A Recursive Paradigm for Aligning Observed Behavior of Large Structured Process Models

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 crucial problem in process mining, since it opens the door for conformance checking and enhancement of process models. The state of the art techniques for the computation of alignments rely on a full exploration of the combination of the model state space and the observed behavior (an event log), which hampers their applicability for large instances. This paper presents a fresh view to the alignment problem: the computation of alignments is casted as the resolution of Integer Linear Programming models, where the user can decide the granularity of the alignment steps. Moreover, a novel recursive strategy is used to split the problem into small pieces, exponentially reducing the complexity of the ILP models to be solved. The contributions of this paper represent a promising alternative to fight the inherent complexity of computing alignments for large instances.

1 Introduction

As business processes become more complex and change frequently, companies and organizations use information systems to handle the processing and execution of their business transactions. These systems generate *event logs* which are the footprints left by process executions. *Process mining* is an emerging field that focuses on analyzing these event logs with the purpose of extracting, analyzing and enhancing evidence-based process models [13].

One of the current challenges for process mining techniques is the computation of an *alignment* of a process model with respect to observed behavior [1]. Intuitively, given a trace representing a real process execution, an optimal alignment provides the best trace the process model can provide to mimic the observed behavior. Then observed and model traces are rendered in a two-row matrix denoting the synchronous/asynchronous moves between individual activities of model and log, respectively. Alignments are extremely important in the context of process mining, since they open the door to evaluate the metrics that asses the quality of a process model to represent observed behavior: *fitness* and *generalization* [1] and precision [2]. Additionally, alignments are a necessary step to enhance the information provided in a process model [13].

Unfortunately, the current algorithmic support to compute alignments is defined as search for a minimal path on the product of the state space of the process model and the observed behavior, an object that is worst-case exponential with respect to the size of the model. This hampers the application of these techniques for medium/large instances.

The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-45348-4_12

Hence, in the process mining field we are facing an interesting paradox: while the current research for process discovery is capable to be applied to large inputs (e.g., [7]), the obtained models often cannot be optimally aligned due to their size. Addressing this paradox is the main motivation of the work presented in this paper.

This paper presents a technique to compute a particular type of alignments, called *approximate alignments*. In an approximate alignment, the granularity of the moves is user-defined (from singletons like in the original definition of alignments, to non-unitary sets of activities), thus allowing for an abstract view, in terms of *step-sequences*, of the model capability of reproducing observed behavior. The implications of generalizing the concept of alignment to non-singleton steps are manifold: conformance checking techniques can be discretized to a desired (time) granularity, e.g., when the ordering of activities in a period is not important for the diagnosis. Also, other techniques like *model repair* [6] may be guided to only repair coarse-grain deviating model parts. Finally, in domains where a fine-grained ordering of activities is not needed approximate alignments can play an important role (e.g., health care [9]).

We assume the input models to be specified as Petri nets. This is without loss of generality, since there exist transformations from other notations to Petri nets. Given a Petri net and a trace representing the observed behavior, we use the *structural theory* of Petri nets [12] to find an approximate alignment. This means that at the end we solve *Integer Linear Programming* (ILP) models whose resolution provide a model firing sequence that mimics the observed behavior. Importantly, these ILP models are extended with a cost function that guarantees (under certain structural conditions on the process model) a global optimality criteria: the obtained firing sequence is mostly similar to the observed trace in terms of the number of firings of each transition. This optimality capability represents one clear difference with respect to current distributed approaches for conformance checking which focus on the decisional problem of checking fitness, but not to compute optimal alignments [14,10].

Since ILP is NP-hard, casting the problem of computing approximate alignments as the resolution of ILP models is not sufficient for alleviating the complexity of the problem. As the complexity of ILP is dominated by the number of variables and constraints, we present a recursive framework to compute approximate alignments that transforms the initial ILP encoding into several smaller and bounded ILP encodings. This approach reduces drastically both the memory and the CPU time required for computing approximate alignments. Remarkably, it can be applied not only with the ILP encoding used in this paper, but also in combination with current techniques for computing alignments.

The organization of this paper is as follow, related work is presented in Section 2. Preliminaries will be presented in section 3. The formalization of approximate alignments is described in Section 4. Section 5 describes the ILP encoding for computing approximate alignments, while Section 6 presents the recursive framework. Experiments, conclusions and future work are presented in sections 7 and 8 respectively.

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: for instance, the use of ILP techniques on each visited state to prune the search space [1]. In contrast to the technique of this paper, these ILP techniques only alleviate the search space while in our case they form the basis to compute an alignment. Alignment techniques from [1] have been extended recently in [3] for the case of *process trees*, presenting techniques for the state space reduction with *stubborn sets*[1]. Also, high-level deviations are proposed in [1] in form of *deviation patterns* that, as the work in this paper, aim at providing less detailed diagnostics.

Decompositional techniques have been presented [14,10] that instead of computing optimal alignments, they focus on the *decisional problem* of whereas a given trace fits or not a process model. The underlying idea is to split the model into a particular set of transition-bordered fragments which satisfy certain conditions, and local alignments can be computed for each one of the fragments, thus providing a upper bound on the cost of an alignment. In contrast, the technique presented in this paper does not split the model, hence enabling the computation of alignments at a global (model) level.

Finally, the work in [9,8] focuses on dealing with partially ordered information, a common situation in contexts like health care. The notion of *partially ordered alignment* is introduced, and a variation of the techniques presented in [1] described.

3 Preliminaries

3.1 Petri nets, Process Mining and Step Sequences

A Petri Net [11] 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 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. When a transition t is enabled, it can *fire* 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 $t_1 t_2 \ldots t_n$ that transforms m into m', denoted by $m[t_1 t_2 \ldots t_n\rangle m'$. A sequence of transitions $t_1 t_2 \ldots t_n$ is a *feasible sequence* if it is firable from the initial marking m_0.

Definition 1 (Trace, Event Log, Parikh vector). *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*[2]. $|\sigma|_a$ *represents the number of occurrences*

[1] There is no fundamental difference between aligning Petri nets or process trees: only the latter allows for a slightly better memory representation.

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

of a in σ. The Parikh vector of a sequence of events σ is a function $\widehat{\ }: T^* \to \mathbb{N}^n$ defined as $\widehat{\sigma} = (|\sigma|_{t_1}, \ldots, |\sigma|_{t_n})$. For simplicity, we will also represent $|\sigma|_{t_i}$ as $\widehat{\sigma}(t_i)$. The support of a Parikh vector $\widehat{\sigma}$, denoted by supp$(\widehat{\sigma})$ is the set $\{t_i | \widehat{\sigma}(t_i) > 0\}$. Finally, given a multiset m, tr(m) provides a trace σ such that supp$(\widehat{\sigma}) = \{x | m(x) > 0\}$.

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. For the sake of simplicity, the techniques of this paper assume models are specified with WF-nets[3].

In this paper we are interested not only in sequential observations of a model, but also in *steps*. A step is a sequence of multisets of activities. The following definitions relate the classical semantics of models and its correspondence to step semantics. Likewise, we lift the traditional notion of *fitness* to this context.

Definition 2 (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.*

Definition 3 (Full Model Step-Sequence). *A* step-sequence *$\bar{\sigma}$ is a sequence of multisets of transitions. Formally, given an alphabet T: $\bar{\sigma} = V_1 V_2 \ldots V_n$, with $V_i \in \mathcal{B}(T)$. Given a system net $N = (\langle P, T, \mathcal{F}\rangle, m_{start}, m_{end})$, a full step-sequence in N is a step-sequence $V_1 V_2 \ldots V_n$ such that there exists a full firing sequence $\sigma_1 \sigma_2 \ldots \sigma_n$ in N such that $\widehat{\sigma_i} = V_i$ for $1 \leq i \leq n$.*

The main metric in this paper to asses the adequacy of a model in describing a log is *fitness* [13], which is based on the reproducibility of a trace in a model:

Definition 4 (Fitting Trace). *A trace $\sigma \in T^*$ fits $SN = (N, m_{start}, m_{end})$ if σ coincides with a full firing sequence of SN, i.e.,$(N, m_{start})[\sigma\rangle(N, m_{end})$.*

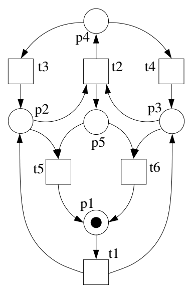
Definition 5 (Step-Fitting Trace). *A trace $\sigma_1 \sigma_2 \ldots \sigma_n \in T^*$ step-fits SN if there exists full model step-sequence $V_1 V_2 \ldots V_n$ of SN such that $V_i = \widehat{\sigma_i}$ for $1 \leq i \leq n$.*

3.2 Petri nets and Linear Algebra

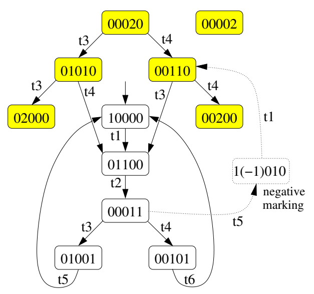
Let $N = \langle P, T, \mathcal{F}\rangle$ be a Petri net with initial marking m_0. Given a feasible sequence $m_0 \xrightarrow{\sigma} m$, the number of tokens for a place p in m is equal to the tokens of p in m_0 plus the tokens added by the input transitions of p in σ minus the tokens removed by the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

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

[3] The theory of this paper can deal with models having *silent* transitions. For the sake of simplicity, we do not consider them in the formalization.



(a)



(b)

$$\begin{array}{ccccccc} m & & m_0 & & N & & \boldsymbol{\sigma} \\ \begin{bmatrix}0\\0\\1\\1\\0\end{bmatrix} & = & \begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix} & + & \begin{bmatrix}-1 & 0 & 0 & 0 & +1 & +1\\ +1 & -1 & +1 & 0 & -1 & 0\\ +1 & -1 & 0 & +1 & 0 & -1\\ 0 & +1 & -1 & -1 & 0 & 0\\ 0 & +1 & 0 & 0 & -1 & -1\end{bmatrix} & & \begin{bmatrix}2\\1\\0\\0\\1\\0\end{bmatrix}\end{array}$$

(c)

Fig. 1: (a) Petri net, (b) Potential reachability graph, (c) Marking equation.

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 $m_0 \xrightarrow{\sigma} m$, 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* [12]. Figure 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. Figure 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 sufficient condition for reachability of a marking and replayability for a solution of (1).

For well-structured Petri nets classes equation (1) characterizes reachability. The largest class is *free-choice* [11], *live*, bounded and *reversible* nets. For this class, equation (1) together with a collection of sets of places (called *traps*) of the system completely characterizes reachability [4]. For the rest of cases, the problem of the spurious solutions can be palliated by the use of traps [5], or by the addition of some special places named *cutting implicit places* [12] to the original Petri net that remove spurious solutions from the original marking equation.

4 Approximate Alignment of Observed Behavior

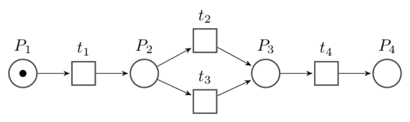


Fig. 2: Process model

As outlined above, the fitness dimension requires an *alignment* of trace and model, i.e., transitions or events of the 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 notation of aligning event log and process model was introduced by [1]. To achieve an alignment between process model and event log we need to relate *moves* in the trace to *moves* in the model. It may be the case that some 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 instance, consider the model in Fig. 2 and the trace $\sigma = t_1t_1t_4t_2$; some possible alignments are:

$$\gamma_1=\begin{array}{|c|c|c|c|c|} t_1&t_1&\perp&t_4&t_2\\ \hline t_1&\perp&t_2&t_4&\perp \end{array}\ \gamma_2=\begin{array}{|c|c|c|c|c|} t_1&t_1&\perp&t_4&t_2\\ \hline \perp&t_1&t_2&t_4&\perp \end{array}\ \gamma_3=\begin{array}{|c|c|c|c|c|} t_1&t_1&t_4&t_2&\perp\\ \hline t_1&\perp&\perp&t_2&t_4 \end{array}\ \gamma_4=\begin{array}{|c|c|c|c|c|} t_1&t_1&t_4&t_2&\perp\\ \hline \perp&t_1&\perp&t_2&t_4 \end{array}$$

The moves are represented in tabular form, where moves by trace log are at the top and moves by model are at the bottom of the table. For example the first move in γ_2 is $(t_1, \perp)$ and it means that the log moves t_1 while the model does not make any move. Cost can be associated to alignments, with asynchronous moves having greater cost than synchronous ones [1]. For instance, if unitary costs are assigned to asynchronous moves and zero cost to synchronous moves, alignment γ_2 has cost 3.

In this paper we introduce a different notion of alignment. In our notion, denoted as *approximate alignment*, moves are done on multisets of activities (instead of singletons, as it is done for the traditional definition of alignment). Intuitively, this allows for observing step-moves at different granularities, from the finest granularity ($\eta = 1$, i.e., singletons) to the coarse granularity ($\eta = |\sigma|$, i.e., the Parikh vector of the model's trace). To illustrate the notion of approximate alignment, consider again the process model in Fig. 2 and trace $\sigma = t_1t_1t_4t_2$. Some possible approximate alignments with different level of granularities are:

$$\alpha_1=\begin{array}{|c|} \{t_1,t_1,t_4,t_2\}\\ \hline \{t_2,t_1,t_4\} \end{array}\ \alpha_2=\begin{array}{|c|c|c|} t_1&t_1&\{t_4,t_2\}\\ \hline t_1&\perp&\{t_4\ ,t_2\} \end{array}\ \alpha_3=\begin{array}{|c|c|c|c|c|} t_1&t_1&t_4&t_2&\perp\\ \hline \perp&t_1&\perp&t_2&t_4 \end{array}$$

For instance, approximate alignment α_2 computes a step-sequence $t_1\{t_4, t_2\}$, meaning that to reproduce σ, the model first fires t_1 and then the step $\{t_4, t_2\}$ is computed, i.e., the order of the firings of the transitions of this step is not specified.

Definition 6 (Approximate Alignment). *Let A_M and A_L be the set of transitions in the model and the log, respectively, and $\perp$ denote the empty multiset.*

- *(X, Y) is a synchronous move if $X \in \mathcal{B}(A_L)$, $Y \in \mathcal{B}(A_M)$ and $Y = X$*
- *(X, Y) is a move in log if $X \in \mathcal{B}(A_L)$ and $Y = \perp$.*

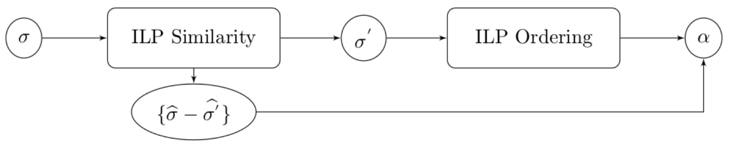


Fig. 3: Schematic of ILP approach for computing approximate alignments.

- *(X, Y) is a move in model if $X = \perp$ and $Y \in \mathcal{B}(A_M)$.*
- *(X, Y) is a approximate move if $X \in \mathcal{B}(A_L)$, $Y \in \mathcal{B}(A_M)$, $X \neq \perp$, $Y \neq \perp$, $X \neq Y$, and $X \cap Y \neq \perp$*
- *(X, Y) is an illegal move, otherwise.*

The set of all legal moves is denoted as A_{LM}. Given a trace σ, an approximate alignment is a sequence $\alpha \in A^*_{LM}$. The projection of the first element (ignoring $\perp$ and reordering the transitions in each move as the ordering in σ) results in the observed trace σ, and projecting the second element (ignoring $\perp$) results in a step-sequence.

Similar to the classical alignment, for a given trace different alignments can be defined with respect to the level of agreement with the trace. Hence, a distance function $\Psi : \mathcal{B}(A_L) \times \mathcal{B}(A_M) \to N$ must be defined for this goal. We propose the following implementation of the function: $\Psi(X, Y) = |X \Delta Y|$, although other possibilities could be considered[4]. For example $\Psi(\alpha_2) = \Psi(\{t_1\}, \{t_1\}) + \Psi(\{t_1\}, \perp) + \Psi(\{t_2, t_4\}, \{t_2, t_4\}) = 0 + 1 + 0 = 1$. For the other approximate alignments $\Psi(\alpha_1) = 0$ and $\Psi(\alpha_2) = 3$. Notice that the optimality (according to the distance function) of an approximate alignment depends on the granularity allowed.

5 Structural Computation of Approximate Alignments

Given an observed trace σ, in this paper we will compute approximate alignments using the structural theory introduced in Section 3.2. The technique will perform the computation of approximate alignments in two pipelined phases, each phase considering the resolution of an Integer Linear Programming (ILP) model containing the marking equation of the net corresponding to the model. The overall approach is described in Figure 3. In the first ILP model (ILP Similarity) a solution (the Parikh vector of a full firing sequence of the model) is computed that maximizes the similarity to $\widehat{\sigma}$. Elements in σ that cannot be replayed by the model in the Parikh vector found are removed for the next ILP, resulting in the projected sequence σ'. These elements are identified as *moves on log* cf. Definition 6, and will be inserted in the approximate alignment computed α. In the second ILP model (ILP Ordering), it is guaranteed that a feasible solution containing at least the elements in σ' exists. The goal of this second ILP model is to compute the approximate alignment given a user-defined granularity: it can be computed from the finest level ($\eta = 1$) to the most coarse level ($\eta = |\sigma|$).

[4] $X \Delta Y = (X \setminus Y) \cup (Y \setminus X)$.

5.1 ILP for Similarity: Seeking for an Optimal Parikh Vector

This stage will be centered on the marking equation of the input Petri net. Let $J = T \cap \text{supp}(\widehat{\sigma})$, the following ILP model computes a solution that is as similar as possible with respect to the firing of the activities appearing in the observed trace:

$$\begin{aligned} \text{Minimize} \sum_{t \in J} X^s[t] - \sum_{t \in J} X[t], \text{ Subject to:} & \\ m_{end} = m_{start} + \mathbf{N}.X & \\ \forall t \in J: \quad \widehat{\sigma}[t] = X[t] + X^s[t] & \\ X, X^s \geq \mathbf{0} & \end{aligned} \tag{2}$$

Hence, the model searches for a vector X that both is a solution to the marking equation and maximizes the similarity with respect to $\widehat{\sigma}$. Notice that the ILP problem has an additional set of variables $X^s \in \mathbb{N}^{|J|}$, and represents the slack variables needed when a solution for a given activity cannot equal the observed number of firings. By minimizing the variables X^s, and the variables X^s (negated), solutions to (2) clearly try to both assign zeros as much as possible to the X^s variables, and the opposite for the X variables in J (i.e., variables denoting activities appearing in σ).

Given an optimal solution X to (2), activities a_i such that $X[i] < \widehat{\sigma}(a_i)$ are removed from σ; in the simplest case, when $X[i] = 0$ and $\widehat{\sigma}(a_i) > 0$, every occurrence of a_i in σ will not appear in σ'. However, if $X[i] > 0$ and $X[i] < \widehat{\sigma}(a_i)$, all possibilities of removal should be checked when computing σ'[5].

5.2 ILP for Ordering: Computing an Aligned Step-Sequence

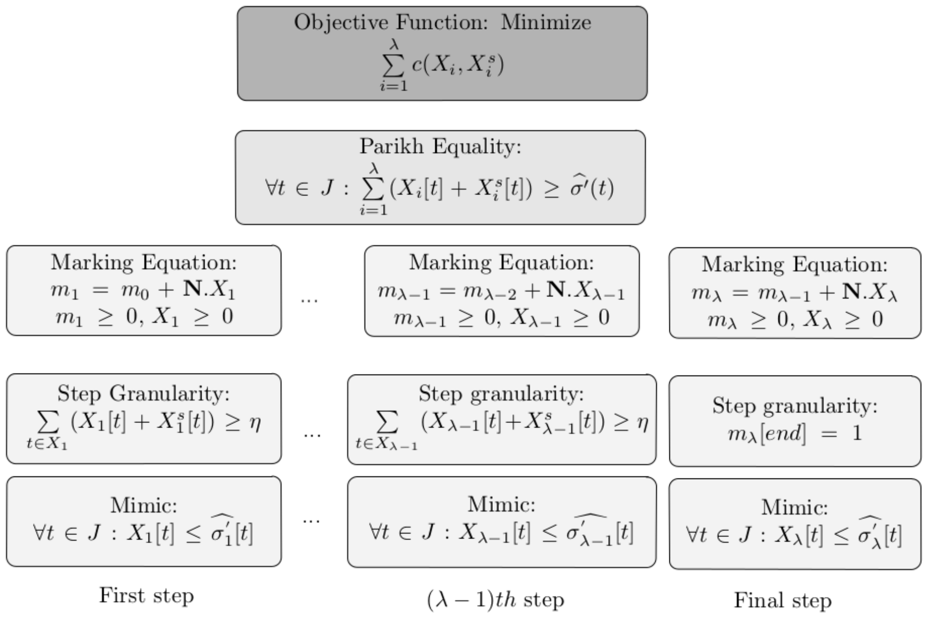
The schematic view of the ILP model for the ordering step is shown in Fig. 4. Given a granularity η, $\lambda = \lceil \frac{|\sigma'|}{\eta} \rceil$ steps are required for a step-sequence in the model that is aligned with σ'. Accordingly, the ILP model has variables $X_1 \ldots X_\lambda$ with $X_i \in \mathbb{N}^{|T|}$ to encode the λ steps of the marking equation, and variables $X_1^s \ldots X_\lambda^s$, with $X_i^s \in \mathbb{N}^{|J|}$ and $J = T \cap \text{supp}(\sigma')$, to encode situations where the model cannot reproduce observed behavior in some of these steps. We now describe the ILP model in detail.

Objective Function The goal is to compute a step-sequence which resembles as much as possible to σ'. Therefore transitions in supp(σ') have cost 0 in each step X_i whilst the rest have cost 1. Also, the slack variables X_i^s have cost 1.

Marking Equation Constraints The computation of a model's step-sequence $m_0 \xrightarrow{X_1} m_1 \xrightarrow{X_2} m_2 \ldots m_{\lambda-1} \xrightarrow{X_\lambda} m_{end}$ is enforced by using a chain of λ connected marking equations.

Parikh Equality Constraints To enforce the similarity of the Parikh vectors $X_1 \ldots X_\lambda$ with respect to $\widehat{\sigma'}$, this constraints require the sum of the assignments to variables X_i and X_i^s for every variable $t \in J$ should be greater or equal to $\widehat{\sigma'}(t)$. Given the cost function, solutions that minimize the assignment for the X_i^s variables are preferred.

[5] In our experiments, only the simplest cases were encountered.



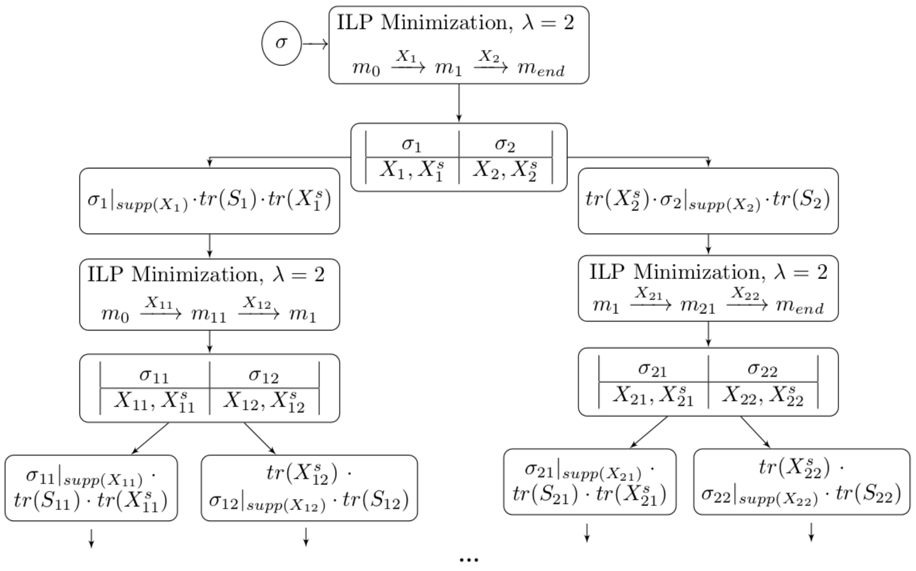
Objective Function: Minimize
$\sum_{i=1}^{\lambda} c(X_i, X_i^s)$
Parikh Equality:
$\forall t \in J : \sum_{i=1}^{\lambda} (X_i[t] + X_i^s[t]) \geq \widehat{\sigma'}(t)$
Marking Equation:
$m_1 = m_0 + \mathbf{N}.X_1$
$m_1 \geq 0, X_1 \geq 0$
Marking Equation:
$m_{\lambda-1} = m_{\lambda-2} + \mathbf{N}.X_{\lambda-1}$
$m_{\lambda-1} \geq 0, X_{\lambda-1} \geq 0$
Marking Equation:
$m_\lambda = m_{\lambda-1} + \mathbf{N}.X_\lambda$
$m_\lambda \geq 0, X_\lambda \geq 0$
Step Granularity:
$\sum_{t \in X_1} (X_1[t] + X_1^s[t]) \geq \eta$
Step granularity:
$\sum_{t \in X_{\lambda-1}} (X_{\lambda-1}[t] + X_{\lambda-1}^s[t]) \geq \eta$
Step granularity:
$m_\lambda[end] = 1$
Mimic:
$\forall t \in J : X_1[t] \leq \widehat{\sigma_1'}[t]$
Mimic:
$\forall t \in J : X_{\lambda-1}[t] \leq \widehat{\sigma_{\lambda-1}'}[t]$
Mimic:
$\forall t \in J : X_\lambda[t] \leq \widehat{\sigma_\lambda'}[t]$
First step
$(\lambda - 1)th$ step
Final step

Fig. 4: ILP model schema for the ordering step of Figure 3.

Step Granularity Constraints Require that the sum of model's steps X_i and the slack variables X_i^s is lower bounded by the given granularity η. Since the cost of variables X_i is lesser than the cost of X_i^s variables, the solutions will tend to assign as much as possible to X_i. Last step X_λ is not constrained in order to ensure the feasibility of reaching the final marking m_{end}.

Mimic Constraints The input sequence σ' is split into λ consecutive chunks, i.e., $\sigma' = \sigma_1'\sigma_2' \ldots \sigma_\lambda'$, with $|\sigma_i'| = \eta$, for $1 \leq i < \lambda$. This set of constraints require at each step that the multiset of observed transitions (X_i) must only happen if it has happened in the corresponding chunk σ_i'. It is worth to note that events with multiple occurrences are distinguished based on their positions.
Once the two steps of Fig. 3 are performed, the gathered information is sufficient to obtain an approximate alignment: on the one hand, the removed activities from the ILP model (2) are inserted as "moves in the log". On the other hand, the solution obtained from the ILP model of Fig. 4 provide the steps that can be appended to construct the final approximate alignment.

A note on completeness and optimality The global optimality guarantee provided in the approach of this paper is with respect to the similarity between the Parikh vectors of the computed and the observed trace. Informally, the technique searches for traces as similar as possible (c.f., ILP models (2)) and then computes the ordering (with respect to a given granularity). However, as the reader may have realized, by relying on the marking equation the approach presented in this section may be sensible to the existence of spurious solutions (see Sect. 3.2). This may have negative consequences since the



σ
ILP Minimization, $\lambda = 2$
$m_0 \xrightarrow{X_1} m_1 \xrightarrow{X_2} m_{end}$
σ_1 | σ_2
X_1, X_1^s | X_2, X_2^s
$\sigma_1|_{supp(X_1)} \cdot tr(S_1) \cdot tr(X_1^s)$
$tr(X_2^s) \cdot \sigma_2|_{supp(X_2)} \cdot tr(S_2)$
ILP Minimization, $\lambda = 2$
$m_0 \xrightarrow{X_{11}} m_{11} \xrightarrow{X_{12}} m_1$
ILP Minimization, $\lambda = 2$
$m_1 \xrightarrow{X_{21}} m_{21} \xrightarrow{X_{22}} m_{end}$
σ_{11} | σ_{12}
X_{11}, X_{11}^s | X_{12}, X_{12}^s
σ_{21} | σ_{22}
X_{21}, X_{21}^s | X_{22}, X_{22}^s
$\sigma_{11}|_{supp(X_{11})} \cdot tr(S_{11}) \cdot tr(X_{11}^s)$
$tr(X_{12}^s) \cdot \sigma_{12}|_{supp(X_{12})} \cdot tr(S_{12})$
$\sigma_{21}|_{supp(X_{21})} \cdot tr(S_{21}) \cdot tr(X_{21}^s)$
$tr(X_{22}^s) \cdot \sigma_{22}|_{supp(X_{22})} \cdot tr(S_{22})$
...

Fig. 5: Schema of the recursive approach.

marking computed may not be possible in the model, and/or the Parikh vectors may not correspond to a real model trace. For the former problem (marking reachability), in case of free-choice, live, bounded and reversible nets, this problem does not exists since the structural theory completely characterizes reachability [12]. For non-structured process models (e.g., spaghetti-like) or when the Parikh vector is spurious, the technique of this paper may still be applied, if the results obtained are verified a-posteriori by replaying the step-sequence computed. In Section 7 an evaluation over both well-structured and unstructured process models is reported, showing the potentials of the technique in practice for both situations.

6 The Recursive Algorithm

Section 5 shows how to compute approximate alignments using the structural theory of Petri nets through the marking equation. The complexity of the approach, which is NP-hard, can be measured by the size of the ILP formulation in the minimization step, in terms of number of variables: given a trace σ and a model with $|T|$ transitions and $|P|$ places, $(|T|+|J|+|P|)\cdot(|\sigma|/\eta)$ variables are needed, where η is the desired granularity and $J = T \cap \text{supp}(\widehat{\sigma})$. This poses a problem for handling medium/large process models.

In this section we will present a way to fight the aforementioned complexity, by using a recursive strategy that will alleviate significantly the approach presented in the previous section. The first step will be done as before, so we will focus on the second step (Ordering), and will assume that σ is the input sequence for this step. The overall idea is, instead of solving a large ILP instance, solve several small ILP instances that combined represent a feasible solution of the initial problem. Figure 5 illustrates the recursive approach: given a trace σ, on the top level of the recursion a couple of Parikh vectors X_1, X_2 are computed such that $m_0 \overset{X_1}{\to} m_1 \overset{X_2}{\to} m_{end}$, by using the Ordering ILP strategy of the previous section with granularity $|\sigma|/2$, with $\sigma = \sigma_1\sigma_2$. Some crucial observations can now be made:

1. X_1 and X_2 represent the optimal Parikh vectors for the model to mimic the observed behavior *in two steps*.
2. Elements from X_1 precede elements from X_2, but no knowledge on the orderings within X_1 or within X_2 is known yet.
3. Marking m_1 is the intermediate marking, being the final marking of X_1, and the initial marking of X_2.
4. Elements in supp$(X_1) \cap$ supp$(\widehat{\sigma_1})$ denote those elements in σ_1 that can be reproduced by the model if one step of size $|\sigma|/2$ was considered.
5. Elements in $S_1 = X_1 \setminus$ supp$(\sigma_1|_{\text{supp}(X_1)})$, denote the additional transitions in the net that are inserted to compute the final ordering. They will denote skipped "model moves" in the final alignment.
6. Elements in supp(X_1^s) denote those elements in σ_2 that the model needs to fire in the first part (but they were observed in the second part). They will denote asynchronous "model moves" in the final alignment.
7. 4, 5, and 6 hold symmetrically for X_2, X_2^s and σ_2.

The combination of these observations implies the independence between the computation of an approximate alignment for $\sigma_1|_{\text{supp}(X_1)} \cdot$ tr$(S_1) \cdot$ tr(X_1^s) and tr$(X_2^s) \cdot \sigma_2|_{\text{supp}(X_2)} \cdot$ tr(S_2), if the intermediate marking m_1 is used as connecting marking between these two independent problems[6]. This gives rise to the recursion step: each one of these two problems can be recursively divided into two intermediate sequences, e.g., $m_0 \overset{X_{11}}{\to} m_{11} \overset{X_{12}}{\to} m_1$, and $m_1 \overset{X_{21}}{\to} m_{21} \overset{X_{22}}{\to} m_{end}$, with $X_1 = X_{11} \cup X_{12}$ and $X_2 = X_{21} \cup X_{22}$. By consecutive recursive calls, more precedence relations are computed, thus progressing towards finding the full step sequence of the model.

Now the complexity analysis of the recursive approach can be measured: at the top level of the recursion one ILP problem consisting of $(|T| + |J_1|) \cdot 2 + |P|$ variables is solved, with $J_1 = T \cap$ supp$(\widehat{\sigma})$. In the second level, two ILP problems consisting of at most $(|T| + |J_2|) \cdot 2 + |P|$ variables, with $J_2 = \max(T \cap$ (supp$(\widehat{\sigma_1}) \cup X_1 \cup X_1^s), T \cap$ (supp$(\widehat{\sigma_2})) \cup X_2 \cup X_2^s)$. Hence as long as the recursion goes deeper, the ILP models have less variables. The depth of the recursion is bounded by $\log(|\sigma|)$, but in practice we limit the depth in order to solve instances that are small enough.

Let us show how the method works step by step for an example. Consider the model in Fig. 6 and a given non-fitting trace like $\sigma = t_5t_1t_3t_4t_4t_3t_4t_3$. On this trace ILP model (2) will not remove any activity from σ. We then concentrate on the recursive ordering step. First at the top level of Fig. 5 the solutions X_1, X_1^s, X_2 and X_2^s will be computed, with $\lambda = 2$.

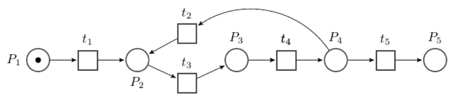


Fig. 6: Example with loop

[6] Note the different way the traces are obtained, e.g., in the right part tr(X_2^s) is the leftmost part since it denotes log moves that the model can produce on the left step.

$$\alpha_0 = \begin{array}{|c|c|} \sigma_1 = t_5 t_1 t_3 t_4 & \sigma_2 = t_4 t_3 t_4 t_3 \\ \hline X_1 \cup X_1^s = \{t_1, t_3, t_4, t_2\} & X_2 \cup X_2^s = \{t_3, t_3, t_4, t_5^s, t_2, t_4\} \end{array}$$

Notice that when seeking for an optimal ordering, t_5 does not appears in X_1 since then its firing will empty the net, and hence it appears in X_2^s (to guarantee reaching the final marking). The intermediate marking computed is $m_1 = \{P_2\}$. Accordingly, $\sigma_1|_{\text{supp}(X_1)} \cdot \text{tr}(S_1) \cdot \text{tr}(X_1^s) = t_1 t_3 t_4 \cdot t_2 \cdot \emptyset$, and $\sigma_2|_{\text{supp}(X_2)} \cdot \text{tr}(S_2) \cdot \text{tr}(X_2^s) = t_5 \cdot t_4 t_3 t_4 t_3 \cdot t_2$. Let us assume the recursion stops with subtraces of length less than 5, and then the ILP approach (with granularity 1 in this example) is applied. The left part will then stop the recursion, providing the optimal approximate alignment:

$$\begin{array}{|c|c|c|c|c|} t_5 & t_1 & t_3 & t_4 & \perp \\ \hline \perp & t_1 & t_3 & t_4 & t_2 \end{array}$$

For the subtrace on the right part, i.e., $t_5 t_4 t_3 t_4 t_3 t_2$ the recursion continues. Applying again the ILP with two steps, with $m_1 = \{P_2\}$ as initial marking, results in the following optimal approximate alignment:

$$\alpha_1 = \begin{array}{|c|c|} \sigma_{21} = t_5 t_4 t_3 & \sigma_{22} = t_4 t_3 t_2 \\ \hline X_{21} \cup X_{21}^s = \{t_3, t_4, t_2\} & X_{22} \cup X_{22}^s = \{t_4, t_3, t_5^s\} \end{array}$$

With $m_1 = \{P_2\}$ as intermediate marking. Whenever the recursion goes deeper, transitions are re-arranged accordingly in the solutions computed (e.g., t_2 moves to the left part of α_1, whilst t_5 moves to the right part). The new two subtraces induced from α_1 are $t_4 t_3 t_2$ and $t_5 t_4 t_3$. Since the length of both is less than 5, the recursion stops and the ILP model with granularity 1 is applied for each one, resulting in the solutions:

$$\alpha_{31} = \begin{array}{|c|c|c|} t_4 & t_3 & \perp \\ \hline \perp & \{t_3, t_4\} & t_2 \end{array} \quad \alpha_{32} = \begin{array}{|c|c|c|} t_4 & t_3 & \perp \\ \hline \perp & \{t_3, t_4\} & t_5 \end{array}$$

So the final optimal approximate alignment can be computed by concatenating the individual alignments found in preorder traversal:

$$\alpha = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|} t_5 & t_1 & t_3 & t_4 & \perp & t_4 & t_3 & \perp & t_4 & t_3 & \perp \\ \hline \perp & t_1 & t_3 & t_4 & t_2 & \perp & \{t_3, t_4\} & t_2 & \perp & \{t_3, t_4\} & t_5 \end{array}$$

which represents the step-sequence $\bar{\sigma} = t_1 t_3 t_4 t_2 \{t_3, t_4\} t_2 \{t_3, t_4\} t_5$ from the model of Fig. 6. Informally, the final approximate alignment reports that two activities t_2 were skipped in the trace, the ordering of two consecutive pair of events ($t_4 t_3$) was wrong, and transition t_5 was observed in the wrong order. Also, as mentioned in previous sections, the result of proposed method is an approximation to the corresponding optimal alignment, since some moves have non-singleton multisets (e.g., $\{t_3, t_4\}$). For these moves, the exact ordering is not computed although the relative position is known.

7 Experiments

The techniques of this paper have been implemented in Python as prototype tool that uses Gurobi for ILP resolution[7]. The tool has been evaluated over two different families of examples: on the one hand, large and well-structured synthetic benchmarks used

[7] The experiments have been done on a desktop computer with Intel Core i7-2.20GHz, and 5GB of RAM. Source code and benchmarks can be provided by contacting the first author.

in [10] for the distributed evaluation of fitness (see Table 1). On the other hand, a collection of large realistic examples from the literature has been also considered, some of them very unstructured (see Table 2). We compare our technique over $\eta = 1$ with the reference three approaches for computing optimal alignments from [1][8]: With or without ILP state space pruning, and the swap+replacement aware[9].

Table 1: BPM2013 artificial benchmark datasets

Model	$\|P\|$	$\|T\|$	$\|Arc\|$	Cases	Fitting	$\|\sigma\|_{avg}$
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

Table 2: Real benchmark datasets

Model	$\|P\|$	$\|T\|$	$\|Arc\|$	Cases	Fitting	$\|\sigma\|_{avg}$
Banktransfer	121	114	276	2000	No	58
Documentflow	334	447	2059	12391	No	5
Documentflow2	337	456	2025	12391	No	5
BPIC15_2	78	420	848	832	No	53
BPIC15_4	178	464	954	1053	No	44
BPIC15_5	45	277	558	1156	No	51

Comparison for Well-Structured and Synthetic Models Figure 7 provides the comparison in CPU time for the two families of approaches. One can see that for event logs with many short traces the approach from [1] takes advantage of the optimizations done in the implementation, e.g., caching and similar. Notice that those optimizations can also be implemented in our setting. But clearly, in large models and event logs with many long traces (*prDm6*, *prFm6* and *prGm6*) the three approaches from [1] either provide a solution in more than 12 hours or crash due to memory problems (N/A in the figure), while the recursive technique of this paper is able to find approximate alignments in a reasonable time. We have monitored the memory usage: our techniques use an order of magnitude less memory than the techniques from [1]. Finally, for these well-structured benchmarks, the approach presented in this technique never found spurious solutions.

[8] In spite of using $\eta = 1$, still the objects computed by our technique and the technique from [1] are different, and hence this comparison is only meant to provide an estimation on the speedup/memory/quality one can obtain by opting for approximate alignments.

[9] The plugin "Replay a log on Petri net for conformance analysis" from ProM with parameters "A^* cost-based fitness express with/without ILP and being/not being swap+replacement aware". We instructed the techniques from [1] to compute *one-optimal* alignment.

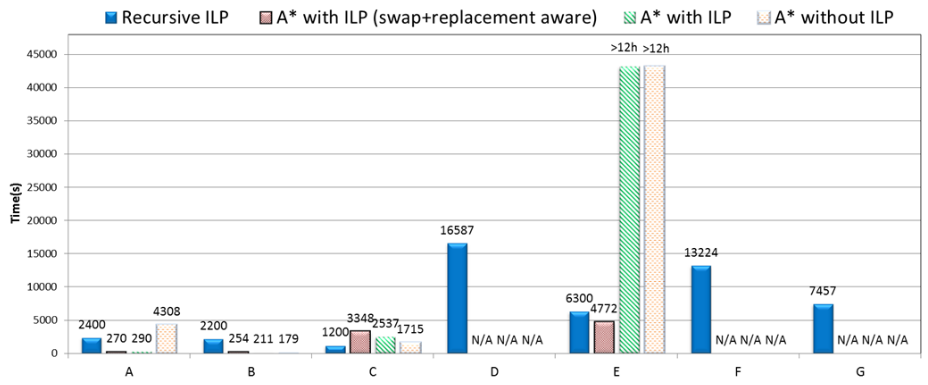


Fig. 7: Comparison of computation time for well-structured synthetic benchmarks.

Comparison for Realistic Benchmarks Figure 8 provides the comparison for the realistic examples from Table 2. The figure is split into structured and unstructured models[10]. Benchmark Banktransfer is taken from [15] and Documentflow benchmarks are taken from [16]. Some event logs from the last edition of the *BPI Challenge* were used, for which the models BPIC15_ 2, BPIC15_ 4, BPIC15_ 2 were generated using Inductive Miner plugin of ProM with noise threshold 0.99, 0.5 and 0.2, respectively. For the structural realistic models, the tendency of the previous structured benchmarks is preserved. For the two unstructured benchmarks, the technique of this paper is able to produce approximate alignments in considerably less time than the family of A^*-based techniques. Moreover, for the benchmarks from the BPI challenge, the A^*-based techniques crashes due to memory problems, whilst our technique again can handle these instances. The memory usage of our technique is again one order of magnitude less than the compared A^*-based techniques, but for the unstructured models spurious solutions were found.

Quality of Approximate Alignments Table 3 reports the evaluation of the quality of the results obtained by the two approaches for the cases where [1] provides a solution. We considered two different comparisons: i) fine-grained comparison between the sequences computed by [1] and the step-sequences of our approach, and ii) coarse-grained comparison between the fitness value of the two approaches. For i), we considered two possibilities: using the *Edit* or *Jaccard* distances. For the first, given a trace σ and a step-sequence $\bar{\gamma}$, we simply take the minimal edit distance between σ and any of the linearizations of $\bar{\gamma}$. For the Jaccard distance, which measures similarities between sets, we considered both objects as sets and used this metric to measure their similarity. In the table, we provide the average of these two

Table 3: Quality comparison.

Model/ Case	ED	Jaccard	MSE
prAm6	0.25	0	0.0002
prBm6	0	0	0
prCm6	2.99	0.01	0.0093
prEm6	0	0	0
Banktransfer	4.30	0.04	0.0400
Documentflow	3.16	0.27	0.0310
Documentflow2	3.17	0.29	0.0330

[10] Most of the realistic benchmarks in Table 2 have silent transitions.

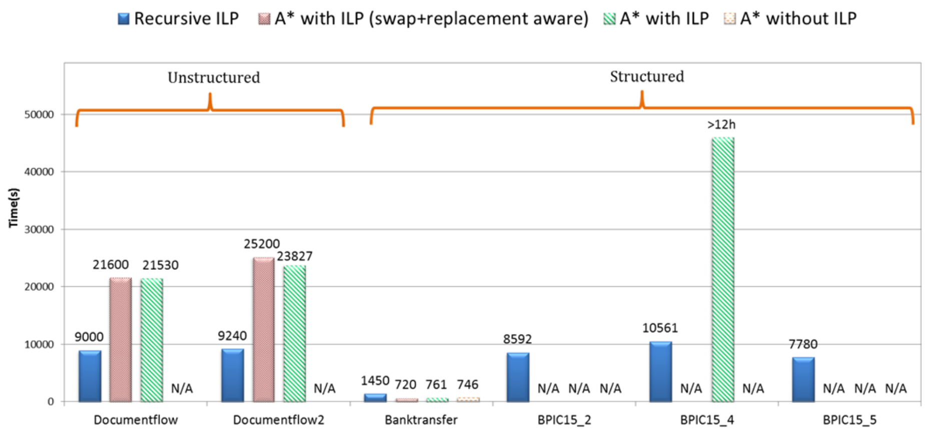


Fig. 8: Comparison of computation time for realistic benchmarks.

metrics per trace, e.g. for prAm6 the two approaches are less than 1 edit operation (0.25) different on average. For measuring ii), the *Mean Square Root* (MSE) over the fitness values provided by both metrics is reported. Overall, one can see that both in fine-grained and coarse-grained comparisons, the approach of this paper is very close to the optimal solutions computed by [1], specially for well-structured models.

8 Conclusions and Future Work

Approximate alignments generalize the notion of alignment by allowing moves to be non-unitary, thus providing a user-defined mechanism to decide the granularity for observing deviations of a model with respect to observed behavior. A novel technique for the computation of approximate alignments has been presented in this paper, based on a divide-and-conquer strategy that uses ILP models both as splitting criteria and for obtaining partial alignments. The technique has been implemented as a prototype tool and the evaluation shows promising capabilities to handle large instances.

As future work, we see many possibilities. On the one hand, a thorough evaluation of the quality of the obtained results over a large set of benchmarks will be carried out. Second, extending the current theory to deal with models having duplicate transitions will be considered. Also, the incorporation of natural optimizations like parallelization and caching would have an strong impact. Finally, as the recursive method presented in this paper can be used as a high-level strategy for partitioning the alignment computations, we plan to combine it with the A^* approach from [1] for computing partial alignments on the leafs of the recursion.

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. Arya Adriansyah. *Aligning observed and modeled behavior*. PhD thesis, Technische Universiteit Eindhoven, 2014.
2. Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Measuring precision of modeled behavior. *Inf. Syst. E-Business Management*, 13(1):37–67, 2015.
3. Joos C. A. M. Buijs. *Flexible Evolutionary Algorithms for Mining Structured Process Models*. PhD thesis, Technische Universiteit Eindhoven, 2014.
4. J. Desel and J. Esparza. Reachability in cyclic extended free-choice systems. *TCS 114, Elsevier Science Publishers B.V.*, 1993.
5. J. Esparza and S. Melzer. Verification of safety properties using integer programming: Beyond the state equation. *Formal Methods in System Design*, (16):159–189, 2000.
6. Dirk Fahland and Wil M. P. van der Aalst. Model repair - aligning process models to reality. *Inf. Syst.*, 47:220–243, 2015.
7. Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Scalable process discovery with guarantees. In *Enterprise, Business-Process and Information Systems Modeling - 16th International Conference, BPMDS 2015, 20th International Conference, EMMSAD 2015, Held at CAiSE 2015, Stockholm, Sweden, June 8-9, 2015, Proceedings*, pages 85–101, 2015.
8. Xixi Lu, Dirk Fahland, and Wil M. P. van der Aalst. Conformance checking based on partially ordered event data. In *Business Process Management Workshops - BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers*, pages 75–88, 2014.
9. Xixi Lu, Ronny Mans, Dirk Fahland, and Wil M. P. van der Aalst. Conformance checking in healthcare based on partially ordered event data. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation, ETFA 2014, Barcelona, Spain, September 16-19, 2014*, pages 1–8, 2014.
10. Jorge Munoz-Gama, Josep Carmona, and Wil M. P. van der Aalst. Single-entry single-exit decomposed conformance checking. *Inf. Syst.*, 46:102–122, 2014.
11. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–574, April 1989.
12. M. Silva, E. Teruel, and J. M. Colom. Linear algebraic and linear programming techniques for the analysis of place/transition net systems. In Reisig, W. and Rozenberg, G., editors, *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, volume 1491, pages 309–373. Springer-Verlag, 1998.
13. Wil M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
14. Wil M. P. van der Aalst. Decomposing Petri nets for process mining: A generic approach. *Distributed and Parallel Databases*, 31(4):471–507, 2013.
15. Seppe K. L. M. vanden Broucke, Jorge Munoz-Gama, Josep Carmona, Bart Baesens, and Jan Vanthienen. Event-based real-time decomposed conformance analysis. In *On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings*, pages 345–363, 2014.
16. Jochen De Weerdt, Seppe K. L. M. vanden Broucke, Jan Vanthienen, and Bart Baesens. Active trace clustering for improved process discovery. *IEEE Trans. Knowl. Data Eng.*, 25(12):2708–2720, 2013.