraints				20 constraints			
1-50	0.59	1.75	-	4.34	2.29	3.41	-	5.23	21.29	12.41	-	6.12
51-100	0.93	3.35	-	7.1	3.55	5.01	-	8.12	38.39	19.88	-	9.02
101-150	1.36	5.66	-	9.81	5.66	7.71	-	10.96	53.97	23.83	-	11.83
151-200	1.85	9.11	-	14.4	8.91	12.14	-	16.3	74.27	26.25	-	17.51

Table 3: Experimental results for the *synthetic* case study. The time (in *seconds*) is the average per trace.

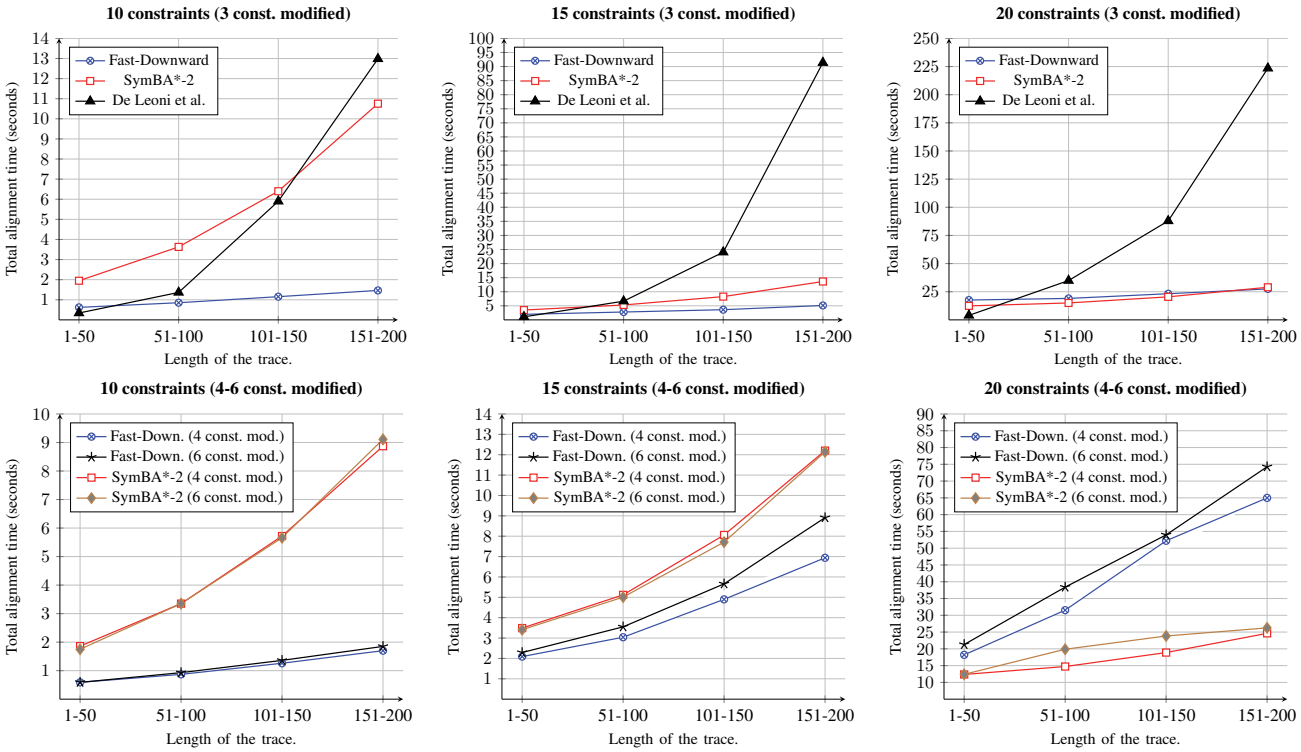

Figure 1: Performance of computing optimal alignments for the *synthetic* case study.

the lower plots of Fig. 1. They suggest that the approach is feasible also in case of traces requiring a large number of alignment actions. It is interesting to notice that the bidirectional A* search employed in SYMBA*-2 scales better than the blind A* search of FAST-DOWNWARD when the tested models contain a higher number of constraints.

7 Concluding Remarks

The scientific literature reports several works in the field of *conformance checking* (van der Aalst 2011). In (Cook and Wolf 1999; Rozinat and van der Aalst 2008), for the first time, the concept of conformance checking with respect to (procedural) process models was investigated. In (Adriansyah, van Dongen, and van der Aalst 2011), the authors introduce conformance checking augmented with the notion of trace alignment. Recently, in (Di Francescomarino et al. 2015), an approach for trace alignment against procedural process models based on automated planning has been proposed.

In recent years, an increasing number of researchers are focusing on the conformance checking with respect to declarative models. Some of these works (Chesani et al. 2009; Montali et al. 2010; Burattin et al. 2012) 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). Specific works on trace alignment for declarative specifications include (de Leoni, Maggi, and van der Aalst 2012; 2015; De Giacomo et al. 2016).

In this work, we have presented a sound and complete

technique to synthesize the trace alignment for declarative specifications relying on finite automata theoretic manipulations. The technique has been implemented using automated planning technology. Planning provides a mature and "elaboration tolerant" technology and, in fact, in the BPM literature, there exists a number of works utilizing 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; 2016).

Through an in-depth experimental study, we showed that our proposed plan-based technique significantly outperforms the state-of-the-art ad-hoc alignment approaches presented in (de Leoni, Maggi, and van der Aalst 2012; 2015; De Giacomo et al. 2016). In addition, differently from these approaches, we propose a fully general approach that is not limited to the DECLARE language but is able to cope with the whole LTL_f.

In the next future, we would like to employ the same planning-based technology for advanced types of trace alignment in which *data* and *time* are taken into account, something that is being recognized as a necessary though challenging problem in the BPs community (Montali et al. 2013; Burattin, Maggi, and Sperduti 2016; Maggi and Westergaard 2014).

Acknowledgments. This work has been partly supported by the Italian projects RoMA and NEPTIS, the Italian cluster SM&ST and the Sapienza project "Immersive Cognitive Environments".

References

Adriansyah, A.; van Dongen, B. F.; and van der Aalst, W. M. P. 2011. Conformance Checking Using Cost-Based Fitness Analysis. In *15th Int. Ent. Dist. Object Comp. Conf. (EDOC 2011)*.

Burattin, A.; Maggi, F. M.; van der Aalst, W. M. P.; and Sperduti, A. 2012. Techniques for a Posteriori Analysis of Declarative Processes. In *16th Int. Ent. Dist. Object Comp. Conf. (EDOC 2012)*.

Burattin, A.; Maggi, F. M.; and Sperduti, A. 2016. Conformance Checking Based on Multi-Perspective Declarative Process Models. *Expert Syst. Appl.* 65:194–211.

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*.

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):147–176.

De Giacomo, G., and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *23th Int. Conf. on AI (IJCAI'13)*.

De Giacomo, G., and Vardi, M. Y. 2015. Synthesis for LTL and LDL on Finite Traces. In *24th Int. Conf. on AI (IJCAI'15)*.

De Giacomo, G.; Maggi, F. M.; Marrella, A.; and Sardiña, S. 2016. Computing Trace Alignment against Declarative Process Models through Planning. In *26th Int. Conf. on Automated Planning and Scheduling (ICAPS 2016)*.

de Leoni, M.; Maggi, F. M.; and van der Aalst, W. M. P. 2012. Aligning Event Logs and Declarative Process Models for Conformance Checking. In *10th Int. Conf. on Business Process Management (BPM 2012)*.

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.* 47:258–277.

Di Ciccio, C.; Bernardi, M. L.; Cimitile, M.; and Maggi, F. M. 2015. Generating event logs through the simulation of declare models. In *11th Int. Workshop on Ent. & Org. Modeling and Simulation (EOMAS 2015)*.

Di Francescomarino, C.; Ghidini, C.; Tessaris, S.; and Sandoval, I. V. 2015. Completing Workflow Traces Using Action Languages. In *27th Int. Conf. on Adv. Inf. Syst. Eng. (CAiSE 2015)*.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.(JAIR)* 20:61–124.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.(JAIR)* 26:191–246.

Maggi, F. M., and Westergaard, M. 2014. Using Timed Automata for *a Priori* Warnings and Planning for Timed Declarative Process Models. *Int. J. Coop. Inf. Syst.* 23(1).

Marrella, A., and Lespérance, Y. 2013. Synthesizing a Library of Process Templates through Partial-Order Planning Algorithms. In *14th Int. Conf. on Business Process Modeling, Development and Support (BPMDS 2013)*.

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 2014)*.

Marrella, A.; Mecella, M.; and Sardiña, S. 2016. Intelligent Process Adaptation in the SmartPM System. *ACM Trans. Intell. Syst. Technol.* 8(2):25:1–25:43.

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

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 2013)*.

Rozinat, A., and van der Aalst, W. M. P. 2008. Conformance Checking of Processes Based on Monitoring Real Behavior. *Inf. Syst.* 33(1):64–95.

Torralba, A.; Alcazar, V.; Borrajo, D.; Kissmann, P.; and Edelkamp, S. 2014. Symba: A symbolic bidirectional a planner. In *International Planning Competition*, 105–108.

van der Aalst, W.; Pesic, M.; and Schonenberg, H. 2009. Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science - R&D* 23(2):99–113.

van der Aalst, W. M. P. 2011. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, Incorporated, 1st edition.

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