Context-Aware Trace Alignment with Automated Planning

Giacomo Acitelli*, Marco Angelini*, Silvia Bonomi*, Fabrizio M. Maggi†, Andrea Marrella*, Alessandro Palma*
*Sapienza Universitá di Roma, Rome, Italy
Email: {acitelli, angelini, bonomi, marrella, palma}@diag.uniroma1.it
†Free University of Bozen-Bolzano, Bozen, Italy
Email: maggi@inf.unibz.it

***Abstract*—Trace alignment is the problem of finding the best possible execution sequence of a business process (BP) model that reproduces an (observed) execution trace of the same BP by pinpointing where it deviates. One limiting assumption that governs the state-of-the-art alignment algorithms relies in a static cost function assigning fixed costs to all the possible types of deviations related to a BP activity, thus neglecting the specific context in which the deviation takes place and flattening the analysis of its potential impact. In this paper, we relax this assumption by providing a technique based on theoretic manipulations of deterministic finite state automata (DFAs) to build optimal alignments driven by dedicated cost models that assign context-dependent variable costs to the deviations. We show how the algorithm can be implemented relying on automated planning in Artificial Intelligence (AI), which is proven to be an effective tool to address the alignment task in the case of BP models and event logs of remarkable size. Finally, we report on the results of experiments conducted in a real-life case study on incident management and on larger synthetic ones performed through three well-known planning systems to showcase the performance, scalability and versatility of our technique.**

I. Introduction

Conformance checking is an active area of research in the field of process mining that aims at verifying whether the observed behavior of a business process (BP), which is stored in a log, matches the intended behavior represented as a BP model [1]. *Trace alignment* is considered to be the most effective family of algorithms for this purpose [2], since it finds the best execution sequence of a BP model (i.e., the *optimal alignment*) that reproduces an execution trace of the same BP by pinpointing where it deviates. Computing alignments is often the starting point to enhance a BP model with the evidences found in the data, e.g., to distinguish the BP frequent behavior from the exceptional one [3].

Many existing works illustrate that trace alignment has a strong practical relevance as a diagnostic tool in several application domains, such as healthcare [4] and security auditing [5], just to name a few. However, as recently observed by Boltenhagen et al. in [6], one limiting assumption that governs the state-of-the-art trace alignment algorithms relies in a static cost function assigning fixed costs to all the possible types of deviations related to a BP activity, thus neglecting the *context*, i.e., the specific position in the trace, in which the deviations take place. For instance, in a loan process, while it is possible to specify that approving a loan when it is not supposed to be approved is more severe than asking for additional information when unnecessary, it is not possible to quantify that approving a loan before assessing the applicant is more problematic than approving a loan before asking for additional information. In both cases, activity "Approve Loan" is executed when not allowed, but in the first case the deviation is more severe and it might be useful to associate it with a higher deviation cost for better interpreting the impact of the misalignment.

In this paper, we tackle this issue by providing a trace alignment technique to synthesize optimal alignments driven by dedicated *cost models* associating context-dependent variable costs to the potential deviations that may occur during the alignment task. For the formal representation of a BP model and its associated cost models, we opted for deterministic finite state automata (DFAs), which have a clear semantics and can be employed to perform formal reasoning over the original BP model. In addition, DFAs are not directly tied to the prescriptive or declarative nature of the BP model to analyze, allowing us to capture both modeling perspectives. Note that the process mining literature is plenty of solutions to convert a BP model represented as a Petri net or as a collection of Declare constraints in the form of a DFA, cf. [7], [8].

We implement our alignment technique in two steps. First, we provide an algorithm, based on DFA-theoretic manipulations, to synthesize the alignment instructions and customize the severity of the deviations based on the context in which BP activities are executed. Then, inspired by our previous works [9], [10], we show how the algorithm can be implemented relying on automated planning technology in Artificial Intelligence (AI), which was proven to be very effective to tackle the alignment problem in case of BP models and event logs of remarkable size [11]. Specifically, we resort to cost-optimal planning [12], 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 planning actions capture alignment steps in the form of synchronous moves (having no cost) and deviations (additions and deletions) to the input trace, with non-zero costs. The goal is to make the input trace conforming with the BP model at a minimal cost.

Pre-print copy of the manuscript published by IEEE (available at: https://ieeexplore.ieee.org/document/9980649) and identified by doi: 10.1109/ICPM57379.2022.9980649

We evaluate the performance and scalability of our technique through an extensive experimentation performed over a real incident management use case and synthetically generated event logs and DFAs of increasing complexity. Notably, three well-known planning systems are employed in the experiments to showcase the versatility of our technique, which allows the process analyst to plug in new planners at no cost.

The paper is organized as follows. In Section II, we overview the state-of-the-art algorithms to compute alignments. In Section III, we discuss the background necessary to understand the paper. In Section IV, we introduce a use case on incident management that is used to explain the advantages of context-aware trace alignment. Section V presents our alignment technique and shows how to reduce it to a planning problem. Finally, in Section VI we evaluate our technique and, in Section VII, we conclude the paper by outlining future work.

II. Related Work

Alignment-based conformance checking is primarily used as an artifact to pinpoint and document deviations. The seminal work of Adriansyah [13] provides an ad-hoc implementation of the A* algorithm to compute optimal alignments for a particular class of BP models. It points out that the worst-case complexity for computing optimal alignments is exponential wrt. the amount of behavior allowed by BP models and the length of the log traces. Thus, even improving the computation, there will always be a BP model for which the computation of alignments for certain traces becomes intractable.

The computation of alignments can be fastened by using divide-and-conquer approaches. In [14], [15], a decomposition technique is proposed that guarantees the model to be decomposed in fragments such that, if sub-traces fit the individual fragments, then they can be composed into a larger trace that fits the overall BP model. Similarly, in [16], a Single-Entry Single-Exit decomposition technique is presented that partitions large BP models and event logs into smaller parts that can be analyzed independently. In [17], the computation of alignments is performed through a set of concurrency-free automata, called S-components, representing the behavior of the BP model. Each S-component is aligned separately against a projected version of the log, leading to one product automaton per S-component. The resulting product automata are then recomposed into a single automaton capturing all the differences between the BP model and the log, but without minimality guarantees. In [18], the notion of approximate alignment is introduced through a recursive paradigm based on the structural theory of Petri nets. In [19], a local search framework is applied to improve the alignment until no further improvement is possible.

The aforementioned works focus on alleviating the computational demand of trace alignment by using a standard (i.e., context-agnostic) cost function that assigns fixed costs to deviations. Conversely, a couple of recent works [6], [20] rely on a variable cost function to improve the performance of trace alignment. In [6], a prefix-first alignment heuristic based on a discounted cost function that penalizes deviations appearing at early stage of the BP execution is proposed. Although this approach does not guarantee the optimality of the alignments, this is compensated by a reduced computation time of the alignments. In [20], an approach that maximizes the number of synchronous moves in the alignment is proposed. It relies on milestone (i.e., unskippable) activities and on a cost function that dynamically changes to penalize log moves.

We notice that both [6] and [20] do not build their cost function based on the context in which the BP execution takes place. Conversely, in this paper, we present a planning-based alignment technique that relies on context-aware cost models for the construction of optimal alignments, where the cost of a deviation dynamically changes based on where the deviation occurs within the trace.

III. Background

Given a BP model represented as a DFA and a collection of log traces, the problem we want to address is to build an optimal alignment of each trace wrt. the model using a cost function that quantifies the severity of a deviation based on the context in which the deviation takes place. The context is determined by the position in the trace in which the deviation occurs and, more specifically, by the activities or sequence of activities that occur before or after the deviation. The notion of "context" is expressed through a dedicated *cost model*, which will be introduced in Section V, together with a reformulation of the trace alignment problem in the form of a DFA-theoretic manipulation. The alignment will be built by relying on automated planning technology.

A. Log Traces and Deterministic Finite State Automata

Event logs are the starting point for performing trace alignment. An event log is a multiset of *log traces* describing the life-cycle of a BP instance in terms of the activities executed.

Definition 1 (Event Log and Traces). *Let $\mathcal{A}$ be a set of activities. We define a log L as a finite multiset of sequences $t \in \mathcal{A}^*$, which we refer to as log traces.*

A *Deterministic Finite State Automaton* (DFA) is a finite-state machine that accepts or rejects a given log trace, by running it through a state sequence uniquely determined by the sequence of activities included in the trace. Deterministic refers to the uniqueness of the computation run.

Definition 2 (Deterministic Finite State Automaton). *A DFA is a tuple $\mathcal{N} = \langle \mathcal{A}, Q, q_0, \delta, F \rangle$, where:* (i) *$\mathcal{A}$ is the* input alphabet*;* (ii) *Q is the (non-empty) finite set of* automaton states*;* (iii) *$q_0 \in Q$ is the* initial state*;* (iv) *$\delta \subseteq Q \times \mathcal{A} \times Q$ is the* transition relation*; and* (v) *$F \subseteq Q$ is the set of* final states*.*

Definition 3 (Computation of a trace on a DFA). *Let $t = e_1 \cdots e_n$ be a log trace such that $e_i \in \mathcal{A}$ (with $1 \leq i \leq n$) and $\mathcal{N}$ the DFA representing a BP model. A* computation *of $\mathcal{N}$ on t is a sequence*

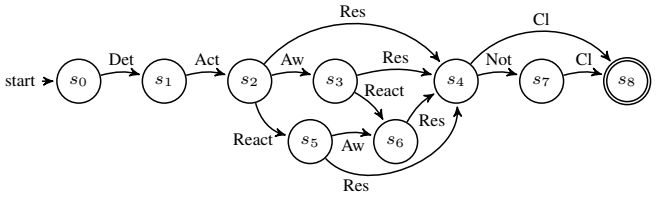

Fig. 1: The DFA representing the IM process

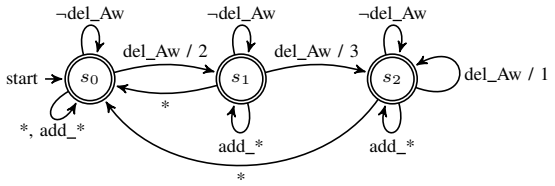

Fig. 2: The cost model related to the IM process, with the character '*' meaning any activity in the input alphabet $\mathcal{A}$

$\rho = q_0 \xrightarrow{e_1} q_1 \cdots q_{n-1} \xrightarrow{e_n} q_n$ *such that, for* $i = 0, \ldots, n-1$, *there exists a transition* $\langle q_i, e_{i+1}, q_{i+1} \rangle \in \delta$, *denoted as* $q_i \xrightarrow{e_{i+1}} q_{i+1}$.

Since δ is a function, it behaves deterministically. That is, at any step of a computation, the DFA has only one "choice" for its next transition, depending on its current state and the input activity. Note that we consider DFAs that are *complete*, i.e., they define from each state a transition for each activity in $\mathcal{A}$. Finally, we say that $\mathcal{N}$ *accepts* t ($t \vDash \mathcal{N}$) if there exists a computation ρ on t whose last state is *final*.

B. Automated Planning

Planning systems are problem-solving algorithms that operate on explicit representations of states and actions [12]. PDDL [21] is the standard Planning Domain Definition Language; it allows us to formulate a *planning problem* with the description of the initial state of the world, the desired goal state, and the planning domain. A planning domain is built from a set of *propositions* describing the state of the world (i.e., the set of propositions that are true) and a set of *operators* Ω (i.e., *actions*) that can be executed. An *action schema* $a \in \Omega$ defines the list of *input parameters* for a, the *preconditions* under which a can be executed, and the *effects* of a on the state of the world. Both preconditions and effects are stated in terms of *propositions* in the planning domain, represented as boolean predicates and numeric fluents.

In recent years, the planning community has developed a plethora of planners that embed very effective (i.e., scaling up to large problems) search heuristics, which have been employed to solve collections of challenging problems from several Computer Science domains [11], [22], [23]. There exist several forms of planning in the AI literature. In this paper, we focus on *cost-optimal classical planning* techniques characterized by *fully observable* and *static* domains. A solution for a cost-optimal planning problem is a sequence of operators—a *plan*—whose execution transforms the initial state into a state satisfying the goal by optimizing a pre-specified metric.

IV. A Use Case on Incident Management

According to the ISO/IEC 27035 standard, a security incident is an unwanted or unexpected set of events that have a significant probability of compromising business operations and threatening security. In this context, Incident Management (IM) is the process of detecting, reporting, assessing, responding to, and dealing with security incidents [24].

In Fig. 1, it is shown a DFA representation of the BP model underlying ISO/IEC 27035. The BP starts when a new incident is detected by the IT team (activity Det) and reported to the security team that is in charge to proceed with the incident registration (e.g., the opening of a ticket, activity Act). Then, many different paths are possible. In the best case, the incident is quickly resolved by the security team (activity Res). However, before resolution, it could be necessary to wait for a second assessment made by third party security companies (activity Aw) and/or to reactivate the ticket associated to the incident (activity React). Once resolved, a resolution notification is sent to the IT team (activity Not) and the ticket is closed (activity Cl). For the sake of readability, we are reporting in the DFA of Fig. 1 only the "prescriptive" view with the expected paths of the IM process, i.e., we are neglecting the transitions with non-allowed activities that would be connected to a sink non-accepting state of the DFA.

Nowadays, to assess that the IM process is enacted by organizations in compliance with ISO/IEC 27035, a manual analysis is performed by expert auditors that detect misalignments and quantify their degree of severity to support the delivery of focused decisions. In principle, trace alignment techniques would be a suitable tool to increase the automation and quality of the incident analysis task [25]. However, since state-of-the-art alignment techniques rely on a static cost function that assigns fixed costs to the deviations (independently of the context in which they arise), the risk is to provide unreliable diagnostics for non-conforming behaviors.

For example, consider the following execution traces of the IM: $t_1 = \langle \text{Act}, \text{Aw}, \text{Aw}, \text{Aw}, \text{Res}, \text{Cl} \rangle$ and $t_2 = \langle \text{Act}, \text{Act}, \text{Act}, \text{Aw}, \text{Res}, \text{Cl} \rangle$. Both traces are misaligned with the DFA in Fig. 1. Specifically, for both traces Det is missing before the activation of the incident. Then, for t_1 and t_2, Aw and Act are unnecessarily repeated three times, respectively. By employing a static cost function that assigns a unitary cost to any deviation, the alignment of both traces would have a cost equal to 3, meaning that the two situations are considered equally problematic. However, this is far to be a realistic analysis. Indeed, the recording of many repeated Aw means that the execution of the BP has been stuck for a quite long time. On the other hand, even if many repeated Act are observed, it is guaranteed that the security team continued to work for resolving the security incident. Therefore, the first misalignment represents a more severe situation than the

second one. In addition, the cost of repeating Aw several times should increase at any occurrence, since it critically delays the incident resolution. In this paper, we will employ dedicated cost models in the form of DFAs, like the one in Fig. 2, to capture the variability of the cost of deviations based on the context in which they are captured. This will allow us, for example, to assign a different cost to the execution of Aw depending on whether it occurs after that Aw has already been executed once, twice, or three times.

V. Context-Aware Trace Alignment as Planning

In this section, after providing a formulation of the context-aware trace alignment problem (Section V-A), we discuss how it can be solved with a technique based on DFA manipulations (Section V-B). Finally, we show how this technique can be reduced to a planning problem in AI (Section V-C).

A. Context-Aware Trace Alignment

Consider a log trace $t = \langle e_1, ..., e_n \rangle$ over $\mathcal{A}$ and a DFA $\mathcal{N}$ representing a BP model such that $t \notin \mathcal{N}$. We are interested in "transforming" t into a new trace $\hat{t}$ such that $\hat{t} \vDash \mathcal{N}$, i.e., t is aligned wrt. $\mathcal{N}$. We assume that traces can be aligned by executing the following operations: *skip* an activity, i.e., leave the activity unchanged; *delete* an activity from a position; *add* a new activity at a certain position. In particular, we extend the set of activities $\mathcal{A}$ with two new activities del_e and add_e for every activity $e \in \mathcal{A}$. We call them *repair activities*, denote the obtained set as $\mathcal{A}_+$, and call the traces over $\mathcal{A}_+$ *repair traces*. In a repair trace, $e \in \mathcal{A}$ stands for the synching e, del_e for the deletion of e, and add_e for the addition of e.

A repair trace t^+ represents a set of modifications that transform, when successfully executed, a trace t into a new trace $\hat{t}$. We say that a repair trace t^+ is *applicable* to a log trace t, if t^+ can be obtained from t by: *(i)* inserting any arbitrarily long sequence of activities add_* (where $*$ matches any activity from $\mathcal{A}$) before the first activity, after the last activity, or between any two consecutive activities of t; *(ii)* replacing any activity e of t by either the activity itself or by del_e. For instance, $t^+ = \langle a, del\_b, add\_c, c, add\_a \rangle$ is applicable to $t_1 = \langle a, b, c \rangle$ but not to $t_2 = \langle a, b, b \rangle$, because t^+ prescribes the synching of c, which is not present in t_2.

We formally capture the result of performing the operations of a repair trace by defining the trace *induced by* t^+, i.e., the trace $\hat{t}$ over $\mathcal{A}$ obtained from t^+ by: *(i)* deleting every occurrence of del_e; and *(ii)* replacing every occurrence of add_e with activity e. For instance, in the case of $t_1 = \langle a, b, c \rangle$, the trace induced by the repair trace $t^+ = \langle a, del\_b, add\_c, c, add\_a \rangle$ is $\hat{t} = \langle a, c, c, a \rangle$. When t^+ is applicable to a trace t and induces $\hat{t}$, we say that t^+ *transforms* t into $\hat{t}$.

Finally, we define the *cost of a repair trace* t^+, denoted with $cost(t^+)$, as the sum of the cost of modifications (expressed as add and del activities) occurring in t^+. We observe that, in the case of a static cost function, it is sufficient to assign (non-negative) fixed costs to additions or deletions of a specific activity. Rather than implementing a static cost function, in this paper, we introduce a *cost model* $\mathcal{C}$ (as a DFA) to associate repair activities with *variable costs* that depend on the context in which they occur during the construction of t^+, i.e., on the activities that occur before or after them. Consequently, $cost(t^+)$ will be a function of $\mathcal{C}$.

Definition 4 (Context-Aware Trace Alignment)**.** *Let t be a log trace, $\mathcal{N}$ the DFA representing a BP model and $\mathcal{C}$ the DFA representing a cost model. Context-Aware Trace Alignment is the problem of finding a repair trace t^+ that transforms t into $\hat{t}$, such that $\hat{t} \vDash \mathcal{N}$, by minimizing $cost(t^+)$.*

B. A DFA Manipulation Technique for Trace Alignment

In this section, we present a DFA-based formalization of the context-aware trace alignment problem. Let $t = e_1 \cdots e_n$ be a log trace over $\mathcal{A}$, and $\mathcal{N} = \langle \mathcal{A}, Q, q_0, \delta, F \rangle$ a DFA representing a BP model.

We start by defining the so-called *trace automaton of* t, i.e., the automaton $\mathcal{T} = \langle \mathcal{A}_t, Q_t, q_0^t, \delta_t, F_t \rangle$, where:

- $\mathcal{A}_t \subseteq \mathcal{A}$;
- $Q_t = \{ q_0^t, \ldots, q_n^t \}$ is a set of $n+1$ states;
- q_0^t is the initial state;
- $\delta_t = \bigcup_{i=0,\ldots,n-1} \langle q_i^t, e_{i+1}, q_{i+1}^t \rangle$;
- $F_t = \{ q_n^t \}$.

We denote the set of traces accepted by $\mathcal{T}$, i.e., the *language* of $\mathcal{T}$, as $\mathcal{L}_\mathcal{T}$. An example of trace automaton, for $t = \langle c, a, b \rangle$, is illustrated in Fig. 3.

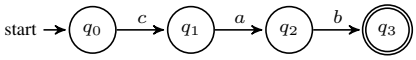

Fig. 3: Trace automaton associated with trace $t = \langle c, a, b \rangle$

Then, we augment $\mathcal{T}$ and $\mathcal{N}$ as follows. From $\mathcal{T}$, we define automaton $\mathcal{T}^+ = \langle \mathcal{A}_+, Q_t, q_0^t, \delta_t^+, F_t \rangle$, where δ_t^+ contains all transitions in δ_t, plus:

- a transition $\langle q, del\_p, q' \rangle$, for all transitions $\langle q, p, q' \rangle \in \delta_t$;
- a loop transition $\langle q, add\_*, q \rangle$, for each $q \in Q_t$.

We call $\mathcal{T}^+$ the *repair automaton of* t. Notice that $\mathcal{T}^+$ accepts repair traces. We denote the set of repair traces accepted by $\mathcal{T}^+$ as $\mathcal{L}_{\mathcal{T}^+}$. The repair automaton of trace $t = \langle c, a, b \rangle$ over $\mathcal{A} = \{ a, b, c, d \}$ is shown in Fig. 4.

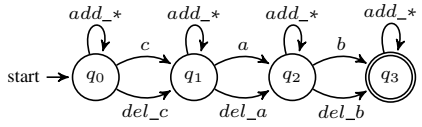

Fig. 4: Repair automaton of trace $t = \langle a, b, c \rangle$

Then, from $\mathcal{N} = \langle \mathcal{A}, Q, q_0, \delta, F \rangle$, we derive the *augmented automaton of* $\mathcal{N}$, i.e., $\mathcal{N}^+ = \langle \mathcal{A}_+, Q, q_0, \delta^+, F \rangle$, where δ^+ contains all transitions in δ, plus:

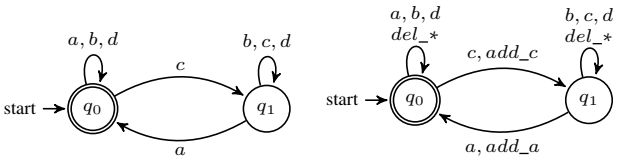

Fig. 5: A DFA over $\mathcal{A} = \{a, b, c, d\}$ and its augmented version

- a transition $\langle q, add\_e, q' \rangle$, for each $\langle q, e, q' \rangle \in \delta$;
- a loop transition $\langle q, del\_*, q \rangle$, for each $q \in Q$.

Notice that the above definition implies that $\mathcal{L}_{\mathcal{N}} \subseteq \mathcal{L}_{\mathcal{N}^+}$, where $\mathcal{L}_{\mathcal{N}}$ and $\mathcal{L}_{\mathcal{N}^+}$ denote the languages of $\mathcal{N}$ and $\mathcal{N}^+$.

Fig. 5 shows a sample DFA $\mathcal{N}$ defined over $\mathcal{A} = \{a, b, c, d\}$ and its augmented version $\mathcal{N}^+$. Intuitively, $\mathcal{N}^+$ accepts all the repair traces t^+ (including those not applicable to t) that induce a trace t^+ accepted by $\mathcal{N}$.

Finally, we introduce a cost model $\mathcal{C}$ in the form of a DFA $\mathcal{C} = \langle \mathcal{A}_+, Q_{\mathcal{C}}, q_0^{\mathcal{C}}, \delta_{\mathcal{C}}, F_{\mathcal{C}} \rangle$ that is used to capture the costs of deviations in certain points of the execution of $\mathcal{N}^+$. $\mathcal{C}$ is a traditional DFA accepting the same input alphabet of $\mathcal{T}^+$ and $\mathcal{N}^+$. The unique characteristics of $\mathcal{C}$ are that: *(i)* all its states are final. This is due to the fact that the paths allowed by $\mathcal{C}$ do not affect the validity of an alignment, but are just used to keep track of the cost of repair activities that will be included in a repair trace; and *(ii)* each repair activity used for labeling a transition is associated with a non-negative cost, i.e., transitions in $\delta^{\mathcal{C}}$ have the form: $\langle q, e/cost(e), q' \rangle$, with $q, q' \in Q_{\mathcal{C}}$. We denote with $\mathcal{L}_{\mathcal{C}}$ the language accepted by $\mathcal{C}$.

In Fig. 2, we have already shown an example of a cost model for the IM use case. We note that it assigns a cost equal to 2 when a first unnecessary occurrence of Aw is captured, a cost equal to 3 for the second occurrence of Aw and a cost equal to 1 for any other further occurrence of Aw in the trace. When repair activities in the cost model are not explicitly related to any cost, we assume that they are associated with the fixed cost captured by the static cost function. In a nutshell, in the presence of a static cost function, our cost model overrides the cost to be assigned to certain repair activities in $\mathcal{C}$.

Based on the above formulation, we can conclude that, given t, $\mathcal{N}$ and $\mathcal{C}$, the *context-aware trace alignment* of t wrt. $\mathcal{N}$ is equivalent to searching for a repair trace t^+ (over $\mathcal{A}_+$) that is accepted by both $\mathcal{N}^+$ and $\mathcal{T}^+$ and has a cost determined by the occurrence of the repair activities in $\mathcal{C}$. Indeed, the acceptance by $\mathcal{T}^+$ and $\mathcal{N}^+$ guarantees that we are considering all (and only) the repair traces t^+ that are *(i)* applicable to t, and *(ii)* induce a log trace $\hat{t}$ satisfying $\mathcal{N}$. t^+ is said to be an *optimal alignment* if $cost(t^+)$ is minimal. In other words, the search space of our problem is the language $\mathcal{L}_{\mathcal{T}^+} \cap \mathcal{L}_{\mathcal{N}^+} \cap \mathcal{L}_{\mathcal{C}}$. In the next section, we show how the search can be actually performed by resorting to planning technology.

C. Context-Aware Trace Alignment as Planning

In this section, based on the formalization presented above, we show how we can reduce trace alignment to a deterministic cost-optimal planning problem.

The idea behind the reduction is to model activities from $\mathcal{A}_+$ as planning actions whose execution triggers state changes in both $\mathcal{T}^+$, $\mathcal{N}^+$, and $\mathcal{C}$, according to their respective transition function. The current state of $\mathcal{T}^+$, $\mathcal{N}^+$ and $\mathcal{C}$ is modeled through suitable fluents. Planning actions are executable based on the current state of the automata: an action a can be executed only if the current states of $\mathcal{T}^+$, $\mathcal{N}^+$ and $\mathcal{C}$ have an outgoing transition labelled with a. The problem then consists in finding a deterministic plan that takes such DFAs from their initial state to a final state at a minimum cost, where actions corresponding to add and del are assigned a cost depending on the occurrence of their respective repair activities in $\mathcal{C}$, while skip actions are assigned a 0 cost. The obtained plan is a representation of the repair trace that solves the problem, i.e., a trace that says how each activity from the input trace must be modified (or left unchanged), in order to satisfy $\mathcal{N}$ with minimum cost. We provide the details of this reduction below.

A *deterministic planning domain with action costs* over a set of propositional fluents $\mathcal{F}$ is a tuple $\mathcal{D} = \langle S, \Omega, cost, \tau \rangle$, where:

- $S \subseteq 2^{\mathcal{F}}$ is the finite set of *domain states*, with each state $s = \{ f_1, \ldots, f_n \}$ representing the propositional interpretation that assigns $true$ to a fluent f if and only if $f \in s$;
- Ω is the finite set of *domain actions*;
- $cost : \Omega \mapsto \mathbb{N}_0$ is a *cost function*, assigning a non-negative cost to each planning action;
- $\tau : S \times \Omega \mapsto S$ is the domain *transition function*.

A *plan for* $\mathcal{D}$ is a finite sequence $\pi = a_1 \cdots a_n \in \Omega^*$ of actions, which is said to be *executable* in $\mathcal{D}$ from a state $s_0 \in S$, if there exists a sequence of states $\sigma = s_0 \cdots s_n$ such that, for $i = 0, \ldots, n-1$, it is the case that $s_{i+1} = \tau(s_i, a_{i+1})$; if σ exists, it is said to be the *(domain) trace induced by* π. Notice that, since $\mathcal{D}$ is deterministic, every plan π induces a unique domain trace. The *cost* of a plan π is defined as $cost(\pi) \doteq \sum_{i=1,\ldots,n} cost(a_i)$.

A *cost-optimal planning problem* is a tuple $\mathcal{P} = \langle \mathcal{D}, s_0, G \rangle$, where:

- $\mathcal{D}$ is a planning domain with action costs;
- $s_0 \in \mathcal{S}$ is the *initial state* of the problem;
- G, the *problem goal*, is a propositional formula over $\mathcal{F}$.

A plan π is a *solution* of $\mathcal{P}$ if the last state s_n of the trace induced by π is such that $s_n \models G$. A solution π of $\mathcal{P}$ is said to be *optimal* if, for all other solutions π', it is the case that $cost(\pi) \leq cost(\pi')$.

We encode the context-aware trace alignment problem into cost-optimal planning as follows. Consider an instance of the trace alignment problem, i.e., a trace t and a DFA $\mathcal{N}$, and let $\mathcal{T}^+ = \langle \mathcal{A}_+, Q_t, q_0^t, \delta_t^+, F_t \rangle$, $\mathcal{N}^+ = \langle \mathcal{A}_+, Q, q_0, \delta^+, F \rangle$ and $\mathcal{C} = \langle \mathcal{A}_+, Q_\mathcal{C}, q_0^\mathcal{C}, \delta_\mathcal{C}, F_\mathcal{C} \rangle$ be the repair, the augmented and the cost automaton. We start by defining the planning domain $\mathcal{D} = \langle S, \Omega, cost, \tau \rangle$ over $\mathcal{F} = Q_t \cup Q \cup Q_\mathcal{C}$ (automata states are used as fluents; Q_t, Q and $Q_\mathcal{C}$ are disjoint), where:

1) $S \subseteq \{ \{ q_t \} \cup \{ q \} \cup \{ q_\mathcal{C} \} \mid q_t \in Q_t, q \in Q, q_\mathcal{C} \in Q_\mathcal{C} \}$;
2) $\Omega = \mathcal{A}_+$, i.e., we use activities from $\mathcal{A}_+$ as planning actions, in particular, an activity $e \in \mathcal{A}$ models a skip action on e, whereas activity add_e and del_e model the addition/deletion of activity e;
3) For all $e \in \mathcal{A}$, $cost(e) = 0$. Then, $cost(add\_e)$ and $cost(del\_e)$ depend on the specific combination of transitions in $\mathcal{N}^+$, $\mathcal{T}^+$ and C, labeled with add_e or del_e, which is encoded in the i-th planning action;
4) for $a \in \Omega$, $q_t, q_t' \in Q_t$, $q, q' \in Q$, $q_\mathcal{C}, q_\mathcal{C}' \in Q_\mathcal{C}$, $\tau(\{ q_t \} \cup \{ q \} \cup \{ q_\mathcal{C} \}, a) = \{ q_t' \} \cup \{ q' \} \cup \{ q_\mathcal{C}' \}$ if and only if:
   a) $\langle q_t, a, q_t' \rangle \in \delta_t^+$;
   b) $\langle q, a, q' \rangle \in \delta^+$;
   c) $\langle q_\mathcal{C}, a/cost(a), q_\mathcal{C}' \rangle \in \delta_\mathcal{C}$.

We call $\mathcal{D}$ the *context-aware trace alignment planning domain* of t wrt. $\mathcal{N}$ and $\mathcal{C}$. $\mathcal{D}$ models the synchronous product of $\mathcal{T}^+$, $\mathcal{N}^+$ and $\mathcal{C}$, where fluents represent the states in which each DFA is. Since we are working with DFAs, each state of the planning domain contains exactly one state from Q_t, Q and $Q_\mathcal{C}$ and one (unique) successor state wrt. action a and the current states of $\mathcal{T}^+$, $\mathcal{N}^+$ and $\mathcal{C}$. Requirements 4a, 4b and 4c capture executability: action a can be executed only if the corresponding activity is accepted by $\mathcal{T}^+$, $\mathcal{N}^+$ and $\mathcal{C}$ in their current state. Notice that the so-defined transition function τ of the planning domain $\mathcal{D}$ is deterministic. In addition, since $\Omega = \mathcal{A}_+$, every plan for $\mathcal{D}$ is also a repair trace.

Now, we define the cost-optimal planning problem $\mathcal{P} = \langle \mathcal{D}, s_0, G \rangle$, where:

- $s_0 = \{ q_0, q_0^t, q_0^\mathcal{C} \}$;
- $G = (\bigvee_{q \in F} q) \wedge q_f^t$, with q_f^t the (unique) final state of $\mathcal{T}^+$ (remind that all the states of $\mathcal{C}$ are final by default).

We call $\mathcal{P}$ the *context-aware trace alignment planning problem* of t wrt. $\mathcal{N}$ and $\mathcal{C}$. A solution of this problem is a minimal-cost plan inducing a domain trace that ends in a state satisfying G. As said above, plans are, in fact, repair traces.

Thus, to actually solve the trace alignment problem, we can resort to automated planning. As discussed before, this can be done by searching for a repair trace that is accepted by the augmented automaton and the DFA representing the cost model. In the planning setting, this amounts to adding new fluents in the domain states, each modeling a state of the augmented automaton and the cost model, and then defining the transition function in such a way that a state transition in the domain accounts for all the state transitions of each DFA. The goal is to reach a state where all DFAs are in a final state.

VI. Experiments

We have developed a planning-based alignment tool as a standard Java application[1] that implements the technique discussed in Sections V. The tool can be run interactively using a GUI interface, and allows the process analyst to load existing logs formatted with the XES (eXtensible Event Stream) standard and to import BP models defined as DFAs in DOT (graph description language) format. In order to find cost-minimal alignments, our tool makes use of three well-known planning systems, namely, FAST-DOWNWARD [26], SYMBA*-2 [27] and COMPLEMENTARY1 [28]. 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 the plan weights, while SYMBA*-2, winner of the sequential optimizing track at the 2014 Int. Planning Competition (IPC) performs a bidirectional A* search. Finally, COMPLEMENTARY1 is a recent cost-optimal planner using a heuristic function driven by a lookup table that estimates the distance from the achievement of an optimal plan. We tested our approach on the grounded version of the problem presented in Section V-C. Specifically, to find a plan, we represented the planning problems by making use of the STRIPS fragment of PDDL 2.1 [21] enhanced with the features provided by the same language for keeping track of the costs of planning actions and synthesizing plans yielding minimal overall cost. We used both a *real-life* log and *synthetic* logs. We performed our experiments with an Intel Core i7-4770S CPU 3.10GHz Quad Core and 8GB RAM.

Real-life Log. The real-life log refers to an incident management process and was extracted from a dataset available in the UCI Machine Learning Repository, cf. http://archive.ics.uci.edu/ml/index.php. The log contains 141,712 events organized in 24,918 traces with various lengths. We ran our technique using the DFA in Fig. 1 and as cost model the DFA in Fig. 2. We employed the FAST-DOWNWARD planner to build the optimal alignments. The results (that do not include duplicate traces) are shown in Table III, and show that the performance of the planner when computing the alignment of one trace is near real-time, but decreases when the trace size increases. Notice that all the optimal alignments include many deviations.

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 with synthetic logs of different complexity. Specifically, to generate synthetic logs, we exploit the well-known equivalence between (regular) languages and DFAs, which states that any LTL_f formula ϕ can be associated with a DFA $\mathcal{N}$ that accepts exactly all traces satisfying ϕ [8].

[1]The tool is available for testing and experiments repeatability at https://github.com/bpm-diag/PL_DEC_ALIGNER.

Trace length	SymbA-2* Preprocessing	SymbA-2* Searching	SymbA-2* Steps	Complementary1 Preprocessing	Complementary1 Searching	Complementary1 Steps	Fast-Downward Preprocessing	Fast-Downward Searching	Fast-Downward Steps	Context-Aware Alignment Cost	Alignment Cost
3 form. modified					***2242 states & 56026 transitions***						
1-50	0.16	0.59	46	2.11	$1 \cdot 10^{-3}$	46	0.33	0.48	46	2.1	1.7
51-100	0.17	1.33	81	3.1	$3 \cdot 10^{-3}$	81	0.47	33.48	81	2.2	2.2
101-150	0.21	3.45	128	5.11	$5 \cdot 10^{-3}$	128	0.71	15.75	127	3.1	3
151-200	0.29	8.43	170	8.68	$7 \cdot 10^{-3}$	170	0.82	63.6	164	4.2	4.2
4 form. modified					***2242 states & 56026 transitions***						
1-50	0.15	0.49	47	2.33	$2 \cdot 10^{-3}$	46	0.32	12.6	47	4.2	3.2
51-100	0.17	1.37	83	3.28	$3 \cdot 10^{-3}$	83	0.49	83.06	83	6.1	6.1
101-150	0.2	2.82	125	5.25	$6 \cdot 10^{-3}$	125	0.68	373.28	119	10.3	10.3
151-200	0.25	5.92	167	7.54	$7 \cdot 10^{-3}$	166	0.76	537.72	138	14.2	14.2
6 form. modified					***2242 states & 56026 transitions***						
1-50	0.14	0.55	53	2.84	$2 \cdot 10^{-3}$	54	0.35	29.54	52	6.4	5.2
51-100	0.14	0.85	69	3.41	$3 \cdot 10^{-3}$	69	0.44	173.28	64	9.4	8.1
101-150	0.18	2.31	123	6.1	$7 \cdot 10^{-3}$	124	0.63	647.64	120	15.3	10.9
151-200	0.28	5.7	181	8.97	$8 \cdot 10^{-3}$	182	-	-	-	20	16.3

TABLE I: Experimental results for the *synthetic* logs. The time (in *seconds*) is the average per trace

Trace length	SymbA-2* Preprocessing	SymbA-2* Searching	SymbA-2* Steps	Complementary1 Preprocessing	Complementary1 Searching	Complementary1 Steps	Fast-Downward Preprocessing	Fast-Downward Searching	Fast-Downward Steps	Context-Aware Alignment Cost	Alignment Cost
3 form. modified					***29182 states & 729526 transitions***						
1-50	0.24	0.92	51	3.14	$2 \cdot 10^{-3}$	50	1.62	0.14	50	1.8	1.8
51-100	0.24	2.44	84	4.79	$4 \cdot 10^{-3}$	84	1.4	35.48	87	3.1	2.6
101-150	0.28	5.36	127	7.46	$6 \cdot 10^{-3}$	127	2.13	146.76	130	3.7	3.3
151-200	0.37	14.23	183	14.76	$9 \cdot 10^{-3}$	182	-	-	-	4.2	4.2
4 form. modified					***29182 states & 729526 transitions***						
1-50	0.3	1.36	48	3.93	$2 \cdot 10^{-3}$	48	1.61	30.9	51	4.8	3.8
51-100	0.24	2.09	77	4.52	$3 \cdot 10^{-3}$	77	2.05	140.86	89	6.9	6.9
101-150	0.3	6.41	132	10.13	$7 \cdot 10^{-3}$	132	-	-	-	10.8	10.8
151-200	0.36	12.45	177	16.12	$9 \cdot 10^{-3}$	177	-	-	-	15.9	15.1
6 form. modified					***29182 states & 729526 transitions***						
1-50	0.22	1.25	50	3.72	$2 \cdot 10^{-3}$	50	-	-	-	7.2	6.1
51-100	0.23	2.57	78	5.76	$4 \cdot 10^{-3}$	79	-	-	-	11.2	9
101-150	0.33	7.63	150	10.2	$8 \cdot 10^{-3}$	151	-	-	-	15.6	11.8
151-200	0.35	12.09	184	17.07	$1.1 \cdot 10^{-2}$	185	-	-	-	22.6	17.5

TABLE II: Experimental results for the *synthetic* logs. The time (in *seconds*) is the average per trace

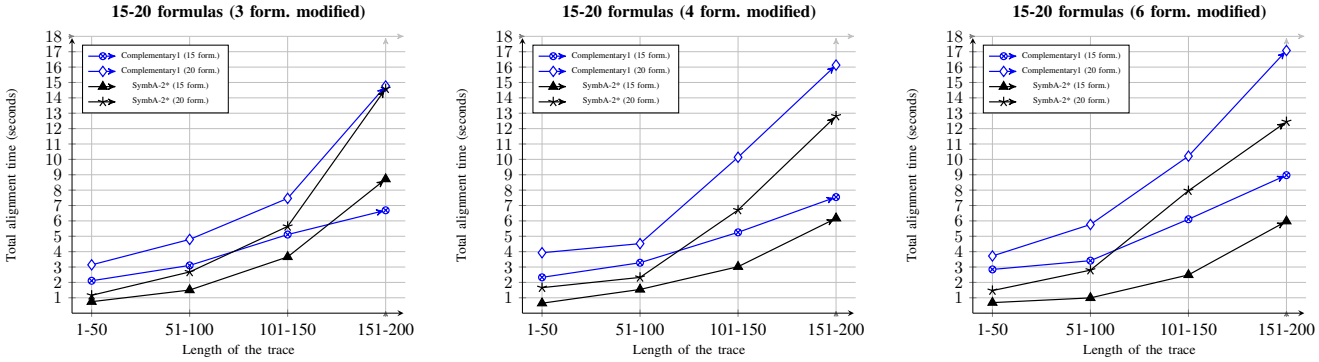

Fig. 6: Execution time for computing optimal alignments for the *synthetic* logs

Trace Length	Number of Traces	Average Preprocessing Time	Average Searching Time	Average Number of Deviations
1 – 10	388	0.90 s	0.05 s	6
11 – 20	1231	1.22 s	0.059 s	11
21 – 30	176	1.73 s	0.071 s	20
31 – 40	25	1.98 s	0.075 s	31
41 – 50	6	2.36 s	0.077 s	40
51 – 60	2	2.83 s	0.078 s	56

TABLE III: Experimental results for the *real-life* log

Having this in mind, we created 2 BP models in the form of DFAs having the same alphabet of activities and obtained by the conjunction of 15 and 20 LTL_f formulas, respectively. The models were both associated with a cost model that assigned a variable cost to the repetition of some specific activities. Then, to create logs containing noise, i.e., behaviors non-compliant with the original DFAs, we replaced 3, 4, and 6 LTL_f formulas with their negative counterparts and generated noisy logs from them. The logs were obtained using the log generator presented in [29]. For any of the 2 DFAs, we generated 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, resp. (see Tables I and II).

Results. The results of the experiments can be seen in Tables I and II and in Fig. 6. Note that columns "Alignment Cost" and "Context-Aware Alignment Cost" indicate, respectively, the cost of an optimal alignment in the case of a static cost function and in the presence of the employed cost model. The results suggest that the performance of the forward A* search of FAST-DOWNWARD decreases exponentially when the

tested DFA contains a higher number of states and transitions. Sometimes, FAST-DOWNWARD was not able to complete the computation of the alignment due to the excessive resource computation requested. On the other hand, both SYMBA*-2 and COMPLEMENTARY1 scale very well when the complexity of the DFA and the length of the log traces increases. Notably, both planners enact around the same number of alignment steps for alignments of the same cost (see column "Context-Aware Alignment Cost"). In addition, they seem not to suffer the presence of noisy logs. Indeed, the plots in Fig. 6 suggest that the computation of the alignment is feasible also in the case of traces requiring a large number of alignment actions. The only visible difference between the two planners is that COMPLEMENTARY1 requires a higher preprocessing time wrt. SYMBA*-2 to process the planning domains and problems generated by our technique.

Therefore, SYMBA*-2 performs slightly better than COMPLEMENTARY1 to resolve the context-aware trace alignment problem, but both perform significantly better than FAST-DOWNWARD. Such results can be explained with the observation that the heuristics adopted by SYMBA*-2 and COMPLEMENTARY1 are able to efficiently cope with the size of the state space, which is exponential wrt. the size of the model, the amount of noise and the trace length.

VII. CONCLUDING REMARKS

Alignments establish the best possible connection between an observed trace and a BP model, exhibiting the model run closest to the given observed trace. The literature approaches for trace alignment are context-agnostic and only support static cost models with fixed costs. In this paper, we have presented a technique that allows end users to define sophisticated cost models that assign costs to alignment operations depending on the context in which the operation is applied within a trace. From a formal perspective, our technique is based on a manipulation of DFAs to reformulate context-aware trace alignment into a cost-optimal planning problem that can be efficiently solved by state-of-the-art planning systems. The approach has been proven to be feasible through an extensive experimentation employing three state-of-the-art planners against a real-life and many synthetic logs. For future work, we plan to extend the current definition of context depending on the (bounded) sequences of activities executed before or after a deviation, to the unbounded case (e.g., the cost of a deviation indefinitely increases with the number of occurrences of a certain activity). In addition, the costs of the repair actions could also be defined as dependent on the value of trace data attributes or on the occurrence of specific values for some event data attributes (commonly available in the event logs). Given the complexity of these sophisticated cost models, we plan to define a number of patterns for recurrent cost models, mitigating the burden related to their definition, which is, at the moment, in charge of the BP designer.

Acknowledgments. This work was supported by the the H2020 project DataCloud and the Sapienza grant BPbots.

REFERENCES

[1] J. Carmona, B. F. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking - Relating Processes and Models*, 2018.

[2] L. Padró and J. Carmona, "Computation of alignments of business processes through relaxation labeling and local optimal search," *Inf. Syst.*, vol. 104, 2022.

[3] R. J. C. Bose and W. M. van der Aalst, "Process diagnostics using trace alignment: opportunities, issues, and challenges," *Inf. Syst.*, vol. 37, 2012.

[4] A. Guzzo, A. Rullo, and E. Vocaturo, "Process mining applications in the healthcare domain: A comprehensive review," *Wiley Interd. Rev.: Data Mining and Kn. Disc.*, vol. 12, no. 2, 2022.

[5] S. Coltellese, F. M. Maggi, A. Marrella, L. Massarelli, and L. Querzoni, "Triage of IoT attacks through process mining," in *COOPIS*, 2019.

[6] M. Boltenhagen, T. Chatain, and J. Carmona, "A Discounted Cost Function for Fast Alignments of Business Processes," in *BPM*, 2021.

[7] P. Bouvier and others, "Automatic decomposition of Petri nets into automata networks - A synthetic account," in *PETRI NETS*, 2020.

[8] G. De Giacomo and M. Y. Vardi, "Synthesis for LTL and LDL on finite traces." in *IJCAI*, 2015, pp. 1558–1564.

[9] M. de Leoni and A. Marrella, "Aligning real process executions and prescriptive process models through automated planning," *Expert Syst. Appl.*, vol. 82, pp. 162–183, 2017.

[10] G. De Giacomo, F. M. Maggi, A. Marrella, and F. Patrizi, "On the Disruptive Effectiveness of Automated Planning for LTL*f*-Based Trace Alignment," in *AAAI*, 2017.

[11] A. Marrella, "Automated planning for business process management," *J. Data Semant.*, vol. 8, no. 2, 2019.

[12] H. Geffner and B. Bonet, "A Concise Introduction to Models and Methods for Automated Planning," *Synth.Lect. on AI and ML*, vol. 8, no. 1, 2013.

[13] A. Adriansyah, N. Sidorova, and B. F. van Dongen, "Cost-Based Fitness in Conformance Checking," in *ACSD 2011*. IEEE, 2011.

[14] W. M. P. van der Aalst, "Decomposing Petri nets for process mining: A generic approach," *Distributed and Parallel Databases*, vol. 31, 2013.

[15] H. M. W. Verbeek and W. M. P. van der Aalst, "Merging alignments for decomposed replay," in *Application and Theory of Petri Nets and Concurrency*. Springer International Publishing, 2016, pp. 219–239.

[16] J. Munoz-Gama, J. Carmona, and W. M. P. van der Aalst, "Single-entry single-exit decomposed conformance checking," *Inf. Syst.*, vol. 46, 2014.

[17] D. Reißner, A. A-Cervantes, R. Conforti, M. Dumas, D. Fahland, and M. La Rosa, "Scalable alignment of process models and event logs: An approach based on automata and s-components," *Inf. Syst.*, vol. 94, 2020.

[18] F. Taymouri and J. Carmona, "A recursive paradigm for aligning observed behavior of large structured process models," in *BPM 2016*, 2016.

[19] ——, "Computing alignments of well-formed process models using local search," *ACM TOSEM*, vol. 29, no. 3, 2020.

[20] V. Bloemen, S. van Zelst, W. van der Aalst, B. van Dongen, and J. van de Pol, "Aligning observed and modelled behaviour by maximizing synchronous moves and using milestones," *Inf. Syst.*, vol. 103, 2022.

[21] M. Fox and D. Long, "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains," *JAIR*, vol. 20, 2003.

[22] A. Marrella and Y. Lespérance, "A planning approach to the automated synthesis of template-based process models," *Serv. Oriented Comput. Appl.*, vol. 11, no. 4, 2017.

[23] G. D. Giacomo, F. M. Maggi, A. Marrella, and S. Sardina, "Computing Trace Alignment against Declarative Process Models through Planning," in *ICAPS'16*. AAAI Press, 2016.

[24] N. Shinde and P. Kulkarni, "Cyber incident response and planning: a flexible approach," *Computer Fraud & Security*, 2021.

[25] R. Accorsi and T. Stocker, "On the exploitation of process mining for security audits: the conformance checking case," in *SAC*, 2012.

[26] M. Helmert, "The Fast Down. Planning System," *JAIR*, vol. 26, 2006.

[27] A. Torralba and et al., "Symba: A symbolic bidirectional a planner," in *International Planning Competition*, 2014, pp. 105–108.

[28] S. Edelkamp, "Cost-Optimal Planning in the IPC 2018: Symbolic Search and Planning Pattern Databases vs. Portfolio Planning," 2019.

[29] C. Di Ciccio, M. L. Bernardi, M. Cimitile, and F. M. Maggi, "Generating event logs through the simulation of Declare models," in *(EOMAS)*, 2015.