Model and Event Log Reductions to Boost the Computation of Alignments

Farbod Taymouri(✉) and Josep Carmona

Universitat Politècnica de Catalunya, Barcelona, Spain
{taymouri,jcarmona}@cs.upc.edu

Abstract. The alignment of observed and modeled behavior is a pivotal issue in process mining because it opens the door for assessing the quality of a process model, as well as the usage of the model as a precise predictor for the execution of a process. This paper presents a novel technique for reduction of a process model based on the notion of *indication*, by which, the occurrence of an event in the model reveals the occurrence of some other events, hence relegating the later set as less important information when model and log alignment is computed. Once indications relations are computed in the model, both model and log can be reduced accordingly, and then fed to the state of the art approaches for computing alignments. Finally, the (macro)-alignment derived is expanded in these parts containing high-level events that represent a set of indicated events, by using an efficient algorithm taken from bioinformatics that guarantees optimality in the local parts of the alignment. The implementation of the presented techniques shows a significant reduction both in computation time and in memory usage, the latter being a significant barrier to apply the alignment technology on large instances.

Keywords: Process mining · Conformance checking · Alignment
SESE · Model abstraction

1 Introduction

Nowadays many systems generate *event logs*, which are footprints left by process executions. *Process mining* delves into this information and examines it to extract, analyze and enhance evidence-based process models [15]. One of the challenges in process mining is how to align a process model to a set of traces forming an event log. Given a trace representing a real process execution, an *optimal alignment* provides the best trace the process model can provide to imitate the observed trace [1]. Alignments are crucial for important metrics like fitness, precision and generalization [1,2].

This paper presents a model-based technique for reduction of a process model and observed behavior that both preserves the semantics of the process model and retains the information of the original observed behavior as much as possible. The technique is meant to fight the main problem current approaches for

P. Ceravolo et al. (Eds.): SIMPDA 2016, LNBIP 307, pp. 1–21, 2018.
https://doi.org/10.1007/978-3-319-74161-1_1

alignment computation have: the complexity both in space and time. In other words the main goal of the reduction of a process model and event log presented in this paper is to alleviate the current computational challenge of computing an alignment rather than abstracting a process model to capture its essential fragments and hiding details [11]. Therefore given a process model a particular kind relation between transitions of the model which implies *causality* is of interest and the presented technique seeks corresponding fragments of the process model for this issue by which reduces the model. Also, other kinds of relation between transitions of the model for the aim of abstraction or consistency verification between process model are presented in [19] which are not suitable for the mentioned challenge. More specific, the overall idea of this paper relies on the notion of *indication* between activities of the process model when it is represented as a Petri net. An indication relation between a set of transitions (indicated set) and another transition (indicator) denotes a deterministic causal firing relation in the model, which expresses that the presence in any model's sequence of the indicator transition requires the presence of the indicated set as well. The notion of indication is inspired from the *reveals* relation from [3] and *co-occurrence* relation in [19]. We use a well-known technique to find logically independent parts of a graph (known as *fragment with entry-exit pair* in [12] or the so-called *Single Entry Single Exit* (SESE) in [6]), which are then used to gather indication relations efficiently. These relations dictate which parts of a process model are abstracted as a single, high-level node. Once the model is reduced, the observed trace to align is projected (hence, reduced as well) into the reduced model's alphabet. This way, not only the model but also the trace are reduced, which in turn makes the alignment techniques to be significantly alleviated, specially for well-structured process models where many indication relations may exist. Once alignments are computed, the final step is also an interesting contribution of this paper: to cast the well-known Needleman-Wunsch algorithm [9] to expand locally each high-level part of the alignment computed, using the indication relation.

2 Related Work

The seminal work in [1] proposed the notion of alignment, and developed a technique to compute optimal alignments for a particular class of process models. For each trace σ in the log, 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 is implemented in ProM, and can be considered as the state-of-the-art technique for computing alignments. Several optimizations have been proposed to the basic approach to speed up and improve memory consumption. The recent work in [14] proposed a divide and conquer strategy based on Integer Linear Programming (ILP) approach to compute approximate alignments. Despite its memory and time efficiency, it cannot guarantee the obtention of an (optimal) alignment. The similar approach which combines the ideas of two mentioned techniques and can always guarantee a solution (not optimal) and

heavily uses the resolution of ILP and marking equation in combination with a bounded backtracking is presented in [16].

The work in [7] presented a decomposition approach using SESEs for conformance checking of the model and observed behavior. The proposed approach decomposes a given model to smaller parts via SESE and then applies conformance checking for each part independently. This technique is very efficient, but the result is decisional (a yes/no answer on the fitness of the trace). Recently [18] proposed a new approach which provides an algorithm that is able to obtain such an optimal alignment from the decomposed alignments if this is possible, which is called proper optimal alignment. Otherwise, it produces a so-called pseudo-alignment which as in the case of [14], may not be executable in the net.

The *Refined Process Structure Tree* (RPST), proposed by [17], is a graph parsing technique that provides well-structured parts of a graph. The resulting parse tree is unique and modular, i.e., local change in the local workflow graph results in a local change of the parse tree. It can be computed in linear time using the method proposed in [13] which is based on the triconnected components of a given biconnected graph. The proposed approach only works with single sink, single source workflow graphs which hampers its applicability to real world problems with many sink, source nodes. The work in [12] presents a more efficient way to compute RPST which can deal with multiple source, sink workflow graphs.

Abstraction of business process models is presented in [11]. The core idea is to replace the process fragments inside a given process model with the process tasks of higher abstraction levels to simplify the given process models for non-technical stakeholders. The key property of the presented approach is *order preservation*, by which the abstraction mechanism ensures that neither new task execution order constraints are produced nor existing ones gone after abstraction. Stated differently the mentioned property secures the overall process logic to be reflected in the abstracted model. To identify process fragments, the paper uses the notion of *process component* i.e., a process fragment which is connected to the rest of the model by only two nodes namely fragment entry and fragment exit. Identifying process components in a given process model amounts to finding *triconnected* components of a graph. To this end the presented approach lies on *SPQR-tree* decomposition, by which triconnected components can be obtained. Afterwards, the proposed abstraction rules utilize these components. Four abstraction rules are presented which depend on the structure types returned from the decomposition stage. Since the proposed approach relays on identifying triconnected components of a process model therefore it must have some structural characteristics like being free of self-loop structural patterns and must contain no places with multiple incoming and multiple outgoing arcs. Similarly the work in [19] presents *causal behavioural profile* notion for consistency verification between a normative model and its workflow implementation, i.e., to what degree the behavior of the later is captured by the former. The mentioned notion represents a behavioural abstraction that includes dependencies in terms of *order, exclusiveness* and *causality* between pairs of activities of a process model.

The general idea of consistency measure is as follows, given the correspondence relation between the sets of transitions of two WF-nets, all respective transitions of two models are aligned and for each pair of aligned transitions it is checked whether those transitions show the same constraints as defined by the causal behavioural profile. To compute causal behavioural profile efficiently, the presented approach concretises RPST fragments by annotating them with behavioural characteristics. Stated differently, an explicit relation between structural and behavioural characteristics is established.

The seminal work [5] first introduced the notion of *reveals relation*, which determines that whenever an action a happens, then the occurrence of another action b is inevitable. The notion of indication in this paper on the one side is inspired on the reveals relation and on the other side is an extension over co-occurrence relation between two transitions of the process model defined in [19].

3 Preliminaries

3.1 Petri Nets, Structural Deadlock

A *Petri Net* 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. Marking of a Petri net represents the number of tokens each place has. 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. 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 $t_1 t_2 \ldots t_n \in T^*$ that transforms m into m', denoted by $m[t_1 t_2 \ldots t_n \rangle m'$. For a given model N and initial marking m_0, the set $RS(N, m_0) = \{m | \exists w \in T^*.m_0[w\rangle m\}$ is the *reachability* set [8].

A *structural deadlock* or simply deadlock in a Petri net is a set of places such that every transition which outputs to one of the places in the deadlock also inputs from one of these places. Formally, a nonempty subset of places P_d of a net N is a deadlock if ${}^\bullet P_d \subseteq P_d^\bullet$, See Fig. 2. Deadlocks have the following properties [4,10]:

- If marking $m \in RS(N, m_0)$ is a deadlock state then $P_d = \{p | m[p] = 0\}$, is an unmarked set of places.
- Once all of the places in the deadlock become unmarked, the entire set of places will always be unmarked; no transition can place a token in the deadlock because there is no token in the deadlock to enable a transition which outputs to a place in the deadlock.

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*. Figure 1(a) represents a WF-net.

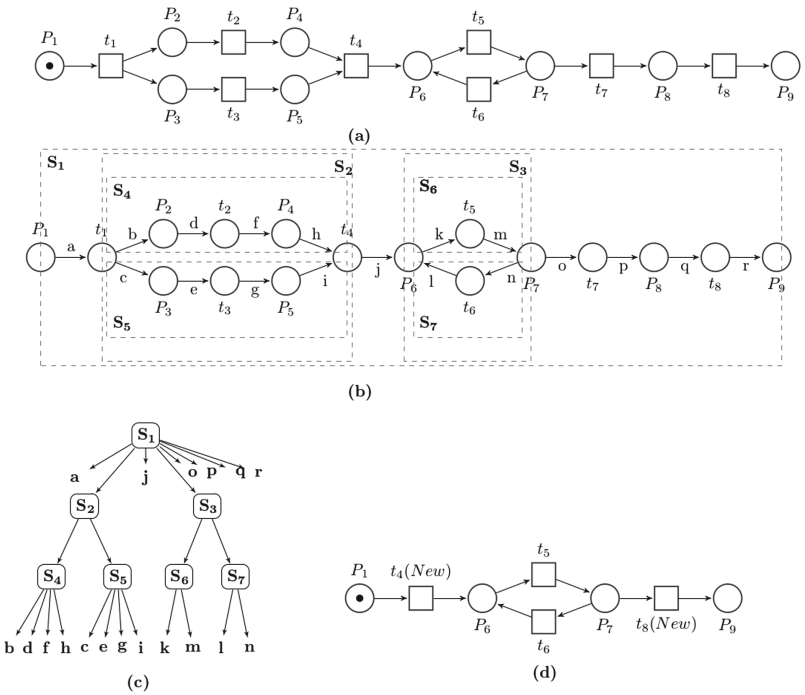

Fig. 1. (a) WF-net, (b) Workflow graph, (c) RPST, (d) Reduced WF-net

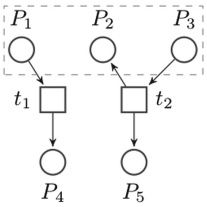

Fig. 2. $P_d = \{P_1, P_2, P_3\}$, ${}^\bullet P_d = \{t_2\}$, $P_d^\bullet = \{t_1, t_2\}$

3.2 Trace, Event Log, Alignment

Given an alphabet of events $T = \{t_1, \ldots, t_n\}$, a trace is a word $\sigma \in T^*$ that represents a finite sequence of events. An *event log* $L \in \mathcal{B}(T^*)$ is a multiset of traces[1]. An *alignment* is represented by a two-row matrix where the top and bottom rows represent moves on log and the model respectively. Such an alignment reveals how the given trace can be replayed on the process model. The classical notion of aligning event log and process model was introduced by [1]. To achieve an alignment between a process model and an event log, we need to relate *moves* in the trace to *moves* in the model. It may be the case that some

[1] $\mathcal{B}(A)$ denotes the set of all multisets of the set A.

of the moves in the trace can not be mimicked by the model and vice versa, i.e., it is impossible to have synchronous moves by both of them. For example given trace $t_1t_4t_2t_5t_8$ and the model in Fig. 1(a), an example of alignment is:

$$\alpha=\begin{array}{|c|c|c|c|c|c|c|}\hline t_1 & \bot & t_4 & t_2 & t_5 & \bot & t_8 \\ \hline t_1 & t_2 & t_4 & \bot & t_5 & t_7 & t_8 \\ \hline \end{array}$$

where the model is able to mimic t_1, t_4, t_5 and t_8 hence they are called *synchronous* moves and the rest are *asynchronous* moves. If weight is assigned to each move such that synchronous moves get less weight than asynchronous moves then an *optimal* alignment which is of interest is the one with minimum cost.

3.3 Interior and Boundary Nodes, SESE

Let $F \subseteq E$ represents a set of edges of a directed graph $\langle V, E, \ell \rangle$, $G_F = \langle V_F, F \rangle$ is the subgraph formed by F if V_F is the smallest set of nodes such that G_F is a subgraph. A node in V_F is *boundary* with respect to G_F if it is connected to nodes in V_F and in $V - V_F$, otherwise it is *interior*. A boundary node u of G_F is an *entry* node if no incoming edge of u belongs to F or if all outgoing edges of u belong to F. A boundary node v of G_F is an *exit* node of G_F if no outgoing edge of v belongs to F or if all incoming edges of v belong to F. G_F with one entry and one exit node is called *Single Entry Single Exit (SESE)*. If a SESE contains only one edge it is called trivial. A SESE of G is called *canonical* if it does not overlap with any other SESEs of G, but it can be nested or disjoint with other SESEs. For example in Fig. 1(b) all SESEs are canonical, S_2 and S_4 are nested, S_3 and S_2 are disjoint. A WF-net can be viewed as a Workflow graph if no distinctions are made between its nodes. WF-graph of Fig. 1(a) is presented in Fig. 1(b). Let G be a graph, then its *Refined Process Structure Tree* (RPST) is the set of all canonical SESEs of G. Because canonical fragments are either nested or disjoint, they form a hierarchy. In a typical RPST, the leaves are trivial SESE and the root is the whole graph. Figure 1(c) is the RPST of WF-graph in Fig. 1(b), S_1 which is the entire graph is at root and leaves are trivial SESEs which only contain one edge.

4 Overall Framework

Given a process model N, represented by a Petri net, and σ as observed behavior, the strategy of this paper is sketched in Fig. 3. We now provide descriptions of each stage.

- *Model Reduction*: N will be reduced based on the notion of *indication* relation which results in N_r. It contains some abstract events representing the indicators of certain indicated sets of transitions. Section 5.1 explains it in detail.
- *Log Reduction*: Using the indication relations computed in the model, σ is projected into the remaining labels in N_r, resulting in σ_r. Section 5.2 describes this step.

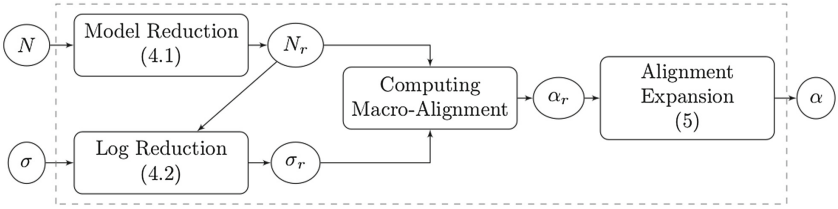

Fig. 3. Overall framework for boosting the computation of alignments

- *Computing Alignment*: Given N_r and σ_r, approaches like [1,14] can be applied to compute alignments. At this point because both N_r and σ_r contain abstract events, the computed alignment will have them as well. We call it *macro-alignment*.
- *Alignment Expansion*: For each abstract element of a macro-alignment, the modeled and observed indications are confronted. Needleman–Wunsch algorithm [9] is adapted to compute optimal alignments for these abstracted elements. Section 6 will be centered on this.

It must be stressed that for the proposed framework, obtaining an optimal alignment is not guaranteed due to expansion and reduction issues but the experimental outcomes which are presented in Sect. 7 revealed the results are closed to optimal solutions.

5 Reduction of Model and Observed Behavior

5.1 The Indication Relation

Let us consider the model in Fig. 1(a). For any sequence of the model, whenever transition t_4 fires it is clear that transitions t_1, t_3, and t_2 have fired as well or firing of t_8 indicates that t_1, t_5 and t_7 must be happened already. Formally:

Definition 1 (Universal-Indication Relation). *Let $N = \langle P, T, \mathcal{F} \rangle$, $\forall t \in T$, indication is defined as a function, $I(t)$ where, $I : T \rightarrow [P(T)^+]^{+2}$ such that for any sequence $\sigma \in \mathcal{L}(N)$, if $t \in \sigma$ then $I(T) \in \sigma$. If $I(t) = \omega_1\omega_2...\omega_n$, then elements of ω_m precede the elements of ω_n in σ for $1 \leq m < n$. It is called linear if it contains only singleton sets, i.e. $\forall \omega_i \in I(t), |\omega_i| = 1$ otherwise it is non-linear.*

Model reduction can be done through the subclass of universal-indication relation, which is called *flow-indication* relation. Stated formally:

Definition 2 (Flow-Indication Relation). *Given Definition 1, If $I(t) = \omega_1\omega_2...\omega_n$, it represents a flow-indication if and only if, for all consecutive elements ω_i, ω_{i+1}, firing the whole elements of the former enable all elements in the later, exclusively, for $1 \leq i < n$.*

[2] $P(T)$ is powerset of the set of transitions of the model.

For example in Fig. 1(a), $I(t_4) = \{t_1\}\{\{t_2\},\{t_3\}\}\{t_4\}$ (non-linear), which is a flow-indication as well, and $I(t_8) = \{t_1\}\{t_5\}\{t_7\}\{t_8\}$ (linear), but it is not a flow-indication because firing of t_1 will not enable t_5 exclusively. From now on, because the flow-indication relation is our concern for the remaining parts of the paper, for the sake of simplicity, by indication we mean flow-indication relation, unless otherwise stated explicitly.

5.1.1 Detecing Flow-Indication Relation Through SESE.

SESEs are potential candidates for identifying indication relations inside a WF-net: the exit node of a SESE is the potential indicator of the nodes inside the SESE. Since entry/exit nodes of a SESE can be either place or transitions, SESEs are categorized as (P, P), (P, T), (T, P) or (T, T). In case the SESE is linear, indication relations can be extracted easily and the corresponding SESE is reduced (see Fig. 4).

Non-linear cases are decomposed into linear ones such that indication relations can be computed directly on the linear components extracted. After that, the indication relation of the corresponding linear SESEs are computed and they are reduced as well. This procedure should be done with caution to avoid reaching a deadlock situation. Hence a *deadlock-free post-verification* must be done after reduction of these linear parts. Informally, the verification is only needed for particular type of linear SESEs ((T, T)), and consists on validating the property of the SESE after the reduction. Notice the verification is necessary in these cases because, non-linear SESEs may contain linear universal-indications at nested level, which cannot be extracted as flow-indication relations due to choice or loop constructs. For example in Fig. 5(a), (b) t_5 can not be the indicator of transitions in the corresponding SESEs due to choice and loop structures.

Stated differently, the reduction of non-linear SESEs must be done alongside by a deadlock-free post-verification; for instance, Fig. 6 shows that in spite of the indication arising from SESE S_2, the net cannot be reduced without changing the language. To put it another way, this reduction will cause a deadlock in the reduced model, and hence must be avoided. Looking at the reduced result

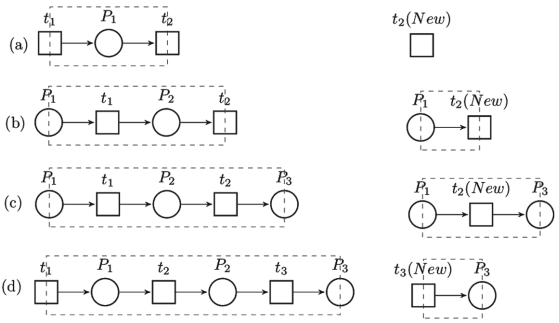

Fig. 4. Linear SESEs and corresponding reductions.

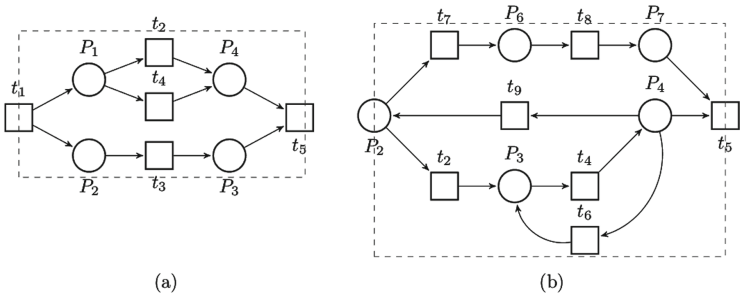

Fig. 5. (a) Non-Linear (T,T), (b) Non-Linear (P,T)

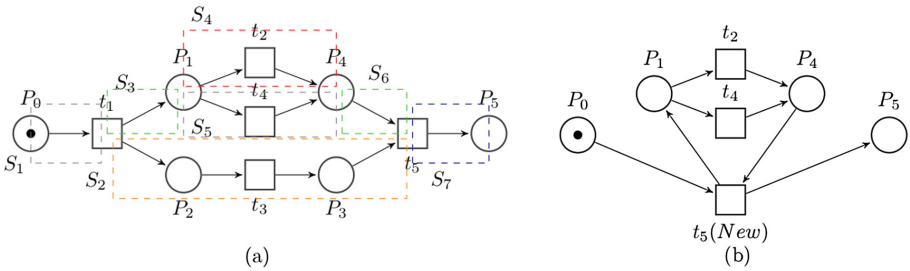

Fig. 6. Incorrect indication-based reduction: a deadlock is introduced.

in Fig. 6(b), transition $t_5(New)$ never fires because after the reduction it won't be enabled since P_4 never gets marked. To shed more light on the examination of the deadlock-free post-verification, more details are stated in the following theorem.

Theorem 1. *Let S be a reduced linear SESE or the combination of other reduced linear SESEs with entry, exit nodes (t_u, t_v) of the (T,T) category. If $OUT(t_u)$ and $IN(t_v)$ represent the set outgoing and incoming arcs of t_u and t_v respectively, then the reduction is deadlock-free if and only if:*

$$(a) \quad \forall e \in OUT(t_u), \quad then \quad e \in S \quad (b) \quad \forall e \in IN(t_v), \quad then \quad e \in S$$

Proof. First of all, assume that the original model before the reduction does not have any deadlock and T_S and $t_{v(New)}$ represent internal transitions of S and the reduced SESE respectively. The proof is presented by contradiction as follow:

Suppose that conditions in Theorem 1 hold and the reduction of S causes deadlock in the system. Namely, there is a set of places, P_d, which attributes deadlock or in other words $t_{v(New)}$ outputs to one of places in P_d and inputs from one of them. Due to the fact that all transitions in T_S are internal and do not have direct access to any places in P_d, the only incoming and outgoing arcs of $t_{v(New)}$ belong to t_u and t_v respectively. So it can be concluded that once the places in

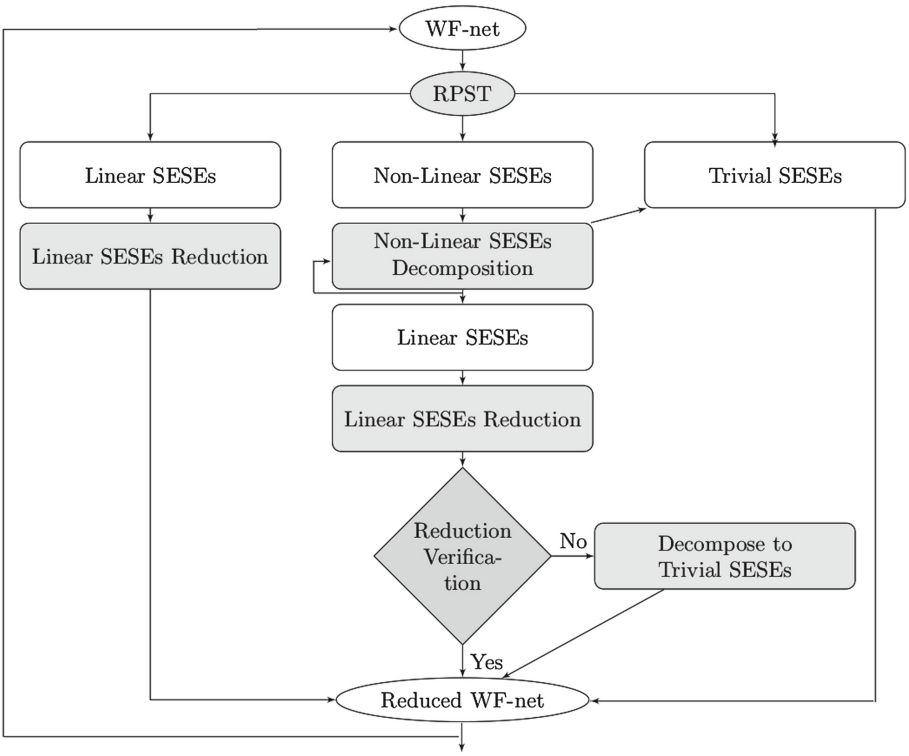

Fig. 7. Schema for reduction of a WF-net.

P_d become unmarked they will always be unmarked and neither t_u nor t_v can place a token in the deadlock, but this contradicts with the assumption that the original model does not have deadlock due to the fact that $IN(t_u)$ and $OUT(t_v)$ remain unchanged before and after reduction. Thus the theorem is true. □

The reduction schema is depicted in Fig. 7. From the RPST, a top-down approach is applied that searches for indication-based reductions that do preserve the language of the initial model, once the net is expanded back, i.e., the language of the model must be preserved after reduction.

Notice that the reduction can be applied more than once till saturation (hence the arc back from the node "Reduced WF-net" to the node "WF-net" in Fig. 7).

Figure 8 shows an example (for the sake of simplicity only linear SESEs are shown). Obviously, SESE S_2 is inherently a linear SESE but the rest come from the decomposition of non-linear SESEs. The reduction schema is as follows: Since S_2 is inherently a linear SESE, hence it can be reduced easily according to Fig. 4 without any deadlock-free post-verification. The rest of linear SESEs also will be reduced accordingly and the deadlock-free post-verification will be done after each reduction to check that no deadlock arises. One can see all reductions will pass the verification, except for S_7, whose reduction induces a deadlock

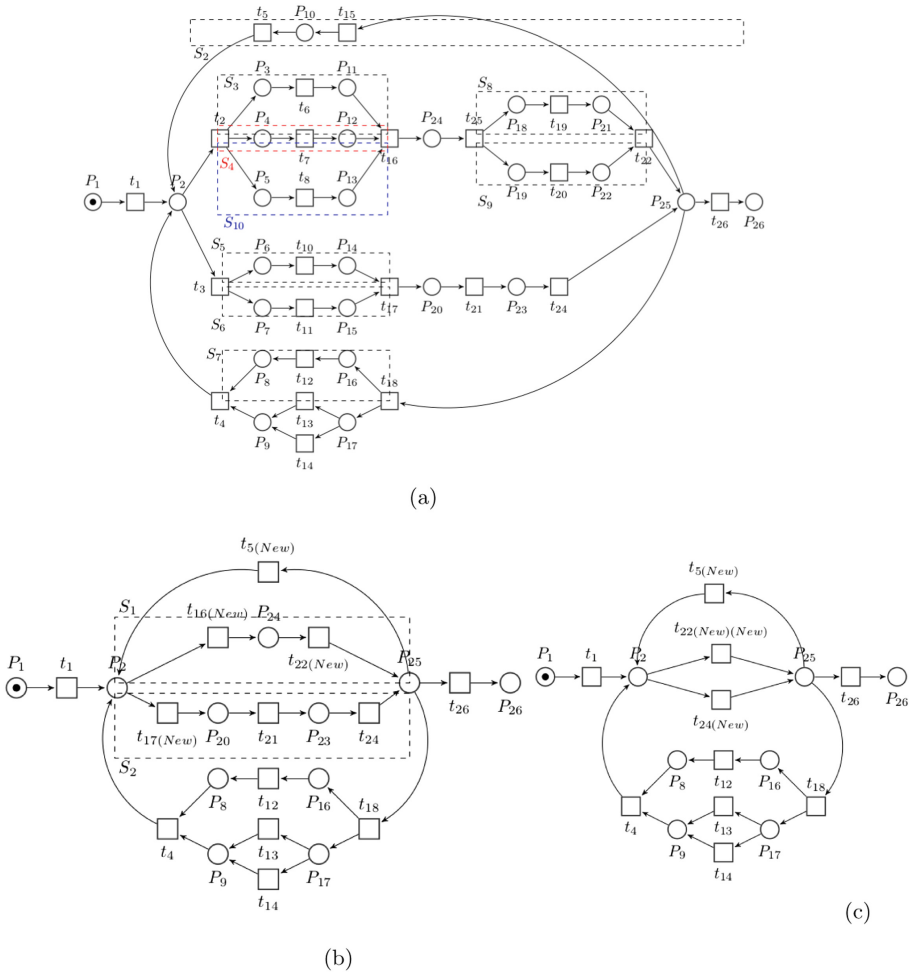

Fig. 8. (a) Process model, (b) One-time reduced (c) Two-times reduced.

hence must be excluded from abstraction. Applying the reduction once, results in Fig. 8(b). As mentioned earlier, the reduction can be applied more than once until no reduction can be made. Figure 8(c) is the reduction of the model in Fig. 8(b) and it is clear that no more reduction can be made from this model.

5.2 Reduction of Observed Behavior

Given a reduced model N_r and σ, we show how to produce σ_r. We will use the reduced model in Fig. 8(b) and the trace $\sigma_1 = t_1t_5t_3t_{11}t_{10}t_{21}t_6t_2t_7t_{16}t_{25}t_{19}t_{20}t_{26}$. The indication of $t_{5(New)}$ in Fig. 8(b) which is linear, equals to $\{t_5\}\{t_{15}\}$. So the *observed indication* for this abstract node is ${\sigma_1}_{\downarrow_{I(t_{5(new)})}} = t_5$. After computing

the observed indication the reduced trace is $t_1t_{5(new)}t_3t_{11}t_{10}t_{21}t_6t_2t_7t_{16}t_{25}$ $t_{19}t_{20}t_{26}$. For $t_{17(New)}$, $I(t_{17(New)}) = \{t_3\}\{\{t_{10}\},\{t_{11}\}\}\{t_{17}\}$, which is non-linear and merged of two linear indications, $I_1(t_{17(New)})$=$\{t_3\}\{t_{10}\}\{t_{17}\}$ and $I_2(t_{17(New)}) = \{t_3\}\{t_{11}\}\{t_{17}\}$. So the projection must be done for each linear indication separately, $\sigma_{1_{\downarrow_{I_1(t_{17(New)})}}} = t_3t_{10}$ and $\sigma_{1_{\downarrow_{I_2(t_{17(New)})}}} = t_3t_{11}$, removing transitions t_3, t_{10}, t_{11} and t_{17} from the current trace (notice that t_{17} does not appear originally, hence it is not projected). Finally, we need to insert $t_{17(New)}$ into the reduced trace; it will be inserted at the position of t_{10}, because the end transition of the abstract node, i.e. t_{17} did not happen in σ, and t_{10} happened last in σ. Therefore the reduced trace so far is $t_1t_{5(new)}t_{17(new)}t_{21}t_6t_2t_7t_{16}t_{25}$ $t_{19}t_{20}t_{26}$. By applying this process for the rest of abstract nodes ($t_{16(New)}$, $t_{22(New)}$), we reach $\sigma_r = t_1t_{5(new)}t_{17(new)}t_{21}t_{16(New)}$ $t_{22(New)}t_{26}$.

6 Expansion Through Local Optimal Indication Alignments

After reducing a given process model and corresponding observed behavior, we can use current methods for computing alignments [1,14] to align N_r and σ_r, deriving α_r. For example the following is the macro alignment of $\sigma_{1r} = t_1t_{5(new)}t_{17(new)}t_{21}t_{16(New)}$ $t_{22(New)}t_{26}$ and the model in Fig. 8(b) obtained by the approach in [1].

$$\alpha_r = \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline t_1 & t_{5(New)} & t_{17(New)} & t_{21} & \perp & \perp & t_{16(New)} & t_{22(New)} & t_{26} \\ \hline t_1 & \perp & t_{17(New)} & t_{21} & t_{24} & t_{5(New)} & t_{16(New)} & t_{22(New)} & t_{26} \\ \hline \end{array}$$

When mapped to linear indications, indication of an abstract node and the corresponding observed indication are both sequence of events; hence for each linear combination of modeled/observed indication, we can adapt the dynamic programming approach from [9] (used in bioinformatics) to align two sequences. As an example, we use indication of $t_{17(New)}$ and its observed indication computed in the previous section.

Table 1. Aligning modeled and observed indications

Table 1: Aligning modeled and observed indications

		t_3	t_{11}
	0	-1	-2
t_3	-1	0	-1
t_{11}	-2	-1	0
t_{17}	-3	-2	-1

(a)

		t_3	t_{10}
	0	-1	-2
t_3	-1	0	-1
t_{10}	-2	-1	0
t_{17}	-3	-2	-1

(b)

$$\alpha_1 = \begin{array}{|c|c|c|} \hline t_3 & t_{11} & \perp \\ \hline t_3 & t_{11} & t_{17} \\ \hline \end{array} \qquad \alpha_2 = \begin{array}{|c|c|c|} \hline t_3 & t_{10} & \perp \\ \hline t_3 & t_{10} & t_{17} \\ \hline \end{array}$$

To achieve this goal, we create a table for each linear indication, where the first row and column are filled by observed and abstract node indications respectively, as depicted in Table 1(a), (b). The second row and second column are initialized with numbers starting from 0, −1, −2,..., they are depicted in yellow color. The task then is to fill the remaining cells as follows:

$$SIM(t_i, t_j) = MAX(SIM(t_{i-1}, t_{j-1}) + s(t_i, t_j), SIM(t_{i-1}, t_j) - 1, SIM(t_i, t_{j-1}) - 1)$$

where $SIM(t_i, t_j)$ represents the similarity score between t_i and t_j. $s(t_i, t_j)$ is the substitution score for aligning t_i and t_j, it is 0 when they are equal and −1 otherwise.

The final step in the algorithm is the trace back for the best alignment. In the above mentioned example, one can see the bottom right hand corner in for example Table 1, score as −1. The important point to be noted here is that there may be two or more alignments possible between the two example sequences. The current cell with value −1 has immediate predecessor, where the maximum score obtained is diagonally located and its value is 0. If there are two or more values which points back, suggests that there can be two or more possible alignments. By continuing the trace back step by the above defined method, one would reach to the 0th row, 0th column. Following the above described steps, alignment of two sequences can be found.

Alignments can be represented by a sequence of paired elements, for example $\alpha_1 = (t_3, t_3)(t_{11}, t_{11})(\perp, t_{17})$, $\alpha_2 = (t_3, t_3)(t_{10}, t_{10})(\perp, t_{17})$ and final alignment which represent the non-linear indication is $\alpha = (t_3, t_3)\{(t_{11}, t_{11}), (t_{10}, t_{10})\}(\perp, t_{17})$. This information is booked for each abstract node.

After computing local alignments for abstract nodes, we can use them to expand corresponding abstract nodes in a given α_r. The policy of expansion depends on whether the abstract node is in synchronous or asynchronous move.

In α_r, $t_{17(New)}$ is in a synchronous move so we can expand it by its local alignment, which results in:

$$\alpha = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} t_1 & t_{5(New)} & t_3 & t_{11} & t_{10} & \perp & t_{21} & \perp & \perp & t_{16(New)} & t_{22(New)} & t_{26} \\ t_1 & \perp & t_3 & t_{11} & t_{10} & t_{17} & t_{21} & t_{24} & t_{5(New)} & t_{16(New)} & t_{22(New)} & t_{26} \end{array}$$

The same story also happens for $t_{16(New)}$ and $t_{22(New)}$, which results in:

$$\alpha = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} t_1 & t_{5(New)} & t_3 & t_{11} & t_{10} & \perp & t_{21} & \perp & \perp & t_6 & t_2 & t_7 & \perp & \perp & t_{16} & t_{25} & t_{19} & t_{20} & \perp & t_{26} \\ t_1 & \perp & t_3 & t_{11} & t_{10} & t_{17} & t_{21} & t_{24} & t_{5(New)} & \perp & t_2 & t_7 & t_6 & t_8 & t_{16} & t_{25} & t_{19} & t_{20} & t_{22} & t_{26} \end{array}$$

On the other hand $t_{5(New)}$ in α_r is a asynchronous move both on the model and observed trace. The policy of expansion is to expand move on log and move on model independently. To put it in another way, move on log will be expanded using observed indication and move on model will be expanded using the abstract node' indication, which results:

$$\alpha = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} t_1 & t_5 & t_3 & t_{11} & t_{10} & \perp & t_{21} & \perp & \perp & \perp & t_6 & t_2 & t_7 & \perp & \perp & t_{16} & t_{25} & t_{19} & t_{20} & \perp & t_{26} \\ t_1 & \perp & t_3 & t_{11} & t_{10} & t_{17} & t_{21} & t_{24} & t_{15} & t_5 & \perp & t_2 & t_7 & t_6 & t_8 & t_{16} & t_{25} & t_{19} & t_{20} & t_{22} & t_{26} \end{array}$$

7 Experiments

The technique presented in this paper has been implemented in Python as a prototype tool. The tool has been evaluated over different family of examples with variety of difficulties, alongside with the state of the art techniques for computing alignments [14] ($ILP.R$), [1] (A^*). We used benchmark datasets from [7,14], and new generated datasets.

Reduction of Models. Table 2 provides the results of one-time reduction by applying the proposed method to benchmark datasets. Significant reductions are found often. Obviously one can see that the results of reduction are more representative for models without loops like *(prAm6,..,prGm6)* or for models that contain small loops, like (*Banktransfer*).

Table 2. Reduced benchmark datasets

Model	$\|P\|$ (Before)	$\|T\|$ (Before)	$\|Arc\|$ (Before)	$\|\sigma\|_{avg}$ (Before)	$\|P\|$ (After)	$\|T\|$ (After)	$\|Arc\|$ (After)	$\|\sigma\|_{avg}$ (After)
prAm6	363	347	846	31	175(52%)	235(32%)	498	22(29%)
prBm6	317	317	752	43	188(40%)	225(29%)	490	33(23%)
prCm6	317	317	752	42	188(40%)	225(29%)	490	33(21%)
prDm6	529	429	1140	248	270(49%)	248(42%)	618	148(40%)
prEm6	277	275	652	98	180(35%)	205(26%)	454	75(23%)
prFm6	362	299	772	240	181(50%)	172(42%)	406	137(42%)
prGm6	357	335	826	143	195(45%)	221(34%)	498	94(34%)
M_1	40	39	92	13	25(37%)	28(28%)	62	9(30%)
M_2	34	34	80	17	26(23%)	28(18%)	64	13(23%)
M_3	108	123	276	37	76(30%)	98(20%)	212	29(21%)
M_4	36	52	106	26	31(14%)	48(8%)	96	23(11%)
M_5	35	33	78	34	27(23%)	27(18%)	62	28(18%)
M_6	69	72	168	53	51(26%)	59(18%)	132	43(19%)
M_7	65	62	148	37	43(34%)	46(26%)	104	28(24%)
M_8	17	15	36	17	6(65%)	7(53%)	14	9(47%)
M_9	47	55	120	44	26(45%)	39(29%)	78	34(23%)
M_{10}	150	146	354	58	91(39%)	105(28%)	236	42(28%)
Bank transfer	121	114	272	58	61(46%)	72(37%)	152	38(34%)

Executable Property of Alignments. Since the alignment technique $ILP.R$ may be approximate or the results contain spurious elements, Table 3 provides an overview of how many of the computed alignments can be replayed for $ILP.R$ method when combined with the technique of this paper. Also the corresponding results for the technique in [1] are presented as well. One can see that the expanded alignments provided by A^* were replayed 100% for all datasets.

Table 3. Replaying of computed step-sequences

Model	Cases	Replay% (Before)ILP.R	Replay% (After)ILP.R	Replay% (Before)A^*	Replay% (After)A^*
prAm6	1200	100%	100%	100%	100%
prBm6	1200	100%	100%	100%	100%
prCm6	500	100%	100%	100%	100%
prDm6	1200	100%	100%	100%	100%
prEm6	1200	100%	100%	100%	100%
prFm6	1200	100%	100%	100%	100%
prGm6	1200	100%	100%	100%	100%
M_1	500	94.2%	86%	100%	100%
M_2	500	95.4%	86.2%	100%	100%
M_3	500	98%	88.8%	100%	100%
M_4	500	90%	81%	100%	100%
M_5	500	94.8%	95.2%	100%	100%
M_6	500	98.6%	90.8%	100%	100%
M_7	500	97.2%	96%	100%	100%
M_8	500	100%	100%	100%	100%
M_9	500	100%	98.8%	100%	100%
M_{10}	500	100%	99.8%	100%	100%
Bank transfer	2000	97.25%	88.9%	100%	100%

Table 4. Quality of computed step-sequences

Model	ED($A^* vs$ $EXP.R.A^*$)	Jaccard($A^* vs$ $EXP.R.A^*$)	MSE($A^* vs$ $EXP.R.A^*$)	ED(ILP.R vs EXP.R.ILP.R)	Jaccard(ILP.R vs EXP.R.ILP.R)	MSE(ILP.R vs EXP.R.ILP.R)
prAm6	7.49	0	0.065	9.25	0.017	0.00081
prBm6	7.87	0	0	18.31	0	0
prCm6	8.65	0.016	0.005	11.60	0.0019	0.00646
prDm6	NA	NA	NA	93.28	0.0101	0.00041
prEm6	37.14	0	0.02	37	0	0
prFm6	NA	NA	NA	67	0.013	0.0074
prGm6	NA	NA	NA	77	0.011	0.00064
M_1	4	0.085	0.021	4	0.025	0.0165
M_2	6	0.012	0.0193	6	0	0.018
M_3	8	0.046	0.021	5	0.011	0.016
M_4	4	.12	0.028	2	0.015	0.025
M_5	11	0.0022	0.0045	15	0.00024	0.0103
M_6	NA	NA	NA	12	0.0012	0.0088
M_7	NA	NA	NA	15	0.0027	0.019
M_8	4	0.073	0.039	4	0.0078	0.035
M_9	NA	NA	NA	3	0.0044	0.0085
M_{10}	NA	NA	NA	13	0.00038	0.012
Bank transfer	18	0.031	0.025	13	0.0118	0.0067

Comparing with Original Alignments. Table 4 reports the evaluation of the quality of the results for both approaches [1,14] with and without applying the technique of this paper. Columns ED/Jaccard report the edit/Jaccard distances between the sequences computed, while (Mean Square Error) MSE columns

Table 5. The atrerage of required variables for ILP.R

Model	$\|Var\|_{avg}$(Before)	$\|Var\|_{avg}$(After)	Model	$\|Var\|_{avg}$(Before)	$\|Var\|_{avg}$(After)
prAm6	10757	5170 (52%)	M_2	578	364 (37%)
prBm6	13631	7425 (45%)	M_3	4551	2842 (37%)
prCm6	13314	7425 (44%)	M_4	1352	1104 (18%)
prDm6	106392	36704 (65%)	M_5	1122	756 (32%)
prEm6	26950	15375 (43%)	M_6	3816	2537 (33%)
prFm6	71760	23564 (67%)	M_7	2294	1288 (44%)
prGm6	47905	20774 (56%)	M_8	255	63 (75%)
banktransfer	6612	2736 (58%)	M_9	2420	1326 (45%)
M_1	507	252 (50%)	M_{10}	8468	4410 (48%)

report the mean square error between the corresponding fitness values. Edit distances are often large, but interestingly this has no impact on the fitness, since when expanding abstract nodes although the final position may differ, the model still can replay the obtained sequences very often.

Memory Usage. By one-time reduction, the memory usage[3] of computing alignments using [1], is reduced significantly. See Fig. 9(a)–(b) which represents the required memory for [1] without and with using the proposed framework respectively. For large models, *prDm6, prFm6, prGm6*, it can only compute alignments if applied in combination with the technique of this paper otherwise it runs out of memory for the machine by which the experiment are done, denoted by (>5500 MB) in Fig. 9(a), (b). For the approach in [14], due to the fact that it is based on Integer Linear Programming (ILP), to accentuate the effect of reduction, the evaluation was done based on number of required variables for computing alignments with and without the proposed approach. The results in Table 5[4] represent, in average, significant reduction to the number of variables when an ILP instance needs to be solved a given problem.

Computation Time Comparison. Figures 10 and 11(a)–(b) report execution times for BPM-2013 and other benchmark datasets for the computation of alignments by techniques in [1,14] with and without using the presented technique in this paper (denoted by *EXP.R.*) respectively. It is evident that A^* approach combined with the proposed method is significantly faster than the other approach in nearly all datasets except (*prGm6, prDm6,* M_6, M_{10}). Still A^* approach cannot compute alignments for models M_6 *and* M_{10} even after applying the presented technique, which are denoted by (N/A), and in that case the combination of *ILP.R* with the presented technique is the best choice.

[3] Each dataset during its execution was monitored every 0.15 s, and the portion of memory occupied by the corresponding process that is held in main memory (RSS) was booked. Based on the gathered data 95% CI was computed.

[4] For a given model with $|T|$ transitions and an event log σ, the required number of variables for the ILP based technique in [14] is $\Theta(|\sigma| \times |T|)$, totally.

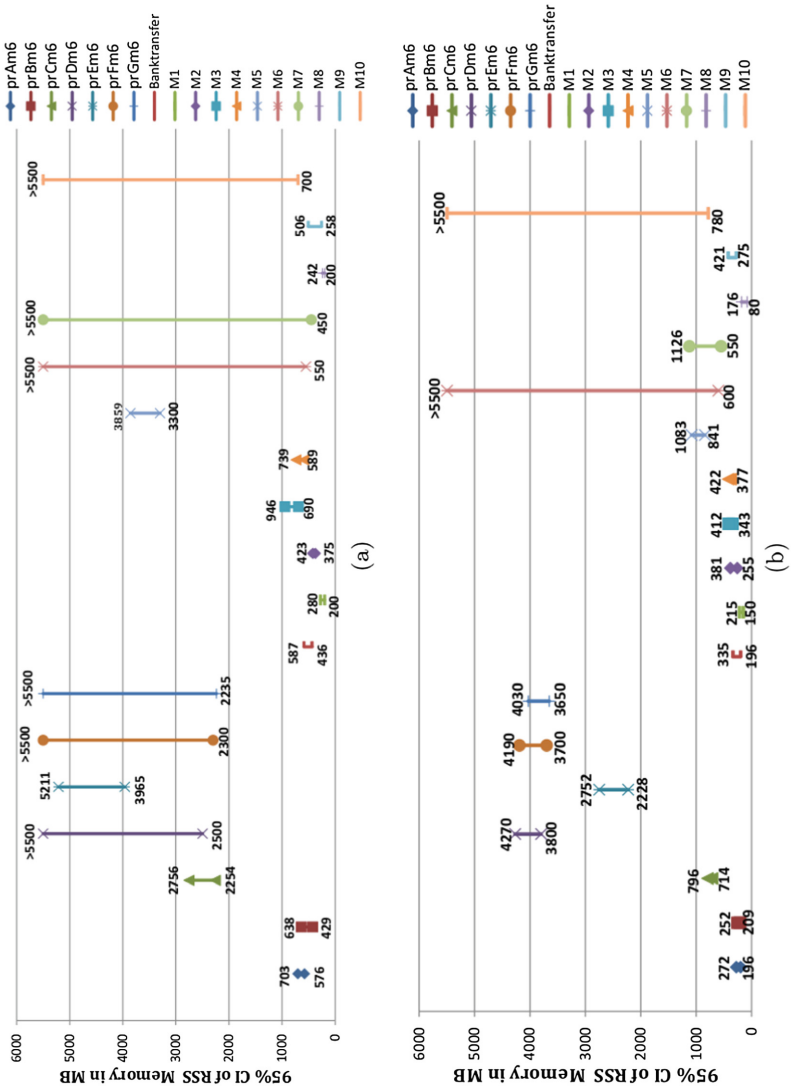

Fig. 9. Memory usage for [1] (a) Before reduction and (b) After reduction

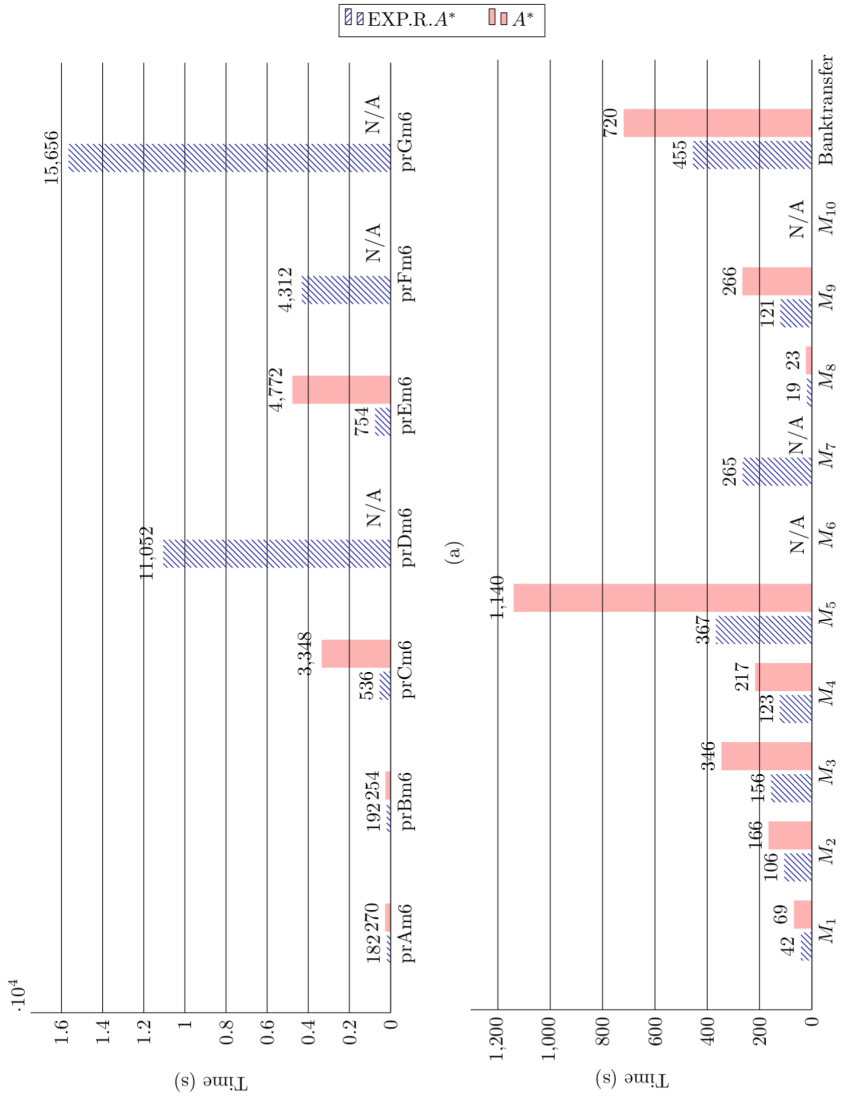

Fig. 10. (a) BPM-2013 datasets [7], (b) Synthetic datasets

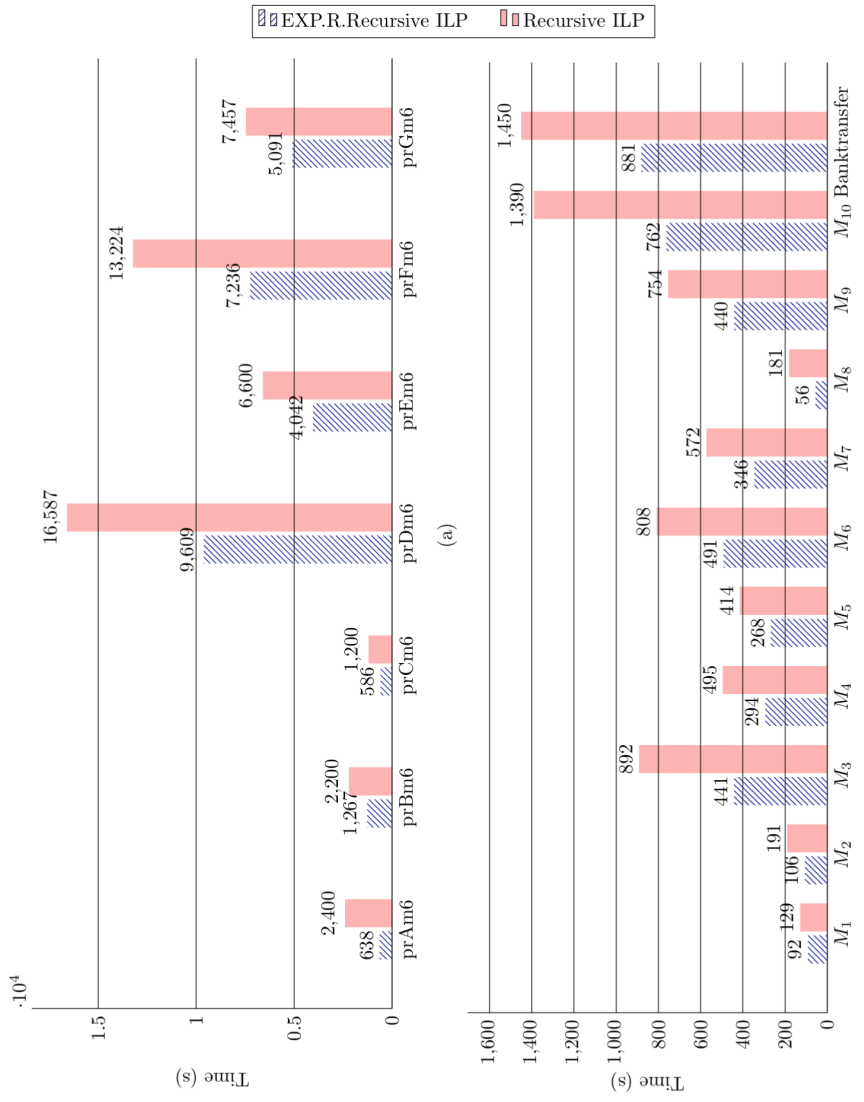

Fig. 11. (a) BPM-2013 datasets [7], (b) Synthetic datasets

8 Conclusion and Future Work

We have presented a technique that can be used to significantly alleviate the complexity of computing alignments. The technique uses the indication relation to abstract unimportant parts of a process model so that global computation of alignments focus on a reduced instance. The reduced part of computed alignments then will be expanded to represent local deviations as well. Experiments are provided that witness the capability of the technique when used in combination with state-of-the-art approaches for alignment computation. Future work will be devoted to apply the technique on more unstructured inputs and examining other methods to extract indication relations more efficiently.

Acknowledgments. This work was supported by the Spanish Ministry for Economy and Competitiveness (MINECO) and the European Union (FEDER funds) under grant COMMAS (ref. TIN2013-46181-C2-1-R).

References

1. Adriansyah, A.: Aligning observed and modeled behavior. Ph.D. thesis, Technische Universiteit Eindhoven (2014)
2. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Measuring precision of modeled behavior. Inf. Syst. E-Bus. Manage. **13**(1), 37–67 (2015)
3. Balaguer, S., Chatain, T., Haar, S.: Building occurrence nets from reveals relations. Fundam. Inform. **123**(3), 245–272 (2013)
4. Colom, J.M., Teruel, E., Silva, M., Haddad, S.: Structural methods. In: Girault, C. (ed.) Petri Nets for Systems Engineering, pp. 277–316. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-662-05324-9_16
5. Haar, S.: Unfold and cover: qualitative diagnosability for petri nets. In: Proceedings of the 46th IEEE Conference on Decision and Control (CDC 2007), New Orleans, LA, USA, pp. 1886–1891. IEEE Control System Society (2007)
6. Johnson, R., Pearson, D., Pingali, K.: The program structure tree: Computing control regions in linear time. SIGPLAN Not. **29**(6), 171–185 (1994)
7. Munoz-Gama, J., Carmona, J., Van Der Aalst, W.M.P.: Single-entry single-exit decomposed conformance checking. Inf. Syst. **46**, 102–122 (2014)
8. Murata, T.: Petri nets: Properties, analysis and applications. Proc. IEEE **77**(4), 541–574 (1989)
9. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. **48**(3), 443–453 (1970)
10. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice Hall PTR, Upper Saddle River (1981)
11. Polyvyanyy, A., Smirnov, S., Weske, M.: The triconnected abstraction of process models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 229–244. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03848-8_16

12. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. In: Bravetti, M., Bultan, T. (eds.) WS-FM 2010. LNCS, vol. 6551, pp. 25–41. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19589-1_2
13. Tarjan, R.E., Valdes, J.: Prime subprogram parsing of a program. In: Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 1980, pp. 95–105. ACM, New York (1980)
14. Taymouri, F., Carmona, J.: A recursive paradigm for aligning observed behavior of large structured process models. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 197–214. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_12
15. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer, Heidelberg (2011)
16. van Dongen, B., Carmona, J., Chatain, T., Taymouri, F.: Aligning modeled and observed behavior: a compromise between computation complexity and quality. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 94–109. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_7
17. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 100–115. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85758-7_10
18. Verbeek, H.M.W., van der Aalst, W.M.P.: Merging alignments for decomposed replay. In: Kordon, F., Moldt, D. (eds.) PETRI NETS 2016. LNCS, vol. 9698, pp. 219–239. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39086-4_14
19. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Causal behavioural profiles - efficient computation, applications, and evaluation. Fundam. Inf. **113**(3–4), 399–435 (2011)