Aligning Real Process Executions and Prescriptive Process Models through Automated Planning

M. de Leoni[a], A. Marrella[b]

[a]*Eindhoven University of Technology, The Netherlands*
[b]*Sapienza University of Rome, Italy*

Abstract

Modern organizations execute processes to deliver product and services, whose enactment needs to adhere to laws, regulations and standards. Conformance checking is the problem of pinpointing where deviations are observed. This paper shows how instances of the conformance checking problem can be represented as planning problems in PDDL (Planning Domain Definition Language) for which planners can find a correct solution in a finite amount of time. If conformance checking problems are converted into planning problems, one can seamlessly update to the recent versions of the best performing automated planners, with evident advantages in term of versatility and customization. The paper also reports on results of experiments conducted on two real-life case studies and on eight larger synthetic ones, mainly using the FAST-DOWNWARD planner framework to solve the planning problems due to its performances. Some experiments were also repeated though other planners to concretely showcase the versatility of our approach. The results show that, when process models and event logs are of considerable size, our approach outperforms existing ones even by several orders of magnitude. Even more remarkably, when process models are extremely large and event-log traces very long, the existing approaches are unable to terminate because they run out of memory, while our approach is able to properly complete the alignment task.

Keywords: Conformance Checking, Automated Planning, Business Process Management, Process Mining

1. Introduction

Process mining is about extracting knowledge from event logs commonly available in today's information systems. These techniques provide new means to discover, monitor and improve processes in a variety of application domains.

Email addresses: `m.d.leoni@tue.nl` (M. de Leoni), `marrella@diag.uniroma1.it` (A. Marrella)

Whereas large attention has been paid to discovery models that summarize the behavior observed in the event log, less effort has been put on conformance checking, which aims to find commonalities and discrepancies between the modeled and the observed behavior.

However, a number of previous works advocates the use of conformance checking for business alignment and auditing (see, e.g., (Accorsi & Stocker, 2012; Hosseinpour & Jans, 2016)): It allows one to check whether business processes are executed within certain boundaries set by managers, governments, laws, national and international regulations and standards, etc. Conformance checking starts from an event log, which records the actual process executions, and a model that encodes the boundaries and the constraints mentioned above. The practical relevance of conformance checking is demonstrated through several successful case studies in multiple domains, of which a partial list is reported in Section 6.1.

Several notions were introduced for conformance checking, such as token-based replay and comparison of footprints (see, e.g., (van der Aalst, 2016)) but they were unable to exactly pinpoint the deviations causing nonconformity. To overtake this limitation, the notion *alignment* has been introduced (van der Aalst et al., 2012). 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*) cannot 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*. 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*.

Alignments are powerful artifacts to detect nonconformity between the observed behavior as recorded in the event log and the prescribed behavior as represented by process models. 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 represented by the process model. As a matter of fact, the moves in log/model indicate where the execution is not conforming by pinpointing the deviations that have caused this nonconformity. Pinpointing the actual reasons of nonconformity is crucial for, e.g., auditors.

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 et al., 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, in fact, one of the alignments with the least expensive deviations (one of the so-called *optimal alignments*), according to some function assigning costs to deviations.

The existing techniques to compute optimal alignments (Adriansyah et al., 2011, 2013a) provide ad-hoc implementations of the A* algorithm based on an ad-hoc heuristics. This paper starts from the belief that re-implementing standard planning techniques for solving specific planning problems in an ad-hoc way is not ideal. The main drawback is that it is not possible to seamlessly plug in new planning algorithms and evaluate the performance levels. As a consequence, the possibility of evaluating different alternatives is hampered; also, if the literature proposes new algorithms that clearly overtake the existing ones for solving instances of the alignment problem, a massive amount of work is needed to modify the implementation.

Hence, in order to facilitate the integration of different planning algorithms, this paper illustrates how the problem of computing optimal alignments can be formulated as a planning problem in PDDL (Planning Domain Definition Language) (McDermott et al., 1998), which can be solved through off-the-shelf automated planners. PDDL is the standard encoding language for planning tasks. It allows one to explicitly represent states of the world and actions through a *planning domain*, and to instantiate such a domain with concrete objects, an initial state and a goal specification (*planning problem*).

In a nutshell, given a process model and a real process execution recorded in an event log, this paper shows 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 that can be solved by planning systems in a finite amount of time.

An evaluation was performed on real-life process models and event logs and on synthetic models and logs of increasing sizes to analyze its scalability. The evaluation results show that, when the process models are larger (more than 100 activities), it is significantly faster than the existing techniques reported in (Adriansyah et al., 2011, 2013a). The latter are the only techniques that compute alignments with guarantee of optimality. Even more remarkably, when process models are extremely large and event-log traces very long, the existing techniques were unable to compute alignments because they were unable to operate with 16 Gb of RAM (they went out of memory after several hours of computation), while our plan-based approach was able to properly complete the alignment task.

The need of approach that scales better is also advocated by the IEEE Task Force in Process Mining (Van Der Aalst et al., 2011): "*In some domains, mind-boggling quantities of events are recorded. [...] Therefore, additional efforts are needed to improve performance and scalability.*". While experiments were largely conducted through FAST-DOWNWARD because of being the best performing, the paper also reports on experience with other planners. The intention is also to showcase how the approach is versatile and allows one to plug in new planners at basically no cost.

The rest of the paper is organized as follows. In Section 2 we provide the relevant background necessary to understand the paper. In Section 3 we illustrate our approach to convert alignment problems to planning problems, and we provide correctness results for such problems. In Section 4 we describe the architecture of the software tool implementing our planning-based alignment approach. Then, Section 5 reports on experiment results, while in Section 6 we discuss related works. Finally, Section 7 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 event logs and process models represented as Petri nets. In Section 2.3 we give an overview on automated planning.

2.1. Petri Nets

Many notations have been introduced to represent business processes, such as BPMN, EPC, YAWL or UML Activity Diagrams (Dumas et al., 2013), and some of those are characterized by an ambiguous semantics. Since we need a simple language with clear semantics to explain our technique, we opted for Petri nets. Despite of its simplicity, it has been proven to be sufficiently adequate to model crucial aspects of business processes (van der Aalst, 1998) and to check their conformance against real executions recorded in event logs (see, e.g., the case studies reported in (van der Aalst, 2016)). The practical suitability of Petri nets for conformance checking is also showcased through two real-life case studies in Section 5.1.

It is undoubtable that there are business-process aspects that cannot be modeled through Petri nets, including the objects manipulated by activities, the activity guards or constraints on activities to only be executed by resources belonging to certain organizational units. They can certainly be modeled by richer languages, e.g. BPMN, but this addition aspects requires an increase of the language expressiveness to an extent that makes the alignment computation an undecidable problem (cf. Section 7). However, our approach can easily be applied to any other process modeling language with the same expressiveness or, for more expressive languages, if the constructs providing more expressiveness are not employed.

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

Definition 1 (Petri Net). *A Petri net is a tuple* (P, T, F) *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:

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

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* $(P, T, F, A, \ell, m_i, m_f)$ *where:*

- (P, T, F) *is a Petri net;*

- A *is the set of activity labels;*

- $\ell : T \not\rightarrow A$ *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 $N = (P, T, F, A, \ell, m_i, m_f)$, 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.

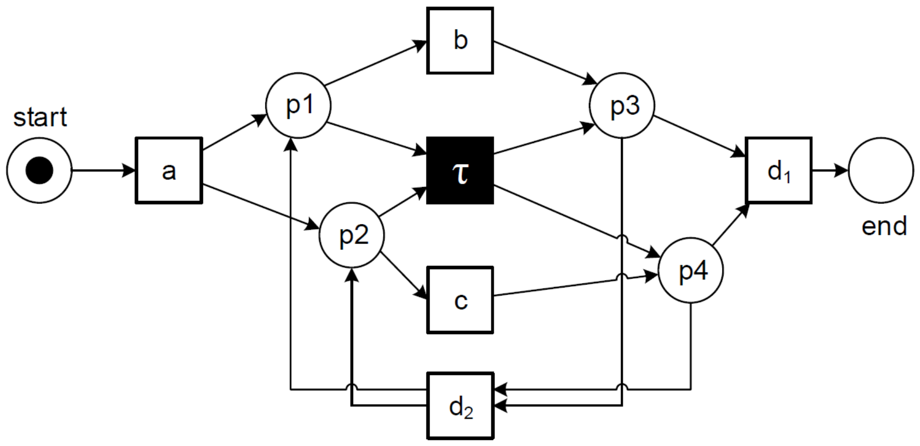


Figure 1: The Petri net used as working example.

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 markings with respectively one token in place *start* or in place *end* are the initial and final marking (and no tokens in any other place). 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.

In the remainder of this paper, we assume Labelled Petri nets to be 1-bounded, also known as safe. A Petri net is 1-bounded (or safe) if in any reachable marking from the initial marking m_i, no place ever contains more than 1 token. The following is the general definition of k-boundness (van der Aalst, 1998; van der Aalst & van Hee, 2002):

Definition 3 (k-boundedness)**.** *A Labelled Petri net $N = (P, T, F, A, \ell, m_i, m_f)$ is **k-bounded** iff, for each marking m s.t. $m_0 \xrightarrow{\sigma} m$ for some $\sigma \in T^*$, $m(p) \leq k$ for all $p \in P$.*

One-boundness is not a large limitation as the behavior allowed by most of business processes can be represented as 1-bounded Petri nets (see (Kiepuszewski et al., 2003; van der Aalst & van Hee, 2002)). In particular, every k-bounded Petri net for some $k \in \mathbb{N}$ can be transformed in 1-bounded Petri net: first, the reachability graph, which is finite, is built; second, a 1-bounded Petri net is built by transforming each reachability-graph state in a place and each reachability-graph transition in a Petri-net transition; finally, Petri-net arcs are added to connect the places to transition as guided by the reachability graph. In general, 1-bounded Petri nets can represent any BPMN model that only uses any of the following constructions: activities, sequence flows, start events, end events and exclusive, inclusive and parallel gateways (Kalenkova et al., 2015).

As a further indication of the practical relevance of k-bounded Petri nets, readers can refer to Section 6.1 where a partial list of successful cases of the conformance-checking applications is discussed. All these cases have in common that the processes can be represented as k-bounded Petri nets.

2.2. Alignment between Event Logs and Petri Nets

Event logs are the starting point for process mining. An event log is a multi-set of *traces*. A trace describes the life-cycle of a *process instance* in terms of the *activities* executed.

Definition 4 (Event Log)**.** *Let $N = (P, T, F, A, \ell, m_i, m_f)$ be a Labelled Petri net. Let $\mathcal{E}$ be the universe of events. A trace $\sigma_L \in \mathcal{E}^*$ is a finite sequence of events. An event log L over N is a pair (L, λ_N) consisting of a multi-set $L \in \mathbb{B}(\mathcal{E}^*)$*[1] *of traces and a function $\lambda_N : \mathcal{E} \rightarrow A$, which associates each event e with an activity $\lambda_N(e)$.*

Multiple instances of a process may consist of the same sequence of transition firings and, hence, result in the same trace. This motivates the definition of an event log as a multi-set. The following $[\langle a, b, c, d\rangle, \langle a, b, c, d\rangle, \langle a, b, c, d, b, c, d\rangle, \langle a, d\rangle, \langle a, d\rangle]$ is an example of event log, assuming an event universe $\mathcal{E} = \{a, b, c, d\}$ and, for each event $e \in \mathcal{E}$, $\lambda_N(e) = e$ Transition firings in an event log are known as *events.* The concept of time is not explicitly modeled but, without losing generality, we assume that events in a trace are sorted according to the timestamp of their occurrence. As mentioned in Section 1, we perform conformance checking by constructing an *alignment* of event log (L, λ_N) and process model N (van der Aalst et al., 2012), which allows us to exactly pinpoint where deviations occur. On this aim, the events in the event log need to be related to transitions in the model, and vice versa. Building this alignment is far from trivial, since the log may deviate from the model at an arbitrary number of places.

We need to relate "moves" in the log to "moves" in the model in order to establish an alignment between a process model and an event log. 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 $\gg$.

Definition 5 (Legal Alignment Moves)**.** *Let $N = (P, T, F, A, \ell, m_i, m_f)$ be a Labelled Petri net. Let (L, λ_N) be an event log. A* legal alignment move *for N and (L, λ_N) 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 $s_M \in T$,*

[1]Given a set X, $\mathbb{B}(X)$ is the set of all multisets constituted by elements of X.

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

$$\gamma_3 = \begin{array}{|c|c|c|c|c|c|} a & \gg & d & b & c & \gg \\ \hline a & \tau & d_2 & b & c & d_1 \end{array}$$

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

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

An alignment is a sequence of alignment moves:

Definition 6 (Alignment)**.** *Let $N = (P, T, F, A, \ell, m_i, m_f)$ be a Labelled Petri net and (L, λ_N) be a event log. Let Γ_N be the universe of all legal alignment moves for N and (L, λ_N). Let $\sigma_L \in L$ be a log trace. Sequence $\gamma \in \Gamma_N^*$ is an* alignment *of N and σ_L if, ignoring all occurrences of $\gg$, the projection on the first element yields σ_L and the projection on the second 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 $\sigma_1 = \langle a, d, b, c\rangle$.[2] Note how moves are represented vertically. For example, as shown in Fig. 2, the first move of γ_1 is (a, a), i.e., a synchronous move of a, while the the second and fifth move of γ_1 are a move in log and model, respectively.

However, we aim at finding a complete alignment of σ_L 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. The alignment with the lowest cost is called an *optimal alignment*.

Definition 7 (Cost Function)**.** *Let $N = (P, T, F, A, \ell, m_i, m_f)$ be a Labelled Petri net and $\sigma_L \in A^*$ 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 σ_L 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 σ_L, $\mathcal{K}(\gamma) \leq \mathcal{K}(\gamma')$. The definition of the cost function κ is domain dependent and is manually defined by process analysts/auditors according to the knowledge of the domain and of the severity of the different sorts of nonconformity. For instance, in a loan process, it is significantly more severe to approve a loan when not supposed (i.e., a move in log for a certain activity *Approve Loan*) than to send a letter and ask for additional information when unnecessary (i.e., a move

[2]For this example, for the sake of simplicity, we again assume that $\mathcal{E} = \{a, b, c, d\}$ and, for each event $e \in \mathcal{E}$, $\lambda_N(e) = e$.

in log for a certain *Ask for additional information*) or to forget to send it when expected (i.e., a move in model for *Ask for additional information*). In other words, the cost of each legal move depends on the specific model and process domain and, hence, the cost function needs to be defined specifically for each setting and cannot be automated. While approaches could be researched to support stakeholders with the cost-function definition, this would be beyond the scope of this paper. Also note that an optimal alignment does not need to be unique, i.e., multiple alignments with the same minimal cost may exist.

In case that a customized cost function cannot be provided, one can use a *standard cost function*, which assigns cost 1 to every move in log or move in model for visible transitions and cost 0 to synchronous moves and moves in model for invisible transitions. For instance, the standard cost function for the model in Fig. 1:

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

Let us consider the alignments in Fig. 2. Using the standard cost function in Equation 2, alignments γ_1 and γ_2 have cost 2 since they have two moves in model and/or log and, also, moves for τ have cost 0. Conversely, γ_3 takes on a cost 1. So, γ_3 is a less expensive alignment. Since no alignment exists with cost 0, γ_3 is among the least expensive alignments. Therefore γ_3 is an optimal alignment.

2.3. Automated Planning

The *Automated planning* field is a branch of Artificial Intelligence (AI) that aims to the realization of automated systems for the synthesis of organized sequences of real-world activities. Automated planning operates on explicit representations of states and actions (Bonet & Geffner, 2001; Ghallab et al., 2004). The Planning Domain Definition Language (PDDL) (McDermott et al., 1998) is a de-facto standard 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 (a state is characterized by the set of propositions that are true) and a set of *actions* Ω that can be executed in the domain. An *action schema* $a \in \Omega$ is of the form $a = \langle Par_a, Pre_a, Eff_a \rangle$, 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 fluents. In the remainder of the paper, we remain consistent with PDDL terminology (McDermott et al., 1998; Fox & Long, 2003): a *predicate* is a boolean property of the world and *fluents* are used to express numeric properties, such as the actions' cost. Both the values of predicates and fluents can change as result of the execution of actions.

PDDL includes also the ability of *typing* the parameters that appear in actions and *constraining* the types of arguments to predicates and fluents.

There exist several forms of planning in the AI 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). Concretely, this implies that: *(i)* any planning action only provides deterministic and observable effects; *(ii)* a complete knowledge of the initial state I is available.

Example *Consider the PDDL planning domain PD_{ex} and its associated planning problem PR_{ex}:*

```
(define (domain PDex)
  (:types  simple complex - object)
  (:predicates
    (pred1 ?x - simple ?y - complex) (pred2) (pred3) (pred4))
  (:functions
    (single-cost ?x - simple) (total-cost))
  (:action act1
     :parameters (?k1 - simple ?k2 - complex)
     :precondition (and (pred1 ?k1 ?k2) (pred3))
     :effect (and (not(pred4)) (pred2)
                  (increase (total-cost) (single-cost ?k1))))
  (:action act2
     :precondition (pred4)
     :effect (pred3))
)

(define (problem PRex) (:domain PDex)
  (:objects obj1 obj2 - simple obj3 - complex)
  (:init (pred1 obj1 obj3) (pred4)
         (= (single-cost obj1) 1) (= (single-cost obj2) 2)
         (= (total-cost) 0))
  (:goal (and (pred2) (pred3) (not (pred4))
         (= (total-cost) 1))
)
```

The example planning domain PD_{ex} consists of a unique abstract type `object` *capturing any possible object instance involved in some predicate or fluent, and two object sub-types named respectively* `simple` *and* `complex` *objects. Each proposition (e.g., the predicate* `pred1` *and the fluent* `single-cost`*) may be defined over variables of a certain type. Variables are distinguished by a "*`?`*" character at front, and the dash "*`-`*" is used to assign types to the variables. In the example above,* `?x` *is defined to be of type* `simple` *and* `?y` *to be a* `complex` *object. Planning actions may involve some input parameters. For example, action* `act1` *has two parameters defined in terms of variables* `?k1` *of type* `simple` *and* `?k2` *of type* `complex`*. The action schema of* `act1` *states that - for being executed - the predicates* `pred1` *(properly grounded on two object instances of kind* `simple` *and* `complex`*, resp.) and* `pred3` *must hold.*

Then, it states that a successful execution of `act1` *guarantees, on the one hand, that* `pred2` *and the negation of* `pred4` *will hold together, and on the other hand that the fluent* `total-cost` *is increased of a value equal to the action cost* `(single-cost ?k1)`*. In the PDDL planning problem PR_{ex}, which has been defined over PD_{ex}, two object instances of type* `simple` *and one of type* `complex` *are defined. In the initial state I only the predicates* `(pred1 obj1 obj3)` *and* `pred4` *are known to be true, while the other predicates are set to $false$. Furthermore, numeric fluents are initialized to their respective initial values. The goal condition G is a formula where the conjunction of* `pred2`, `pred3` *and the negation of* `pred4` *is true and the value of the* `total-cost` *fluent is equal to 1.* □

Under the assumptions of a "perfect world description", a solution to a planning problem is a sequence of actions—*a plan*—whose execution brings from initial state I to some state that satisfy goal G. The plan is said to be *optimal* if it minimizes the sum of action costs.

Example *A simple plan satisfying PR_{ex} consists in, first, executing action* `act2` *(possible because* `pred4` *is true in I), which makes* `pred3` *true. Then, the plan continues executing* `act1`*, where object instances* `obj1` *and* `obj3` *are properly grounded (i.e., instantiated). This turns the value of* `pred2` *to true and that of* `pred4` *to $false$. With these actions, the value of* `total-cost` *is increased of the value of* `(single-cost obj1)`*, i.e. 1. Notice that this plan is optimal.* □

Automated planning has made huge advances in the last twenty years, leading to solvers able to create plans with thousands of actions for problems described by hundreds of propositions. In this work, we represent planning domains and problems making use of the STRIPS fragment of PDDL 2.1 (Fox & Long, 2003), enhanced with the numeric features provided by the "level 2" of the same language. Such features 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 (L, λ_N) and Labelled Petri net $N = (P, T, F, A, \ell, m_i, m_f)$, the problem of the alignment of N and $\sigma_L = \langle e_1, \ldots, e_n \rangle \in L$ 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 σ_L and N.

3.1. Predicates and fluents

In the planning domain $\mathcal{P_D}$, we provide three abstract 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 σ_L that must be checked for conformance. For trace $\sigma_1 = \langle a, d, b, c \rangle$ 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
          e1 e2 e3 e4 evEND - event)
```

In addition to the four events, denoted as $e1, ..., e4$, we introduce the invisible transition *inv* and a fictitious $evEND$, which identifies the end of trace. The introduction of this fiction $evEND$ is not specific of this example, but it is general for any instance of the alignment problem.

To capture all possible markings of N and the evolution of σ_L during an alignment, we define four 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.

succ. For each $1 \leq n < |\sigma_L|$, `(succ ?e1 - event ?e2 - event)` holds if $e1 = e_i$ and $e2 = e_{i+1}$.

tracePointer. For each event $e \in \sigma_L$, `(tracePointer ?e - event)` holds when, during the computation of the alignment, e is the next trace event to align.

associated For each event $e \in \sigma_L$, `(associated ?e - event ?t - transition)` holds when $\ell(t) = \lambda_N(e)$.

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 $e \in \sigma_L$, 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 costs and, hence, no fluent needs to be introduced. For trace $\sigma_1 = \langle a, d, b, c \rangle$ 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) (tracePointer e1)
       (succ e1 e2) (succ e2 e3)
       (succ e3 e4) (succ e4 evEND)
       (associated e1 a) (associated e2 d1)
       (associated e2 d2) (associated e3 c)
       (associated e4 b) (= (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 e1) 1)
        (= (log-move-cost e2) 1)
        (= (log-move-cost e3) 1)
        (= (log-move-cost e4) 1))
```

Readers can observe that the last ten 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 the 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 the trace pointer has reached `evEND`:

```
(:goal (and (tracePointer evEND) (token end)
            (not (token p1)) (not (token p2))
            (not (token p3)) (not (token p4))
            (not (token start))))
```

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 transition $t \in T \setminus \mathsf{Inv}(N)$ to represent a synchronous move for t. For instance, let us consider transition a of the model in Fig. 1; synchronous move for a is associated with the following action:

```
(:action moveSync-a
 :parameters (?e1 - event ?e2 - event)
 :precondition (and (token start)
                    (tracePointer ?e1)
                    (associated ?e1 a)
                    (succ ?e1 ?e2))
 :effect (and (not (token start))
              (token p1) (token p2)
              (not (tracePointer ?e1))
              (tracePointer ?e2)))
```

The preconditions of the action are *(i)* that a is enabled (as defined in Section 2.1): each place $p \in {}^{\bullet}a$ contains a token; *(ii)* $e1$ is the actual event of σ_L under analysis; *(iii)* $e1$ corresponds to the execution of a in σ_L; *(iv)* $e1$ is succeeded by the event $e2$ in σ_L. The effect is such that the trace pointer moves from $e1$ to $e2$ and that the marking changes according to the firing rules in Equation 1 of Section 2.1: The token in place *start* is consumed and one token is produced in places *p1* and *p2*.

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 σ_L. For instance, let us consider transition a of the model in Fig. 1; a model move for a is associated with the following action:

```
(:action moveInTheModel-a
 :precondition (token start)
 :effect (and (not (token start))
              (token p1) (token p2)
              (increase (total-cost)
                        (move-model-cost a))))
```

The effects are accordant to the firing rules in Equation 1: The token in place *start* is consumed and one token is produced in places *p1* and *p2*. The difference with synchronous moves is that the value of any predicate `tracePointer` does not change. It is worthy observing how the execution of a model move for a makes total cost (of the alignment) increases of a value equal to `(move-model-cost a)`.

Log moves. All log moves can be represented by a single generic action, which is independent of N and σ_L:

```
(:action moveInTheLog
 :parameters (?e1 - event ?e2 - event)
 :precondition (and (tracePointer ?e1)
                    (succ ?e1 ?e2))
 :effect (and (not (tracePointer ?e1))
              (tracePointer ?e2)
              (increase (total-cost)
                        (log-move-cost ?e1))))
```

A log move for any event $e_i \in \sigma_L$ is represented as `(moveInTheLog ?e1 ?e2)` where $e1 = e_i$ and $e2 = e_{i+1}$. The preconditions are that $e2$ follows $e1$, and that $e1$ is the actual event under analysis. The effect is to move the trace pointer from $e1$ to $e2$ and to increase the total cost of the alignment of a value equal to `(log-move-cost ?e1)`.

3.3. Discussion on Correctness and Termination

The proposed encoding of an alignment problem $\mathcal{A}$ as planning problem $\mathcal{P}$ is such that one can find a solution for $\mathcal{A}$ by solving $\mathcal{P}$. Also, planning algorithms always terminate when the input is a planning problem $\mathcal{P}$ that encodes an alignment problem according to the procedure proposed in this paper.

Proposition 8 (Correctness)**.** *Let $N = (P, T, F, A, \ell, m_i, m_f)$ be any 1-bounded Labelled Petri net, let (L, λ_N) be an event log and let $\sigma_L \in L$ be any trace. Let $\mathcal{A}$ be the problem of building an alignment of σ_L and N. If there exists a sequence $\sigma_N \in T^*$ s.t. $m_i \xrightarrow{\sigma_N} m_f$, our encoding procedure for $\mathcal{A}$ generates a planning problem that has a solution.*

In a nutshell, if such a sequence σ_N exists, it is always possible to build an alignment that consists of moves in log for all events $e \in \sigma_L$ followed by moves in model for all transitions $t \in \sigma_N$ (see Definition 6). Therefore, there exists a solution to $\mathcal{P}$ that consists of a finite sequence of steps, each of which mapping one alignment move.

Proposition 9 (Termination)**.** *Let $N = (P, T, F, A, \ell, m_i, m_f)$ be any 1-bounded Labelled Petri net, let (L, λ_N) be an event log and let $\sigma_L \in L$ be any trace. Let $\mathcal{A}$ be the problem of building an alignment of σ_L and N. If there exists a sequence $\sigma_N \in T^*$ s.t. $m_i \xrightarrow{\sigma_N} m_f$, our encoding procedure for $\mathcal{A}$ generates a planning problem that can be solved by planning algorithms in a finite amount of time.*

Intuitively, our encoding procedure define object types in the planning domain that refer to a finite number of places and transitions of N and events of σ_L (cf. Definitions 2 and 4). Therefore, the number of planning problem object values is finite. Since predicates and actions specified in the planning domain provide a finite number of input parameters defined over the above object types, also their grounding over the object values is finite. Hence, the domain and the planning problem can be phrased as a propositional one; and the thesis follows from the decidability of propositional planning (Ghallab et al., 2004). As discussed in (Helmert, 2002; Haslum & Geffner, 2014), fluent `total-cost` does not affect termination: it only appears among the effects of actions and not in the preconditions of actions or in the planning goals. Termination is also not affected by the definition of the metric associated with fluent `total-cost`, since the plan's cost never decreases when new actions are added and actions have constant costs.

Last but not least, *if planning systems are used that guarantee the optimality of the solution, the solution is one of alignments with the lowest cost, which is an optimal alignment.*

4. Implementation

We have developed a planning-based alignment tool that implements the technique discussed in Section 3. The tool has been developed as a standard Java application, using the Java SE 7 Platform, and relies on two main architectural layers as shown in Fig. 3.

The *Design Layer* provides a GUI that assists the process designer in *(i)* the definition of a new event log from scratch (cf. Fig. 4(a)), *(ii)* the import of existing event logs and Petri nets stored in external repositories, *(iii)* the customization of the cost function and of the initial/final marking of the Petri net under analysis as well as *(iv)* the selection of a search heuristic within an available repertoire.[3] (cf. Fig. 4(b)) Notably, the tool supports the standard XES (eXtensible Event Stream (Günther & Verbeek, 2014)) and PNML (Petri Net

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

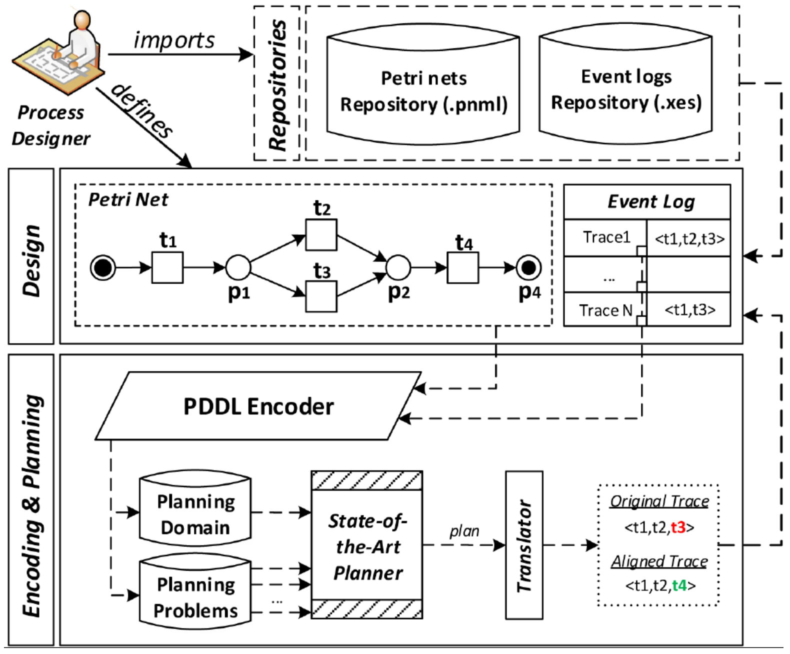


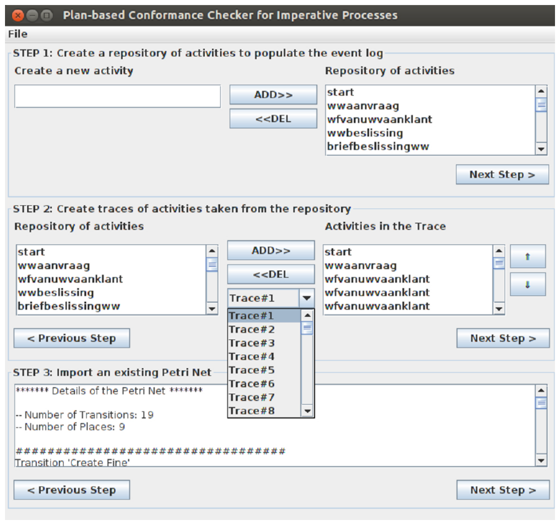
Figure 3: The architecture of the planning-based alignment tool.

Markup Language (Billington et al., 2003)) for respectively storing, exchanging, and analyzing event logs and Petri nets.

The *Encoding & Planning Layer* is in charge of translating the input specifications in PDDL format, which is readable by any state-of-the-art PDDL planner, and performing the alignment task. Specifically, starting from a Petri net and an event log to be aligned, the *PDDL Encoder* component builds a single PDDL planning domain file and as many PDDL planning problem files as are the traces stored in the event log. At this point, for any trace of the event log (i.e., for any of the generated planning problems) a state-of-the-art PDDL planner is invoked. With such inputs, the planner synthesizes a plan, i.e., a sequence of alignment moves, whose quality depends on the specific search heuristic selected.

Our tool is integrated with the FAST-DOWNWARD planning framework (Helmert, 2006) to find optimal alignments of event logs and process models. FAST-DOWNWARD is a progression planner that uses hierarchical decompositions of planning tasks for computing its heuristic function, called the *causal graph heuristic*, which approximates goal distances by solving a hierarchy of "local" planning problems. To produce optimal alignments, FAST-DOWNWARD uses a best-first search in first iteration to find a plan and a weighted A* search to iteratively decreasing weights of plans.

When a plan satisfying the goal is found, the *Translator* component makes it available to the process designer through the GUI. Furthermore, together with



STEP 1: Create a repository of activities to populate the event log

Create a new activity

Repository of activities

```
start
wwaanvraag
wfvanuwvaanklant
wwbeslissing
briefbeslissingww
```

STEP 2: Create traces of activities taken from the repository

Repository of activities

```
start
wwaanvraag
wfvanuwvaanklant
wwbeslissing
briefbeslissingww
```

Trace#1

```
Trace#1
Trace#2
Trace#3
Trace#4
Trace#5
Trace#6
Trace#7
Trace#8
```

Activities in the Trace

```
start
wwaanvraag
wfvanuwvaanklant
wfvanuwvaanklant
wfvanuwvaanklant
```

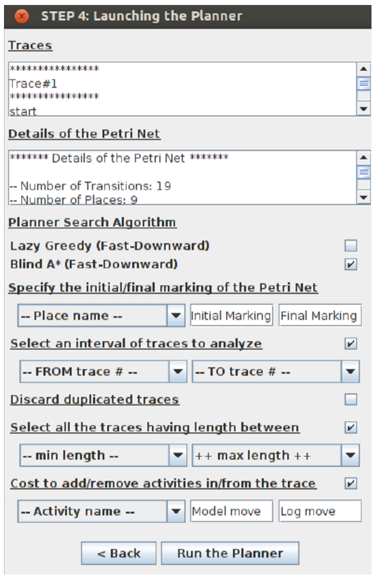
STEP 3: Import an existing Petri Net

```
******* Details of the Petri Net *******

-- Number of Transitions: 19
-- Number of Places: 9

##################################
Transition 'Create Fine'
```

(a)



Traces

```
****************
Trace#1
****************
start
```

Details of the Petri Net

```
******* Details of the Petri Net *******

-- Number of Transitions: 19
-- Number of Places: 9
```

Planner Search Algorithm

Lazy Greedy (Fast-Downward) ☐

Blind A* (Fast-Downward) ☑

Specify the initial/final marking of the Petri Net

-- Place name -- | Initial Marking | Final Marking

Select an interval of traces to analyze ☑

-- FROM trace # -- | -- TO trace # --

Discard duplicated traces ☐

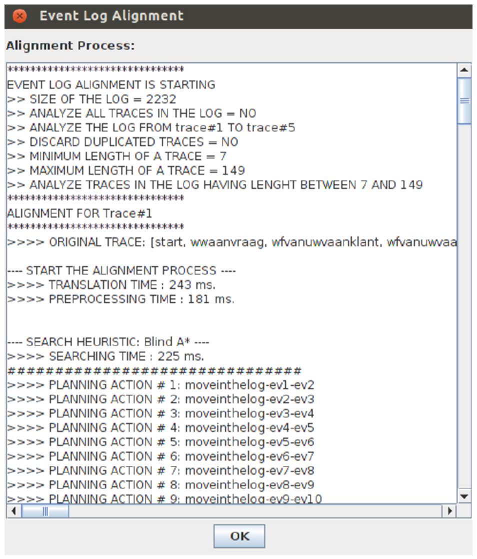
Select all the traces having length between ☑

-- min length -- | ++ max length ++

Cost to add/remove activities in/from the trace ☑

-- Activity name -- | Model move | Log move

(b)



Alignment Process:

```
******************************
EVENT LOG ALIGNMENT IS STARTING
>> SIZE OF THE LOG = 2232
>> ANALYZE ALL TRACES IN THE LOG = NO
>> ANALYZE THE LOG FROM trace#1 TO trace#5
>> DISCARD DUPLICATED TRACES = NO
>> MINIMUM LENGTH OF A TRACE = 7
>> MAXIMUM LENGTH OF A TRACE = 149
>> ANALYZE TRACES IN THE LOG HAVING LENGHT BETWEEN 7 AND 149
******************************
ALIGNMENT FOR Trace#1
******************************
>>>> ORIGINAL TRACE: [start, wwaanvraag, wfvanuwvaanklant, wfvanuwvaa

---- START THE ALIGNMENT PROCESS ----
>>>> TRANSLATION TIME : 243 ms.
>>>> PREPROCESSING TIME : 181 ms.


---- SEARCH HEURISTIC: Blind A* ----
>>>> SEARCHING TIME : 225 ms.
################################
>>>> PLANNING ACTION # 1: moveinthelog-ev1-ev2
>>>> PLANNING ACTION # 2: moveinthelog-ev2-ev3
>>>> PLANNING ACTION # 3: moveinthelog-ev3-ev4
>>>> PLANNING ACTION # 4: moveinthelog-ev4-ev5
>>>> PLANNING ACTION # 5: moveinthelog-ev5-ev6
>>>> PLANNING ACTION # 6: moveinthelog-ev6-ev7
>>>> PLANNING ACTION # 7: moveinthelog-ev7-ev8
>>>> PLANNING ACTION # 8: moveinthelog-ev8-ev9
>>>> PLANNING ACTION # 9: moveinthelog-ev9-ev10
```

(c)

Figure 4: Screenshots of the GUI provided by our planning-based alignment tool.

the alignment moves, when all the traces of a log have been analyzed, the tool produces several statistics to evaluate the quality of the alignments (e.g., the average cost of the alignment, the percentage of dirty traces in the log, etc.). Conversely, if the planner component is not able to find any plan for a specific trace (i.e., for a specific planning problem), this means that no alignment is possible for that trace.

Finally, we notice that the manual effort required to use our planning based alignment tool is comparable to the effort needed to use other established tools for conformance checking such as the implementation by Adriansyah's et al. (Adriansyah et al., 2011, 2013a). They all take as input *(i)* a Petri Net (formatted in PNML), *(ii)* an event log (formatted in XES) and *(iii)* an explicit specification of the cost function.

5. Validation

The approach has been positively assessed using both real-life and synthetic processes. For real-life processes, we employed a road-traffic management a financial-related process. For synthetic processes of different sizes, processes contained up to 236 activities, and we compared the results with those obtained by the existing approach by Adriansyah et al. (Adriansyah et al., 2011, 2013a). The results show that the approach has a significantly better scalability than the Adriansyah's approach. The latter is outperformed by our planning-based approach when the model becomes larger. For the two larger processes, the existing approach by Adriansyah et al. was unable to compute the alignment with the our experiment machine. In particular, we performed our experiments with a machine with an Intel Core i7-4770S CPU 3.10GHz Quad Core and 16GB RAM. The FAST-DOWNWARD planning framework was set to use the Blind A* search strategy, which guarantees to produce optimal solutions. Notice that it is formally proven (Geffner & Bonet, 2013) that the A* algorithm is *optimal* when, for all states $s \in S$ of the search space, the algorithm uses a heuristics $h(s)$ that is *admissible* (i.e., $h(s)$ never overestimates the cost of reaching the goal) and *safe* (i.e., if $h(s)$ declares s to be a dead-end, then s is really a dead-end). Since the Blind heuristics used in the FAST-DOWNWARD planning framework is proven to be admissible and safe (see, e.g., `http://www.fast-downward.org/Doc/Heuristic`), the Blind A* search strategy employed always find solutions guaranteed to be optimal.

In the experiments, we use a cost function that assigns cost 1 to log moves and model moves for non-invisible transition and cost 0 to any other legal alignment move (similar to Equation 2). Readers should observe that the approaches by Adriansyah and et. exploit the fact that, if multiple process instances are executed in the same way, they are recorded in the event log as multiple instances of the same trace and, hence, the alignment can be computed once for all occurrences. Therefore, those approaches do not recompute the alignment multiple times for the same traces, thus speeding up the entire process. In order to make the comparison of the two approaches more fair, we also employ the same speed improvement.

In order to assess the scalability with respect to the "complexity" of the event logs, we artificially injected noise into the event logs of both the real-life and synthetic processes. The injection of $X\%$ of noise in a log corresponds to injecting $X\%$ of noise in each trace of the log: an event is swapped with the next

Algorithm 1: Injection of $X\%$ of noise in a trace

```
**Data:** A trace $\sigma_L = \in \mathcal{E}^*$ and a probability $\overline{X} \in (0, 100)$
**Result:** A new trace $\sigma'_L \in \mathcal{E}^*$
$\sigma'_L \leftarrow \sigma_L$;
**for** $i \leftarrow 1$ **to** $|\sigma'_L|$ **do**
    $X \leftarrow ExtractNumber(0, 100)$;
    **if** $X \leq \overline{X}$ **then**
        $SwapEventsInPositions(i, i + 1)$;
```

one with a probability of $X\%$ (see Algorithm 1). Note that an event initially at position i can be swapped with event at position $i + 1$ and, later, can be swapped again with the event at position $i + 2$, etc., thus moving further apart from its initial position (with decreasing probability). Readers may wonder why we opted for swapping events rather than adding artificial or remove existing events. The reason is related to the fact that swapping events is the hardest type of noise when building an alignment; for instance, supposing that a followed by b is allowed, if the trace contains b followed by a, then the alignment would consists in a model move for a, which is followed by a synchronous move for b and a log move for a.

Section 5.1 reports on the experiments conducted on two real-life case studies, where Section 5.2 illustrates the results of the synthetic processes of larger sizes. The analysis on the synthetic processes has a twofold purpose. On the one hand, it shows how our approach scales with models of increasing sizes. On the other hand, it provides a comparison of our approach with the approach by Adriansyah et al. (Adriansyah et al., 2011, 2013a). Section 5.3 reports on the experience of using planners different from Fast-downward to showcase the versatility and easiness of plugging in different planners at almost no cost.

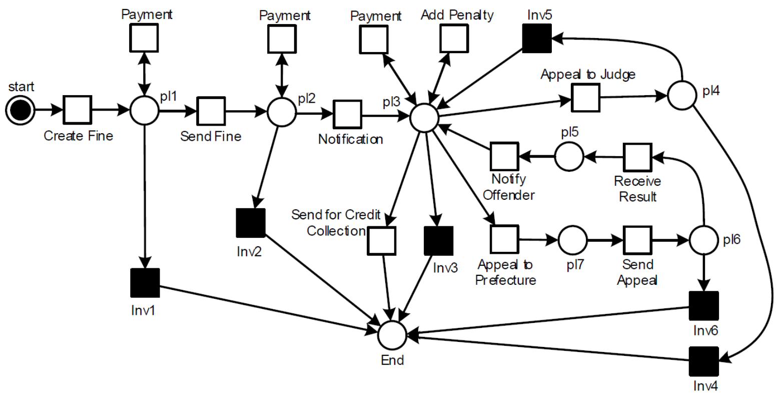
5.1. Evaluation on the Real-life Case Studies

The evaluation on real-life case studies is based on the process enacted in an Italian local police for the management of road-traffic violations and on the process enacted by a Dutch financial institute.

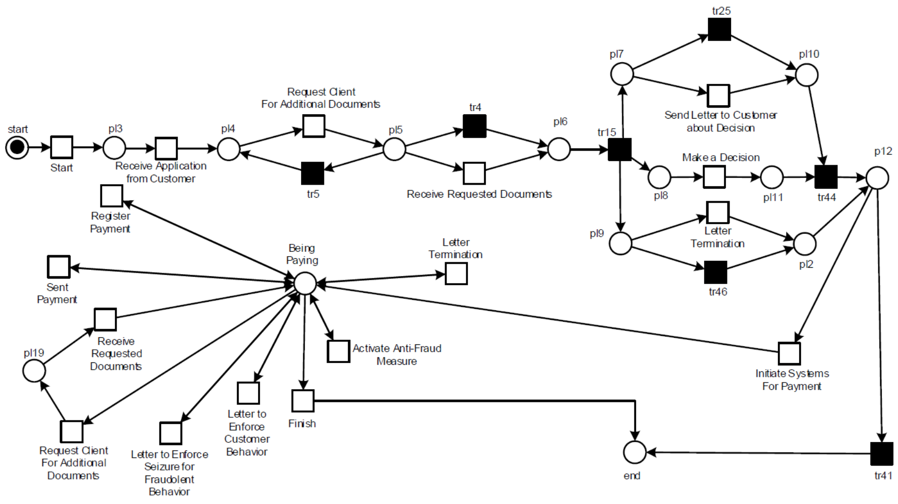
5.1.1. Description of the processes

The road-traffic violations are managed by the Italian local police through an ad-hoc information system that was explicitly built for this purpose by an external company. The functioning of this information system is not supported by process models, nor does the system documentation provide any textual description of how the process is actually managed through the system.

Therefore, we had several interviews with the stakeholders involved in the use of the systems. During these interviews, we asked them how the system manages the violations in the different steps. During the interview process, we also became acquainted with the Italian laws that constrain the process of



(a) The Petri net describing the road traffic violations process model.



start
Start
pl3
Receive Application from Customer
pl4
Request Client For Additional Documents
tr5
pl5
tr4
Receive Requested Documents
pl6
tr15
pl7
tr25
pl10
Send Letter to Customer about Decision
pl8
Make a Decision
pl11
tr44
p12
pl9
Letter Termination
pl2
tr46
Register Payment
Being Paying
Letter Termination
Sent Payment
Receive Requested Documents
pl19
Activate Anti-Fraud Measure
Initiate Systems For Payment
Request Client For Additional Documents
Letter to Enforce Seizure for Fraudolent Behavior
Letter to Enforce Customer Behavior
Finish
end
tr41

(b) The Petri net describing the model of the process at the Dutch financial institute.

Figure 5: The Petri nets that model the processes used as real-life case studies to validate our approach.

managing of these violations. It should be noticed that the present constraints still leave space to the different local polices to slightly vary how the process is actually management in the details. Fig. 5(a) depicts the model that illustrates how the road-traffic violations should be managed, which was finally validated with the local-police stakeholders.

A new process instance starts by firing the *Create Fine* transition. In general, the offender can pay the fine (partly or fully) at many moments in time: Right after the creation, after a road fine notification is sent by the police to the offender's place of residence, or when such a notification is received by the offender herself. If the entire amount is paid (or, even, by mistake, more than that amount), the fine management is closed. This motivates the presence of the invisible transitions *Inv1*, *Inv2* and *Inv3*. If a notification is sent, the offender needs to also pay the postal expenses. If the offender does not pay within 180 days, a penalty is added, usually as much as the fines amount. After being notified by post, the offender can appeal against the fine through a judge and/or the prefecture. If the appeal is successful, the process instance ends (by firing either *Inv4* or *Inv6*). Otherwise, the case continues by firing *Inv5* or *Receive Result*. If the offender does not pay, eventually the fine ends by handing over the case for credit collection. Interested readers can refer to (Mannhardt et al., 2015) for a detailed discussion of the domain and the process model. An event log that records the execution of more than 200000 process instances is publicly available (de Leoni & Mannhardt, 2015), out of which we randomly extracted 30000 traces for the experiments.

For the case study with the Dutch financial institute, we followed a similar process as for the Italian local police to draw the process model. We interviewed two company stakeholders, specifically one manager and one business analyst, to understand their process and we were introduced to the Dutch laws on this topic. The result was the model in Fig. 5(b), which was ultimately validated with the same individuals. A new instance of this process is created when a customer applies for a monetary benefit for their financial product. The initial marking is a marking with one token in place *Start* and no tokens in any other place; the final marking is one token in place *End* and no tokens in any other place. The process starts with customer applying for the benefits (transition *Receive Application from the Customer*). In order to assess the eligibility of the applicant, a number of letters can be asked to the customers depending on the situation (see the transitions *Request Client For Additional Documents* and *Receive Requested Documents*). After that, a decision is made about the eligibility for the credit and the customer is informed (see transition *Make a Decision* and *Send Letter to Customer about Decision*). If the decision is negative, invisible transition *tr41* is performed and the process instance terminates. If the decision is positive, transition *Initiate Systems For Payments* is eventually executed. This transition is then followed by a number of transitions that are executed during the provision of the credit, such as sending the payment installments (see transition *Send Payment*). During the phase, the institute may perform a number of fraud checks to ensure that the information provided by the applications is truthful. This may require the institute to ask the customer for further information. If a fraud is detected, anti-fraud measures can be activated and letters can be sent to the customer to claim a number of payments back. Eventually, when the credit provision ends, transition *Finish* is executed and the process instance terminates. For this case study, we were provided with an event log composed of 2232 traces, each of which refers to the execution to deal with a different applicant.

trace length	number of traces	search time (ms.)	total time (ms.)	alignment cost	search time (ms.)	total time (ms.)	alignment cost	search time (ms.)	total time (ms.)	alignment cost	search time (ms.)	total time (ms.)	alignment cost
Financial Institute		original log			10% noise			20% noise			30% noise		
1-25	798	15.63	66.32	1.25	15.7	66.48	1.86	15.8	67.35	1.97	15.9	67.54	2.17
26-50	1,166	16.31	76.7	2.48	16.39	78.33	3.91	16.4	78.85	4.39	16.72	79.39	4.87
51-75	212	17.46	99.59	3.53	17.51	105.54	4.75	17.92	106.72	5.71	18.13	108.42	6.54
76-100	44	19.06	122.27	4.18	20.43	131.41	5.18	20.57	132.52	6.93	20.66	135.04	8.41
101-125	10	20.2	149.3	5.67	21.6	162.3	7.21	21.9	164.2	9.12	22.1	164.8	9.87
126-150	1	22	178	7	22.2	201	9	22.4	207	11	23	208	12
Traffic Violations		original log			10% noise			20% noise			30% noise		
1-5	27,200	14.63	53.94	0.24	14.65	54.12	0.28	14.72	54.15	0.34	14.73	54.27	0.46
6-10	2,792	14.71	55.93	0.25	14.71	55.94	0.31	14.98	55.98	0.67	14.99	56.13	0.94
11-15	7	14.86	64.57	0.27	14.88	64.85	0.33	15.28	65.14	1	15.43	65.57	1.27
16-20	1	15.25	79.12	2	15.74	79.71	2	16.43	81.07	2	17.14	82.98	2

Table 1: Experimental results for both real-life case studies. Numbers reported are averages over all traces within a certain trace cluster.

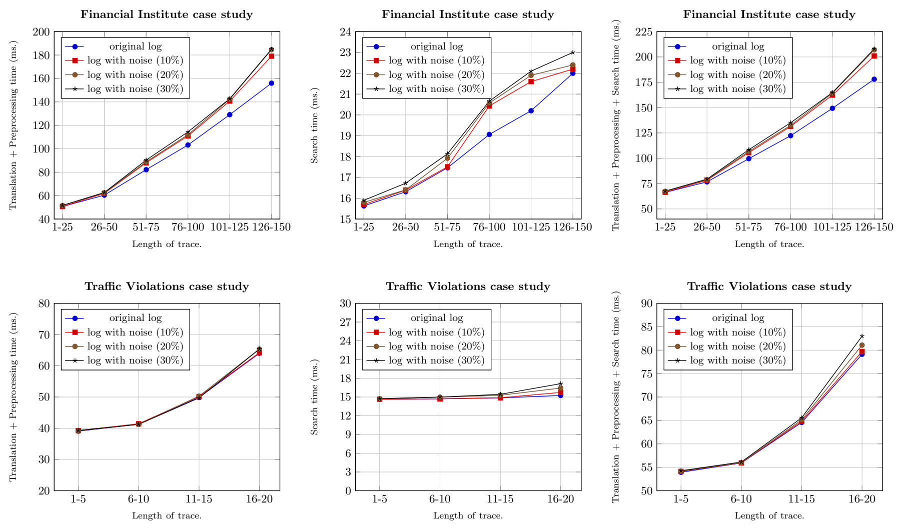


Figure 6: Performance of computing alignments through planning for the two real-life case studies investigated in this paper. The time is the average per trace for each category. The upper figures refer to the *financial institute* case study, while the lower figure refer to the *traffic violations* case study. The figures on left-hand side and center refer to, respectively, to the average translation plus pre-processing time and to the average plan search time. The figures on right refer to the total time, which is the sum of the graphs of the left-hand side and center graphs.

5.1.2. Results of the Experiments and Discussion

Table 1 and Fig. 6 show the results of our experiments. The traces are split in clusters according to their length. The values for "original log" refer to the clusters before injecting noise. By showing the results separately for each cluster and different values of additional noise, we can evaluate how the performance scales up with longer and/or more noisy traces. The upper charts of Fig. 6 are about the financial institute and the lower are about the Italian local police. The each case study, left-hand side center charts respectively show the average time for translating and pre-processing and for an optimal plan searching; the

PN Length	search time (ms.)	total time (ms.)	search time (ms.)	total time (ms.)	alignment cost	search time (ms.)	total time (ms.)	alignment cost	search time (ms.)	total time (ms.)	alignment cost
	original log		10% noise			20% noise			30% noise		
25	17.76	82.33	20.39	84.79	3.08	22.41	86.21	4.24	24.62	88.87	5.32
36	18.91	94.98	19.36	94.13	4.95	19.55	92.78	8.03	19.71	92.67	10.44
68	21.73	126.45	26.89	130.68	3.07	31.23	134.57	4.38	40.71	144.99	6.53
95	21.2	120.94	23.67	123.65	4.43	25.58	125.78	7.1	27.23	126.82	9.25
115	24.43	157.71	71.75	205.47	4.45	247.02	384.26	7.42	580.46	716.68	10.19
136	25.33	172.49	31.12	178.66	7.91	35.29	182.88	13.87	39.17	187.17	19.04
175	42.49	330.29	18,473.59	18,759.91	6.88	83,955.49	84,195.41	13.36	$1.71 \cdot 10^5$	$1.71 \cdot 10^5$	19.45
263	53.18	415.79	1,982.93	2,343.92	7.84	13,165.97	13,524.63	15.22	30,219.76	30,582.11	21.46

Table 2: Experimental results of the synthetic case studies for the eight synthetic processes and when varying the amount of noise injected in the event log. The time refers to the average per trace.

right-hand side charts show the sum of these two times. This way, we can have a better understanding of the weight of these two components in the total time.

The results show that the planner is able to align even the most complicated traces (i.e., the largest traces with the 30% of noise injected) in no more than 208 milliseconds. The results of the experiments seem to suggest that the time for computing an optimal alignment grows polynomially with the trace length and the amount of noise. Similarly results are also observed for the synthetic process models, which are discussed in Section 5.2. This means that, whereas the worst-case complexity of computing alignments is certainly exponential wrt. the noise amount and trace length, the planner's heuristics are able to considerably limit the exploration of the search space.

5.2. Evaluation with Synthetic Data Sets

As mentioned above, the evaluation with synthetic data sets aims to assess the scalability of the planning-based approach with model of increasingly larger sizes and to compare it with the scalability of existing approaches.

Specifically, we generate eight Petri nets with respectively 25, 36, 68, 95, 115, 136, 175 and 263 (visible and invisible) transitions (the Petri nets are attached as an Appendix at the end of the paper). For each Petri net, we generate an event log with 1000 traces. The average trace length is respectively 20.8, 26.5, 51.6, 27.8, 42.8, 58.3, 148.07, 156.21 for the eight processes in question. We used the PLG2 tool (Burattin, 2015) to generate the Petri nets and the event logs, using the default parameters to configure the generation. In the same way as for the real-life case studies, we injected 10%, 20% and 30% of noise in the event log so as to be able to assess the correlation between computation time and amount of noise.

Table 2 shows the results of the experiments conducted on the eight Petri nets generated by PLG2 when varying the injected noise from no noise to 30% noise. The table reports for each combination model-noise the search time, the total time (translation, pre-processing plus search time) and the alignment's cost, averaged over all event-log traces. When injecting no noise, the alignment cost is clearly zero and, hence, we do not report it. The average translation + pre-processing time and search times are also plotted on, respectively, the left-hand and right-hand side graphs in Figure 7. The former linearly grows

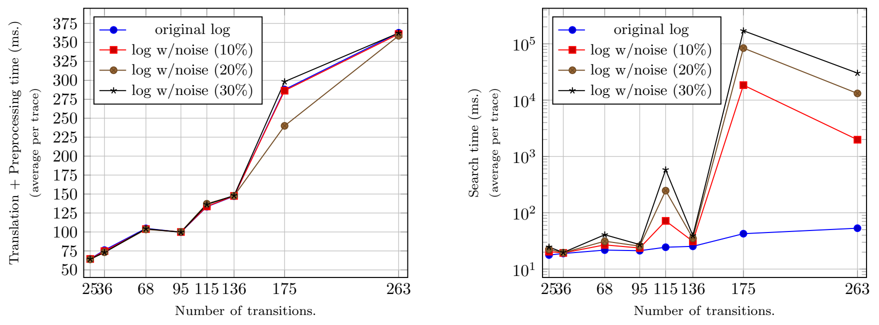


Figure 7: Experimental results of the synthetic case studies for the eight synthetic processes and when varying the amount of noise injected in the event log. The left-hand and right-hand side graphs refer to the average translation time and search time, respectively.

with larger models. This is expected: translation time accounts for the time to encode the alignment problems as planning problems and the pre-processing is the time to parse the planning problems and instantiate the internal structures for searching. These times are certainly linearly proportional to the size of the domain, namely the size of the process model and the respective event log. The slight decrease for the process model with 95 activities compared with the model with 68 activities is undoubtedly related to a smaller average trace size of the event log referring to the process with 95 activities.

Looking at the search time in the right-hand side graph in Figure 7, it seems that the planning-based approach shows a good scalability. With the exception of the model with 175 activities and noise of 20% and 30%, the alignment time is less than 30 seconds per trace. Clearly, the problem has an exponential complexity with respect to the size of the process model and the complexity of log traces (length and noise). Therefore, in certain cases, the computation can take significantly longer. Some readers might be surprised that a model with fewer activities is harder than another with more activities, e.g., comparing the models with 115 and 136 activities and the models with 175 and 263 activities. However, this is easy to explain: The complexity of computing alignments is linked to the amount of behavior that the model allows and, not necessarily does a larger number of activities reflect a larger amount of allowed behavior. As an extreme example, one can think of a model that consists in a sequence of 100 activities: There is only one allowed behavior, namely the execution sequence of these 100 activities. Unfortunately, it is far from being trivial to use the amount of allowed behavior as dependent variable to vary in the experiments. Computing the amount of allowed behavior is an entire direction of research that is still being carried on. Also, all tools to synthesize artificial models and logs are based on parameters linked to number of modeling elements (e.g, the number of transitions/activities) rather than based on the amount of allowed behavior.

The synthetic set of process models have also been used to compare the performances and scalability of our planning-based approach and the approach by

PN Length	Adriansyah's approach (ms.)	Plan-based approach (ms.)	Adriansyah's approach (ms.)	Plan-based approach (ms.)	alignment cost	Adriansyah's approach (ms.)	Plan-based approach (ms.)	alignment cost	Adriansyah's approach (ms.)	Plan-based approach (ms.)	alignment cost
	original log		10% noise			20% noise			30% noise		
25	10.35	82.33	25.85	84.79	3.08	42.8	86.21	4.24	56.87	88.87	5.32
36	2.39	94.98	11.71	94.13	4.95	19.24	92.78	8.03	28.6	92.67	10.44
68	23.68	126.45	112.85	130.68	3.07	164.92	134.57	4.38	278.31	144.99	6.53
95	13.16	120.94	77.56	123.65	4.43	119.89	125.78	7.1	168.32	126.82	9.25
115	75.22	157.71	638.52	205.47	4.45	966.34	384.26	7.42	1,987.12	716.68	10.19
136	64.05	172.49	304.62	178.66	7.91	465.97	182.88	13.87	578.63	187.17	19.04
175	659.25	330.29	—	18,759.91	6.88	—	84,195.41	13.36	—	$1.71 \cdot 10^5$	19.45
263	593.69	415.79	—	2,343.92	7.84	—	13,524.63	15.22	—	30,582.11	21.46

Table 3: Comparison between the experimental results of the synthetic case studies through the Adriansyah's approach (Adriansyah et al., 2011; van der Aalst et al., 2012) and the planning-based approach when varying the amount of noise injected in the event log. The time refers to the average per trace. The missing values refer to experiments that could not be carried out through the Adriansyah's approach because of lack of memory.

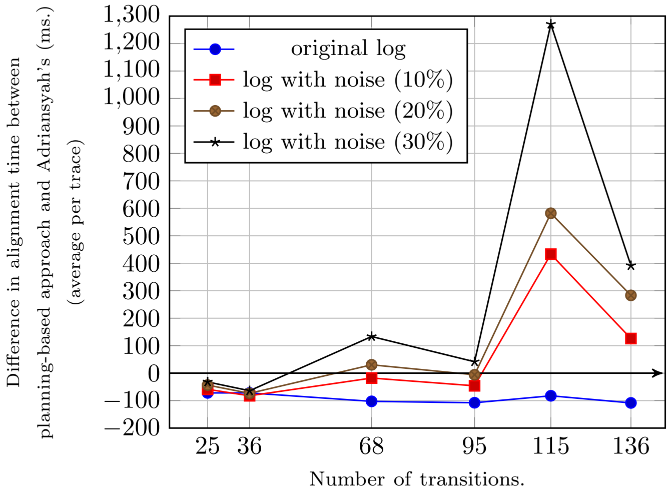


Figure 8: Comparison of the total alignment time between our planning-based approach and the approach proposed by Adriansyah et al. For the different combinations of synthetic processes and event-log noise, the graph plots the difference in the alignment time between the Adriansyah's and the planning-based approach. Values larger than zero indicates that our approach is faster than Adriansyah's; conversely, values smaller than zero indicates that our approach is slower. The results do not show the comparison for the models with 176 and 263 activities because the implementation of the Adriansyah's technique could not compute the optimal alignments and run out of memory. Even when approaches could compute optimal alignments, our approach outperforms the existing approach for larger models with more noise, while the results are comparable for small models and little amount of noise.

Adriansyah et al. (Adriansyah et al., 2011; van der Aalst et al., 2012). We have used the same process models and event logs as before. Table 3 shows the results of the experiments conducted through the Adriansyah's and our planning-based approach. The missing values in the table refer to executions where the approach by Adriansyah et al. (Adriansyah et al., 2011; van der Aalst et al., 2012) was unable to terminate and compute optimal alignments for all traces.

As previously mentioned, this was caused by the fact that the existing approach was running out of memory while computing the optimal alignments for certain traces, even though the machine used for the experiment was equipped with 16 GB of RAM. Conversely, for our planning-based approach, we monitored the memory consumption during the alignment tasks, and we found that the maximum amount of memory used was of 4.8 GB. This is certainly due to the fact that the planning-based approach required to visit a few number of states while building alignments.

It is worthy observing that, given a specific process model and an event log, the alignment steps synthesized by our plan-based approach have exactly the same cost than the ones provided by the Adriansyah's, as both approaches provide optimal solutions (i.e., with the lowest cost) to the alignment problem.

For the different combinations of process models and event logs with different amount of noise, Fig. 8 plots the difference in the average alignment time between the Adriansyah's and our planning-based approach. Values larger than zero indicates that our approach is faster than Adriansyah's; conversely, values smaller than zero indicates that our approach is slower. We used the implementation available in ProM 6.6 to compute the alignment for the Adriansyah's approach.[4] The graph shows that the ProM implementation of the Adriansyah's approach is faster for smaller models and, also, for the models of intermediate sizes but with small amount of noise. When the noise amount increases and/or the models become larger, the planning-based outperforms the existing approach, even of several orders of magnitude (for the hardest combination of model and event log). For instance, for the combination of the model with 115 activities and the respective event log with the injection of 30% of noise, the planning-based approach requires around 716 milliseconds on average to compute the alignment versus 1987 milliseconds (277% of the time) for the Adriansyah's, therefore with a gain of around 1300 milliseconds per trace. Of course, we do not show here the result comparison with the models with 175 and 263 activities because, as mentioned before, the existing approach was unable to compute all optimal alignments without running out of memory.

As a conclusion, *when used to compute alignments between process models and log traces,* FAST-DOWNWARD*, the state-of-the-art planner used in our implementation, was able to compute optimal alignments for every synthetic model used in this experiment. Conversely, the existing approach developed by Adriansyah et al. was unable to compute all optimal alignments without running out of memory. Even when the existing approach was able to carry out the alignment task, it was significantly outperformed for the large models. This clearly motivates the need of our planning-based approach, which scales significantly better with models and log of increasing sizes.*

Readers should notice that saving 1300 milliseconds per trace is not negligible: event logs with 500000 traces are very common and we would lead to saving

[4]ProM is an is a open-source framework for implementing process mining tools and algorithms. ProM 6.6 Web site - `http://www.promtools.org/doku.php?id=prom66`

650000 seconds of computation, namely almost 180 hours (around 7 days and half). It is worth also highlighting that our definition of encoding contributes to improved performances. The encoding is defined such that the planners requires as little time as possible to interpret it. In order to do so, we have minimized the number of predicates and planning actions and of their parameters.

The reason why the Adriansyah's approach is slightly faster for small process models and/or small amount of noise is only related to the implementation. The implementation by Adriansyah et al. can compute the alignment of a log with N traces in a single OS (operating-system) process. Conversely, FAST-DOWNWARD needs to create $2N$ OS processes, namely one process for pre-processing the problem and one for solving it for each of the N traces. The amount of time required by the operating system to create a new OS process is comparable with the actual time to solve the planning problems when these problems are relatively small. Furthermore, FAST-DOWNWARD leverages on the creation of OS files to let the different OS processes communicate whereas the implementation by Adriansyah et al. uses the memory, with the latter obviously faster of several orders of magnitude. When the models are larger or the event logs are more noisy, this implementation's aspects become negligible because the actual search time is significantly larger that the OS time to create OS processes.

5.3. Integration with other planning algorithms

Our plan-based approach is able to produce - for any specific alignment problem - PDDL files that are standard and independent from the specific planning system. Therefore, plugging in new planners and performing new experiments have very limited costs.

In order to show the benefits of versatility and customization of our plan-based approach, we performed further experiments with the same synthetic Petri nets and event logs described in Section 5.2 by using two further optimal state-of-the-art planners: SymBA-2* (Torralba et al., 2014a) and SPM&S (Torralba et al., 2014b). We employed these two planners for a fair comparison because they also use heuristics that guarantee optimality. The SymBA-2* (Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning) planner performs a bidirectional search, while the SPM&S (Symbolic Perimeter Merge-and-Shrink) planner implements an optimal algorithm that combines abstraction heuristics, perimeter and symbolic search.

Table 4 shows that the Blind A* search strategy used by FAST-DOWNWARD performs significantly better in most of cases than the algorithms implemented in SymBA-2* and SPM&S. For the process model with 263 activities, FAST-DOWNWARD, SymBA-2* and SPM&S have comparable results in alignment's time. For the process model with 175 activities, FAST-DOWNWARD is surprisingly performing worse than the other two.

While this indicates that timing performances of automated planners are dependent on the used planning algorithm and system, it is beyond the main scope of the paper to provide guidelines about the preferable planning algorithms/systems in the different scenarios. This papers focuses on showing

PN Length	Fast-Down. (ms.)	SymBA-2* (ms.)	SPM&S (ms.)	Fast-Down. (ms.)	SymBA-2* (ms.)	SPM&S (ms.)	alignment cost	Fast-Down. (ms.)	SymBA-2* (ms.)	SPM&S (ms.)	alignment cost	Fast-Down. (ms.)	SymBA-2* (ms.)	SPM&S (ms.)	alignment cost
	original log			10% noise				20% noise				30% noise			
25	82.33	681.19	810.17	84.79	685.35	812.44	3.08	86.21	692.97	814.04	4.24	88.87	699.42	816.05	5.32
36	94.98	767.05	950.25	94.13	767.05	950.25	4.95	92.78	785.14	956.19	8.03	92.67	793.13	959.25	10.44
68	126.45	1,413.85	1,722.13	130.68	1,413.85	1,722.13	3.07	134.57	1,418.51	1,779.17	4.38	144.99	1,419.5	1,808.61	6.53
95	120.94	2,048.84	2,431.35	123.65	2,048.84	2,431.35	4.43	125.78	2,049.14	2,445.66	7.1	126.82	2,053.92	2,483.98	9.25
115	157.71	2,288.56	2,741.5	205.47	2,289.84	2,787.31	4.45	384.26	2,291.91	2,817.29	7.42	716.68	2,352.81	2,868.77	10.19
136	172.49	2,749.83	3,267.17	178.66	2,752.28	3,282.23	7.91	182.88	2,765.14	3,300.87	13.87	187.17	2,775.46	3,317.48	19.04
175	330.29	5,785.67	6,209.07	18,759.91	5,909.46	6,546.87	6.88	84,195.41	6,092.93	6,837.57	13.36	$1.71 \cdot 10^5$	6,259.99	7,182.13	19.45
263	415.79	11,844.62	13,054.21	2,343.92	11,960.84	13,289.56	7.84	13,524.63	12,044.67	13,744.92	15.22	30,582.11	12,194.09	14,075.09	21.46

Table 4: Comparison between the experimental results of the synthetic case studies obtained through three different planners (Fast-Downward, SymBA-2* and SPM&S) integrated in the planning-based approach when varying the amount of noise injected in the event log. The time refers to the average per trace.

that alignment problems can be encoded in planning problems using a system-independent planner language, PDDL. Since the encoding is system independent, we can seamlessly switch from one planner to the other and look for the planner that outperforms the others for the problem in question. The experiment shows that an outperforming planning system does not really exist and that many state-of-the-art planners perform the existing ad-hoc approaches. The power of our approach is also linked to the versatility: in the future we can improve the performance even further if a better planner will be released. Our mapping to PDDL is independent of the specific planner and, hence, new planners compliant with existing planners can be integrated with almost no effort.

6. Related Works

This section touches on the research works related to this paper.

Section 6.1 reports on the practical relevance of conformance checking with special focus on model-log alignments. It also reports on successful cases where alignments are used to detect deviations between the prescribed behavior and what has been observed in reality.

Section 6.2 illustrates how planning techniques have been employed in the domain of BPM, highlighting the significant differences with respect to the exploitation of planners in the work reported in this paper.

6.1. Conformance Checking and Alignments

Several organizations maintain process models that describe or prescribe how cases (e.g., orders or bank applications) are handled. However, reality may not agree with what is modeled. As indicated in Section 1, conformance checking techniques reveal and diagnose differences between the behavior that is modeled and what is observed.

Several existing works and case studies illustrate that conformance checking is a highly relevant problem with a strong practical relevance. Several successful cases are reported where conformance checking is used to uncover common and frequent deviation patterns in several domains, such as public sector (e.g. (van der Aalst, 2016, pp. 404-408) and (Rozinat & van der Aalst, 2008; Ramezani et al., 2012)), chip production (Rozinat et al., 2009), health care (Mans et al., 2015; Kirchner et al., 2013), finance (Adriansyah & Buijs, 2012) and automotive (vanden Broucke et al., 2013).

In particular, with the exception of (van der Aalst, 2016; Rozinat & van der Aalst, 2008), these papers leverage on optimal model-log alignments and, hence, they can exactly pinpoint which activities are the root-causes of deviations. For further information of the case studies, readers are referred to the corresponding publications. It is still worth highlighting here that, for every mentioned case study, the process model was represented as a 1-bounded Petri net, thus illustrating that Petri nets can be sufficiently expressive to adequately represent many real-life processes.

Alignment-based conformance checking is primarily used as an artifact to pinpoint and document deviations; however, alignments can also be employed to improve existing models (see, e.g., (Fahland & van der Aalst, 2012)) and to evaluate the quality of models discovered through process-mining techniques (see, e.g., (Adriansyah et al., 2015)).

Optimal alignments allow for detecting low-level deviations, such as missing or non-allowed executions. However, in (Adriansyah et al., 2013b), a technique is proposed to "extend alignment-based deviation analysis techniques by supporting the detection of high-level deviations such as activity replacements and swaps". Authors of (Adriansyah et al., 2013b) leverage on an oracle to compute optimal alignments.

Independently of the employed techniques, the worst-case complexity of the problem of computing optimal alignments is exponential with the respect to the amount of behavior allowed by the process models and the length of the log traces. Therefore, no matter how one improves the computation, there will always be a model for which the computation of alignments for certain traces becomes intractable. The computation of alignments can be speeded up by using divide-and-conquer approaches: the model is broken down into fragments and each fragment is aligned separately. In (van der Aalst, 2013), a decomposition technique is proposed that guarantees the model to be decomposed in fragments such that, if the smaller traces fit the individual fragments, then they can be composed into a trace that fits into the overall process model.

6.2. The Use of Planning Techniques in Business Process Management

A number of research works exist on the use of planning techniques in the context of BPM, covering the various stages of the process life cycle.

For the design-time phase, existing literature works focus on exploiting planning techniques to automatically generate candidate process models that are able of achieving some business goals starting from a complete (Gajewski et al., 2005; Ferreira & Ferreira, 2006) or an incomplete (Marrella & Lespérance, 2013a,b) description of the process domain. Some research works also exist that use planning techniques to deal with problems for the run-time phase, i.e. when the process is being executed. Work (Currie & Tate, 1991) reports on the use of planners to allocate process activities to (human) resources, whereas works (Marrella et al., 2016, 2014; van Beest et al., 2014; Marrella et al., 2012, 2011; Marrella & Mecella, 2011; Adams et al., 2006) report on approaches to adapt the running process instances to cope with anomalous situations, including connection anomalies, exogenous events and task faults.

Readers should observe that the entire aforementioned approaches do not automatically use historical information, e.g., extracted from event-log data. Also, in these works, planning techniques are used for completely different purposes.

In the field of conformance checking, the scientific literature reports several work involving the use of planning and other AI-based techniques. In (Regis et al., 2012; Montali, 2010), authors employ model checking to verify whether or not certain properties hold in a process model. It is perhaps possible to represent traces as queries, e.g., in temporal logics. However, these works would return a true/false answer, namely whether or not a trace is confirming the model, without pinpointing where deviations occur and what their severity is.

Relatively close is the work of Di Francescomarino et al. (Di Francescomarino et al., 2015) where authors use planners to recover the missing recording of events in log traces. The concept of missing event recordings is very similar to moves in model in our approach. However, they assume that all executions are compliant with the model and, hence, every event that is present in the incomplete log trace is assumed to be correct. In other words, they do not foresee log moves. Also, when there are multiple ways to complete a log trace, the approach only aims at the explanation with the fewest number of additions, whereas in reality it may expected that events for certain activities are less likely to be missing (i.e., additions for those activities should have a higher weight/cost). Repairing event logs is also the goal of the approach in (Rogge-Solti et al., 2013); however, such approach requires an oracle to compute optimal alignments, such as what proposed in this paper.

In (López et al., 2016), the authors propose a technique to encode the problems of finding optimal alignments as constraint-satisfaction problems. This technique shares the advantage to let it be easy and seamless to plug in the most recent and enhanced off-the-shelf systems; furthermore, in the same way as planners, constraint-satisfaction systems allow one to obtain a good enough solution (i.e., sub optimal) relatively fast. Unfortunately, proposals based on constraint satisfaction are challenged by the fact that linear constraints cannot easily represent sequences. As result, work (López et al., 2016) is characterized by a number of limitations. First, "it is restricted to acyclic process models" (López et al., 2016): The model cannot contain loops, i.e. a set of activities that are repeated an arbitrary number of times (e.g., until a certain condition is met). Second, invisible transitions and "different transitions with the same name are not allowed" (López et al., 2016) (i.e., with the same label). Last, the proposal "cannot always ensure that the found solution is minimal" (i.e., optimal).

Some research approaches focus on verifying the compliance of process models with respect to a set of formulas, which are mostly intended to encode business rules of which one wants to verify the compliance (e.g. (Ly et al., 2011; Belardinelli et al., 2012; Montali, 2010)) A log trace can possibly be represented by a set of formulas (e.g., an event for activity A is followed by an event for activity B) and, hence, its compliance can be checked by applying existing techniques. Unfortunately, their diagnostics are limited to highlighting *which formulas* are not satisfied. We aim to pinpoint *where in the process* deviations occur, such as the case that an activity has not been executed. It is far from easy to derive the same insights on the basis of not-satisfied formulas. This is due to the fact that the same log trace can be "repaired" in multiple ways to satisfy one formula. When multiple, non-satisfied formulas come in to play, we would be interested in finding the least expensive changes that are needed to ensure all formulas are satisfied. In fact, this is again the problem of finding the least expensive solution in a certain search space, i.e. a planning problem. To our knowledge, the same limitation is also shared by techniques to debug the execution of distributed systems (e.g. (Reynolds et al., 2006; Xu et al., 2009))

7. Conclusion

Within the discipline of BPM, conformance checking refers to the problem of comparing if process executions comply with the rules that should be followed. The starting point is a process model, which encodes such rules, and an event log, which records actual executions of the process. Conformance checking techniques aim to verify whether the actual executions deviate process model. The notion of *alignment* (van der Aalst et al., 2012) provides a robust approach to conformance checking, which makes it possible to pinpoint the deviations causing nonconformity. In particular, traces in the log are aligned with traces in the model, thus making deviations be highlighted.

This paper has reported on how the problem of finding an alignment can be represented as a planning problem in PDDL, which can be solved by off-the-shelf planners. If conformance checking problems are converted into planning problems, one can seamlessly update to the recent versions of the best performing automated planners, with evident advantages in term of versatility and customization. The theoretical results have shown that the planning problems always have solutions (correctness) and such solutions can be found by planners in a finite amount of time (termination). From a practical perspective, we integrated our tool with a planning system out-of-the-box (i.e., without specific optimizations) and tested it in two real-life case studies and, also, with several combinations of synthetic process models and event logs. The results suggest that not only is the approach highly scalable but it does also outperform existing approaches when the process models and event logs are of larger sizes.

As future work, we plan to also detect nonconformity that relates to time, resource and data aspects, such as deviations for activities that are not performed by authorized employees or within given deadlines. This is far from being trivial since the problem is in general undecidable; however, we aim to explore what kind of expressiveness limitation we need to introduce to ensure decidability. Another interesting avenue for future work is to consider the interdependencies among the moves in the cost function. For instance, a certain model move can be associated with a lower cost if it appears after/before certain log moves.

Acknowledgements. The work of Andrea Marrella has been partly supported by the Italian projects SM&ST, RoMA, FilieraSicura and NEPTIS, and by the Sapienza project "Data-aware Adaptation of Knowledge-intensive Processes in Cyber-Physical Domains through Action-based Languages".

References

van der Aalst, W. M. P. (1998). The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, *8*, 21–66. doi:10.1142/S0218126698000043.

van der Aalst, W. M. P. (2013). Decomposing Petri nets for process mining: A generic approach. *Distributed and Parallel Databases*, *31*, 471–507. doi:10.1007/s10619-013-7127-5.

van der Aalst, W. M. P. (2016). *Process Mining: Data Science in Action.* (2nd ed.). Springer-Verlag Berlin Heidelberg. doi:10.1007/978-3-662-49851-4.

van der Aalst, W. M. P., Adriansyah, A., & van Dongen, B. (2012). Replaying History on Process Models for Conformance Checking and Performance Analysis. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, *2*, 182–192. doi:10.1002/widm.1045.

van der Aalst, W. M. P., & van Hee, K. M. (2002). *Workflow Management: Models, Methods, and Systems*. MIT Press.

Accorsi, R., & Stocker, T. (2012). On the exploitation of process mining for security audits: The conformance checking case. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12* (pp. 1709–1716). New York, NY, USA: ACM. doi:10.1145/2245276.2232051.

Adams, M., ter Hofstede, A. H. M., Edmond, D., & van der Aalst, W. M. P. (2006). Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In *Proceedings of OnTheMove Federated Conferences OTM 2006)*. Springer Berlin Heidelberg volume 4275 of *Lecture Notes in Computer Science*. doi:10.1007/11914853_18.

Adriansyah, A., & Buijs, J. (2012). *Mining Process Performance from Event Logs: The BPI Challenge 2012 Case Study*. Technical Report. Available at http://www.win.tue.nl/bpi/lib/exe/fetch.php?media=2012:adriansyah.pdf. Retrieved on January, 19th, 2017.

Adriansyah, A., van Dongen, B. F., & van der Aalst, W. M. P. (2013a). *Memory-Efficient Alignment of Observed and Modeled Behavior*. BPM Center Report BPM-03-03 BPMcenter.org. Available at http://bpmcenter.org/wp-content/uploads/reports/2013/BPM-13-03.pdf. Retrieved on January, 19th, 2017.

Adriansyah, A., Dongen, B. F. v., & Zannone, N. (2013b). Controlling Break-the-Glass Through Alignment. In *Proceedings of the 2013 International Conference on Social Computing, SOCIALCOM '13* (pp. 606–611). IEEE Computer Society. doi:`10.1109/SocialCom.2013.91`.

Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B. F., & van der Aalst, W. M. P. (2015). Measuring precision of modeled behavior. *Information Systems and e-Business Management*, *13*, 37–67. doi:`10.1007/s10257-014-0234-7`.

Adriansyah, A., Sidorova, N., & van Dongen, B. F. (2011). Cost-Based Fitness in Conformance Checking. In *Proceedings of the 11th International Conference on Application of Concurrency to System Design, ACSD 2011* (pp. 57–66). IEEE. doi:`10.1109/ACSD.2011.19`.

van Beest, N. R. T. P., Kaldeli, E., Bulanov, P., Wortmann, J. C., & Lazovik, A. (2014). Automated runtime repair of business processes. *Inf. Syst.*, *39*, 45–79. doi:`10.1016/j.is.2013.07.003`.

Belardinelli, F., Lomuscio, A., & Patrizi, F. (2012). Verification of GSM-Based Artifact-Centric Systems through Finite Abstraction. In *Proceedings of the 10th International Conference on Service-Oriented Computing, ICSOC'12* (pp. 17–31). Springer Berlin Heidelberg volume 7636 of *Lecture Notes in Computer Science*. doi:`10.1007/978-3-642-34321-6_2`.

Billington, J., Christensen, S., van Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., & Weber, M. (2003). The Petri Net Markup Language: Concepts, Technology, and Tools. In *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets, ICATPN 2003* (pp. 483–505). Springer Berlin Heidelberg volume 2679 of *Lecture Notes in Computer Science*. doi:`10.1007/3-540-44919-1_31`.

Bonet, B., & Geffner, H. (2001). Planning and Control in Artificial Intelligence: A Unifying Perspective. *Applied Intelligence*, *14*, 237–252. doi:`10.1023/A:1011286518035`.

vanden Broucke, S., Vanthienen, J., & Baesens, B. (2013). Volvo IT belgium VINST. In *Proceedings of the 3rd Business Process Intelligence Challenge co-located with 9th International Business Process Intelligence Workshop BPI 2013*. CEUR-WS.org volume 1052 of *CEUR Workshop Proceedings*.

Burattin, A. (2015). PLG2: Multiperspective Processes Randomization and Simulation for Online and Offline Settings. *CoRR*, *abs/1506.08415*. Available at `https://arxiv.org/pdf/1506.08415v3.pdf`. Retrieved on January, 19th, 2017.

Currie, K., & Tate, A. (1991). O-Plan: The Open Planning Architecture. *Artificial Intelligence*, *52*, 49–86. doi:`10.1016/0004-3702(91)90024-E`.

de Leoni, M., & Mannhardt, F. (2015). Road Traffic Fine Management Process. Eindhoven University of Technology. `http://dx.doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5`.

Di Francescomarino, C., Ghidini, C., Tessaris, S., & Sandoval, I. V. (2015). Completing Workflow Traces Using Action Languages. In *Proceedings of the 27th International Conference on Advanced Information Systems Engineering, CAiSE 2015* (pp. 314–330). Springer International Publishing volume 9097 of *Lecture Notes in Computer Science*. doi:`10.1007/978-3-319-19069-3_20`.

Dumas, M., La Rosa, M., Mendling, J., & Reijers, H. A. (2013). *Fundamentals of Business Process Management*. Springer Berlin Heidelberg. doi:`10.1007/978-3-642-33143-5`.

Fahland, D., & van der Aalst, W. M. P. (2012). Repairing Process Models to Reflect Reality. In *Proceedings of the 10th International Conference on Business Process Management, BPM'12* (pp. 229–245). Springer Berlin Heidelberg volume 7481 of *Lecture Notes in Computer Science*. doi:`10.1007/978-3-642-32885-5_19`.

Ferreira, H. M., & Ferreira, D. R. (2006). An Integrated Life Cycle for Workflow Management Based on Learning and Planning. *Int. J. Cooperative Inf. Syst.*, *15*, 485–505. doi:`10.1142/S0218843006001463`.

Fox, M., & Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res. (JAIR)*, *20*, 61–124. doi:`10.1613/jair.1129`.

Gajewski, M., Meyer, H., Momotko, M., Schuschel, H., & Weske, M. (2005). Dynamic Failure Recovery of Generated Workflows. In *Proceedings of the 16th International Workshop on Database and Expert Systems Applications, DEXA 2005* (pp. 982–986). IEEE. doi:`10.1109/DEXA.2005.78`.

Geffner, H., & Bonet, B. (2013). A Concise Introduction to Models and Methods for Automated Planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *8*, 1–141. doi:`10.2200/S00513ED1V01Y201306AIM022`.

Ghallab, M., Nau, D. S., & Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers Inc.

Günther, C. W., & Verbeek, H. (2014). *IEEE Task Force on Process Mining: XES Standard Definition*. Technical Report BPM Center Report BPM-14-09 BPMcenter.org. Available at `http://bpmcenter.org/wp-content/uploads/reports/2014/BPM-14-09.pdf`. Retrieved on January, 19th, 2017.

Haslum, P., & Geffner, H. (2014). Heuristic Planning with Time and Resources. In *Proceedings of the Sixth European Conference on Planning, ECP-01*. AAAI.

Helmert, M. (2002). Decidability and Undecidability Results for Planning with Numerical State Variables. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, AIPS 2002* (pp. 44–53). AAAI.

Helmert, M. (2006). The Fast Downward Planning System. *J. Artif. Int. Res. (JAIR)*, *26*, 191–246. doi:`10.1613/jair.1705`.

Hosseinpour, M., & Jans, M. (2016). Categorizing Identified Deviations for Auditing. In *Proceedings of the 6th International Symposium on Data-driven Process Discovery and Analysis, SIMPDA 2016* (pp. 125–129). CEUR-WS.org volume 1757 of *CEUR Workshop Proceedings*.

Kalenkova, A. A., van der Aalst, W. M. P., Lomazova, I. A., & Rubin, V. A. (2015). Process mining using BPMN: relating event logs and process models. *Software & Systems Modeling*, (pp. 1–30). doi:`10.1007/s10270-015-0502-0`.

Kiepuszewski, B., ter Hofstede, A., & van der Aalst, W. (2003). Fundamentals of control flow in workflows. *Acta Informatica*, *39*, 143–209. doi:`10.1007/s00236-002-0105-4`.

Kirchner, K., Herzberg, N., Rogge-Solti, A., & Weske, M. (2013). Embedding Conformance Checking in a Process Intelligence System in Hospital Environments. In *Process Support and Knowledge Representation in Health Care: BPM 2012 Joint Workshop, ProHealth 2012/KR4HC 2012* (pp. 126–139). Springer Berlin Heidelberg. doi:`10.1007/978-3-642-36438-9_9`.

López, M. T. G., Borrego, D., Carmona, J., & Gasca, R. M. (2016). Computing alignments with constraint programming: The acyclic case. In *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2016* (pp. 96–110). CEUR-WS.org volume 1592 of *CEUR Workshop Proceedings*.

Ly, L. T., Rinderle-Ma, S., Knuplesch, D., & Dadam, P. (2011). Monitoring Business Process Compliance Using Compliance Rule Graphs. In *Proceedings of OnTheMove Federated Conferences, OTM 2011* (pp. 82–99). Springer Berlin Heidelberg volume 7044 of *Lecture Notes in Computer Science*. doi:`10.1007/978-3-642-25109-2_7`.

Mannhardt, F., de Leoni, M., Reijers, H. A., & Aalst, W. M. P. (2015). Balanced multi-perspective checking of process conformance. *Computing*, *98*, 407–437. doi:`10.1007/s00607-015-0441-1`.

Mans, R. S., van der Aalst, W. M. P., & Vanwersch, R. J. B. (2015). Applications of process mining. In *Process Mining in Healthcare: Evaluating and Exploiting Operational Healthcare Processes* (pp. 53–78). Springer International Publishing. doi:`10.1007/978-3-319-16071-9_5`.

Marrella, A., & Lespérance, Y. (2013a). Synthesizing a Library of Process Templates through Partial-Order Planning Algorithms. In *Proceedings of the 14th International Conference on Business Process Management, Development and Support, BPMDS 2013* (pp. 277–291). Springer Berlin Heidelberg volume 147 of *Lecture Notes in Business Information Processing*. doi:`10.1007/978-3-642-38484-4_20`.

Marrella, A., & Lespérance, Y. (2013b). Towards a Goal-Oriented Framework for the Automatic Synthesis of Underspecified Activities in Dynamic Processes. In *Proceedings of the 6th International Conference on Service-Oriented Computing and Applications, SOCA 2013* (pp. 361–365). IEEE. doi:`10.1109/SOCA.2013.43`.

Marrella, A., & Mecella, M. (2011). Continuous Planning for Solving Business Process Adaptivity. In *Proceedings of the 12th International Conference on Business Process Management, Development and Support, BPMDS 2011* (pp. 118–132). Springer Berlin Heidelberg volume 81 of *Lecture Notes in Business Information Processing*. doi:`10.1007/978-3-642-21759-3_9`.

Marrella, A., Mecella, M., & Russo, A. (2011). Featuring Automatic Adaptivity through Workflow Enactment and Planning. In *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 2011* (pp. 372–381). doi:`10.4108/icst.collaboratecom.2011.247096`.

Marrella, A., Mecella, M., & Sardiña, S. (2014). SmartPM: An Adaptive Process Management System through Situation Calculus, IndiGolog, and Classical Planning. In *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning, KR 2014*. AAAI.

Marrella, A., Mecella, M., & Sardina, S. (2016). Intelligent Process Adaptation in the SmartPM System. *ACM Trans. Intell. Syst. Technol.*, *8*, 25:1–25:43. doi:`10.1145/2948071`.

Marrella, A., Russo, A., & Mecella, M. (2012). Planlets: Automatically Recovering Dynamic Processes in YAWL. In *Proceedings of OnTheMove Federated Conferences, OTM 2012*. Springer Berlin Heidelberg volume 7565 of *Lecture Notes in Computer Science*. doi:`10.1007/978-3-642-33606-5_17`.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C. A., Ram, A., Veloso, M., Weld, D. S., & 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. Available at `http://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf`. Retrieved on January, 19th, 2017.

Montali, M. (2010). *Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach* volume 56 of *Lecture Notes in Business Information Processing*. Springer Berlin Heidelberg. doi:`10.1007/978-3-642-14538-4`.

Ramezani, E., Fahland, D., & van der Aalst, W. M. P. (2012). Where Did I Misbehave? Diagnostic Information in Compliance Checking. In *Proceedings of the 10th International Conference on Business Process Management, BPM 2012* (pp. 262–278). Springer Berlin Heidelberg volume 7481 of *Lecture Notes in Computer Science*. doi:`10.1007/978-3-642-32885-5_21`.

Regis, G., Ricci, N., Aguirre, N. M., & Maibaum, T. (2012). Specifying and Verifying Declarative Fluent Temporal Logic Properties of Workflows. In *Proceedings of the 15th Brazilian Conference on Formal Methods: Foundations and Applications, SBMF'12*. Springer Berlin Heidelberg volume 7498 of *Lecture Notes of Computer Science*. doi:`10.1007/978-3-642-33296-8_12`.

Reynolds, P., Killian, C., Wiener, J. L., Mogul, J. C., Shah, M. A., & Vahdat, A. (2006). Pip: Detecting the Unexpected in Distributed Systems. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation, NSDI'06* (pp. 115–128). USENIX Association.

Rogge-Solti, A., Mans, R. S., van der Aalst, W. M. P., & Weske, M. (2013). Improving Documentation by Repairing Event Logs. In *Proceedings of 6th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling, PoEM 2013* (pp. 129–144). Springer Berlin Heidelberg volume 81 of *Lecture Notes in Business Information Processing*. doi:`10.1007/978-3-642-41641-5_10`.

Rozinat, A., & van der Aalst, W. M. P. (2008). Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, *33*, 64–95. doi:`10.1016/j.is.2007.07.001`.

Rozinat, A., de Jong, I. S. M., Gnther, C. W., & v. der Aalst, W. M. P. (2009). Process mining applied to the test process of wafer scanners in asml. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *39*, 474–479. doi:`10.1109/TSMCC.2009.2014169`.

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

Torralba, A., Alcázar, V., López, C. L., Borrajo, D., Kissmann, P., & Edelkamp, S. (2014b). SPM&S Planner: Symbolic Perimeter Merge-and-Shrink. *International Planning Competition, IPC 2014*, (pp. 101–104).

Van Der Aalst, W. et al. (2011). Process Mining Manifesto. In *Proceedings of the 9th International Conference on Business Process Management (BPM 2011)* (pp. 169–194). Springer Berlin Heidelberg. doi:`10.1007/978-3-642-28108-2_19`.

Wilkins, D. E. (1988). *Practical planning: extending the classical AI planning paradigm*. Morgan Kaufmann.

Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M. I. (2009). Detecting Large-scale System Problems by Mining Console Logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, SOSP'09* (pp. 117–132). ACM. doi:10.1145/1629575.1629587.

Appendix

In this appendix we provide the diagrams of the synthetic Petri nets used to test the scalability of our planning-based approach (see Section 5.2).

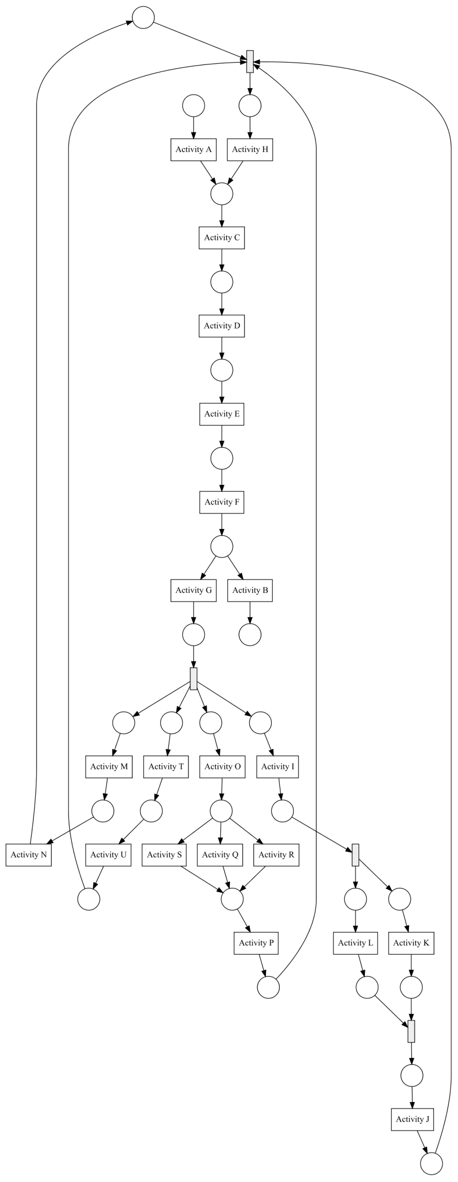


Figure 9: A synthetic Petri net with 25 transitions and 27 places.

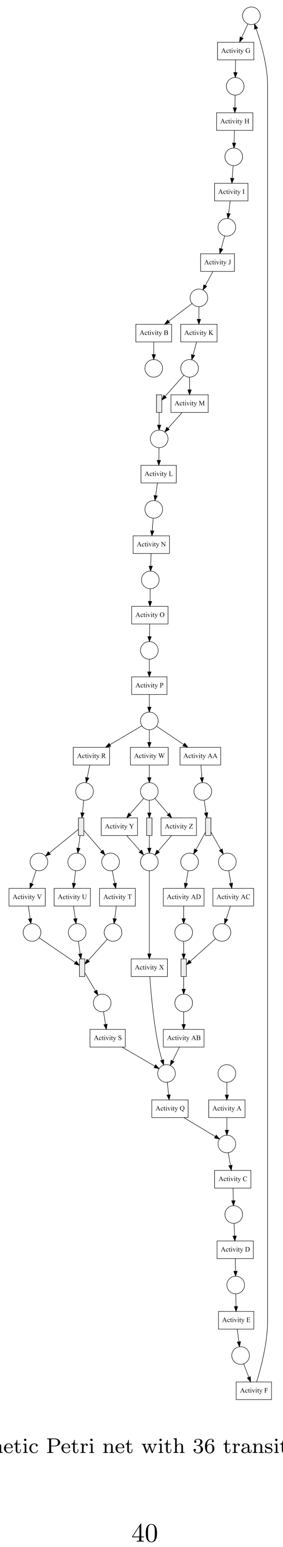


Figure 10: A synthetic Petri net with 36 transitions and 34 places.

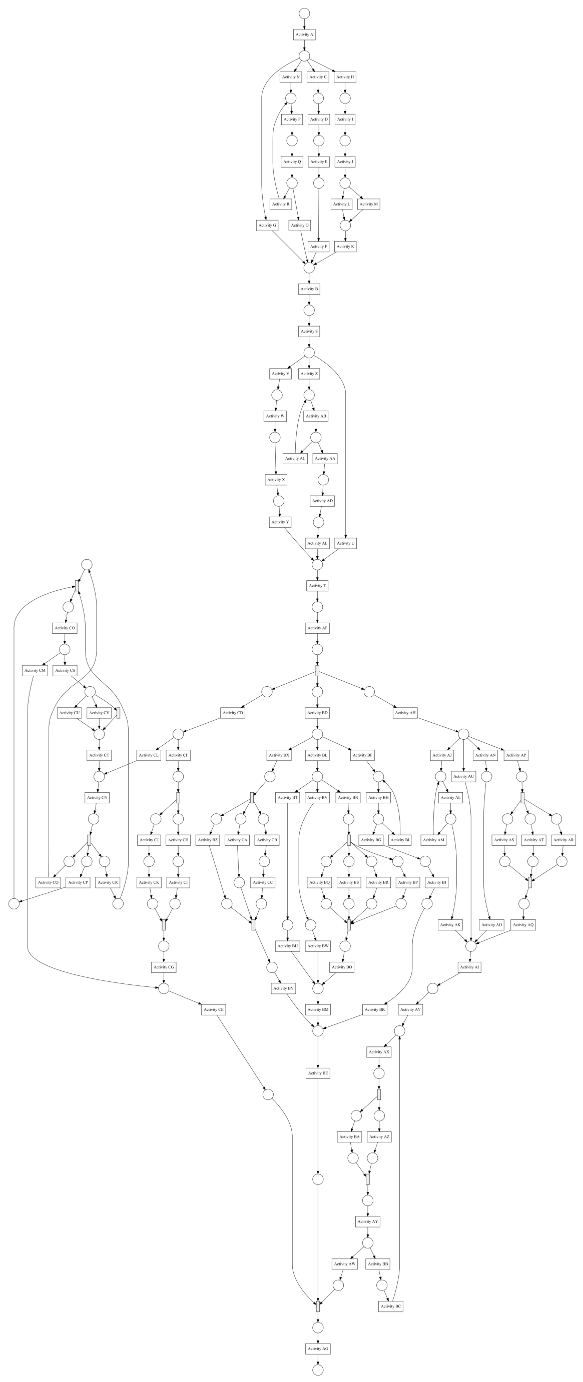


Figure 11: A synthetic Petri net with 115 transitions and 107 places.

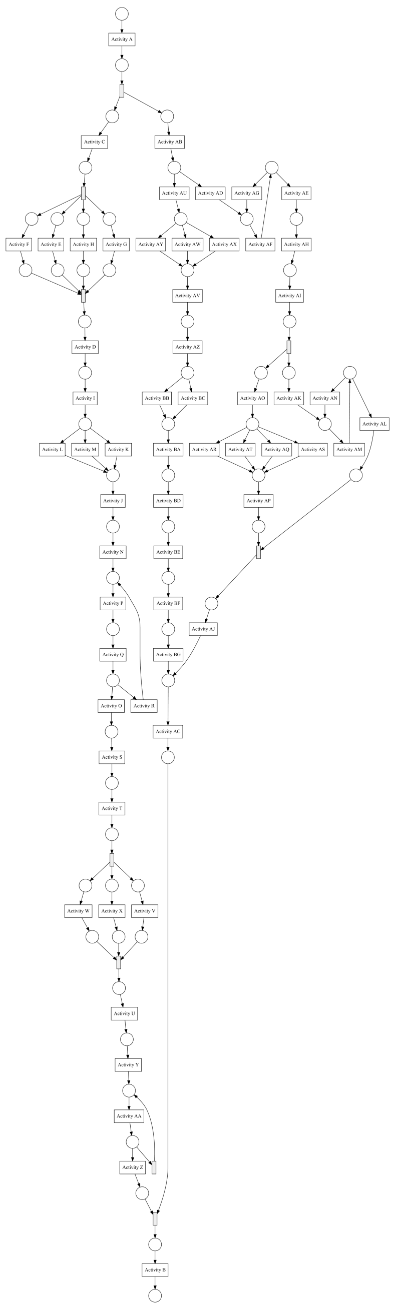


Figure 12: A synthetic Petri net with 68 transitions and 63 places.

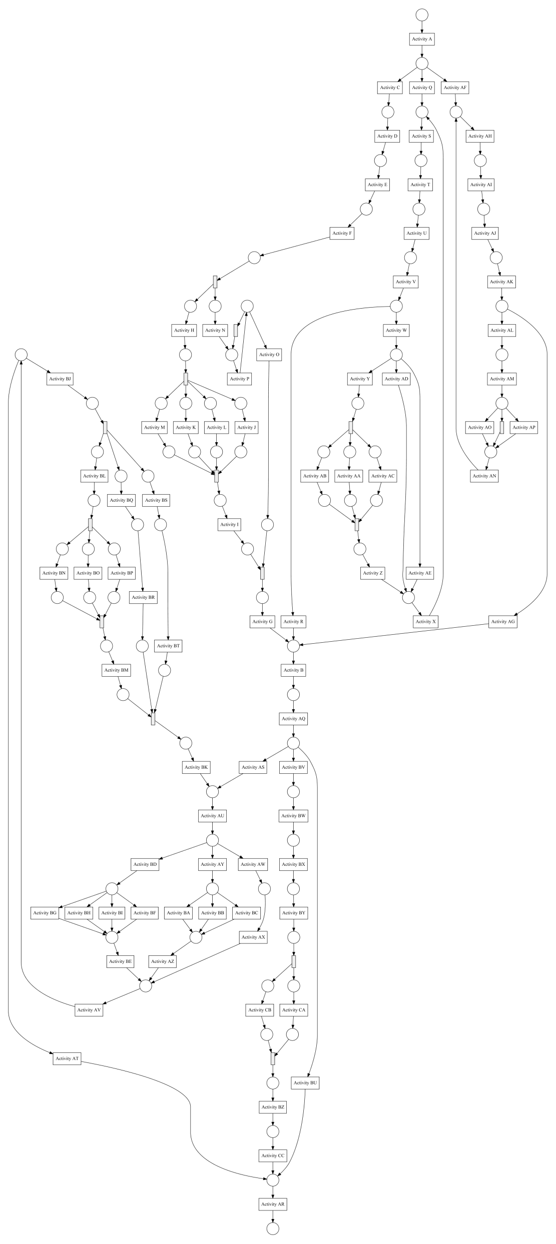


Figure 13: A synthetic Petri net with 95 transitions and 88 places.

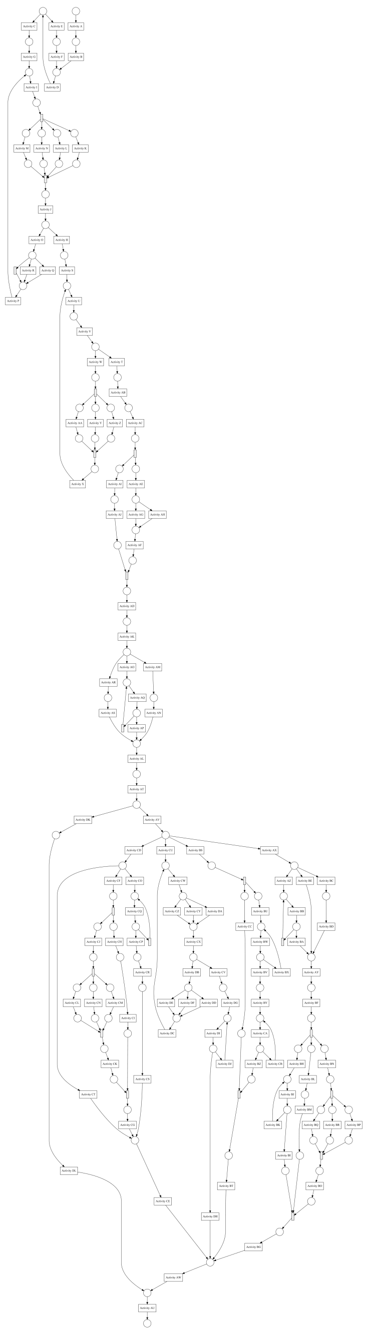


Figure 14: A synthetic Petri net with 136 transitions and 123 places.

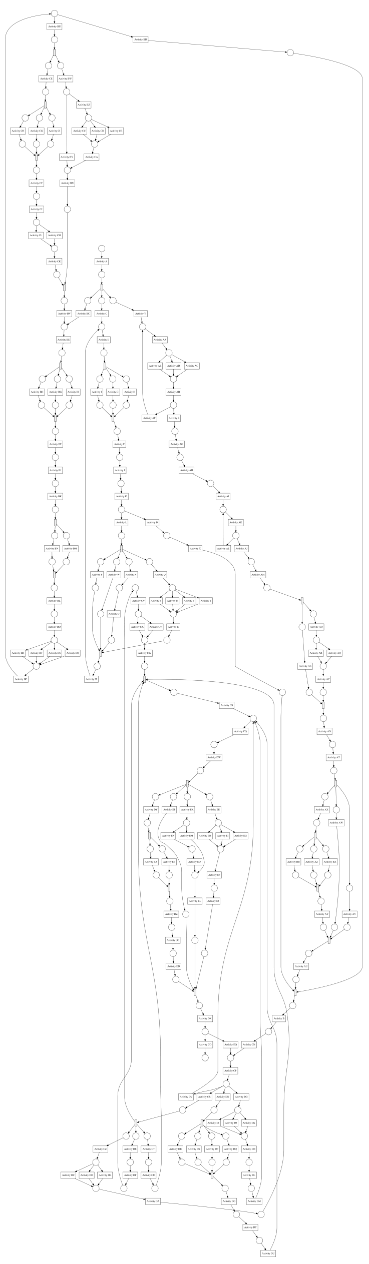


Figure 15: A synthetic Petri net with 175 transitions and 175 places.

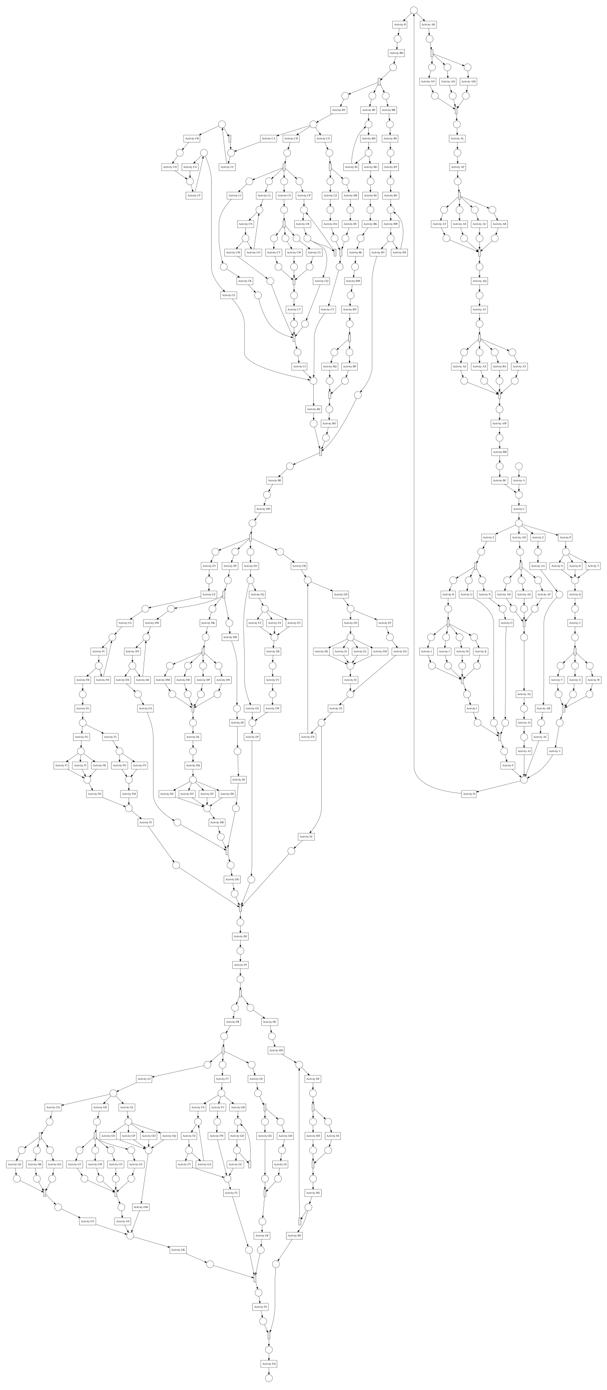


Figure 16: A synthetic Petri net with 263 transitions and 267 places.