Aligning Partially-Ordered Process-Execution Traces and Models Using Automated Planning

Massimiliano de Leoni
Eindhoven University of Technology
m.d.leoni@tue.nl

Giacomo Lanciano
Sapienza - University of Rome, Italy
lanciano.1487019@studenti.uniroma1.it

Andrea Marrella
Sapienza - University of Rome, Italy
marrella@diag.uniroma1.it

Abstract

Conformance checking is the problem of verifying if the actual executions of business processes, which are recorded by information systems in dedicated event logs, are compliant with a process model that encodes the process' constraints. Within conformance checking, alignment-based techniques can exactly pinpoint where deviations are observed. Existing alignment-based techniques rely on the assumption of a perfect knowledge of the order with which process' activities were executed in reality. However, experience shows that, due to logging errors and inaccuracies, it is not always possible to determine the exact order with which certain activities were executed. This paper illustrates an alignment-based technique where the perfect knowledge assumption of the execution's order is removed. The technique transforms the problem of alignment-based conformance checking into a planning problem encoded in PDDL, for which planners can find a correct solution in a finite amount of time. We implemented the technique as a software tool that is integrated with state-of-the-art planners. To showcase its practical relevance and scalability, we report on experiments with a real-life case study and several synthetic ones of increasing complexity.

1 Introduction

Modern organizations are centered on the business processes needed to deliver products and services in an efficient and effective manner. *Business Process Management* (BPM) is the discipline that focuses on overseeing how work is performed in an organization by managing and optimising a company's business processes (Dumas et al. 2013).

Processes often need to comply with regulations and laws so that, e.g., the organizations can be certified as conforming the ISO standards. *Process models* can be used for this purpose. Within BPM, *conformance checking* aims at verifying whether the process' observed behavior, which is stored in an *event log*, matches the intended behavior represented as a process model (van der Aalst 2011). Event logs are broken down in *traces*, each of which containing the events referring to the execution of one process instance.

The notion of *alignment* (van der Aalst, Adriansyah, and van Dongen 2012) provides a robust approach to conformance checking, which makes it possible to pinpoint the deviations causing nonconformity. An alignment between a recorded process execution and a process model is a pairwise matching between activities recorded in the log and activities allowed by the model. Sometimes, activities as recorded in the log (*events*) can not be matched to any of the activities allowed by the model (*process activities*).

For instance, an activity is executed when not allowed (e.g., a loan is opened before assessing the applicant). In this case, we match the event with a special *null* activity (hereafter, denoted as $\gg$), thus resulting in so-called *moves in log*, which indicate that the log-trace pointer "moves" forward to consider the next event in trace while the model stays still. Other times, an activity should have been executed but is not observed in the event log (e.g., a loan is not opened after positively assessing the applicant). This results in a process activity that is matched to a $\gg$ event, thus resulting in a so-called *move in model*, which indicates that conversely the log-trace pointer remains still while the model progresses by firing exactly one transition. If an alignment between a log trace and process model contains at least one move in log/model, it means that the trace refers to a process execution that is not compliant with the allowed behavior that is encoded in a process model.

In general, a large number of possible alignments exist between a process model and a log trace, since there may exist manifold explanations why a trace is not conforming. It is clear that one is interested in finding the most probable explanation. In (van der Aalst, Adriansyah, and van Dongen 2012), an approach is proposed that is based on the principle of the Occam's razor: the most parsimonious explanation is preferable. Therefore, one should not aim to find any alignment but, precisely, one of the alignments with the least expensive deviations (one of the so-called *optimal alignments*), according to some function assigning costs to deviations.

Several existing techniques, such as (Adriansyah, Sidorova, and van Dongen 2011; Adriansyah, van Dongen, and van der Aalst 2013; van der Aalst, Adriansyah, and van Dongen 2012; Di Francescomarino et al. 2015; López et al. 2016; de Leoni and Marrella 2017; De Giacomo et al. 2016; 2017), rely on the assumption that each event in a trace has a different timestamp, which can be leveraged on to define the total ordering of events in traces. However, in several settings, this assumption is not met in reality: the logging mechanisms of information systems can sometimes record event's

timestamps with a course granularity. For instance, a logging system may not be able to record the minutes in which events occurred; as a consequence, events at, e.g., *5:05pm* or *5:10pm* take on the same timestamp: *5:00pm*.

In these settings, existing techniques must pick an arbitrary total ordering. If this is not chosen correctly, the alignments may pinpoint more deviations that in reality. As an example, let consider a process execution that consists of the following set of events: $\phi = \{a_{5:05pm}, b_{5:07pm}, c_{5:06pm}, d_{6:02pm}\}$ (the subscript is the timestamp when the event occurred). This corresponds to the process execution $\langle a, c, b, d\rangle$, which - assume - is compliant with its underlying process model. If the logging system has the hour granularity, this is recorded as the following events' set: $\overline{\phi} = \{a_{5:00pm}, b_{5:00pm}, c_{5:00pm}, d_{6:00pm}\}$. We lose the total-ordering information about events a, b and c. Existing techniques have to force one total ordering, such as the process execution $\langle a, b, c, d\rangle$, which it is not compliant. Consequently, the trace would be diagnosed as non-compliant, but it really was. The reason of this misdiagnosis is related to the logging-mechanism that has a course granularity.

A too-course granularity is not infrequent: event logs are typically extracted from information-system databases, where timestamps are often recorded through, e.g., "date" fields. Possibly one may modify the database fields and the information systems to log the exact timestamp; however, this would only affect the future logging, making it impossible to leverage on the existing logging data recorded until that moment in time, which may even refer to several years of information-system use.

This paper aims at releasing this total-ordering assumption: events with the same timestamp are considered as partially ordered and will be totally ordered so as to minimize the number of deviations. In relation to the example above, among all possible total orders, our technique will again choose $\langle a, c, b, d\rangle$, reporting the trace as compliant. Compared with existing techniques, we guarantee an important property: if a trace was compliant in reality, we will detect it as such, even in case of partially-ordered traces. Of course, the other way around is not guaranteed: if a trace was not compliant in reality, our technique may underestimate the deviations because it may re-order events differently from what occurred in reality with the aim of minimizing the deviations, in line with the principle of Occam's razor.

This paper extends (de Leoni and Marrella 2017) and illustrates how the problem of computing optimal alignments between process models and partially-ordered traces can be encoded as a planning problem, which can be solved through off-the-shelf PDDL planners. Specifically, given a process model and a real process execution recorded in an event log, we show how to build a planning domain and problem instance such that the solution steps of the problem are guaranteed to correspond to the alignment steps. Also, the paper illustrates how the encoding of alignment problem always generates planning problems for which planning systems find optimal solutions in a finite amount of time.

We notice that, to this date, no existing technique in the BPM literature can deal with the problem of partially-ordered traces. Furthermore, to motivate the employment of planning techniques, the experiments have shown that the alignment-problem complexity is large and a technique is necessary that can scale up adequately.

The rest of the paper is organized as follows. Section 2 elaborates on the relevant background to understand the paper, namely introducing the concept of process model, event log and process-trace alignment. Section 3 reports on our technique to convert alignment problems to planning problems, discussing correctness results for the resulting planning problems. Section 4 reports on the experiments conducted on both real-life and synthetic process models and event logs. The real-life experiments show a case study where timestamps do not have the right granularity, thus illustrating that the problem is practically relevant. The experiments of synthetic data sets demonstrates a good level of scalability of the technique. Section 5 concludes the paper.

2 Preliminaries

In this section, we provide some preliminary concepts used throughout the paper. In Section 2.1 we introduce the Petri Net modeling language. In Section 2.2 we describe the problem we want to solve through planning: constructing an optimal alignment between partially-ordered traces and process models represented as Petri nets. In Section 2.3 we provide an overview on automated planning, indicating which portion of PDDL is required for the encoding of our technique.

2.1 Petri Nets

Many notations have been introduced to represent business processes, such as BPMN, EPC, or UML Activity Diagrams (Dumas et al. 2013). These languages allow process designers to specify aspects linked to different perspectives, ranging from expressing the ordering with which activities need to be executed (control-flow perspective) till modelling the objects manipulated by activities or the constraints on the resources allowed to execute activities. Some languages are characterized by an ambiguous semantics; however, we need a simple language with clear semantics to explain our technique. Therefore, we opted for Petri nets, which have proven to be adequate for modelling business processes (van der Aalst 1998). This is especially true when the focus is only on the control-flow perspective, which is the case in this paper.

A Petri net is a directed graph with two node types called *places* (represented by circles) and *transitions* (represented by rectangles), which are connected via directed arcs. Arcs between two nodes of the same type are not allowed.

Definition 1 (Petri Net)**.** *A Petri net is a tuple $\langle P, T, F\rangle$ where:*

- *P is a finite set of places;*
- *T is a finite set of transitions;*
- *$F \subseteq (P \times T) \cup (T \times P)$ is the flow relation between places and transitions (and between transitions and places).*

Given a transition $t \in T$, ${}^\bullet t$ is used to indicate the set of *input places* of t, which are the places p with a directed arc from p to t (i.e., such that $(p, t) \in F$). Similarly, $t^\bullet$ indicates the set of *output places*, namely the places p with a direct arc from t to p. At any time, a place can contain zero or more

tokens, drawn as black dots. The state of a Petri net, a.k.a. *marking*, is determined by the number of tokens in places. Therefore, a marking m is a function $m : P \to \mathbb{N}$.

In any run of a Petri net, the number of tokens in places may change, i.e., the Petri net marking, according to the following enablement and firing rules:

- A transition t is **enabled** at a marking m iff each input place contains at least one token: $\forall\, p \in {}^\bullet t,\ m(p) > 0$.
- A transition t can **fire** at a marking m iff it is enabled. As result of firing a transition t, one token is "consumed" from each input place and one is "produced" in each output place. More formally, firing a transition t at marking m leads to a marking m' such that:

$$m'(p) = \begin{cases} m(p) - 1, & \text{if } p \in {}^\bullet t \setminus t^\bullet, \\ m(p) + 1, & \text{if } p \in t^\bullet \setminus {}^\bullet t, \\ m(p), & \text{otherwise.} \end{cases} \quad (1)$$

This is denoted as $m \xrightarrow{t} m'$. In the remainder, given a sequence of transition firing $\sigma = \langle t_1, \ldots, t_n \rangle \in T^*$, $m_0 \xrightarrow{\sigma} m_n$ is used to indicate $m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} m_n$.

When Petri nets are used to represent business processes, transitions are associated with process activities, more specifically to activity labels, and markings indicate the process state (van der Aalst 1998). Executions of business processes have a start and end. Therefore, Petri nets need to be associated with an *initial* and *final marking*.

Definition 2 (Labelled Petri Net)**.** *A Labelled Petri net is a tuple $\langle P, T, F, A, \ell, m_i, m_f \rangle$ where:*

- *$\langle P, T, F \rangle$ is a Petri net;*
- *A is the set of activity labels;*
- *$\ell : T \not\to A$ is a function that associates a label with some transitions in T;*
- *m_i and m_f are the initial and final marking.*

Given a Labelled Petri net $N = \langle P, T, F, A, \ell, m_i, m_f \rangle$, the labeling function ℓ does not need to be total, as certain transitions are not associated with a label. Such transitions do not represent actual pieces of work in processes but their introduction is sometimes necessary to properly represent the process behaviour. In the remainder, shortcuts $\mathsf{Inv}(N) \subset T$ indicates those transitions of N that are associated with no labels (i.e., not in the domain of ℓ). Fig. 1 shows an example of a Labelled Petri net that will be used throughout the paper. The transition's names are depicted inside the transitions. The initial and final marking have respectively one token in place *start* or in place *end* (and no tokens in any other place).

We conclude this section by introducing the concepts of **k-boundness**. A Labelled Petri net $N = \langle P, T, F, A, \ell, m_i, m_f \rangle$ is k-bounded if, in every marking m reachable from the initial marking m_i, every place $p \in P$ always contains at most k tokens, i.e., $m(p) \leq k$.

2.2 Alignment between Event Logs and Petri Nets

Event logs are the starting point for conformance checking. An event log is a set of *traces*. A trace describes the lifecycle of a *process instance* in terms of the *activities* (i.e., Petri net transitions) executed.

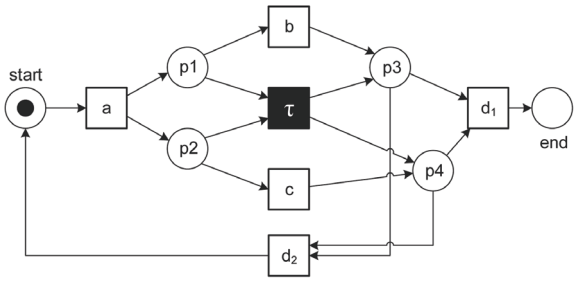

Figure 1: The Petri net used as working example. Transitions *a*, *b* and *c* are associated with labels of the same name. The black-colored transition τ is invisible and transitions d_1 and d_2 are both associated with label d.

Definition 3 (Event Log)**.** *Let $N = \langle P, T, F, A, \ell, m_i, m_f \rangle$ be a Labelled Petri net. Let $\mathcal{E}$ be the universe of events and $\mathcal{T}$ be the universe of timestamps. A trace $\phi \subset \mathcal{E}$ is a finite set of events. An event log L over N is a tuple $\langle L, \lambda_N, \lambda_T \rangle$ consisting of a set $L \subset 2^{\mathcal{E}}$ of traces, a function $\lambda_N : \mathcal{E} \to A$, which associates each event $e \in \mathcal{E}$ with an activity $\lambda_N(e) \in A$, and a function $\lambda_T : \mathcal{E} \to \mathcal{T}$, which associates each event $e \in \mathcal{E}$ with a timestamp $\lambda_T(e) \in \mathcal{T}$.*

The partial ordering of the events in a trace $\phi \subset \mathcal{E}$ is inducted through function λ_T, which assigns timestamps to events. Events with different timestamps are thus ordered but events with the same timestamp are not. A trace can also be regarded as a sequence of **isochronous groups**, each of which contains the trace events with the same timestamp. For example, let us consider the following event log:

$$[\langle \{a^1\}, \{b^2, c^3\}, \{d^4\} \rangle, \langle \{a^5, b^6\}, \{c^7\}, \{d^8\} \rangle, \langle \{a^9, b^{10}\}, \{c^{11}, d^{12}, b^{13}, c^{14}\}, \{d^{15}\} \rangle, \langle \{a^{16}\}, \{d^{17}\} \rangle, \langle \{a^{18}\}, \{d^{19}\} \rangle]$$

Notice that each event e is here represented as l^i where $\lambda_N(e) = l$ and i is some event identifier. This identifier is not used hereafter and is only introduced in this event log to highlight the uniqueness of events. This event log contains five partially-ordered traces. Readers should notice that, in this example, λ_T is not explicitly reported. The event-log traces are expressed in terms of isochronous groups, which are inducted by an appropriate λ_T function. In fact, the precise timestamps are irrelevant since they are only used to define the partial ordering. In the remainder of this paper, given an event $e \in \phi$, notation ${}^\triangleleft e$ denotes the set of all events $e' \in \phi$ such that $\lambda_T(e') < \lambda_T(e)$ (e.g., in the first trace, ${}^\triangleleft b^2 = [a^1]$, ${}^\triangleleft c^3 = [a^1]$ and ${}^\triangleleft d^4 = [a^1, b^2, c^3]$).

As mentioned in Section 1, we perform conformance checking by constructing an *alignment* of event log $\langle L, \lambda_N, \lambda_T \rangle$ and process model N, which allows us to exactly pinpoint where deviations occur. Building this alignment is far from trivial, since the log may deviate from the model at an arbitrary number of places. On this aim, the events in the event log need to be related to transitions in the model, and vice-versa. Specifically, we need to relate "moves" in the log to "moves" in the model. However, it may be that some of the moves in the log cannot be mimicked by the model and vice versa. We explicitly denote such "no moves" by $\gg$.

$$\gamma_1 = \begin{array}{|c|c|c|c|c|c|}\hline b & a & c & d & \gg & \gg \\ \hline \gg & a & c & \gg & b & d_1 \\ \hline \end{array} \qquad \gamma_2 = \begin{array}{|c|c|c|c|c|}\hline a & b & d & c & \gg \\ \hline a & b & \gg & c & d_1 \\ \hline \end{array}$$

$$\gamma_3 = \begin{array}{|c|c|c|c|}\hline a & b & c & d \\ \hline a & b & c & d_1 \\ \hline \end{array}$$

Figure 2: Alignments of trace $\langle\{a,b\},\{d,c\}\rangle$ and the process model in Fig. 1.

Definition 4 (Alignment Moves)**.** *Let $N = \langle P, T, F, A, \ell, m_i, m_f\rangle$ be a Labelled Petri net. Let $\langle L, \lambda_N, \lambda_T\rangle$ be an event log. A legal* alignment move *for N and $\langle L, \lambda_N, \lambda_T\rangle$ is represented by a pair $(s_L, s_M) \in (\mathcal{E} \cup \{\gg\} \times T \cup \{\gg\}) \setminus \{(\gg, \gg)\}$ such that:*

- *(s_L, s_M) is a* move in log *if $s_L \neq \gg$ and $s_M = \gg$,*
- *(s_L, s_M) is a* move in model *if $s_L = \gg$ and $s_M \in T$,*
- *(s_L, s_M) is a* synchronous move *if $s_L \neq \gg$, $s_M \in T \setminus \mathit{Inv}(N)$ and $\lambda_N(s_L) = \ell(s_M)$*

To properly define the concept of alignment, we need to introduce the notion of *prefix* of a sequence $\sigma = \langle x_1, \ldots, x_n\rangle$ for each $x_i \in \sigma$: $prefix(\sigma, x_i) = \langle x_1, \ldots, x_i\rangle$. An *alignment* is a sequence of alignment moves defined as:

Definition 5 (Alignment)**.** *Let $N = \langle P, T, F, A, \ell, m_i, m_f\rangle$ be a Labelled Petri net and $\langle L, \lambda_N, \lambda_T\rangle$ be an event log. Let Γ_N be the universe of all alignment moves for N and $\langle L, \lambda_N, \lambda_T\rangle$. Let $\phi \in L$ be a log trace. Sequence $\gamma \in \Gamma_N^*$ is an alignment of N and ϕ if, ignoring all occurrences of $\gg$:*

- *the projection on the first element of γ yields a sequence $\sigma' \in \mathcal{E}^*$ such that*
  - *$e \in \sigma' \Leftrightarrow e \in \phi$, and*
  - *$\forall e' \in \sigma'.\ e'' \in prefix(\sigma', e') \Rightarrow \lambda_T(e'') \leq \lambda_T(e')$.*
- *the projection on the second element of γ yields a sequence $\sigma'' \in T^*$ such that $m_i \xrightarrow{\sigma''} m_f$.*

Many alignments are possible for the same trace. For example, Fig. 2 shows three possible alignments for a trace $\phi_{ex} = \langle\{a,b\},\{d,c\}\rangle$. Note how moves are represented vertically. For example, as shown in Fig. 2, the first move of γ_1 is $(b, \gg)$ (i.e., a move in the log of b), the second is (a, a) (i.e., a synchronous move of a) and the fifth move is $(\gg, b)$ (i.e., a move in the model of b). However, we aim at finding a complete alignment of ϕ and N with minimal deviation cost. In order to define the severity of a deviation, we first introduce a cost function on legal moves and, then, generalize it to alignments. Any alignment with the lowest possible cost is named an *optimal alignment*.

Definition 6 (Cost Function)**.** *Let $N = \langle P, T, F, A, \ell, m_i, m_f\rangle$ be a Labelled Petri net and $\phi \in L$ a log trace, respectively. Assuming Γ_N as the set of all legal alignment moves, a* cost function *κ assigns a non-negative cost to each legal move: $\Gamma_N \rightarrow \mathbb{N}_0^+$. The* cost of an alignment *$\gamma \in \Gamma_N$ between ϕ and N is computed as the sum of the cost of all constituent moves: $\mathcal{K}(\gamma) = \sum_{(s_L, s_M)\in\gamma} \kappa(s_L, s_M)$.*

Alignment γ is an **optimal alignment** if, for any alignment γ' of N and ϕ, $\mathcal{K}(\gamma) \leq \mathcal{K}(\gamma')$. This cost function can be used to favor one type of explanation for deviations over the other. The cost of each legal move depends on the specific model and process domain and, hence, the cost function κ needs to be defined specifically for each setting. Note that an optimal alignment does not need to be unique, i.e., multiple alignments with the same minimal cost may exist.

Let us consider the following cost function for the model in Fig. 1:

$$\kappa\big((s_L, s_M)\big) = \begin{cases} 1 & \text{if } s_M = \gg, \\ 1 & \text{if } s_L = \gg \text{ and } s_M \neq \tau, \\ 0 & \text{otherwise.} \end{cases} \qquad (2)$$

With reference to the alignments in Fig. 2, alignments γ_1 and γ_2 have cost 4 and 2 respectively. Conversely, γ_3 takes on a cost 0. So, γ_3 is a less expensive alignment and, in particular, is among the least expensive alignments. Therefore γ_3 is an optimal alignment. Specifically, since the cost is 0, trace ϕ_{ex} is compliant with the model in Fig. 1. As mentioned in Section 1, the existing techniques based on total-ordering would enforce one of possible total ordering of events in trace ϕ_{ex}, such as $\langle a, b, d, c\rangle$, with the latter yielding uncorrect diagnosis of deviations.

2.3 Automated Planning

Planning systems are problem-solving algorithms that operate on explicit representations of states and actions (Bonet and Geffner 2001; Ghallab, Nau, and Traverso 2004). PDDL (McDermott et al. 1998) is the standard Planning Domain Definition Language; it allows us to formulate a *planning problem* $\mathcal{P} = \langle I, G, \mathcal{P}_\mathcal{D}\rangle$, where I is the description of the initial state of the world, G is the desired goal state, and $\mathcal{P}_\mathcal{D}$ is the planning domain. A planning domain $\mathcal{P}_\mathcal{D}$ is built from a set of *propositions* describing the state of the world (i.e., by the set of propositions that are true) and a set of *operators* Ω (i.e., *actions*) that can be executed in the domain. An *action schema* $a \in \Omega$ is of the form $a = \langle \mathit{Par}_a, \mathit{Pre}_a, \mathit{Eff}_a\rangle$, where Par_a is the list of *input parameters* for a, Pre_a defines the *preconditions* under which a can be executed, and Eff_a specifies the *effects* of a on the state of the world. Both preconditions and effects are stated in terms of the *propositions* in $\mathcal{P}_\mathcal{D}$. Propositions can be represented through boolean predicates and numeric fluents.

There exist several forms of planning in the literature. In this paper, we focus on planning techniques characterized by *fully observable*, *static* and *deterministic* domains, i.e., we rely on the *classical* planning assumption of a "perfect world description" (Wilkins 1988). A solution for a planning problem is a sequence of operators—a *plan*—whose execution transforms the initial state I into a state satisfying the goal G. To this end, we represent planning domains and problems making use of the STRIPS fragment of PDDL 2.1 (Fox and Long 2003), enhanced with the numeric features provided by the "level 2" of the same language, which are used to keep track of the costs of planning actions and to synthesize plans satisfying pre-specified metrics.

3 Encoding the Alignment Problem in PDDL

This section illustrates how, given an event log $\langle L, \lambda_N, \lambda_T\rangle$ and Labelled Petri net $N = \langle P, T, F, A, \ell, m_i, m_f\rangle$, the

problem of the alignment of N and of a partially-ordered trace $\phi \in L$ can be encoded as a PDDL planning problem, which can be solved by state-of-the-art planners. The synthesized plan will consist of a sequence of alignment moves that establish an alignment between ϕ and N.

In the remainder of this paper, we assume Labelled Petri nets to be 1-bounded. This is not a large limitation as the behavior allowed by most of business processes can be represented as 1-bounded Petri nets (van der Aalst 2002). As further support of the practical relevance of 1-bounded Petri nets, readers can look at large body of recent work on the usage of block-structured process models, a special subclass of 1-bounded Petri nets (see, e.g., (Leemans, Fahland, and van der Aalst 2014)). Also, any k-bounded Petri net N' for any k can be transformed into a 1-bounded Petri net: first, the reachability graph of N' is built (van der Aalst 1998; 2002), which is a finite transition system because N' is k-bounded; then, the reachability graph is trivially translated into a 1-bounded Petri net, constructing one Petri net place for each reachability graph state and one Petri net transition for each reachability graph transition.

3.1 Predicates and fluents

In the planning domain $\mathcal{P}_\mathcal{D}$, we provide three object types called `place`, `transition` and `event`. The types `place` and `transition` represent respectively the places and the transitions of N. The type `event` is used to record the list of events that occur in the specific trace ϕ that must be checked for conformance. For example, for trace $\phi = \langle\{a, b\}, \{d, c\}\rangle$ and the Petri net in Fig. 1, the following objects are introduced in the planning problem $\mathcal{P}$:

```
(:objects a b c d1 d2 inv - transition
          start p1 p2 p3 p4 end - place
          e_a e_b e_d e_c - event)
```

Notice that we have explicitly introduced the invisible transition `inv`. To capture all possible markings of N and the evolution of ϕ during an alignment, we define two boolean predicates in $\mathcal{P}_\mathcal{D}$, as follows:

token. For each $p \in P$, `(token ?p - place)` holds iff p contains a token in the currently reached marking.

aligned. For each event $e \in \phi$, `(aligned ?e - event)` holds iff the event has been aligned in the current (partial) solution, namely the alignment built so far.

In addition to the predicates above, assuming a cost function κ, the following fluents are also introduced:

move-model-cost. For each transition $t \in T$, fluent `(move-model-cost ?t - transition)` takes on the cost of a model move for t, i.e., the value of $\kappa((\gg, t))$.

move-log-cost. For each event $e \in \phi$, fluent `(move-log-cost ?e - event)` takes on the cost of a log move for e, i.e., the value of $\kappa((e, \gg))$.

total-cost. `(total-cost)` keeps track of the cost of the alignment currently found.

Synchronous moves are associated with no cost and, hence, no fluent needs to be introduced. For trace $\phi = \langle\{a, b\}, \{d, c\}\rangle$ and the Petri net in Fig. 1, the initial state of $\mathcal{P}$ with the definition of predicates and fluents is as follows:

```
(:init (token start)
 (= (total-cost) 0)
 (= (move-model-cost a) 1)    (= (move-model-cost b) 1)
 (= (move-model-cost c) 1)    (= (move-model-cost d1) 1)
 (= (move-model-cost d2) 1)   (= (move-model-cost inv) 0)
 (= (log-move-cost e_a) 1)    (= (log-move-cost e_b) 1)
 (= (log-move-cost e_c) 1)    (= (log-move-cost e_d) 1))
```

Readers can observe that the last five lines are the representation of the cost-function example in Equation (2) of Section 2.2. At the end, for the example in question, when an alignment is found, the Petri net needs to be in the final marking, i.e., with one token in place `end` and zero tokens in any other place, and `aligned` must hold for all the events in the trace:

```
(:goal (and (token end) (not (token a)) (not (token b))
 (not (token c)) (not (token d1)) (not (token d2))
 (not (token inv)) (aligned e_a) (aligned e_b)
 (aligned e_c) (aligned e_d)))
```

Readers should notice that, since our purpose is to minimize the total cost of the alignment, the planning problem also contains the following specification: `(:metric minimize (total-cost))`.

3.2 Planning Actions

The plan to reach the final goal from the initial state is constituted by a sequence of alignment moves, each of which is a planning action. Therefore, three classes of actions exist in $\mathcal{P}_\mathcal{D}$: synchronous moves, model moves and log moves.

Synchronous moves. A separate action exists for each pair $(e, t) \in \phi \times (T \setminus \mathsf{Inv}(N))$ such that $\ell(t) = \lambda_N(e)$ (i.e., for each pair (e, t) such that the event is associated with the same label as transition t) to represent a synchronous move for event e and transition t. For instance, let us consider transition c of the model in Fig. 1; synchronous move for c is associated with the following action:

```
(:action moveSync-c-e_c
 :precondition (and (token p2) (aligned e_a)
                    (aligned e_b))
 :effect (and (not (token p2)) (token p4) (aligned e_c)))
```

The preconditions of the action are: *(i)* c is enabled, i.e., all input places of c contain a token; *(ii)* each event $e_x \in {}^\triangleleft e$ has already been aligned. In this example, ${}^\triangleleft e\_c = \{e\_a, e\_b\}$. The effects denote the firing of c and the fact that the event e_c is marked as aligned.

Model moves. A separate action exists for each transition $t \in T$. A model move focuses on making an enabled transition t fire without performing any move in ϕ. For instance, let us consider transition c of the model in Fig. 1; a model move for c is associated with the following action:

```
(:action moveInTheModel-c
   :precondition (token p2)
   :effect (and (not (token p2)) (token p4)
        (increase (total-cost) (move-model-cost c))))
```

The effects are accordant to the firing rules in Equation (1) of Section 2.1. It is worthy observing how the execution of a model move for c makes the total cost (of the alignment) increase of a value equal to the cost of a model move for c.

Log moves. A separate action exists for each event $e \in \phi$ to represent a log move for e. For instance, consider event e_c:

```
(:action moveInTheLog-e_c
   :precondition (and (aligned e_a) (aligned e_b))
   :effect (and (aligned e_c) (increase (total-cost)
                  (log-move-cost e_c))))
```

The preconditions are that all the events belonging to ${}^{\triangleleft}e\_c$ have been already aligned (notice that e_d belongs to the same isochronous group of e_c). The effects allow one to mark the event e_c as aligned and increase the cost of the alignment of a value equal to the cost of a log move for e_c.

3.3 Discussion on Correctness and Termination

The proposed encoding of an alignment problem $\mathcal{A}$ as planning problem $\mathcal{P}$ maintains the propositions of correctness and termination discussed in (de Leoni and Marrella 2017). Therefore, it is always possible to find a solution for $\mathcal{A}$ by solving $\mathcal{P}$ in a finite amount of time. Also, *if planning systems are used that guarantee the optimality of the solution, the solution is one of the alignments with the lowest cost, which is an optimal alignment*.

In the remainder of this section, we consider a compliant trace as a negative diagnosis, and the other way around for non-compliant traces. The proposition below illustrates that, differently from (de Leoni and Marrella 2017), we guarantee that every compliant trace ϕ (negative) is diagnosed as such, even if event timestamps are not recorded at the right granularity, leading to partially-ordered trace ϕ_p:

Proposition 7 (Absence of False Positives)**.** *Let $N = \langle P, T, F, A, \ell, m_i, m_f \rangle$ be a labelled Petri net. Let $L = \langle L, \lambda_N, \lambda_T \rangle$ be an event log over N. Let $\phi \in L$ be a totally-ordered trace; namely, for all events $e', e'' \in \phi$, $\lambda_T(e') = \lambda_T(e'') \Leftrightarrow e' = e''$.*

Let ϕ_p be any partial trace constructed from ϕ as follows. For each event $e_i \in \phi$, ϕ_p contains an event e_i^p, such that $\lambda_N(e_i^p) = \lambda_N(e_i)$. Furthermore, for each $e_i, e_j \in \phi$, the two events $e_i^p \in \phi_p$ and $e_j^p \in \phi_p$ are constructed such that if $\lambda_T(e_i) < \lambda_T(e_j)$, then $\lambda_T(e_i^p) \leq \lambda_T(e_j^p)$.

Let $\mathcal{P}$ be the planning problem to compute the optimal alignment of ϕ_p and N. If ϕ is compliant and $\mathcal{P}$ is solved by a planning system that guarantees optimality, the solution of $\mathcal{P}$ (i.e., the optimal alignment) has cost 0.

Proof. For the sake of simplicity, we use the notation with isochronous groups. Since ϕ is totally ordered, all isochronous groups are one-sized: $\phi = \langle \{e_1\}, \ldots, \{e_{i-1}\}, \{e_i\}, \ldots, \{e_j\}, \ldots, \{e_k\}, \{e_{k+1}\}, \ldots, \{e_n\} \rangle$. Without losing of generality, let us take a partially-ordered trace ϕ_p built from ϕ with one isochronous group of size greater than 1: $\phi_p = \langle \{e_1\}, \ldots, \{e_{i-1}\}, \{e_i, ..., e_j, ..., e_k\}, \{e_{k+1}\}, \ldots, \{e_n\} \rangle$. Let us assume by contradiction that $\mathcal{P}$ will return an alignment γ of cost greater than 0. However, there is an alignment γ_O of cost 0:[1]

$$\gamma_O = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|} \hline e_1 & \ldots & e_{i-1} & e_i & \ldots & e_j & \ldots & e_k & e_{k+1} & \ldots & e_n \\ \hline t_1 & \ldots & t_{i-1} & t_i & \ldots & t_j & \ldots & t_k & t_{k+1} & \ldots & t_n \\ \hline \end{array}$$

where $\lambda_N(e_i) = \ell(t_i)$ for all $1 \leq i \leq n$. Since the cost $K(\gamma) > K(\gamma_O) = 0$, $\mathcal{P}$ was solved through a planner that does not guarantee optimality. The latter breaks the hypothesis of optimality of the employed planning system. □

[1] N may have invisible transitions that need to fire to execute $\langle e_1, \ldots, e_n \rangle$. In this case, some model moves for invisible transitions, with cost 0, are intertwined with the synchronous moves. Clearly, this does not invalidate the proof.

4 Implementation and Validation

The technique discussed in Section 3 is implemented as a Java plug-in for ProM, an established open-source framework for implementing Process Mining tools and algorithms (Verbeek et al. 2010). Through a GUI, it is also possible to customize the cost function. Specifically, the plug-in is available in the `PlanningBasedAlignment` package of the so-called "nightly-build" version of ProM under the name *"Planning-Based Alignment of Event Logs and Petri Nets"*.[2] The tool is integrated with the FAST-DOWNWARD planning framework (Helmert 2006) to find optimal alignments. To speed up the search, we employed the BLIND heuristic, which is admissible and safe and, thus, guarantees to produce optimal solutions.

In the remainder of this section, we report on the experiments conducted on a real-life case study and on synthetic models and event logs. We performed every experiment with a machine with an Intel Xeon W3530 CPU 2.80GHz and 8GB RAM. We used a cost function that assigns a cost 1 to log moves and model moves for non-invisible transition, similar to Equation (2). *We also employed heuristics that are admissible and safe, which guarantees the solutions to be always of the lowest cost*, namely the returned alignments are always optimal. The synthetic event logs and process models are available for download (Lanciano and de Leoni 2018).

4.1 Real-life Case Study

The real-life case study refers to the process to manage applications by a Dutch financial institute. Fig. 3 shows the Petri net that models this process. The initial marking M_i consists in one token in both places *Start* and *F Storage*, while the final marking M_f consists in one token in place *End*. For this case study, we were provided with an event log composed of 115295 traces, which records the execution of so many concrete process instances. For the sake of confidentiality, the actual names of the activities are obfuscated and the event log cannot be shared. Process executions consist of 4 events in average, and there are no traces longer than 19 events. Besides, the size of the isochronous groups in a trace is equal to 3.23 events on average. The largest isochronous group in a log trace is made of 16 events.

Using our tool, we computed the optimal alignment of the event log on the Petri net, employing both the technique based on the total-order assumption in (de Leoni and Marrella 2017) and that discussed in this paper, which does not require the assumption. Fig. 3 shows the projection of the deviations onto the Petri net for both of these cases. The colour of a Petri net transition t ranges from white to red with different shades of orange and yellow in accordance with the sum of the number of log and model moves for t

[2] The nightly-build can be downloaded at http://www.promtools.org/doku.php?id=nightly. The PackageManager of ProM can be used to install the required package.

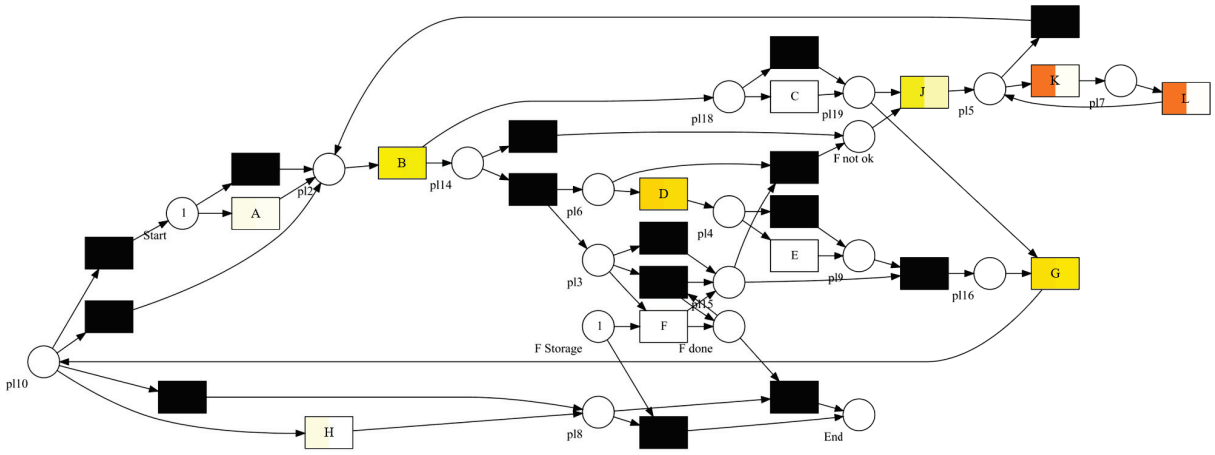

Figure 3: The projection of the deviations onto the Petri net used in the real-life case study. The left-hand half of each activity is related to the settings assumed in (de Leoni and Marrella 2017), the right-hand half is related to the partial ordering setting.

for all optimal alignments divided by the sum of all moves for t for all optimal alignments (i.e., the latter includes synchronous moves, as well). In particular, for each log trace, we consider one optimal alignment, namely that returned by the conformance-checker techniques.

The left-hand side of a transition shows the colors computed through the technique in (de Leoni and Marrella 2017) and can be easily compared with the right-hand side, which refers to the technique discussed in this paper. Our technique detects a smaller fraction of deviations (the transition color is "less red"), compared with what computed in (de Leoni and Marrella 2017). This holds, for instance, for transitions J, K and L. This case study shows that considering the trace as partially-ordered allows one to overcome, at least in part, the issue of having a coarse granularity for the event timestamps. As mentioned before, this could be partly underestimating the number of deviations, probably not as much as the previous technique overestimated; but, certainly, our technique remarkably guarantees that compliant traces will never diagnose as deviant (see Proposition 7).

Although this experiment was based on rather short traces, the results show that the problem of checking conformance of partially-ordered traces exist in reality, thus motivating the practical relevance of finding a solution for it. The real-life case study is not intended to study the level of scalability of the technique with models/logs of increasing size. This will be achieved through the case studies with synthetic data sets, which is reported in the following section.

4.2 Synthetic Experiments

In order to assess the scalability with respect to the complexity of the alignment problem, we take into account four synthetic models of different size and their related event logs. Specifically, we generate Petri nets *d53*, *d62*, *d63* and *d64*, with respectively 95, 136, 175 and 263 (visible and invisible) transitions, which are available for download in PNML format (Lanciano and de Leoni 2018). For each Petri net, we generate an event log with 1000 traces. The average trace length is respectively 42.8, 58.3, 148.07, 156.21 for the four processes in question. We used the *PLG2* tool (Burattin 2017) to generate the Petri nets and the event logs, using the default parameters to configure the generation.

The initially-generated event logs only contained traces that are compliant. To compute the scalability wrt. event logs with traces affected by an increasing number of deviations, we artificially injected noise. The injection of the $Y\%$ of noise in an event log means that, for each log trace, an event is swapped with the next one with a probability of $Y\%$, thus introducing further potential deviations with respect to the underlying process model. In our experiments, Y is set to 10%, 20% and 30%. Orthogonally, we aimed to evaluate the scalability wrt. isochronous event groups of growing size within each trace. Intuitively, larger isochronous groups lower the number of constraints and, hence, the search space becomes larger, increasing the complexity of the problem of aligning partially-ordered traces. For each of the four event logs, we manipulate the timestamps so as to obtain the isochronous groups that, on average, contain X events. In our experiments, X is set to 10, 20 and 30.

Because of the extreme complexity of the resulting alignment problems, the FAST-DOWNWARD planning framework, which was used for the real-life case study, failed in most of the experiments based on synthetic data. Therefore, we switched to *SymBA*-2* (Torralba et al. 2014), which appears more suitable for the class of our trace-model alignment problems, while keeping guarantees of optimality. To this date, due to some implementation issues, this planner is not yet integrated in our graphical tool. However, our tool is versatile and allows one to extract the PDDL files to be used as input to any PDDL planning system. The significantly-better performances of SymBA*-2 are remarkable and likely related to the use of bidirectional A* search, which appears in practice to be more efficient for the class of planning problems derived from alignment problems.

Table 1 shows the results of the experiment for different combinations of models, noise amount and isochronous-

Table 1: Experimental results for synthetic case studies. Numbers reported are average times (ms) over all traces in the logs.

groups size	prepr.	search	prepr.	search	prepr.	search	prepr.	search
d53	original log		10% noise		20% noise		30% noise	
10	17.39	1,076.88	16.25	1,083.63	16.04	1,082.71	16.2	1,089.36
20	16.19	1,057.39	16.16	1,078.46	16.42	1,272.76	16.19	1,376.38
30	16.13	1,303.4	16.1	1,255.78	16.36	1,250.42	16.35	1,278.75
d62	original log		10% noise		20% noise		30% noise	
10	43.86	2,556.3	44.67	2,586.87	44.26	2,600.58	44.49	2,612.92
20	44.27	2,641.21	44.41	2,545.33	44.88	2,668.87	44.37	2,685.34
30	42.04	2,666.71	41.28	2,774.24	42.04	2,636.65	41.96	2,564.27
d63	original log		10% noise		20% noise		30% noise	
10	129.77	14,891.66	129.81	15,316.17	128.71	15,482.54	129.4	17,282.78
20	131.85	19,867.1	130.15	21,107.42	129.58	22,652.23	129.36	25,339.51
30	132.09	51,293.36	130.87	46,830.37	130.57	52,304.72	131.75	60,037.62
d64	original log		10% noise		20% noise		30% noise	
10	211.43	20,798.12	211.1	18,839.27	212.05	19,213.27	209.87	19,539.5
20	207.34	25,852.17	210.86	27,072.33	204.69	27,886.47	211.31	29,540.09
30	210.39	62,675.21	208.06	—	213.84	—	217.07	—

group size. For each combination, the table reports the pre-processing and the search time. The latter is the time needed to compute the solution, whereas the pre-processing time is the time to convert the alignment problem to a planning problem in PDDL along with the time to instantiate the opportune data structure inside the planners to start the search.

From Table 1, some conclusions can be drawn. The pre-processing only depends on the size of the process model and, consequently, of the trace size. This could be expected because the pre-processing depends on the number of predicates, fluents and actions, which is mostly influenced by the model size. Regarding the search time, the amount of noise has a small influence, compared with the size of the models and of the isochronous groups. The latter two factors have a huge influence of the computation time: as expected, the computation time grows exponentially wrt. the size of the models and of the isochronous groups. This is more evident with the two larger models. As a matter of fact, our technique was unable to compute alignments for the largest model with the largest configurations of isochronous-group sizes. This is merely caused by the limited amount of memory at disposal. We repeated the tests that failed by using a machine with 12 GB of memory; and the technique was able to complete the experiments for those complex configurations. It is worth noting that having groups of 30 events with the same timestamp is fairly uncommon in real settings, as illustrated in the real-life case study, so like having models of those sizes.

5 Concluding Remarks

Within the BPM field, conformance checking is the problem of verifying whether executions as recorded in the event logs are compliant with a normative model of process. The notion of *alignment* (van der Aalst, Adriansyah, and van Dongen 2012) provides a robust approach to conformance checking, which makes it possible to pinpoint the deviations causing nonconformity. Existing techniques rely on a total ordering of events in traces, which is an assumption that is not always met in reality, due to course logging of event's timestamps.

This paper has reported on a planning-based technique that extends what was proposed in (de Leoni and Marrella 2017) by removing the total-ordering assumption. We have shown that automated planning offers a mature paradigm for tackling the problem of alignment-based conformance checking (in case of partially-ordered traces) in a theoretically grounded and domain-independent way. Our technique is based on the idea that alignment problems can be represented as planning problems in PDDL, which can be solved by off-the-shelf planners. The real-life experiments illustrate that the problem of partially-ordered traces is encountered in practice and that the corresponding operationalization can be applied to real-life case studies. The results with synthetic event logs and process models of growing complexity showcase that the technique scales sufficiently well. Theoretical results show that, in a finite amount of time, either the technique returns a correct, optimal solution when a solution exists or it stops when no solution exists.

As previously indicated, it is theoretically possible that, in case of partially-ordered traces, the number of deviations is underestimated because the partially-order events are ordered to minimize the number of deviations, which might not coincide with the real ordering. However, the real ordering is an unknown information which we cannot rely on. As future work, instead of classifying a trace either compliant or not, it is worthwhile exploring the possibility to diagnose if a trace is (1) always or (2) never conforming, irrespectively of how the partial orders are resolved, or, conversely, is (3) conforming under the assumption of a certain resolution of the partial order. This is far from trivial: an exhaustive analysis of all partial-order resolutions leads to combinatorial explosion, making the problem intractable.

Acknowledgements. This work is partly supported through the European Community's Seventh Framework Program FP7 under grant agreement n. 603993 (CORE), the Sapienza grant DAKIP ("Data-aware Adaptation of Knowledge-intensive Processes in Cyber-Physical Domains through Action-based Languages") and the Italian projects Social Museum and Smart Tourism (CTN01_00034_23154), RoMA - Resilience of Metropolitan Areas (SCN_00064) and FilieraSicura.

References

Adriansyah, A.; Sidorova, N.; and van Dongen, B. F. 2011. Cost-Based Fitness in Conformance Checking. In *Proc. of Int. Conf. on Application of Concurrency to System Design (ACSD 2017)*. IEEE.

Adriansyah, A.; van Dongen, B. F.; and van der Aalst, W. M. P. 2013. Memory-Efficient Alignment of Observed and Modeled Behavior. BPM Center Report BPM-03-03, BPMcenter.org.

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

Burattin, A. 2017. PLG2: Multiperspective Process Randomization with Online and Offline Simulations. In *Proc. of the BPM Demo Track 2016 Co-located with the 14th Int. Conf. on Business Process Management (BPM 2016)*, volume 1789 of *CEUR Workshop Proc.* CEUR-WS.org.

De Giacomo, G.; Maggi, F. M.; Marrella, A.; and Sardiña, S. 2016. Computing Trace Alignment against Declarative Process Models through Planning. In *Proc. of the Twenty-Sixth Int. Conf. on Automated Planning and Scheduling (ICAPS 2016)*. AAAI Press.

De Giacomo, G.; Maggi, F. M.; Marrella, A.; and Patrizi, F. 2017. On the Disruptive Effectiveness of Automated Planning for LTL$_f$-Based Trace Alignment. In *Proc. of the Thirty-First AAAI Conf. on Artificial Intelligence (AAAI-17)*. AAAI Press.

de Leoni, M., and Marrella, A. 2017. Aligning Real Process Executions and Prescriptive Process Models through Automated Planning. *Expert Systems with Applications* 82.

Di Francescomarino, C.; Ghidini, C.; Tessaris, S.; and Sandoval, I. V. 2015. Completing Workflow Traces Using Action Languages. In *Proc. of the 27th Int. Conf. on Advanced Information Systems Engineering (CAiSE 2015)*. Springer.

Dumas, M.; La Rosa, M.; Mendling, J.; and Reijers, H. A. 2013. *Fundamentals of Business Process Management*. Springer Berlin Heidelberg.

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

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

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.* 26.

Lanciano, G., and de Leoni, M. 2018. Models and Logs used by de Leoni et al. in the paper accepted for ICAPS 2018. http://doi.org/10.4121/uuid:a02afec8-b7c7-42b7-8dff-36d3de3032be.

Leemans, S. J.; Fahland, D.; and van der Aalst, W. M. P. 2014. Discovering Block-Structured Process Models from Incomplete Event Logs. In *Proc. of the 35th Int. Conf. on Application and Theory of Petri Nets and Concurrency (Petri Nets 2014)*. Springer.

López, M. T. G.; Borrego, D.; Carmona, J.; and Gasca, R. M. 2016. Computing Alignments with Constraint Programming: The Acyclic Case. In *Proc. of the Int. Workshop on Algorithms & Theories for the Analysis of Event Data*, volume 1592 of *CEUR Workshop Proc.* CEUR-WS.org.

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, New Haven, Connecticut.

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

van der Aalst, W. M. P.; Adriansyah, A.; and van Dongen, B. 2012. Replaying History on Process Models for Conformance Checking and Performance Analysis. *WIREs Data Mining and Knowledge Discovery* 2(2).

van der Aalst, W. M. P. 1998. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers* 8(1).

van der Aalst, W. M. P. 2002. *Workflow Management: Models, Methods, and Systems*. MIT Press.

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

Verbeek, E.; Buijs, J. C. A. M.; van Dongen, B. F.; and van der Aalst, W. M. P. 2010. ProM 6: The Process Mining Toolkit. In *Proceedings of the Business Process Management 2010 Demonstration Track, Hoboken, NJ, USA, September 14-16, 2010*, volume 615 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann.