Approximate Computation of Alignments of Business Processes Through Relaxation Labelling

Lluís Padró(✉) and Josep Carmona

Computer Science Department, Universitat Politècnica de Catalunya, Barcelona, Spain
{padro,jcarmona}@cs.upc.edu

Abstract. A fundamental problem in conformance checking is aligning event data with process models. Unfortunately, existing techniques for this task are either complex, or can only be applicable to restricted classes of models. This in practice means that for large inputs, current techniques often fail to produce a result. In this paper we propose a method to approximate alignments for unconstrained process models, which relies on the use of relaxation labelling techniques on top of a partial order representation of the process model. The implementation on the proposed technique achieves a speed-up of several orders of magnitude with respect to the approaches in the literature (either optimal or approximate), often with a reasonable trade-off on the cost of the obtained alignment.

1 Introduction

Conformance checking is expected to be the fastest growing segment in process mining for the next years[1]. The main reason for this forthcoming industrial interest is the promise of having event data and process models aligned, thus increasing the value of process models within organizations [5]. On its core, most conformance checking techniques rely on the notion of *alignment* [1]: given an observed trace σ, query the model to obtain the run γ most similar to σ. The computation of alignments is a computational challenge, since it encompasses the exploration of the model state space, an object that is worst-case exponential with respect to the size of the model or the trace.

Consequently, the process mining field is facing the following paradox: whilst there exist techniques to discover process models arbitrarily large, most of the existing alignment computation techniques will not be able to handle such models. This hampers the widespread applicability of conformance checking in industrial scenearios.

In some situations, one can live with approximations: For instance, when the model must be *enhanced* with the information existing in the event log (e.g.,

[1] https://www.marketsandmarkets.com/Market-Reports/process-analytics-market-254139591.html.

T. Hildebrandt et al. (Eds.): BPM 2019, LNCS 11675, pp. 250–267, 2019.
https://doi.org/10.1007/978-3-030-26619-6_17

performance, decision point analysis), or when one aims to *animate* the model by replaying the log on top of it (two of the most celebrated functionalities of commercial process mining tools). Examples of approximations are *token-replay* techniques [14], which do not guarantee optimality, or the techniques in [16,17], which do not guarantee replayability in general, but that significantly alleviate the complexity of the alignment computation. The method presented in this paper is of this latter type.

We propose a method that is applied on a partial order representation of the process model [7]. A pre-processing step is then done once on the partial order, to gather information (shortest enabling paths between event activations and computing the behavioral profiles) that is used for aligning traces. We assume this is a plausible scenario in many situations, where the model is well-known and it is admissible to have some pre-processing before of aligning traces. For computing alignments, the method uses *Relaxation Labeling* algorithm to map events in each trace to nodes in the partial order. On a training phase, the weights that guide the relaxation labelling problem are tuned. Once this information is obtained, the approach is ready to be applied in the second phase. It is remarkable that several modes can be considered corresponding to different objectives, e.g., strive for replayability, optimality, or a weighted combination.

Experimental results computed over existing benchmarks show promising speedups in computation time, while still being able to derive reasonable approximations when compared to reference techniques.

The paper is organized as follows: next section provides related work for the problem considered in this paper. Then in Sect. 3 we introduce the background of the paper, necessary for understanding the main content in Sect. 4. Experimental evaluation and tool support is provide in Sect. 5, before concluding the paper.

2 Related Work

The work in [1] proposed the notion of alignment, and developed a technique based on A^* to compute optimal alignments for a particular class of process models. Improvements of this approach have been presented in [20]. Alternatives to A^* have appeared very recently: in the approach presented in [6], the alignment problem is mapped as an *automated planning* instance. Automata-based techniques have also appeared [10,13].

The work in [17] presented the notion of *approximate* alignment to alleviate the computational demands by proposing a recursive paradigm on the basis of structural theory of Petri nets. In spite of resource efficiency, the solution is not guaranteed to be executable. A follow-up work of [17] is presented in [21], which proposes a trade-off between complexity and optimality of solutions, and guarantees executable results. The technique in [16] presents a framework to reduce a process model and the event log accordingly, with the goal to alleviate the computation of alignments. The obtained alignment, called *macro-alignment* since some of the positions are high-level elements, is expanded based on the information gathered during the initial reduction. Techniques using local search have recently been also proposed [15]. Decompositional techniques have been presented [11,19] that instead of computing optimal alignments, they focus on the *decisional problem* of whereas a given trace fits or not a process model.

Recently, two different approaches have appeared: the work in [3] proposes using binary decision diagrams to alleviate the computation of alignments. The work in [4], which has the goal of maximizing the synchronous moves of the computed alignments, uses a pre-processing step on the model.

The method of this paper is an alternative to the methods in the literature, useful when computation time and/or memory requirements hamper their applicability, and suboptimal solutions are acceptable. In such a scenario, our approach produces solutions close to the optimum with a much smaller computational cost.

3 Preliminaries

3.1 Petri Nets, Unfoldings and Process Mining

A Process Model defined by a *labeled Petri net system* (or simply *Petri net*) consists of a tuple $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda \rangle$, where P is the set of places, T is the set of transitions (with $P \cap T = \emptyset$), $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, m_0 is the initial marking, m_f is the final marking, Σ is an alphabet of actions, and $\lambda : T \rightarrow \Sigma \cup \{\tau\}$ labels every transition with an action or as silent. The semantics of Petri nets is given in terms of *firing sequences*. A *marking* is an assignment of a non-negative integer to each place. A transition t is *enabled* in a marking m when all places in its preset ${}^\bullet t \overset{\text{def}}{=} \{y \in P \cup T \mid (t, y) \in F\}$ 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 its postset $t^\bullet \overset{\text{def}}{=} \{y \in P \cup T \mid (y, t) \in F\}$. A marking m' is *reachable* from m if there is a sequence of firings $\langle t_1 \ldots t_n \rangle$ that transforms m into m', denoted by $m[t_1 \ldots t_n\rangle m'$. The set of reachable markings from m_0 is denoted by $[m_0\rangle$. A Petri net is *k-bounded* if no marking in $[m_0\rangle$ assigns more than k tokens to any place. A Petri net is *safe* if it is 1-bounded. In this paper we assume safe Petri nets. A firing sequence $u = \langle t_1 \ldots t_n \rangle$ is called a *run* if it can fire from the initial marking: $m_0[u\rangle$; it is called a *full run* if it additionally reaches the final marking: $m_0[u\rangle m_f$. We write $\mathit{Runs}(N)$ for the set of full runs of Petri net N. Given a full run $u = \langle t_1 \ldots t_n \rangle \in \mathit{Runs}(N)$, the sequence of actions $\lambda(u) \overset{\text{def}}{=} \langle \lambda(t_1) \ldots \lambda(t_n) \rangle$ is called a *(model) trace of* N.

A *finite and complete unfolding prefix* π of a Petri net N is a finite acyclic net which implicitly represents all the reachable states of N, together with transitions enabled at those states. It can be obtained through unfolding N by successive firings of transitions, under the following assumptions: (a) for each new firing, a fresh transition (called an *event*) is generated; (b) for each newly produced token a fresh place (called a *condition*) is generated. The unfolding is infinite whenever N has an infinite run; however, if N has finitely many reachable states, then the unfolding eventually starts to repeat itself and can be truncated (by identifying a set of *cut-off* events) without loss of information, yielding a finite and complete prefix. We denote by B, E and $E_{cut} \subseteq E$ the sets of conditions, events and cut-off events of the prefix, respectively. Efficient algorithms exist for building such prefixes [7–9].

In this paper we use *behavioral profiles* [23] to guide the search for alignments.

Definition 1 (Behavioral Profiles [23]). *Let x, y be two transitions of a Petri net N. $x \succ y$ if there exists a run of N where x appears before of y. A pair of transitions (x, y) of a Petri net is in at most one of the following behavioral relation:*

- *The* strict *order relation $x \rightsquigarrow y$, if $x \succ y$ and $y \not\succ x$*
- *The* exclusiveness *order relation $x + y$, if $x \not\succ y$ and $y \not\succ x$*
- *The* interleaving *order relation $x \| y$, if $x \succ y$ and $y \succ x$*

Definition 2 (Log, Alignment). *A* log *over an alphabet Σ is a finite set of words $\sigma \in \Sigma^*$, called* log traces. *Given a Petri net $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda \rangle$, and a log trace σ, an alignment is a full run of the model $\gamma \in Runs(N)$ with minimal edit distance to σ, i.e., $\forall \gamma' \in Runs(N) : \gamma' \neq \gamma \implies dist(\sigma, \gamma') \geq dist(\sigma, \gamma)$.*

3.2 Relaxation Labelling Algorithm

Relaxation labelling (RL) is a generic name for a family of iterative algorithms which perform function optimization based on local information, from a constraint satisfaction approach. Although other optimization algorithms could have been used (e.g. genetic algorithms, simulated annealing, or even ILP) we found RL to be suitable to our purposes, given its ability to use models based arbitrary context constraints, to deal with partial information, and to provide a solution even when fed with inconsistent information (though the solution will not necessarily be consistent if that is the case).

Given a set of variables $\mathcal{V} = \{v_1, \ldots, v_n\}$, the algorithm goal is to assign a value (label) to each of them. Values for each $v_i \in \mathcal{V}$ are chosen from a finite discrete set of labels $\mathcal{L}(v_i) = \{t_{i_1}, \ldots, t_{i_{m_i}}\}$. Variable-label assignments are rewarded or penalized by a set of constraints $\mathcal{C}$. Each constraint $r \in \mathcal{C}$ has the form:

$$C_r \quad (v_i : t_{ij}) \quad [(v_{i_1} : t_{i_1 j_1}), \ldots, (v_{i_{d_r}} : t_{i_{d_r} j_{d_r}})]$$

where $(v_i : t_{ij})$ is the *target assignment* of the constraint (i.e the assignment that is rewarded or penalized by the constraint), $[(v_{i_1} : t_{i_1 j_1}), \ldots, (v_{i_{d_r}} : t_{i_{d_r} j_{d_r}})]$ are the constraint *conditions* (i.e. the assignments of other variables required for the constraint to be satisfied), and C_r is a real value expressing *compatibility* (or *incompatibility* if negative) of the target assignment with respect to the conditions.

Algorithm 1 shows the pseudo-code of the used RL variant (a variety of formulas can be used to compute S_{ij} or $p_{ij}(s+1)$. See [18] for a summary), where:

- p_{ij} is the current weight for the assignment $(v_i : t_j)$. Assignment weights are normalized so that $\forall i \ \sum_{j=1}^{m_i} p_{ij} = 1$.

- $Inf(r) = C_r \times p_{i_1 j_1}(s) \times \ldots \times p_{i_{d_r} j_{d_r}}(s)$, is the *influence* of constraint r on its target assignment, computed as the product of the current weights (at time step s) of the assignments in the constraint *conditions* (representing *how satisfied* the conditions are in the current context) multiplied by the constraint compatibility value C_r (stating *how compatible* is the target assignment with the context).
- $\mathcal{C}_{ij} \subseteq \mathcal{C}$ is the subset of constraints that have the pair $(v_i : t_j)$ as target assignment.
- S_{ij} is the total support received by pair $(v_i : e_j)$ from all constraints targeting it. Since S_{ij} depends on the conditioning pairs, it will change over time.

```
/* Start in a uniformly distributed labelling $P$                          */
$P := \{\{p_{11} \ldots p_{1m_1}\}, \ldots, \{p_{n1} \ldots p_{nm_n}\}\}$;
/* Time step counter                                                       */
$s := 0$;
repeat
    /* Compute the support $S_{ij}$ that each label receives from the
       current weights for the labels of the other variables and the
       constraints contributions                                           */
    for each variable $v_i \in \mathcal{V}$ do
        for each label $t_{ij} \in \mathcal{L}(v_i)$ do
            $S_{ij} := \sum_{r \in \mathcal{C}_{ij}} Inf(r)$
        end
    end
    /* Compute (and re-normalize) weights for each variable label at
       time step $s+1$ according to the support they receive              */
    for each variable $v_i \in \mathcal{V}$ do
        for each label $t_{ij} \in \mathcal{L}(v_i)$ do
            $p_{ij}(s+1) := \dfrac{p_{ij}(s) \times (1 + S_{ij})}{\sum_{k=1}^{m_i} p_{ij}(s) \times (1 + S_{ik})}$
        end
    end
    $s := s + 1$
until no more changes;
```

Algorithm 1. Pseudo code of the RL algorithm.

At each time step, the algorithm updates the weights of each possible labels for each variable. The results are normalized per variable, raising weights for labels with higher support, and reducing them for those with lower support. Advantages of the algorithm are:

- Its expressivity: The problem is stated in terms of assigning *labels* to variables, and a set of constraints between variable-label assignments, allowing to model any discrete combinatorial problem. The algorithm can deal with any kind of constraints encoding any relevant domain information.

– Its highly local character (each variable can update its label weights given only the state at previous time step), which makes the algorithm highly parallelizable.
– Its flexibility: Total consistency or completeness of constraints is not required.
– Its robustness: It can give an answer to problems without an exact solution (incomplete or partially incompatible constraints, insufficient data, etc.)
– Its complexity. Being n the number of variables, v the average number of possible labels per variable, c the average number of constraints per label, and I the average number of iterations until convergence, the average cost is $n \times v \times c \times I$. Note that some of these factors can be made constants: The algorithm can be stopped if convergence is not reached after a maximum number of iterations. In most problems v and c do not depend on n, or if they do, they can be bounded (e.g. generating constraints only for nearby neighbors instead of all variables, or pre-filtering unlikely values). In general, for problems with a large amount of variables, the complexity can be controlled at the price of reducing expressivity and/or result accuracy, obtaining accurate enough models with linear or quadratic asymptotic costs.

Drawbacks of the algorithm are:

– Found optima are local, and convergence is not guaranteed in the general case.
– Constraints must be designed manually, since they encode the domain knowledge about the problem.
– Constraint weights must be assigned manually and/or optimized on tuning data.

4 Framework to Approximate Alignments

Figure 1 presents an overall description of the framework: A preprocessing step, *(a)* inside the gray box, is executed only once per model to compute the model unfolding, its behavioural profile, and the shortest enabling path between each pair of nodes. Then, it is used as many times as needed to align log traces. The alignment algorithm, *(c)* relaxation labeling, uses weighted constraints *(b)*, and although their weights can simply be set manually, better results are obtained if they are tuned using available training data. The algorithm produces partial alignments without model moves, which are added –if needed– by a completion post-process *(d)*. The weight tuning procedure is exactly the same: The system is run on different combinations of constraint weights on a separate section of the dataset, and the combination producing the best results is chosen to be used on test data (or used in production).

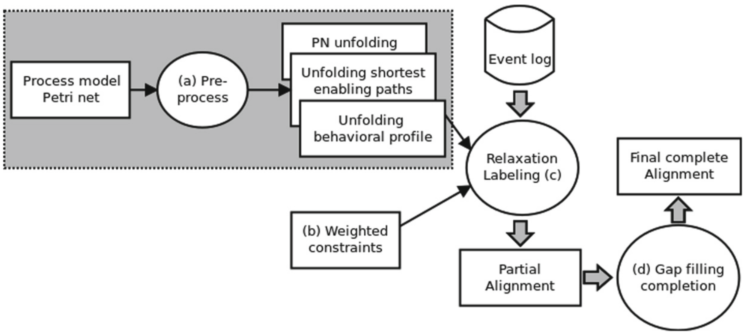

Fig. 1. Overall framework representation.

4.1 Stage 1: Pre-computation of Model Unfolding and Additional Information

We use one of the state-of-the-art techniques to compute an unfolding π of the Petri net [9]. There are two main reasons to use the unfolding instead of the Petri net. First, events in the unfolding correspond to a particular firing of a transition in a Petri net, thus making the correspondence between events in the trace and events in the unfolding meaningful[2]. Second, by being well-structured (e.g., having a clear initial and final node), the computation of alignments is facilitated.

Two types of information between any pair of events in the unfolding are required in our setting: (1) Behavioral relations are used to guide RL in order to reward/penalize particular assignments between events in the trace and unfolding transitions. (2) Shortest enabling paths are necessary for completing the alignment when gaps exist in alignment arising from the solution found by the RL algorithm. Notice that this information is computed only once per model, before aligning each trace in the log.

Behavioral Relations Between Unfolding Events. As it has been pointed out [2,22], not all runs of the Petri net are possible in the complete unfolding, which impacts the behavioral information between events in the unfolding. To amend this, either the unfolding is extended beyond cut-off events so that all relations are visible [2], or the behavioral relations are adapted to consider the discontinuities due to cut-off events [22].

In this paper we opted instead for a pragmatic setting: next to the original unfolding π, a copy π^r where the backward-conflicts branches and loops corresponding to the cut-off events are computed (see Fig. 2). We call π^r *reconnected unfolding*. Notice that, in contrast to the original unfolding, in a reconnected unfolding all the runs of the original Petri net are possible.

[2] Notice that a transition can correspond to several different firing modes, that depend on the context, which will be represented as different events in the unfolding.

Next, the behavioral profiles (c.f. Definition 1) for both π and π^r are computed. Apart from obtaining the behavioral relations for events, computing these relations both in π and π^r is useful to elicit loop behavior: for two events e_1, e_2, if $e_1 \nparallel e_2$ in π, but $e_1 \| e_2$ in π^r, then the concurrency of e_1 and e_2 is due to the existence of a loop in the original Petri net[3], while if $e_1 \| e_2$ in π, then e_1 and e_2 are in a parallel section (which may or may not be inside a loop). These behavioral relations (ordering, exclusiveness, interleaving and loop relations) are then used to assign different constraint weights in the created constraint satisfaction problem instance (see next Section).

Shortest Enabling Paths Between Unfolding Events. Given two events e_1, e_2 in π^r, the *shortest enabling path* is the minimal set of events needed to enable e_2 after e_1 fires. Since we pose the problem as choosing an event (transition) in the unfolding for each event in the trace, the RL algorithm will not suggest new events to be inserted in the trace (i.e. *model moves*). To complete the alignment with required model moves, we fill the gaps in the trace with the shortest enabling path between events, which is precomputed off-line, only once per model.

The length of the shortest enabling path between two nodes is also used to modulate the weight of the constraints (see Sect. 4.2).

4.2 Stage 2: Computation of Mapping Through RL

Given π^r and a trace $\sigma = a_1 \dots a_n \in L$, we post the alignment problem as a *consistent labelling problem* (CLP), which can be solved via suboptimal constraint satisfaction methods, such as RL. We will illustrate how we build our labelling problem, as well as how it is handled by the RL algorithm, with the example M8 model from the dataset described in [12][4], and shown in Fig. 2. Below, to avoid ambiguities, we will refer to events in the unfolding as *transitions*.
The CLP is built as follows:

- Each event $a_i \in \sigma$ is a variable v_i for the CLP problem. The set of variables is $\mathcal{V} = \{v_1, \dots, v_n\}$.
- For each variable $v_i \in \mathcal{V}$, we have a set of labels $\mathcal{L}(v_i) = \{e_{i_1}, \dots, e_{i_{m_i}}, \texttt{NULL}\}$, containing all transitions e_{i_k} in π^r such that $\lambda(e_{i_k}) = a_i$, plus one NULL label to allow for the option to not align a particular event in the trace (a *log move*). Figure 3 shows the aforementioned encoding for the trace $BCGHEFDA$ and the M8 model in Fig. 2[5]. Notice that selecting the possible labels for each

[3] In case models do not have duplicate labels, the detection of loops can alternatively be performed as it was done in [2].

[4] https://data.4tu.nl/repository/uuid:44c32783-15d0-4dbd-af8a-78b97be3de49.

[5] Notice that, for the sake of simplicity, the example in Fig. 2 only contains one unfolding event per label. In general several events in the unfolding can have the same label, and our technique handles that general case.

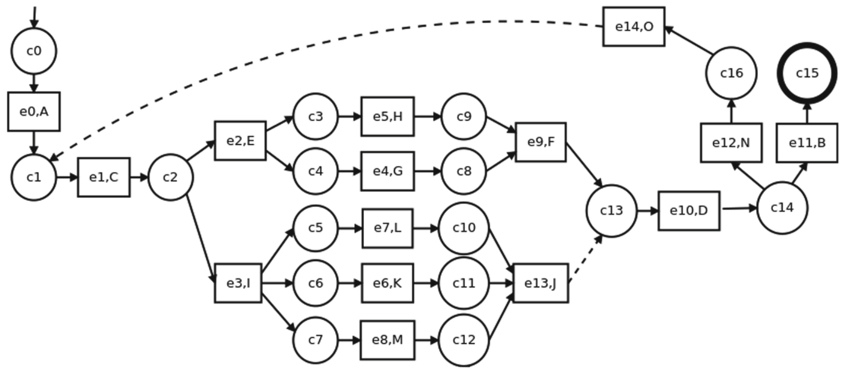

Fig. 2. Reconnected unfolding π^r for model M8. Dashed edges indicate reconnected cut-offs.

variable we can introduce hard constraints (e.g. if an assignment is not possible because of, e.g., data type incompatibility, that value is excluded from $\mathcal{L}(v_i)$, avoiding the need of having to encode this fact as a constraint).

– In this work, we base our constraints on the similarity between the order of the events in the trace and the model, although other kinds of information could be used if available. For instance, a possible constraint for the trace in Fig. 3 and the M8 process model in Fig. 2 could be +10.0 $(v_3 : e4)$ $[(v_2 : e1), (v_4 : e5)]$, stating that the assignment of label $e4$ to variable v_3 gets a positive reward of +10.0 from a context where v_2 is assigned label $e1$ and v_4 is assigned label $e5$.

Trace events (variables)	v_1 (B)	v_2 (C)	v_3 (G)	v_4 (H)	v_5 (E)	v_6 (F)	v_7 (D)	v_8 (A)
Possible alignments (labels)	e11 **NULL**	**e1** NULL	**e4** NULL	**e5** NULL	e2 **NULL**	**e9** NULL	**e10** NULL	e0 **NULL**

Fig. 3. Mapping of the trace *BCGHEFDA* alignment with model M8 as a consistent labelling problem. Boldface labels indicate the solution selected by the RL algorithm.

To avoid an explosion of the number of constraints, we restricted ourselves to use binary constraints –that is, involving just one target assignment and one condition–, except in the case of the *Deletion* constraint (see below). We now provide a description of the constraints used.

Compatibility Constraints. Each constraint has a *compatibility value* that may be either positive (to reward consistent assignments) or negative (to penalize inconsistent combinations). The weight for each constraint is tuned experimentally.

In what follows, $d(v_i, v_j) \overset{\text{def}}{=} |i - j|$ refers to the distance between events a_i and a_j in σ, and $d(e_i, e_j)$ corresponds to the length of a shortest enabling path between transitions e_i and e_j in π^r.

Constraint Patterns: For each combination of two possible assignments $(v_i : e_p), (v_j : e_q)$, we create the following constraint instances:

$$\begin{array}{lll} C_r & (v_i : e_p) & [(v_j : e_q)] \\ C_r & (v_j : e_q) & [(v_i : e_p)] \end{array}$$

for each of the following cases that are applicable. The compatibility value C_r depends on each case:

- *Right order.* If v_i precedes v_j in σ (i.e., $i < j$), and $e_p \rightsquigarrow e_q$ in π^r, C_r is positive, and inversely proportional to $|d(v_i, v_j) - d(e_p, e_q)|$, rewarding assignments in the right order, with higher rewards for closer assignments.
- *Wrong order.* If v_i follows v_j in σ (i.e., $i > j$), and $e_p \rightsquigarrow e_q$ in π^r, C_r is negative, penalizing assignments with crossed ordering in the trace with respect to the model.
- *Exclusive.* If v_i and v_j co-occur in the trace but $e_p + e_q$ in π^r, C_r is negative, penalizing assignments that should not happen in the same trace.
- *Parallel.* If v_i and v_j co-occur in the trace, and $e_p \| e_q$ in π, indicating the presence of a parallel section, C_r is positive, and inversely proportional to $|d(v_i, v_j) - d(e_p, e_q)|$, rewarding this combination in any order, with higher rewards for closer assignments.
- *Loop.* If v_i and v_j co-occur in the trace, $e_p \| e_q$ in π^r, and $e_p \nparallel e_q$ in π indicating that the interleaving is due to the presence of a loop, C_r is positive, which allows the repetition and alternation of looped events.

Deletion. Also, for each combination of three possible assignments $(v_{i-1} : e_m), (v_i : e_p), (v_{i+1} : e_q)$ such that $1 < i < n$ (i.e. three consecutive events in the trace) if the shortest enabling path from e_m to e_q via e_p in π^r is longer than the shortest enabling path from e_m to e_q not crossing e_p, we create the constraint instance:

$$C_r \quad (v_i : e_p) \quad [(v_{i-1} : e_m), (v_{i+1} : e_q)]$$

where C_r is negative. This constraint penalizes the alignment of an event if that would require more model moves (and thus a higher cost) than its deletion.

Figure 4 shows examples of how these patterns are instantiated in the M8 example. Note that the high negative weight of the *wrong order* constraints will cause that in every pair, at least one of the variables (that with less positive contribution from others) will end up selecting any other label (which in this case will be the `NULL` label). Weights for *right order* constraints are inversely proportional to $|d(v_i, v_j) - d(e_p, e_q)|$. The other constraints in the example use a constant value.

It is important to remark that a single constraint does not determine the alignment chosen for a particular event. All constraints affecting the assignment (v_i, e_j) are combined in S_{ij}. The re-normalization of the label weights for each variable ensures that there will always be one value selected: even if all values for a variable had a negative support, the weight for the one with less negative S_{ij} would be increased. In our case, we have the `NULL` value, which has neither penalization nor reward ($S_{ij} = 0$) causing its weight to be raised when all the other possible values have negative support.

Constraint examples

Right order	+8.3	$(v_2 : e1)$	$[(v_3 : e4)]$
	+12.5	$(v_2 : e1)$	$[(v_6 : e9)]$
	+25	$(v_6 : e9)$	$[(v_7 : e10)]$
Wrong Order	-500	$(v_1 : e11)$	$[(v_2 : e1)]$
	-500	$(v_4 : e5)$	$[(v_5 : e2)]$
	-500	$(v_7 : e10)$	$[(v_8 : e0)]$
Parallel	+5	$(v_3 : e4)$	$[(v_4 : e5)]$
Deletion	-200	$(v_5 : e2)$	$[(v_4 : e5), (v_6 : e9)]$

Fig. 4. Some example constraint pattern instantiations for the M8 alignment example.

The algorithm stops when convergence is reached –i.e. no more changes in the weight assignment–. Typical solutions consist of weight assignments of 1 for one label in each variable, and zero for the rest. However, if constraints are incomplete or contradictory, the final state may be a uniform distribution among a subset of values for some variables. Also, since the optimized cost function depends on the constraints, convergence is not theoretically guaranteed (since they may be incomplete or contradictory), although empirical results show that –if constraints are properly defined as it is the case of our formalization– the algorithm normally converges.

As described in Sect. 3.2 the complexity of RL is $n \times v \times c \times I$. In our particular trace alignment problem, v is a small constant (about 2 o 3 possible labels per variable). Since we generate constraints for every pair of trace events, the number of constraints per variable c is proportional to n. The number of required iterations I is in the order of a few dozens, though a safety stop is forced after 500 iterations. Thus, in our case, the complexity is $\mathcal{O}(n \times c \times K) = \mathcal{O}(n^2)$ (though it could be reduced to linear limiting the created constraints to only nearby neighbor events).

4.3 Stage 3: Generation of Approximate Alignment

The CLP solved via RL will produce a partial alignment, where some trace events will be assigned to some transitions in the unfolding, and some events will be assigned the `NULL` label (see Fig. 3). If the solution is consistent, it represents synchronous moves (events in the trace are mapped to a transition in the unfolding) and log moves (events in the trace are assigned to `NULL`). It may only lack model moves, i.e., necessary transitions in the unfolding to recover a full model run.

The approach used to add the needed model moves is to simulate the partial trace on the Petri net, until a mismatch is found (notice that this is a deterministic procedure, since unfolding transitions are unique). Assuming the RL solution alignments and deletions are correct, the mismatch can only be caused by a missing event in the trace. Thus, the shortest enabling path (previously computed) connecting the transition where the mismatch was detected and the transition corresponding to the next event in the trace is inserted at this point, and the simulation is continued. Note that this completion procedure is also able to re-insert events that were wrongly deleted by the RL algorithm. However, if the RL solution contains crucial errors (i.e. alignment of an event that should have been deleted), the resulting alignment may not be fitting.

To handle the insertions at the beginning or end of the trace, we add two *phantom* events, one at the beginning and one the end of the trace, respectively aligned to the initial and final states. In this way, the simulation will detect if there are missing events before the first trace element or after the last one.

Figure 5 shows an example of the results of the completion process, i.e., the technique computes the run $ACEGHFDB$, which is at edit distance 6 (counted as number of insertions and deletions) for the input trace $BCGHEFDA$.

Variables (trace events)	v_1 (B)		v_2 (C)		v_3 (G)	v_4 (H)	v_5 (E)	v_6 (F)	v_7 (D)	v_8 (A)	
Selected (or inserted) labels	**NULL**	*e0 (A)*	**e1**	*e2 (E)*	**e4**	**e5**	**NULL**	**e9**	**e10**	**NULL**	*e11 (B)*

Fig. 5. Complete alignment for example in Fig. 3, after adding necessary insertions to make the trace fitting. Boldface labels correspond to the alignment produced by RL. Transitions in italics are model moves inserted by the completion postprocess.

5 Experiments and Tool Support

To evaluate the performance of our approach, we resorted to datasets previously used in the state-of-the-art to test the performance of alignment techniques [12, 16, 17]. Some of these benchmarks are either very large, and/or contain loops and duplicate activities in the model. We also applied the tool to a real-world case: We used the Inductive Miner [10] (with default parameters) to extract a model for BPIC 2017 loan application data[6], and then we aligned it with the whole set of traces. Source code for our tool is available at https://github.com/lluisp/RL-align.

Since RL results largely depend on the constraint compatibility values, we used part of the data as a development set to tune the constraint weights, and we evaluated on the rest. We compared the solution of our approach with a reference solution: Optimal A* alignment by ProM for the models where it is available, backing off to an approximate method (ILPSDP, see [15]) when ProM failed to process the model trace file due to memory or time limitations. The used partition and some statistics about the models and traces can be found in

[6] https://data.4tu.nl/repository/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b.

Table 1. Statistics about dataset used in the experiments.

	Model	#places	Trace length			#traces	Reference alignment			Preprocess CPU time (s)	
			Avg	Max	Min		Avg. cost	Avg. fitness	Method	Paths	BPs
Tuning	M1	40	13.1	37	8	500	5.8	0.65	ProM	1	2
	M3	108	35.9	217	10	500	8.9	0.79	ProM	4	26
	M5	35	34.0	71	27	500	14.7	0.64	ProM	1	2
	M7	65	37.6	147	20	500	26.3	0.49	IPLSDP	1	5
	M9	47	44.3	216	16	500	21.3	0.61	ProM	1	6
	ML1	27	28.9	123	11	500	17.9	0.51	ProM	1	4
	ML3	45	26.4	194	8	500	22.9	0.35	ProM	1	3
	ML5	159	42.0	595	12	500	30.0	0.55	IPLSDP	12	53
	prAm6	347	31.6	41	19	1,200	4.1	0.90	ProM	133	829
	prCm6	317	42.8	59	15	500	29.3	0.51	IPLSDP	95	394
	prEm6	277	98.7	116	80	1,200	4.0	0.96	IPLSDP	64	153
	prGm6	357	143.0	159	124	1,200	26.3	0.83	IPLSDP	136	134
	TOTAL		*59.3*	*595*	*8*	*8,100*	*16.0*	*0.71*		*447*	*1,611*
Evaluation	M2	34	17.6	52	14	500	10.3	0.56	ProM	1	2
	M4	36	26.8	176	8	500	22.7	0.35	ProM	1	6
	M6	69	53.3	125	42	500	42.3	0.46	IPLSDP	1	5
	M8	17	16.5	109	8	500	7.3	0.65	ProM	1	1
	M10	150	58.2	240	30	500	42.7	0.47	IPLSDP	10	28
	ML2	165	87.4	582	27	500	80.9	0.30	IPLSDP	14	33
	ML4	36	28.1	89	17	500	25.6	0.34	ProM	1	2
	prBm6	317	41.5	59	14	1,200	0.0	1.00	ProM	96	388
	prDm6	529	248.4	271	235	1,200	3.6	0.99	IPLSDP	341	100
	prFm6	362	240.6	245	234	1,200	36.7	0.86	IPLSDP	107	34
	TOTAL		*109,9*	*582*	*8*	*7,100*	*24.0*	*0.70*		*570*	*599*
Total			**83.0**	**595**	**8**	**15,200**	**19.8**	**0.71**		**1,017**	**2,210**
Realistic	BPIC2017	280	38.1	180	10	31,509	38.2	0.10	IPLSDP	27	1,479

Table 1. Cost is computed as edit distance (number of log moves plus number of model moves). Fitness is computed as the ratio of sync moves over the length of the trace. The *average cost* and *average fitness* columns show the average cost/fitness per trace over the whole log. Last two columns show the CPU time required to precompute behavioural profiles and shortest enabling paths.

The tuning procedure consisted on a grid search of weights for each constraint type. Since *Loop* and *Parallel* use the same weight (the former as a constant, the latter in inverse proportion to the distance), we have 5 weights to set. We explored between 6 and 8 possible values for each –totalling over 16,000 combinations– and selected the weight combinations that maximized the desired measure over the tuning dataset.

Tables 2 and 3 show the results for the performed experiments. We report the percentage of cases where a fitting alignment was found, in how many of those

Table 2. Results obtained in scenario 1 (Maximize alignment F_1 score)

	Model	% fitting	% same cost	Obtained alignment				CPU time (sec)		
				Avg. cost	Δ with reference	Avg. fitness	Δ with reference	*RL*	ILPSDP	ProM
Tuning	M1	99.4	81.9	6.0	0.3	0.64	−0.01	1	23	4
	M3	90.8	75.8	8.9	1.3	0.78	−0.02	5	234	142
	M5	44.4	49.5	14.9	1.7	0.64	−0.01	6	59	587
	M7	45.2	25.2	16.8	−1.7	0.61	0.06	5	103	-
	M9	57.0	62.8	16.1	2.0	0.61	−0.02	10	123	51
	ML1	44.8	47.3	14.9	3.8	0.53	−0.03	7	67	18
	ML3	43.8	13.2	46.3	27.3	0.30	−0.08	7	89	61
	ML5	87.3	51.6	20.3	3.9	0.60	0.01	23	688	-
	prAm6	100.0	91.5	4.3	0.2	0.90	−0.003	5	822	58
	prCm6	89.8	21.6	27.0	−2.5	0.54	0.04	4	476	-
	prEm6	100.0	100.0	4.0	0.0	0.96	0.00	21	3, 145	-
	prGm6	0.0	-	-	-	-	-	114	7, 757	-
	TOTAL	*66.8*	*71.2*	*11.7*	*1.6*	*0.75*	*−0.001*	*208*	*13,586*	-
Test	M2	97.6	55.1	11.0	0.8	0.55	−0.004	1	30	20
	M4	54.8	22.3	31.4	14.6	0.34	−0.05	5	99	29
	M6	4.4	4.5	21.7	−7.0	0.68	0.11	8	165	-
	M8	62.6	70.3	6.5	1.9	0.68	−0.03	2	19	3
	M10	22.4	16.1	32.0	−1.6	0.59	0.08	11	411	-
	ML2	52.2	4.6	54.4	−9.0	0.61	0.26	61	1, 743	-
	ML4	28.0	6.4	30.8	11.0	0.33	−0.05	4	63	579
	prBm6	100.0	100.0	0.0	0.0	1.00	0.00	5	856	54
	prDm6	61.0	0.0	42.5	39.1	0.84	−0.15	177	34, 653	-
	prFm6	57.2	0.0	9.1	−27.8	0.96	0.10	159	20, 631	-
	TOTAL	*59.5*	*42.3*	*18.0*	*3.2*	*0.79*	*0.003*	*433*	*58,670*	-
Realistic	BPIC2017	99.9	0.4	43.8	5.6	0.15	0.05	2, 091	8, 702	-

the solution had the same cost than the reference approach (ProM or ILPSDP), the average cost and fitness of the alignments, and their differences with the cost and fitness achieved by the reference approach. In some cases the cost difference is negative (and/or the fitness difference is positive) showing that RL obtained better solutions than ILPSDP.

We also report the required CPU time to process the trace file for each model. Dashes in CPU time columns for ProM correspond to files were ProM run out of memory (using a 8 GB Java heap) or did not end after 8 h (wall clock time). Reported CPU times exclude time required to preprocess each model computing two behavioural profiles (original and reconnected unfolding) and shortest enabling paths for all event pairs (see Table 1). Note that the preprocessing is performed only once per model, so it is amortized in the long run when the number of aligned traces is large enough.

Scenario 1: Maximize Quality of Obtained Alignments. Our first scenario is selecting weights that get better alignments, even this may cause a lower percentage of cases with a fitting solution. In order to keep a balance between the quality of the alignments and the number of solved cases, we measure *precision* ($P = \#sync/(\#sync + \#log)$, maximized when there are no log moves) and *recall* ($R = \#sync/(\#sync + \#model)$, maximized when there are no model moves), and we aim at maximizing their harmonic mean, or F_1 score ($F_1 = 2PR/(P + R)$). The weight combination obtaining higher F_1 on tuning data is: *Right Order* = +15, *Wrong Order* = −100, *Exclusive* = −300, *Deletion* = −20, *Parallel*/*Loop* = +5.

Results of this configuration both on tuning and test data are shown in Table 2.

Scenario 2: Maximize Number of Aligned Traces. A second configuration choice consists of selecting the weights that maximize the number of fitting alignments, even if they have a higher cost. The weight combination obtaining a higher percentage of fitting alignments on tuning data is: *Right Order* = +5, *Wrong Order* = −500, *Exclusive* = −400, *Deletion* = −300, *Parallel*/*Loop* = +5.

Results of this configuration both on tuning and test data are shown in Table 3.

Discussion. Selecting constraint weights that maximize the percentage of fitting traces (scenario 2) results on large negative values for constraints penalizing unconsistent assignments (i.e. *Wrong Order*, *Exclusive*, and *Deletion*), which create a larger number of `NULL` assignments. Thus, the obtained alignments will contain more deletions (including wrong deletions of events that could have been aligned), creating gaps that will be filled by the completion post-process, solving more cases with a fitting alignment, though more likely to differ from the original trace, and thus with a higher cost.

On the other hand, when selecting weights that maximize F_1 score of the obtained solution (scenario 1), milder penalization values are selected. Thus, less events are deleted, causing less alignments to be fitting (a single wrongly aligned event can cause the whole trace to become non-fitting), but for those that are, the cost is closer to the reference (since the alignment does not discard trace events unless there is a strong evidence supporting that decision).

It is interesting to note that the proposed algorithm allows us to choose the desired trade-off between the percentage of fitting alignments and the quality of the obtained solutions. Moreover, it is also worth remarking that we tuned the weights for the dataset as a whole, but that they could be optimized per-model, obtaining configurations best suited for each model, if our use case required so.

Regarding computing time, the polynomial cost of the algorithm offers competitive execution times, making it suitable for real-time conformance checking, and feasible to explore configuration space to customize the weights to specific use cases, even on large models. Specifically, our computation times are about

Table 3. Results obtained in scenario 2 (Maximize number of aligned traces)

	Model	% fitting	% same cost	Obtained alignment				CPU time (sec)		
				Avg. cost	Δ with reference	Avg. fitness	Δ with reference	RL	ILPSDP	ProM
Tuning	M1	100.0	58.8	7.5	1.7	0.59	−0.06	1	23	4
	M3	89.4	62.9	9.9	2.2	0.76	−0.03	5	234	142
	M5	100.0	11.4	21.7	7.0	0.55	−0.10	3	59	587
	M7	99.8	11.8	30.7	4.6	0.49	−0.01	3	103	-
	M9	58.4	18.8	27.4	12.4	0.42	−0.21	5	123	51
	ML1	69.4	19.3	26.2	10.3	0.36	−0.16	3	67	18
	ML3	49.2	8.1	46.5	26.3	0.27	−0.10	2	89	61
	ML5	86.7	13.7	34.1	16.8	0.36	−0.22	24	688	-
	prAm6	100.0	77.1	5.5	1.4	0.88	−0.02	4	822	58
	prCm6	100.0	4.4	61.3	32.0	0.17	−0.33	3	476	-
	prEm6	100.0	100.0	4.0	0.0	0.96	0.00	40	3,145	-
	prGm6	98.9	4.5	35.3	9.1	0.78	−0.05	65	7,757	-
	TOTAL	*90.8*	*42.1*	*22.0*	*7.8*	*0.66*	*−0.05*	*158*	*13,586*	-
Test	M2	100.0	21.4	15.0	4.7	0.44	−0.12	1	30	20
	M4	60.6	11.9	35.0	16.2	0.30	−0.08	2	99	29
	M6	63.6	4.1	37.7	0.5	0.51	0.02	5	165	-
	M8	62.0	59.7	7.0	2.4	0.66	−0.05	1	19	3
	M10	73.2	4.1	57.3	18.0	0.35	−0.13	6	411	-
	ML2	85.4	4.0	65.8	−9.6	0.57	0.26	45	1,743	-
	ML4	54.6	0.4	44.1	20.1	0.14	−0.20	2	63	579
	prBm6	100.0	100.0	0.0	0.0	1.00	0.00	8	856	54
	prDm6	99.6	0.0	57.2	53.6	0.80	−0.19	258	34,653	-
	prFm6	100.0	5.2	35.0	−1.7	0.87	0.01	160	20,631	-
	TOTAL	*85.8*	*26.9*	*33.4*	*12.8*	*0.70*	*−0.05*	*488*	*58,670*	-
Realistic	BPIC2017	100.0	0.0	40.3	2.2	0.06	−0.04	1,576	8,702	-

two orders of magnitude smaller than those offered by ILPSDP and ProM, as presented in Tables 2 and 3.

Our tool also performs well on BPIC 2017 real-world data, achieving results comparable to other state-of-the-art methods, and solving them in a shorter time (although the speed-up is not as large in this case).

We must remark that ProM offers optimal solutions (when computational resources are enough), while relaxation labeling does not. Also, even ILPSDP is also suboptimal, it produces a fitting alignment for all cases, while RL may produce non-fitting solutions for some traces. However, we believe that our approach can be used as fast preprocess to obtain accurate enough suboptimal alignments, before resorting to more complex and computationally expensive approaches. RL solutions, either fitting or not, can also be useful as heuristic information to guide optimal search algorithms such as A*.

6 Conclusions and Future Work

We presented a flexible approach to align log traces with a process model. The used problem representation allows a trade-off between amount of solved cases and quality of the obtained solutions. The behaviour can be customized to particular use cases tuning the weights of the used constraints. Weights can be optimized for a whole dataset (as in presented scenarios 1 and 2), but better results can be obtained if they are optimized for each model, which may be useful for some use cases.

The algorithm requires one-time preprocessing to compute model unfolding, behavioural profile, and shortest enabling paths. Once this is done, any number of traces can be aligned in linear time, with a CPU time orders of magnitude smaller than other state-of-the-art methods. The obtained results show that the method is able to achieve competitive alignments with reasonable costs.

Further research lines include exploring higher-order constraints that allow the algorithm to use more fine-grained context information, and use the results as heuristic information to guide optimal search algorithms.

Acknowledgments. This work has been supported by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R.

References

1. Adriansyah, A.: Aligning observed and modeled behavior. Ph.D. thesis, Technische Universiteit Eindhoven (2014)
2. Armas-Cervantes, A., Baldan, P., Dumas, M., García-Bañuelos, L.: Behavioral comparison of process models based on canonically reduced event structures. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 267–282. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_17
3. Bloemen, V., van de Pol, J., van der Aalst, W.M.P.: Symbolically aligning observed and modelled behaviour. In: 18th International Conference on Application of Concurrency to System Design, ACSD, Bratislava, Slovakia, 25–29 June, pp. 50–59 (2018)
4. Bloemen, V., van Zelst, S.J., van der Aalst, W.M.P., van Dongen, B.F., van de Pol, J.: Maximizing synchronization for aligning observed and modelled behaviour. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 233–249. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_14
5. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99414-7
6. de Leoni, M., Marrella, A.: Aligning real process executions and prescriptive process models through automated planning. Expert Syst. Appl. **82**, 162–183 (2017)
7. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan's unfolding algorithm. Form. Methods Syst. Des. **20**(3), 285–310 (2002)
8. Khomenko, V., Koutny, M.: Towards an efficient algorithm for unfolding Petri nets. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 366–380. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44685-0_25
9. Khomenko, V., Koutny, M., Vogler, W.: Canonical prefixes of Petri net unfoldings. Acta Inf. **40**(2), 95–118 (2003)
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery and conformance checking. Softw. Syst. Model. **17**(2), 599–631 (2018)

11. Munoz-Gama, J., Carmona, J., Van Der Aalst, W.M.P.: Single-entry single-exit decomposed conformance checking. Inf. Syst. **46**, 102–122 (2014)
12. (Jorge) Munoz-Gama, J.: Conformance checking in the large (BPM 2013) (2013)
13. Reißner, D., Conforti, R., Dumas, M., La Rosa, M., Armas-Cervantes, A.: Scalable conformance checking of business processes. In: Panetto, H., et al. (eds.) OTM 2017. LNCS, vol. 10573, pp. 607–627. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69462-7_38
14. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
15. Taymouri, F.: Light methods for conformance checking of business processes. Ph.D. thesis, Universitat Politècnica de Catalunya (2018)
16. Taymouri, F., Carmona, J.: Model and event log reductions to boost the computation of alignments. In: Ceravolo, P., Guetl, C., Rinderle-Ma, S. (eds.) SIMPDA 2016. LNBIP, vol. 307, pp. 1–21. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74161-1_1
17. 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
18. Torras, C.: Relaxation and neural learning: points of convergence and divergence. J. Parallel Distrib. Comput. **6**, 217–244 (1989)
19. van der Aalst, W.M.P.: Decomposing Petri nets for process mining: a generic approach. Distrib. Parallel Databases **31**(4), 471–507 (2013)
20. Dongen, B.F.: Efficiently computing alignments. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 197–214. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_12
21. 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
22. Weidlich, M., Elliger, F., Weske, M.: Generalised computation of behavioural profiles based on Petri-net unfoldings. In: Bravetti, M., Bultan, T. (eds.) WS-FM 2010. LNCS, vol. 6551, pp. 101–115. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19589-1_7
23. Weidlich, M., Mendling, J., Weske, M.: Efficient consistency measurement based on behavioral profiles of process models. IEEE Trans. Softw. Eng. **37**(3), 410–429 (2011)