How Planning Techniques Can Help Process Mining: The Conformance-Checking Case

Massimiliano de Leoni[1], Andrea Marrella[2]

[1] Eindhoven University of Technology, The Netherlands
m.d.leoni@tue.nl
[2] Sapienza - University of Rome, Italy
marrella@dis.uniroma1.it

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 in the process event data. Literature proposes solutions for the conformance-checking problem that, in fact, are implementations of planning algorithms built ad-hoc for the specific problem. Unfortunately, in the era of big data, these ad-hoc implementations do not scale sufficiently compared with robust, well-established planning systems. Furthermore, their ad-hoc nature does not allow for seamlessly plugging in new outperforming planning algorithms or heuristics, causing a massive amount of work to be necessary to incorporate and evaluate alternatives. This paper summarizes our last results on how instances of the conformance checking problem can be represented as classical planning problems in PDDL 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. Experiments with three different planners highlight this versatility. Furthermore, by employing several processes and event logs of different sizes, we show how our planning-based approach outperforms existing approaches of several order of magnitude and, in certain cases, carries out the task while existing approaches run out of memory.

1 Introduction

Process mining is a discipline that sits between data mining and process modeling and analysis and, hence, can be considered one of the links between data science and process science [1]. The idea of process mining is to discover, monitor and improve the processes by extracting knowledge from the data that are stored in information systems about how these systems are used to carry out processes. Differently from a-priori analysis, the focus is not on the assumed processes but on real processes in the way that they are executed. Therefore, the starting point of process mining is event data, which is analyzed to extract useful insights and recurrent patterns about how processes are executed within organizations.

Within Process Mining, *conformance checking* verifies whether the observed behavior stored in an *event log* is compliant with a process model that encodes how the process is allowed to be executed to ensure that norms and regulations are not violated. As an example, suppose that a process model may indicate that orders for more than

100000 Euros require a double check: conformance checking will check whether the actual executions follows this rule or not.

The notion of *alignment* [1, 2] provides a robust approach to conformance checking, which makes it possible to exactly pinpoint the deviations causing nonconformity with a high degree of detail. An alignment between a recorded process execution (*log trace*) 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*).

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 [1, 2], an approach is proposed that is based on the principle of the Occam's razor: the most parsimonious explanation is preferable. Therefore, one should not aim to find any alignment but, precisely, one of the alignments with the least expensive deviations (one of the so-called *optimal alignments*).

The work by Adriansyah et al. [2] provide a technique to compute these optimal alignments, based on the A* algorithm and ad-hoc heuristics. However, experiments (also reported in this paper) show that their technique does not scale sufficiently. In the era of big data, scalable approaches are desperately necessary, as also advocated by the IEEE Task Force in Process Mining [3]: "*In some domains, mind-boggling quantities of events are recorded. [...] Therefore, additional efforts are needed to improve performance and scalability.*".

We believe that re-implementing standard planning algorithms, such as A*, for solving specific planning problems in an ad-hoc way is not ideal. First, ad-hoc implementations cannot compare in scalability with well-established planning systems. Second, ad-hoc implementations prevent one from seamlessly plugging in new algorithms or evaluate alternative heuristics. 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 conformance checking problem, a massive amount of work is needed to modify those ad-hoc implementations.

Hence, in order to facilitate the integration of different planning algorithms, in [4], we illustrate how the problem of computing optimal alignments can be formulated as a planning problem, which can be solved through off-the-shelf PDDL planners.

Here, we summarize the main ideas discussed in [4] and report on the most interesting results from the experiments. The results show that our planning-based approach is versatile, allowing multiple planners to be integrated, and scales better than existing approaches with larger models and event logs.

2 Petri Nets and Event Logs

To model business processes, we opted for Petri nets (see, e.g., [1]), which have proven to be adequate [5] for this purpose. However, the approach described in this paper can be applied to other notations, such as BPMN.

A **Petri net** $N = (P, T, F)$ is a directed graph with a set P of nodes called *places* and a set T of transitions. The nodes are connected via directed arcs $F \subseteq (P \times T) \cup (T \times P)$. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles. Fig. 1 illustrates an example

of Petri nets. 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 $m : P \to \mathbb{N}$.

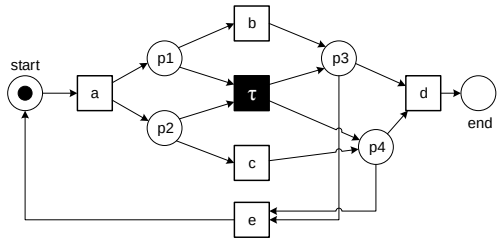

Fig. 1. An example of Petri net.

In any run of a Petri net, the number of tokens in places may change, i.e., the Petri net marking. A transition t is **enabled** at a marking m iff each input place contains at least one token, i.e. $\forall\ p \in {}^\bullet t,\ M(p) > 0$. 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. 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 $m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} m_n$.

Event logs are the starting point for process mining. An event log L is is a multi-set of log *traces* $\sigma_L \in T^*$. A trace is a sequence of transition firings and describes the exection of a *process instance* in terms of the executed *activities*.[3] Transition firings in an event log are known as *events*.

Some transitions do not represent process activities but are necessary to correctly represent the business process through Petri nets. These transitions are **invisible transitions** (similar to τ steps) and are not recorded as log events. For the sake of space, we assume here that each activity is mapped to one and only one Petri-net transition and that events are defined over the same alphabet as the Petri-net transitions. In [4], we show how these assumptions can be removed.

3 Alignment between Event Logs and Petri Nets

As mentioned in Section 1, we perform conformance checking by constructing an *alignment* of event log L and process model N [1, 2], 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 $\gg$.

[3] We use multisets because the same trace can appear multiple times because different process executions can be identical in terms of activities being executed.

Definition 1 (Alignment Moves). *Let $N = (P, T, F)$ be a Petri net and L be an event log. A legal* alignment move *for N and L is represented by a pair $(s_L, s_M) \in (T \cup \{\gg\} \times T \cup \{\gg\}) \setminus \{(\gg, \gg)\}$ such that:*

- *(s_L, s_M) is a* move in log *if $s_L \neq \gg$ and $s_M = \gg$,*
- *(s_L, s_M) is a* move in model *if $s_L = \gg$ and $s_M \in T$,*
- *(s_L, s_M) is a* synchronous move *if $s_L = s_M$.*

An alignment is a sequence of alignment moves:

Definition 2 (Alignment). *Let $N = (P, T, F)$ be a Petri net with an initial marking and final marking denoted with m_i and m_f Let also L be an event log. Let Γ_N be the universe of all alignment moves for N and L. 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 $m_i \xrightarrow{\sigma''} m_f$.*

A move in log for a transition t indicates that t occurred when not allowed; a move in model for a visible transition t indicates that t did not occur, when, conversely, expected.

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

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

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

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

Many alignments are possible for the same trace. For example, Fig. 2 shows three possible alignments for a trace $\sigma_1 = \langle a, d, b, c \rangle$. 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. As discussed in Section 1, we aim at finding a complete alignment of σ_L and N with minimal number of deviations for visible transitions, also known in literature as **optimal alignment**. Aligning an event log L means to compute an optimal alignment for each trace $\sigma_L \in L$. Clearly, different traces in L generally have different alignments, because they contain different events and deviations. With reference to the alignments in Fig. 2, γ_1 and γ_2 have two moves in model and/or log for visible transitions (γ_1 has one move in model for an invisible transition but it does not count for computing optimal alignments). Conversely, γ_3 has three moves for visible transitions, specifically one log move for d (3rd move) and two model moves for e (4th move) and d (last move). Since no alignment exists with only one moves in model and/or log for visible transitions, both γ_1 and γ_2 are optimal alignments and any can equally be returned. For the sake of space, we assume here that all deviations (i.e., moves in model for visible transitions and moves in log) have the same severity. In [4], we show how this assumption can be removed.

4 Encoding the Alignment Problem in PDDL

Planning systems are problem-solving algorithms that operate on explicit representations of states and actions [6]. PDDL [7] is the standard Planning Domain Definition Language; it allows us to formulate a *planning problem* $\mathcal{P} = \langle I, G, \mathcal{P}_\mathcal{D} \rangle$, where I is the description of the initial state of the world, G is the desired goal state, and $\mathcal{P}_\mathcal{D}$ is the planning domain. $\mathcal{P}_\mathcal{D}$ is built from a set of *predicates* and *fluent* describing the state of the world and a set of *actions* that can be executed in the domain.

This section illustrates how, given an event log L and Petri net $N = (P, T, F)$ with given initial and final markings 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 plan synthesized will consist of a sequence of alignment moves that establish an alignment between σ_L and N. As throughout discussed in [4], the proposed encoding of an alignment problem A as planning problem $\mathcal{P}$ is such that one can find a solution for A by solving $\mathcal{P}$. Also, planning algorithms always terminate when the input is a planning problem $\mathcal{P}$ that encodes A according to the procedure proposed in this paper.

In the remainder of this paper, we assume Petri nets to be 1-bounded. This is not a large limitation as the behavior allowed by most of business processes can be represented as 1-bounded Petri nets (see discussion in [4]).

4.1 Predicates and fluents

In the planning domain $\mathcal{P}_\mathcal{D}$, 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 $\mathcal{P}$:

```
(:objects a b c d1 e 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 $t = e$.

In addition to the predicates above, we introduce a fluent **total-cost** to keep track of the number of deviations found in the alignment till that point. 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 d)
        (associated e3 b) (associated e4 c) (= (total-cost) 0))
```

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))`.

4.2 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 visible transition $t \in T$ 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): 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: one 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) 1)))
```

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 1, since a is a visible transition[4]. The effects are accordant to the firing rules: 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.

For what concerns invisible transitions, readers should observe that since the executions of invisible transitions are never recorded in event logs, invisible transitions are always involved in move in models, only. However, these model moves are actually not deviations and, hence, the action `moveInTheModel` does not have `(increase (total-cost) 1)` among the effects. E.g., for transition τ in the model in Fig. 1, action `moveInTheModel-tau` will not have the total-cost increase as an effect.

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

[4] For the sake of simplicity, we are assuming a unitary cost to express the severity of a move in model and/or in log.

	10% noise				20% noise				30% noise			
Petri Net Size	Existing Approach	Fast Downward	SymBA-2*	SPM&S	Existing Approach	Fast Downward	SymBA-2*	SPM&S	Existing Approach	Fast Downward	SymBA-2*	SPM&S
25	**25.85**	84.79	685.35	812.44	**42.8**	86.21	692.97	814.04	**56.87**	88.87	699.42	816.05
36	**11.71**	94.13	767.05	950.25	**19.24**	92.78	785.14	956.19	**28.6**	92.67	793.13	959.25
68	**112.85**	130.68	1,413.85	1,722.13	164.92	**134.57**	1,418.51	1,779.17	278.31	**144.99**	1,419.5	1,808.61
95	**77.56**	123.65	2,048.84	2,431.35	**119.89**	125.78	2,049.14	2,445.66	168.32	**126.82**	2,053.92	2,483.98
115	638.52	**205.47**	2,289.84	2,787.31	966.34	**384.26**	2,291.91	2,817.29	1,987.12	**716.68**	2,352.81	2,868.77
136	304.62	**178.66**	2,752.28	3,282.23	465.97	**182.88**	2,765.14	3,300.87	578.63	**187.17**	2,775.46	3,317.48
175	–	18,759.91	**5,909.46**	6,546.87	–	84,195.41	**6,092.93**	6,837.57	–	$1.71 \cdot 10^5$	**6,259.99**	7,182.13
263	–	**2,343.92**	11,960.84	13,289.56	–	13,524.63	**12,044.67**	13,744.92	–	30,582.11	**12,194.09**	14,075.09

Table 1. Comparison of the experimental results with different planners and with the existing approach of [2], using different models while varying the amount of noise. The values refers to the average time (in ms.) to compute the alignment of a log trace and the respective process model.

```
(:action moveInTheLog
 :parameters (?e1 - event ?e2 - event)
 :precondition (and (tracePointer ?e1) (succ ?e1 ?e2))
 :effect (and (not (tracePointer ?e1))(tracePointer ?e2) (increase (total-cost) 1 )))
```

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 1.

5 Evaluation and Conclusion

The planning-based alignment tool is publicly available as Java application. Implementation details and information about how to download it are available in [4]. To evaluate the versatility of approach to seamlessly integrate different planners, we employed three different planners: Fast Downward[5], SymBA-2* and SPM&S[6]. These three planning systems were employed with searching algorithms that guarantee the optimality of the solution. Therefore, the returned alignments are always guaranteed to be optimal.

To evaluate the scalability and feasibility of the approaches, we tested our implementation with several combinations of real-life and synthetic event logs and processes. We refer to [4] for a thorough discussions of the experimental test bed and of the results. Here, for the sake of space, we focus on the results on the synthetic process models and event logs. Specifically, we artificially generated different process models of increasing sizes in form of Petri nets to evaluate the scalability of the approach and, for each Petri net, we generated artificial event logs with increasing amount of noise. Established techniques and tools exist for this purpose (see details in [4]). The entire set of artificial models is attached as appendix in [4].

In the remainder, an event log with X% of noise indicates that, starting from an event log where each trace is conforming, an event is swapped with the next event in the trace with a probability of X%. This swapping generally causes the trace to be no more conforming. Note also that multiple events may be swapped in a trace and, also, the same event can be swapped with consecutive events, moving it further apart from the original position in the trace.

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

[6] SymBA-2* and SPM&S can be downloaded at: `https://helios.hud.ac.uk/scommv/IPC-14/planners_actual.html`

The experiments with these artificial process models are summarized in Table 1. Each row refers to a different Petri net. For each row, the first column indicates the size of the Petri net in term of number of transitions; the subsequent 12 columns refer to the experimental results when the corresponding event log contain 10%, 20% and 30% noise. For each amount of noise, we compared the result of the existing ad-hoc approach by Adriansyah et al. [2] with the results obtained by our approach when integrated with the three planners mentioned above.

The values refer to the average time to alignment and event-log trace with the respective process model. The results show that the existing ad-hoc approach is faster with models of small sizes and/or when using event logs with smaller amount of noise. Conversely, our approach is faster with larger models. In particular, it seems that, generally speaking, Fast Downward performs better with middle-size models whereas SymBA-2* works better with very large models.

To conclude, the existing approach by Adriansyah et al. [2] is confirmed not to sufficiently scale when event logs and process models become very large. As a matter of fact, when testing with the two largest models, the existing approach was not able to align every event-log trace with the corresponding model and was running out of memory, even though we were using a machine with 16 GBs of RAM. Conversely, tracking the memory consumption, all of three employed planners were never using more than 4 GBs of RAM. Even when the existing approach was able to carry out the alignment tasks, it could do it significantly slower. As an example, compare the results for the model with 115 activities and an event log with 30%: On average, the existing approach requires 1987 ms per trace, versus 716 ms for Fast Downward. A saving of 1.2 seconds per trace means that, to align all traces of an event log with, e.g., 100000 traces (not uncommon in real-world case studies), the use of Fast Downward would allow us to save 120000 seconds: 83.3 hours.

References

1. van der Aalst, W.M.P.: Data Science in Action. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
2. Adriansyah, A., Sidorova, N., van Dongen, B.F.: Cost-based fitness in conformance checking. In: Proc. of Int. Conf. on Application of Concurrency to System Design, IEEE (2011) 57–66
3. Van Der Aalst, W., et al.: Process Mining Manifesto. In: Proc. of the 9th Int. Conf. on Business Process Management (BPM 2011), Springer Berlin Heidelberg (2011) 169–194
4. de Leoni, M., Marrella, A.: Aligning Real Process Executions and Prescriptive Process through Automated Planning. Expert System with Applications **82** (2017) 162 – 183
5. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems, and Computers **8**(1) (1998) 21–66
6. Ghallab, M., Nau, D.S., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann (2004)
7. McDermott, D., Ghallab, M., Howe, A., Knoblock, C.A., Ram, A., Veloso, M., Weld, D.S., Wilkins, D.E.: PDDL—The Planning Domain Definition Language. Technical Report DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, Connecticut (1998)