Structural Computation of Alignments of Business Processes over Partial Orders

Farbod Taymouri
The University of Melbourne
Melbourne, Australia
farbod.taymouri@unimelb.edu.au

Josep Carmona
Universitat Politècnica de Catalunya
Barcelona, Spain
jcarmona@cs.upc.edu

***Abstract*—Relating event data and process models is becoming an important element for organizations. This paper presents a novel approach for aligning traces and process models. The approach is based on the structural theory of Petri nets (the marking equation), applied over an unfolding of the initial process model. Given an observed trace, the approach adopts an iterative optimization mechanism on top of the unfolding, computing at each iteration part of the resulting alignment. In contrast to the previous work that is primarily grounded in the marking equation, this approach is guaranteed to provide real solutions, and tries to mimic as much as possible the events observed in the trace. Experiments witness the significance of this approach both in quality and execution time perspectives.**

***Index Terms*—Conformance Checking; Alignment; Petri Net; Optimization;**

I. Introduction

Conformance checking, by its promise of detecting and explaining deviations of business processes, is gaining attention due to the impact it may have in organizations [1]. On its core, conformance checking techniques rely on replaying event data, materialized in the form of *event logs*, on top of process models to identify discrepancies between them.

Although this replay can be done heuristically, to be confident that the corresponding trace in the process model adheres optimally to the observed trace, the notion of *alignment* was proposed [2]: given an observed trace σ and a process model, an optimal alignment, or simply alignment, provides the closest model trace (e.g., in terms of edit distance) to σ. The computation of alignments has proven to be a computational challenge, illustrated by the number of techniques that have appeared in the last years (see next section for a detailed literature review). Examples of strategies followed in the literature to compute alignments range from state-based/automata approaches, structural techniques, approaches based on planning, symbolic representations and decomposition methods, among others.

In this paper we propose a novel method to compute alignments, which is based on using a partial order representation of the input process model, in combination with a well-known result on structural theory of Petri nets. By using an unfolding of the process model which is acyclic, our method - grounded in the *marking equation* of Petri nets - is free from spurious solutions, thus guaranteeing to obtain a real solution without explicitly traversing the state-space of the process model. When compared to the state-of-the art techniques in the literature, a significant reduction in computation time has been observed, making the proposed approach a good candidate in case of large problem instances.

Moreover, although the approach is not guaranteed to be optimal in the classical sense (minimal number of mismatches), we show that when the focus is instead to maximize the number of replayed events that have been observed, our approach provides solutions that are often significantly better than the aforementioned techniques.

The structure of this paper is as follows: next section provides the context for the work of this paper. Sect. III introduces the necessary background to understand the contribution of this paper, which is described in Sect. IV. An experimental evaluation is reported in Sec. V. Sect. VI concludes the paper and provides pointers for future work.

II. Related Work

The work in [2] proposed the notion of alignment, and developed a technique to compute optimal alignments for a particular class of process models. For each trace σ, the approach consists on exploring the synchronous product of model's state space and σ. In the exploration, the shortest path is computed using the A^* algorithm, once costs for model and log moves are defined. The approach represents the state-of-the-art technique for computing alignments, and can be adapted (at the expense of increasing significantly the memory footprint) to provide all optimal alignments.

Alternatives to the A^* have appeared very recently: in the approach presented in [3], the alignment problem is mapped as an *automated planning* instance. Unlike the A^*, the aforementioned work is only able to produce one optimal alignment (not all optimal), but it is expected to consume considerably less memory. Automata-based techniques have also appeared [4], [5]. In particular, the technique in [4] can compute all optimal alignments. The technique in [4] relies on state space exploration and determinization of automata, whilst the technique in [5] is based on computing several subsets of activities and projecting the alignment instances accordingly.

The work in [6], presented the notion of *approximate* alignment to alleviate the computational demands of the current challenge by proposing a recursive paradigm on the basis of structural theory of Petri nets. In spite of resource efficiency, the solution is not guaranteed to be executable. A follow-up work of [6] is presented in [7], which proposes

 DOI 10.1109/ACSD.2019.00012

a trade-off between complexity and optimality of solutions, and guarantees executable properties of results. The technique in [8], presents a framework to reduce a process model and the event log accordingly, with the goal to alleviate the computation of alignments. The obtained alignment, which is called *macro-alignment* since some of the positions are high-level elements, is expanded based on the gathered information during the initial reduction. Decompositional techniques have been presented [9], [10] that instead of computing optimal alignments, they focus on the *decisional problem* of whereas a given trace fits or not a process model.

Recently, two different approaches from the same authors have appeared; the work in [11] proposes using binary decision diagrams to alleviate the computation of alignments. The work in [12], which has the goal of maximizing the synchronous moves of the computed alignments, uses a pre-processing step on the model.

III. Preliminaries

A. Petri nets and Process Mining

A *Petri Net* [13] is a 3-tuple $N = \langle P, T, \mathcal{F} \rangle$, where P is the set of places, T is the set of transitions, $P \cap T = \emptyset$, $\mathcal{F} : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$ is the flow relation. A *labeled Petri net (LPN)* is a 3-tuple $\langle N, \Sigma, \ell \rangle$, where N is a Petri net, Σ is an alphabet (a set of labels) and $\ell : T \rightarrow \Sigma \cup \{\tau\}$ is a *labeling function* that assigns to each transition $t \in T$ either a symbol from Σ or the empty symbol τ. The set of labeled transitions is represented by T_ℓ. A marking is an assignment of non-negative integers to places. If k is assigned to place p by marking m (denoted $m[p] = k$), we say that p is marked with k tokens. Given a node $x \in P \cup T$, its pre-set and post-set (in graph adjacency terms) are denoted by ${}^\bullet x$ and $x^\bullet$ respectively.

A transition t is *enabled* in a marking m when all places in ${}^\bullet t$ are marked. More formally it is denoted by $(N, m)[t\rangle$, iff ${}^\bullet t \leq m$. When a transition t is enabled, it can *fire* or *execute* by removing a token from each place in ${}^\bullet t$ and putting a token to each place in $t^\bullet$. A marking m' is *reachable* from m if there is a sequence of firings $t_1 t_2 \dots t_n$ that transforms m into m', denoted by $m[t_1 t_2 \dots t_n \rangle m'$. A sequence of transitions $t_1 t_2 \dots t_n$ is a *feasible sequence* if it is firable from the initial marking m_0.

A Petri net is called *live* if no matter what marking has been reached every transitions of the model can be fired through some firing sequences. Also it is said to be *k-bounded* if none of the places can have more than k tokens for any reachable marking. In this paper we assume 1-bounded Petri nets.

Workflow processes can be represented in a simple way by using Workflow Nets (WF-nets). A WF-net is a Petri net where there is a place $start$ (denoting the initial state of the system) with no incoming arcs and a place end (denoting the final state of the system) with no outgoing arcs, and every other node is within a path between $start$ and end. The transitions in a WF-net represent tasks. In conjunction with the 1-boundedness assumption, this paper assumes process models are specified by *weakly sound* WF-nets [14], for which every reachable marking has the option to reach the unique final marking, and there are no dead transitions.

Definition 3.1 (System Net, Full Firing Sequences): A system net is a tuple $SN = (N, m_{start}, m_{end})$, where N is a WF-net and the two last elements define the initial and final marking of the net, respectively. The set $\{\sigma \mid (N, m_{start})[\sigma\rangle(N, m_{end})\}$ denotes all the full firing sequences of SN.

We now turn the focus to event logs and traces:

Definition 3.2 (Trace, Event Log, Parikh vector): Given an alphabet of events $\Sigma = \{a_1, \dots, a_n\}$, a trace is a word $\sigma \in \Sigma^*$ that represents a finite sequence of events. An *event log* $L \in \mathcal{B}(\Sigma^*)$ is a multiset of traces and $|\sigma|_a$ represents the number of occurrences of a in σ. The Parikh vector of a sequence of events σ is a function $\widehat{}: \Sigma^* \rightarrow \mathbb{N}^n$ defined as $\widehat{\sigma} = (|\sigma|_{a_1}, \dots, |\sigma|_{a_n})$. For simplicity, we will also represent $|\sigma|_{a_i}$ as $\widehat{\sigma}[a_i]$. The support of a Parikh vector $\widehat{\sigma}$, denoted by $\text{supp}(\widehat{\sigma})$ is the set $\{a_i | \widehat{\sigma}[a_i] > 0\}$. For a trace σ, $\sigma[1], \sigma[2], .., \sigma[k]$ denote its first, second and k^{th} elements respectively. For two Parikh vectors $\widehat{\sigma_1}$ and $\widehat{\sigma_2}$, $\widehat{\sigma_1} \leqq \widehat{\sigma_2}$ means that each component of the former is less than or equal to each corresponding component of the later.

The main metric in this paper to assess the adequacy of a model in describing a log is *fitness* [15], which is based on the reproducibility of a trace in a model:

Definition 3.3 (Fitting Trace): A trace $\sigma \in \ell^*$ fits $SN = (N, m_{start}, m_{end})$ if σ coincides with a full firing sequence of SN, i.e.,$(N, m_{start})[\sigma\rangle(N, m_{end})$.

B. Petri net Unfoldings

A *finite and complete unfolding prefix* π of a Petri net N is a finite acyclic net which implicitly represents all the reachable states of N, together with transitions enabled at those states. Intuitively, it can be obtained through unfolding N, by successive firings of transitions, under the following assumptions: (1) for each new firing a fresh transition (called an *event*) is generated; (2) for each newly produced token a fresh place (called a *condition*) is generated. The unfolding is infinite whenever N has an infinite run; however, if N has finitely many reachable states, then the unfolding eventually starts to repeat itself and can be truncated (by identifying a set of *cut-off* events) without loss of information, yielding a finite and complete prefix. We denote by B, E and $E_{cut} \subseteq E$ the sets of conditions, events and cut-off events of the prefix, respectively. Associated with an unfolding, there is an homomorphism $\rho : B \cup E \rightarrow P \cup T$ that associates conditions and events of the unfolding to places and transitions of the Petri net. Fig. 1(d) shows a finite and complete unfolding prefix of the Petri net shown in Fig. 1(a).

Efficient algorithms exist for building such prefixes [16]–[19], which ensure that the number of non-cut-off events in a complete prefix can never exceed the number of reachable states of N. However, complete prefixes are often exponentially smaller than the corresponding state graphs, especially for highly concurrent Petri nets, because they represent concurrency directly rather than by multidimensional "diamonds"

as it is done in state graphs. For example, if the original Petri net consists of 100 transitions which can fire once in parallel, the state graph will be a 100-dimensional hypercube with 2^{100} nodes, whereas the complete prefix will coincide with the net itself. Due to its structural properties (such as acyclicity), the reachable markings of N can be represented using *configurations* of π. A configuration C is a downward-closed set of events (being downward-closed means that if $e \in C$ and f is a causal predecessor of e, then $f \in C$) without structural conflicts (i.e., for all distinct events $e, f \in C$, ${}^\bullet e \cap {}^\bullet f = \emptyset$). Intuitively, a configuration is a partial-order execution, i.e., an execution where the order of firing of concurrent events is not important.

C. Petri nets and Linear Algebra

Let $N = \langle P, T, \mathcal{F} \rangle$ be a Petri net with initial marking m_0. Given a feasible sequence $m_0[\sigma\rangle m$, the number of tokens for a place p in m is equal to the tokens of p in m_0 plus the tokens added by the input transitions of p in σ minus the tokens removed by the output transitions of p in σ:

$$m[p] = m_0[p] + \sum_{\forall t \in {}^\bullet p} |\sigma|_t \ \mathcal{F}(t,p) - \sum_{\forall t \in p^\bullet} |\sigma|_t \ \mathcal{F}(p,t)$$

The marking equations for all the places in the net can be written in the following matrix form (see Fig. 1(c)): $m = m_0 + \mathbf{N} \cdot \widehat{\sigma}$, where $\mathbf{N} \in \mathbb{Z}^{P \times T}$ is the *incidence matrix* of the net: $\mathbf{N}[p,t] = \mathcal{F}(t,p) - \mathcal{F}(p,t)$. If a marking m is reachable from m_0, then there exists a sequence σ such that $m_0[\sigma\rangle m$, and the following system of equations has at least the solution $X = \widehat{\sigma}$

$$m = m_0 + \mathbf{N} \cdot X \tag{1}$$

If (1) is infeasible, then m is not reachable from m_0. The inverse does not hold in general: there are markings satisfying (1) which are not reachable. Those markings (and the corresponding Parikh vectors) are said to be *spurious* [20]. Fig. 1(a)-(c) presents an example of a net with spurious markings: the Parikh vector $\widehat{\sigma} = (2,1,0,0,1,0)$ and the marking $m = (0,0,1,1,0)$ are a solution to the marking equation, as is shown in Fig. 1(c). However, m is not reachable by any feasible sequence. Fig. 1(b) depicts the graph containing the reachable markings and the spurious markings (shadowed). The numbers inside the states represent the tokens at each place $(p_1, \ldots, p_5)$. This graph is called the *potential reachability graph*. The initial marking is represented by the state $(1,0,0,0,0)$. The marking $(0,0,1,1,0)$ is only reachable from the initial state by visiting a negative marking through the sequence $t_1 t_2 t_5 t_1$, as shown in Fig. 1(b). Therefore, equation (1) provides only a necessary condition for reachability of a marking and replayability for a solution of (1).

As unfoldings are acyclic Petri nets, one can use the results from [13] to show a fundamental property that serves as key idea of this paper:

Theorem 3.1 (Marking Equation for Acyclic Petri nets [21]): Let N be an acyclic Petri net. If the vector y satisfies the equation $m = m_0 + \mathbf{N} \cdot X$, then there exists a firing sequence σ firable from marking m_0 such that $y = \widehat{\sigma}$.

D. Alignment of Observed Behavior

As outlined above, the fitness dimension requires an *alignment* of an observed trace and a model: events of the observed trace need to be related to elements of the model and vice versa. Such an alignment reveals how the given trace can be replayed on the process model. The classical notion of aligning an event log and process model was introduced by [2]. To achieve an alignment, we need to relate *moves* in the observed trace to *moves* in the model. It may be the case that some of the moves in the observed trace can not be mimicked by the model and vice versa. For instance, consider the model N_1 in Fig. 2, with the following labels, $\ell(t_1) = a_1, \ell(t_2) = a_2, \ell(t_3) = a_3$ and $\ell(t_4) = a_4$, and the trace $\sigma = a_1 a_1 a_4 a_2$; four possible alignments are:

α_1=

a_1	a_1	$\perp$	a_4	a_2
t_1	$\perp$	t_3	t_4	$\perp$

α_2=

a_1	a_1	$\perp$	a_4	a_2
$\perp$	t_1	t_2	t_4	$\perp$

α_3=

a_1	a_1	a_4	a_2	$\perp$
t_1	$\perp$	$\perp$	t_2	t_4

α_4=

a_1	a_1	a_4	a_2	$\perp$
$\perp$	t_1	$\perp$	t_2	t_4

The moves are represented in tabular form, where moves by the trace are at the top, and moves by the model are at the bottom of the table. For example the first move in α_2 is $(a_1, \perp)$ and it means that the observed trace moves a_1, while the model does not make any move. Formally, an alignment is defined as follows:

Definition 3.4 (Alignment): Given a labeled Petri net N and an alphabet of events Σ, Let A_M and A_L be the alphabet of transitions in the model and events in the log, respectively, and $\perp$ denotes no movement, then:

- (X, Y) is a *synchronous move* if $X \in A_L$, $Y \in A_M$ and $X = \ell(Y)$
- (X, Y) is a *move in log* if $X \in A_L$ and $Y = \perp$.
- (X, Y) is a *move in model* if $X = \perp$ and $Y \in A_M$.
- (X, Y) is an *illegal move*, otherwise.

The set of all *legal* moves is denoted as A_{LM} and given an alignment $\alpha \in A_{LM}^*$, the projection of the first element (ignoring $\perp$), $\alpha \downarrow_{A_L}$, results in the observed trace σ, and projecting the second element (ignoring $\perp$), $\alpha \downarrow_{A_M}$, results in the model trace.

For the previous example, $\alpha_1 \downarrow_{A_M} = t_1 t_3 t_4$ and $\alpha_1 \downarrow_{A_L} = a_1 a_1 a_4 a_2$. In this paper we focus on the computation of the model trace given an observed trace. An alignment can be constructed afterwards, e.g., by using classical sequence alignment techniques like [22].

Cost can be associated to the different types of moves in Def. 3.4. Traditionally, the approaches in the literature use a cost function that assigns higher costs to asynchronous moves (move in model/log) than to synchronous moves, and the

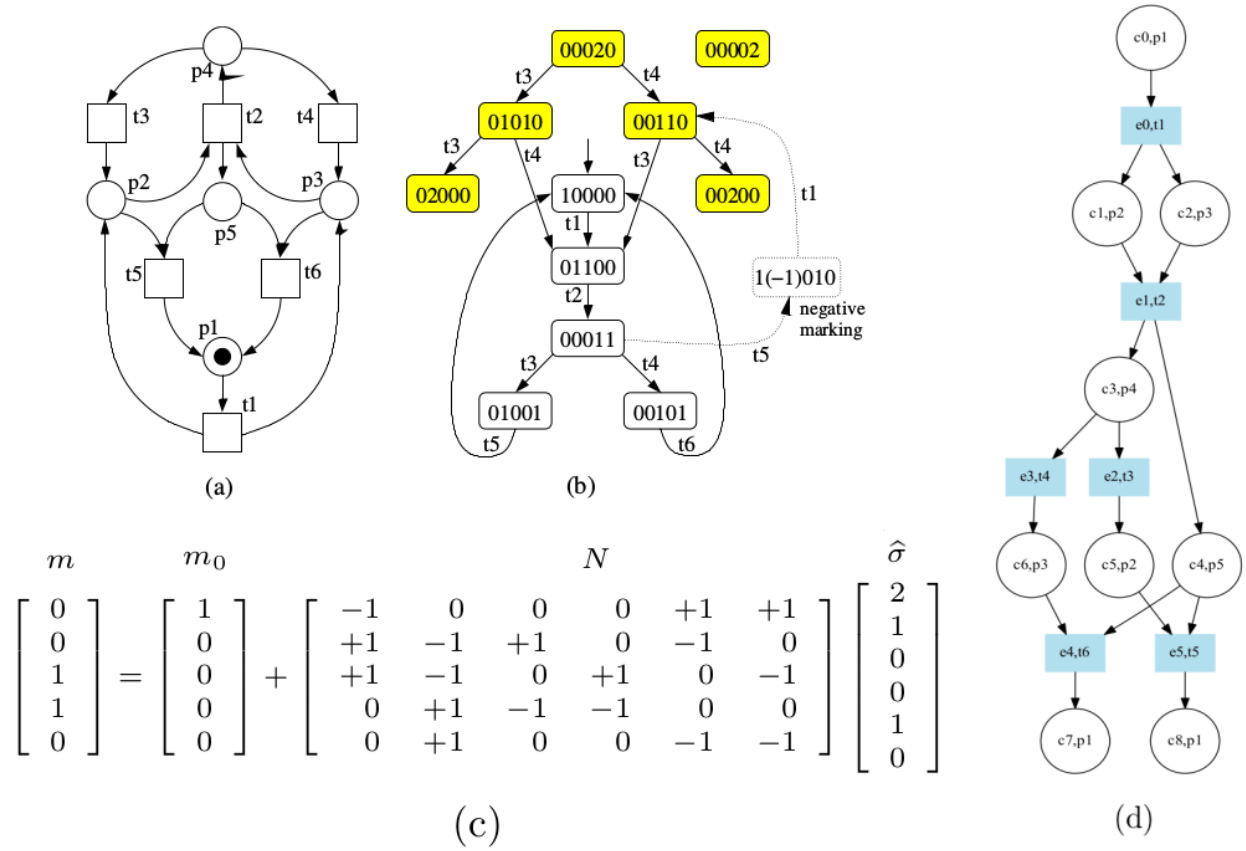

Figure 1. (a) Petri net, (b) Potential reachability graph, (c) Marking equation, (d) Unfolding

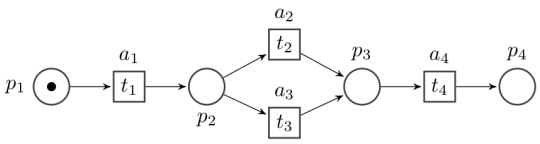

Figure 2. Process model N_1.

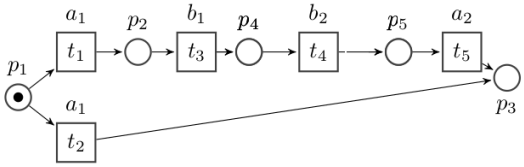

Figure 3. Process model N_2.

model trace that minimizes the cost (hence, minimizing the number of asynchronous moves) is computed.

Recently, a different focus has been proposed for alignment computation, where the maximization of the synchronous moves is considered [12]. The underlying idea of maximizing synchronous moves is to try to adhere as much as possible to the observed trace, instead of minimizing the penalties assigned to asynchronous moves. This can be illustrated with the help of a simple example. For the model of Fig. 3, and the trace a_1a_2, the minimization of asynchronous moves would result in the model trace $a_1(t_2)$, with an alignment containing one synchronous and one asynchronous log move. Instead, the maximization of synchronous moves will provide the model trace $a_1b_1b_2a_2$, with an alignment containing 2 synchronous moves and 2 asynchronous moves. Alignments are shown below.

$\alpha_1 = \begin{array}{|c|c|}\hline a_1 & a_2 \\ \hline a_1 & \bot \\ \hline \end{array}$, $\alpha_2 = \begin{array}{|c|c|c|c|}\hline a_1 & \bot & \bot & a_2 \\ \hline a_1 & b_1 & b_2 & a_2 \\ \hline \end{array}$

IV. Unfolding-Based Alignment Computation over the Marking Equation

This section centers around computing a modeled trace in an iterative way, given an observed trace. It is worth to stress that the approach proposed in this paper assumes the computation of an unfolding of the process model, which should be done just once. The mechanics of this iterative optimization are simple, namely, in each iteration the aim is to maximize the number of synchronous moves that would result on aligning a subset of the obtained model trace and the corresponding part of the observed trace.

Let us assume that there is an unfolding π of the Petri net N and an observed trace σ. Let $J = \ell(\Sigma) \cap \text{supp}(\widehat{\sigma})$, i.e., the labels that appeared in the observed trace; the following ILP model, which applies the marking equation on π and using as markings the initial conditions of the unfolding, computes a

solution, i.e., a selection of events in the unfolding π, that is as similar as possible with respect to the firing of some of the activities appearing in the observed trace:

$$\text{Maximize} \quad (2)$$
$$\Big(\sum_{\ell(e)\in J} X[e] - \delta \times \sum_{\ell(e)\notin J} X[e] + 0 \times \sum_{\ell(e)=\tau} X[e]\Big),$$
$$\text{Subject to:}$$
$$m_i = m_j + \mathbf{N}_\pi . X$$
$$\forall e \in X, \forall a \in \hat{\sigma} \quad \text{If} \quad \ell(e) \in J \quad \text{and} \quad \ell(e) = a :$$
$$\widehat{\sigma}[a] = \sum_{\ell(e)=a} (X[e] + X^s[e]),$$
$$\sum_{\forall \ell(e)\in E_{cut}} X[e] \geq 1,$$
$$X, X^s \geq \mathbf{0}$$

Where $\mathbf{N}_\pi$ denotes the incidence matrix of the unfolding π. In the above optimization, m_i and m_j are intermediate markings of the unfolding, wherein, the later is the unfolding marking corresponding to m_{start} at the beginning, and the former is the unfolding marking corresponding to m_{end} in the final iteration. X^Ss are slack variables that handle situations where the model is unable to reproduce the same number transition for a given event. It must be noted that at each iteration, at least one of the cut-off events must be fired, so as to allow continuing for next iterations. It is worth mentioning that the final marking will be marked by one of the cut-off events.

Considering Eq. (2), the iterative mechanism is as follows: given the vector solution X, for the next iteration the observed trace is set to $\forall_{e\in J}\widehat{\sigma}[e] = \widehat{\sigma}[e] - X[e]$, i.e., we remove from the observed traces those events that correspond to the unfolding run obtained. Accordingly, for the next iteration, we *recharge* the unfolding, i.e., we mark the unfolding adding a token to those conditions b_k for which there is a final condition b_i that is marked and correspond to the same Petri net place, $B(b_k) = B(b_i)$. For example, consider Fig. 4, if $c8$ is marked then it also recharges $c3$, since $B(c8) = B(c3) = p6$.

The following constraint must be added in each intermediate step, to preserve the semantics of firing rules and to allow the iteration over the unfolding model:

$$\sum_{\forall b_k \in B(b_k)=p_i} b_k = 1 \quad (3)$$

Also, the constraint $\sum_{\forall \ell(e)\in E_{cut}} X[e] \geq 1$ in Eq. (2) tries to fire at least one cut-off transition which results in an intermediate marking according to Eq. (3); it lasts until reaching the final marking where no more intermediate marking is available.

This iteration continues until reaching the final marking of the initial model through the unfolding. However, there are some caveats that might prevent to get the final marking. This is the case when after some iterations, we might end up in an intermediate unfolding marking m_i where no more observed events need to be mimicked. This issue causes an unbounded solution for Eq. (2), due to the lack of negative elements in the objective function. Therefore, in such cases the objective function must be changed so as to come up with a solution. In more detail, to overcome the challenge, in these situations the objective function will be substituted with the following one:

$$\sum_{\forall \ell(e)\in E_{cut}} X[e] + \sum_{e\in path} X[e] \quad (4)$$

Wherein, $path$, denotes the set of unfolding events that are on a path from the unfolding marking corresponding to m_{start} to the unfolding marking corresponding to m_{end}. The latest term in Eq. (4) guides the algorithm to converge towards the final marking. Notice that the $path$ is not necessarily unique. The algorithmic perspective of the mentioned technique is presented in Alg. 1.

Algorithm 1 Unfolding-Based Sequence Computing

```
Input: Unfolding π of Petri net (N, m_start, m_end), Observed trace's Parikh σ̂
σ̂_X ← ∅
m_i ← ρ^{-1}(m_start)
while m_i ≠ ρ^{-1}(m_end) do
    result = Solve Eq. (2)
    if result is bounded then
        σ̂_X ← Concatenate(σ̂_X, X)
        ∀_{e∈J} σ̂[e] = σ̂[e] − X[e],   ▷ Updating Parikh of observed trace
        Update the next initial marking m_i
    else
        Modify and solve the optimization according to Eq. (4)
        σ̂_X ← Concatenate(σ̂_X, X)
        Update the next initial marking m_i
    end if
end while
Return σ̂_X
```

In Alg. 1, $\widehat{\sigma}_X$, shows a sequence of Parikh vectors computed at different iterations. Although the iterative approach just mentioned can provide a solution, two important properties hold. Firstly, we will show that this iterative approach converges and, secondly, the obtained result is guaranteed to be executable.

Lemma 4.1: Given the unfolding model π of a system net (N, m_{start}, m_{end}), Alg. 1 has a finite number of iterations.

Proof: The proof is given by contradiction. Assume that the number of iterations is infinite, i.e., the final marking is never reached. But, it contradicts the soundness property of the given model, given that the observed trace is finite. Therefore, the final marking m_{end} is reachable from every intermediate markings. Moreover, if at any intermediate step there is no observed event that can be replayable in the unfolding, then the final marking will be reached by using the alternative cost function (4). □

Lemma 4.2: Given the unfolding model π of a system net (N, m_{start}, m_{end}), the solution provided by Alg. 1 is executable, i.e., $(N, m_{start})[\sigma_X\rangle(N, m_{end})$.

Proof: Let us suppose that there are k iterations, thus, $\widehat{\sigma}_X$ is the sequence of k Parikh vectors, i.e., $\widehat{\sigma}_X = X_1 X_2 \dots X_k$, where, X_i is the solution of Eq. (2) for ith iteration, and, X_i precedes X_j for $i < j$. Notice that by Theorem 3.1,

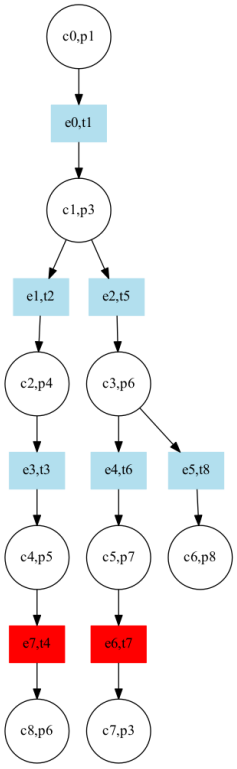

Figure 4. Unfolding example.

any solution to the marking equation on the unfolding is executable, and therefore, any of the vectors X_i is executable in the unfolding and corresponds to a sequence in the original Petri net. We only need to observed that between any contiguous vectors, the recharging of the unfolding does not harm replayability since it preserves the corresponding marking in the Petri net. □

An example helps to drive the concept home. Consider the unfolding in Fig. 4, and the observed trace $\sigma = a_1a_3a_2a_6a_7a_8$ and let us assume that for the original model $\forall t_i \in T, \ell(t_i) = a_i$. The first iteration using Eq. (2), produces the sequence $t_1t_2t_3t_4$ and marked $c8$. Since, $B(c8) = B(c3) = p6$, for the second iteration, $c3$ is marked and considered as the initial marking. This iteration using Eq. (2), produces the sequence t_6t_7 and marks c_7 and due to $B(c7) = B(c1) = p3$, c_1 is marked and considered as the next initial marking. The third iteration, produces t_5t_8 which marks the final marking. The final produced sequence is $t_1t_2t_3t_4t_6t_7t_5t_8$.

V. Experiments

The approach presented in this paper is implemented in Python 2.7 and Gurobi [23] was used as the ILP solver. The experiments are done over different datasets with various characteristics, including different sizes, nested loops, and duplicate transitions. Tables I and II describes the benchmarks. The presented approach in this paper is compared with well-known methods in the literature [2] and [4], on different perspectives that come up next.

Table I
BPM2013 artificial benchmark datasets [9], and synthetic datasets [24]

Model	$\|P\|$	$\|T\|$	$\|Arc\|$	Cases	Fit.	$\overline{\sigma}$
prAm6	363	347	846	1200	No	31
prBm6	317	317	752	1200	Yes	43
prCm6	317	317	752	500	No	43
prDm6	529	429	1140	1200	No	248
prEm6	277	275	652	1200	No	98
prFm6	362	299	772	1200	No	240
prGm6	357	335	826	1200	No	143
M_1	40	39	92	500	No	13
M_2	34	34	80	500	No	17
M_3	108	123	276	500	No	37
M_4	36	52	106	500	No	26
M_5	35	33	78	500	No	34
M_6	69	72	168	500	No	53
M_7	65	62	148	500	No	37
M_8	17	15	36	500	No	17
M_9	47	55	120	500	No	44
M_{10}	150	146	354	500	No	58
Bank.	121	114	272	2000	No	58
Road.	15	23	48	10000	No	3

Table II
Models with duplicate transition names [24]

Model	$\|P\|$	$\|T\|$	$\|Arc\|$	Cases	Fit.	$\overline{\sigma}$	Dup. Trans.
ML_1	27	35	74	500	No	28	2
ML_2	165	177	404	500	No	87	12
ML_3	45	45	106	500	No	26	2
ML_4	36	33	80	500	No	28	6
ML_5	159	172	390	500	No	42	14

a) Unfolding model size:: Table III shows the ratio between the unfolding and the original models in terms of number of places, transitions and arcs. One can see that, unfolding models in general have a slightly greater number of places than the respective original models. This table illustrates that working with unfolding models is not more much expensive than working with original models, and more importantly, contrasts with the aforementioned works that compute alignments at the state-space level.

b) Number of iterations:: Table IV provides in average the number of iterations that each dataset requires to converge. It is clear that, larger models like $prDm6, prEm6$ and $prFm6$ need more iterations. Also, the number of iterations increases for longer traces, and models having many duplicate transitions and including nested loops, see $ML2$, M_3 and M_6.

c) Percentage of reproduced events:: This is one of the crucial properties of the approach presented in this paper. Table V demonstrates in average the percentage of observed events that can be reproduced by the model using the approach in this paper, and state of the art techniques. As this table shows, the method in this paper can often replay more events than the other approaches[1]. For some of the benchmarks (see M_4, ML_3 and ML_4), the difference is quite significant.

[1]For some datasets, state of the art was unable to produce any results due to lack of memory, hence NA is shown instead.

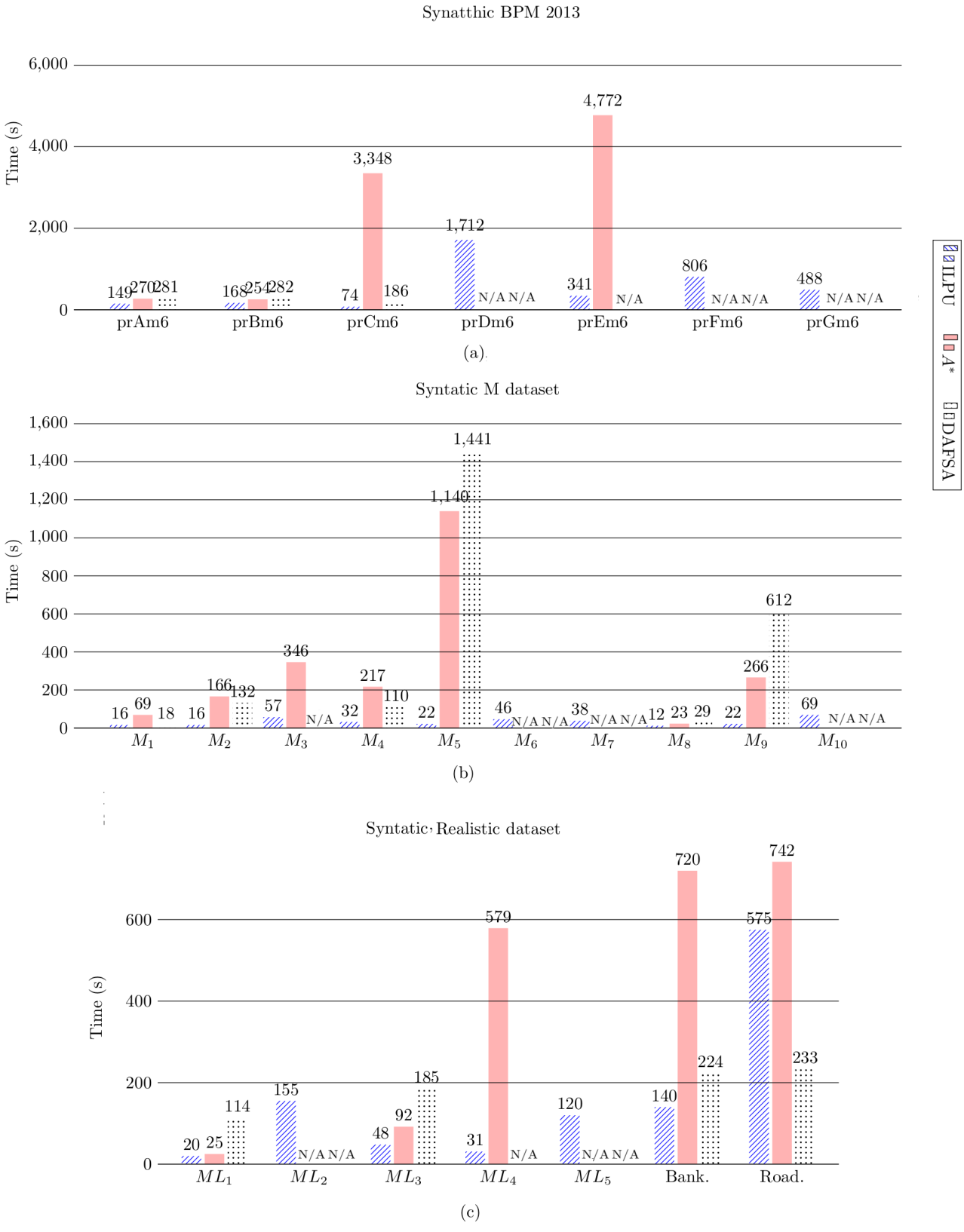

Figure 5. Execution time for the approach in this paper and state of the art techniques

Table III
RATIO OF TRANSITIONS, PLACES AND ARCS NUMBERS BETWEEN UNFOLDING AND THE ORIGINAL MODELS

Model	$\frac{\lvert P_{un.}\rvert}{\lvert P_{or.}\rvert}$	$\frac{\lvert T_{un.}\rvert}{\lvert T_{or.}\rvert}$	$\frac{\lvert Arc_{un.}\rvert}{\lvert Arc_{or.}\rvert}$
prAm6	422/363	363/343	842/846
prBm6	317/317	375/317	748/752
prCm6	317/317	375/317	748/752
prDm6	569/529	429/429	1136/1140
prEm6	325/277	275/275	648/652
prFm6	385/362	299/299	768/772
prGm6	412/357	335/335	822/826
M_1	47/ 40	39/39	92/92
M_2	41/34	34/34	80/80
M_3	139/108	123/123	276/276
M_4	54/36	52/52	106/106
M_5	40/35	33/33	78/78
M_6	85/69	72/72	168/168
M_7	75/65	62/62	148/148
M_8	19/17	15/15	36/36
M_9	61/47	55/55	120/120
M_{10}	178/150	146/146	354/354
ML_1	38/27	35 /35	74/74
ML_2	203/165	177/177	404/404
ML_3	54/45	45/45	106/106
ML_4	41/36	33/33	80/80
ML_5	196/159	172/172	390/390
Road.	25/15	23/23	48/48
Bank.	137/121	114/114	272/272

d) Execution time:: Fig. 5, illustrates the required execution times between the approach in this paper and state of the art techniques. In all models except one (Road), the method in this paper is faster than other techniques in magnitude of time. Also, state of the art techniques are unable to produce solutions for medium and large instance or models with nested loops and duplicate transitions, whereas the presented technique is not very sensitive to these issues.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we provide a novel technique for computing alignments, which both significantly alleviates the computational effort, but also provides often better results when the focus is to explain as much observed events as possible. A prototype implementation has been tested over a well-known set of benchmarks, and initial conclusions show the capabilities of the approach, specially for large problem instance where often the state-of-the-art techniques cannot handle.

As future work, we plan to explore more the role of partial order representations for the problem of alignment computation. In particular, to investigate alternative strategies to the one described in this paper, so that different goals can be defined, depending on the user requirements.

Table IV
AVERAGE NUMBER OF ITERATIONS FOR A MODELED TRACE COMPUTATION

Model	$\overline{It.}$	Model	$\overline{It.}$
prAm6	4.15	M1	3.96
prBm6	5.07	M2	4.14
prCm6	5.08	M3	10.27
prDm6	28.42	M4	7.49
prEm6	10.34	M5	5.92
prFm6	15.61	M6	10.42
prGm6	9.49	M7	8.39
Bank.	8.11	M8	3.81
ML1	5.14	M9	4.44
ML2	20.11	M10	10.27
ML3	9.54	ML4	9.67
ML5	2.78	Road.	8.18

Table V
PERCENTAGE OF REPRODUCED EVENTS

Model	Unfolding	$A*$	Model	Unfolding	$A*$
prAm6	0.91	0.95	M1	1.00	0.77
prBm6	0.84	1.00	M2	1.00	0.71
prCm6	0.86	0.71	M3	0.98	0.92
prDm6	0.89	NA	M4	0.92	0.50
prEm6	0.78	0.98	M5	0.98	0.77
prFm6	0.95	NA	M6	0.98	NA
prGm6	0.83	NA	M7	0.97	NA
Bank.	0.78	0.99	M8	1.00	0.71
ML1	0.69	0.63	M9	0.63	0.73
ML2	0.96	NA	M10	0.95	NA
ML3	0.97	0.47	ML4	0.99	0.48
ML5	0.90	NA	Traffic.	0.68	0.58

Acknowledgments Part of this work was conducted while the first author was at Universitat Politècnica de Catalunya. This research is partly funded by the Australian Research Council (DP150103356), and, Spanish funds MINECO and FEDER (TIN2017-86727-C2-1-R).

REFERENCES

[1] J. Carmona, B. F. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking - Relating Processes and Models*. Springer, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-99414-7

[2] A. Adriansyah, "Aligning observed and modeled behavior," Ph.D. dissertation, Technische Universiteit Eindhoven, 2014.

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

[4] D. Reißner, R. Conforti, M. Dumas, M. L. Rosa, and A. Armas-Cervantes, "Scalable conformance checking of business processes," in *OTM CoopIS, , Rhodes, Greece*, 2017, pp. 607–627.

[5] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Scalable process discovery and conformance checking," *Software and System Modeling*, vol. 17, no. 2, pp. 599–631, 2018.

[6] F. Taymouri and J. Carmona, "A recursive paradigm for aligning observed behavior of large structured process models," in *14th International Conference of Business Process Management (BPM), Rio de Janeiro, Brazil, September 18 - 22*, 2016.

[7] B. F. van Dongen, J. Carmona, T. Chatain, and F. Taymouri, "Aligning modeled and observed behavior: A compromise between computation complexity and quality," in *Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings*, 2017, pp. 94–109. [Online]. Available: https://doi.org/10.1007/978-3-319-59536-8_7

[8] F. Taymouri and J. Carmona, "Model and event log reductions to boost the computation of alignments," in *Proceedings of the 6th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2016), Graz, Austria, December 15-16, 2016.*, 2016, pp. 50–62.

[9] J. Munoz-Gama, J. Carmona, and W. M. P. Van Der Aalst, "Single-entry single-exit decomposed conformance checking," *Inf. Syst.*, vol. 46, pp. 102–122, Dec. 2014.

[10] W. M. P. van der Aalst, "Decomposing Petri nets for process mining: A generic approach," *Distributed and Parallel Databases*, vol. 31, no. 4, pp. 471–507, 2013.

[11] V. Bloemen, J. van de Pol, and W. M. P. van der Aalst, "Symbolically aligning observed and modelled behaviour," in *18th International Conference on Application of Concurrency to System Design, ACSD, Bratislava, Slovakia, June 25-29*, 2018, pp. 50–59.

[12] V. Bloemen, S. J. van Zelst, W. M. P. van der Aalst, B. F. van Dongen, and J. van de Pol, "Maximizing synchronization for aligning observed and modelled behaviour," in *Business Process Management - 16th International Conference, BPM, Sydney, NSW, Australia*, 2018, pp. 233–249.

[13] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–574, Apr. 1989.

[14] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn, "Soundness of workflow nets: classification, decidability, and analysis," *Formal Asp. Comput.*, vol. 23, no. 3, pp. 333–363, 2011.

[15] W. M. P. van der Aalst, *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[16] V. Khomenko and M. Koutny, "Towards an efficient algorithm for unfolding Petri nets," in *CONCUR 2001 - Concurrency Theory, 12th International Conference, Aalborg, Denmark, August 20-25, 2001, Proceedings*, 2001, pp. 366–380.

[17] V. Khomenko, M. Koutny, and W. Vogler, "Canonical prefixes of Petri net unfoldings," in *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, 2002, pp. 582–595.

[18] K. Heljanko, V. Khomenko, and M. Koutny, "Parallelisation of the Petri net unfolding algorithm," in *Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings*, 2002, pp. 371–385.

[19] J. Esparza, S. Römer, and W. Vogler, "An improvement of mcmillan's unfolding algorithm," *Formal Methods in System Design*, vol. 20, no. 3, pp. 285–310, 2002.

[20] M. Silva, E. Teruel, and J. M. Colom, "Linear algebraic and linear programming techniques for the analysis of place/transition net systems." in *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, Reisig, W. and Rozenberg, G., Eds. Springer-Verlag, 1998, vol. 1491, pp. 309–373.

[21] A. Giua, C. Seatzu, and D. Corona, "Marking estimation of petri nets with silent transitions," *IEEE Trans. Automat. Contr.*, vol. 52, no. 9, pp. 1695–1699, 2007.

[22] "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443 – 453, 1970.

[23] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2016. [Online]. Available: http://www.gurobi.com

[24] F. Taymouri and J. Carmona, "An evolutionary technique to approximate multiple optimal alignments," in *16th International Conference of Business Process Management (BPM), Sydney, Australia. September 9-14, Brazil, September 18 - 22*, 2018.