Merging Alignments for Decomposed Replay

H.M.W. Verbeek and W.M.P. van der Aalst

Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, The Netherlands
{h.m.w.verbeek,w.m.p.v.d.aalst}@tue.nl

Abstract. In the area of process mining, conformance checking aims to find an optimal alignment between an event log (which captures the activities that actually have happened) and a Petri net (which describes expected or normative behavior). Optimal alignments highlight discrepancies between observed and modeled behavior. To find an optimal alignment, a potentially challenging optimization problem needs to be solved based on a predefined cost function for misalignments. Unfortunately, this may be very time consuming for larger logs and models and often intractable. A solution is to decompose the problem of finding an optimal alignment in many smaller problems that are easier to solve. Decomposition can be used to detect conformance problems in less time and provides a lower bound for the costs of an optimal alignment. Although the existing approach is able to decide whether a trace fits or not, it does not provide an overall alignment. In this paper, we provide an algorithm that is able provide such an optimal alignment from the decomposed alignments if this is possible. Otherwise, the algorithm produces a so-called pseudo-alignment that can still be used to pinpoint non-conforming parts of log and model. The approach has been implemented in ProM and tested on various real-life event logs.

1 Introduction

The ultimate goal of *process mining* [2] is to gain process-related insights based on an *event log* created by some system. Such an event log contains a sequence of *events* for every case that was handled by the system. As an example, an event could be as follows:

> On October 1st, 2011, the resource 112 has completed the activity a_1.

A sequence of events contained in an event log is commonly referred to as a *trace*. From the data associated with the trace, we can derive for which particular case the activity a_1 was completed.

Within process mining, the field of *process conformance* [2, 4, 5, 7–11, 13, 15, 17] deals with checking to what extent a process model (like a Petri net) and an event log conform to each other, that is, how well they match. For this sake, the event log is first replayed on the Petri net *as best as possible*, which results in an *optimal alignment* between both. Such an optimal alignment relates events in the event log to activities (transition labels) in the Petri net in the best-possible way. Based on this optimal alignment, conclusions can then be drawn on important metrics like *fitness* (how well does

the event log conform to the Petri net?), *precision* (how well does the Petri net conform to the event log?), and *generalization* (how well does the Petri net conform to the system?).

A cutting-edge algorithm to compute an alignment is the *cost-based replayer* [3], which finds a *cost-minimal* alignment between the event log and the Petri net. Although this algorithm is very efficient and effective for smaller logs and smaller nets, it has problems when dealing with larger logs and larger nets. To speed up problematic replays, a *decomposition technique* has been proposed in [1]. This decomposition technique decomposes an overall log and an overall Petri net into a collection of decomposed logs and a collection of matching decomposed Petri nets, and guarantees that the decomposed costs (the costs of the replaying the decomposed logs on the decomposed nets) equal 0 if and only if the non-decomposed costs (the costs of replaying the overall log on the overall net) equal 0. *Hence, the approach is able to accurately identify deviating and non-deviating traces, often in a fraction of the time.* Furthermore, this technique guarantees that the decomposed costs are a lower bound for the non-decomposed costs.

An open issue for the decomposition approach is that it does not prescribe how to *merge the decomposed alignments* into an overall alignment, if possible. Whereas the replay of the overall log on the overall net results in an overall alignment, replaying the decomposed logs on the decomposed nets results in a collection of decomposed alignments. By merging these decomposed alignments into an overall alignment, which may be much faster than computing the non-decomposed alignment, one can also obtain diagnostic information on where the event log and the Petri net do not match. This paper introduces an algorithm to merge decomposed alignments into an overall alignment, again if possible. If not possible, then the algorithm will result in a so-called *pseudo-alignment*, which is a relaxation of the regular alignment. Such pseudo-alignments provide valuable diagnostic information and help to diagnose the misalignments.

The core of this algorithm consist of 3 alignment rules and 2 pseudo-alignment rules. Only applying the alignment rules will result in an alignment, but might not be feasible as in certain situations no alignment rule can be applied anymore. In such situations, we can apply a pseudo-alignment rule, but then the result will be a pseudo-alignment.

The approach has been implemented in ProM and has been applied to a large collection of logs and models showing that the constructed (pseudo-)alignments indeed help to diagnose conformance problems.

The remainder of this paper is organized follows. First, Section 2 provides the preliminaries, that is, logs, nets, alignments, and the decomposition. Second, Section 3 introduces our alignment merge, that is, the 3 alignment rules and the 2 pseudo-alignment rules. Third, Section 4 introduces the implementation of the merge. Fourth, Section 5 discusses the benefits of using the algorithm. Fifth, Section 6 concludes the paper.

2 Preliminaries

2.1 Logs

In this paper, we consider *activity logs*, which are an abstraction of the *event logs* as found in practice. An *activity log* is a collection of traces, where every trace is a se-

Table 1. An example activity log L_1 in tabular form.

Trace	Frequency
$\langle a_1, a_2, a_4, a_5, a_6, a_2, a_4, a_5, a_6, a_4, a_2, a_5, a_7 \rangle$	1
$\langle a_1, a_2, a_4, a_5, a_6, a_3, a_4, a_5, a_6, a_4, a_3, a_5, a_6, a_2, a_4, a_5, a_7 \rangle$	1
$\langle a_1, a_2, a_4, a_5, a_6, a_3, a_4, a_5, a_7 \rangle$	1
$\langle a_1, a_2, a_4, a_5, a_6, a_3, a_4, a_5, a_8 \rangle$	2
$\langle a_1, a_2, a_4, a_5, a_6, a_4, a_3, a_5, a_7 \rangle$	1
$\langle a_1, a_2, a_4, a_5, a_8 \rangle$	4
$\langle a_1, a_3, a_4, a_5, a_6, a_4, a_3, a_5, a_7 \rangle$	1
$\langle a_1, a_3, a_4, a_5, a_6, a_4, a_3, a_5, a_8 \rangle$	1
$\langle a_1, a_3, a_4, a_5, a_8 \rangle$	1
$\langle a_1, a_4, a_2, a_5, a_6, a_4, a_2, a_5, a_6, a_3, a_4, a_5, a_6, a_2, a_4, a_5, a_8 \rangle$	1
$\langle a_1, a_4, a_2, a_5, a_7 \rangle$	3
$\langle a_1, a_4, a_2, a_5, a_8 \rangle$	1
$\langle a_1, a_4, a_3, a_5, a_7 \rangle$	1
$\langle a_1, a_4, a_3, a_5, a_8 \rangle$	1

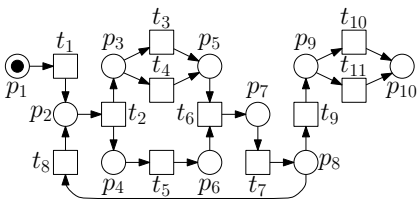

Fig. 1. A Petri net..

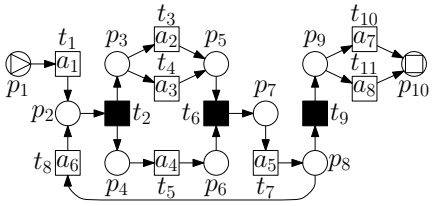

Fig. 2. An accepting Petri net N_1.

quence of *activities* [2]. Table 1 shows the example activity log L_1, which contains information about 20 cases, for example, 4 cases followed the trace $\langle a_1, a_2, a_4, a_5, a_8 \rangle$. In total, the log contains $13{+}17{+}9{+}2{\times}9{+}9{+}4{\times}5{+}9{+}9{+}5{+}5{+}17{+}3{\times}5{+}5{+}5 = 156$ activities.

Definition 1 (Universe of activities). *The set $\mathcal{A}$ denotes the universe of activities.*

To capture an activity log, we use multi-sets. If S is a set of objects, then $\mathcal{B}(S)$ is a multi-set of objects, that is, if $B \in \mathcal{B}(S)$ and $o \in S$, then object o occurs $B(o)$ times in multi-set B.

Definition 2 (Activity log). *Let $A \subseteq \mathcal{A}$ be a set of activities. An activity log L over A is a multi-set of activity traces over A, that is, $L \in \mathcal{B}(A^*)$.*

2.2 Nets

A Petri net is a modelling formalism that contains three different types of elements: *places*, *transitions*, and *arcs* [16]. Figure 1 shows an example Petri net containing 10 places (p_1 through p_{10}), 11 transitions (t_1 through t_{11}), and 24 arcs.

a_1	τ	a_2	a_3	a_4	τ	a_5	a_6	τ	a_7	a_8
t_1	t_2	t_3	$\gg$	t_5	t_6	t_7	$\gg$	t_9	t_{10}	$\gg$
0	0	0	10	0	0	0	10	0	0	10

Fig. 3. A trace alignment extended with costs for the trace $\langle a_1, \ldots, a_8 \rangle$ and net N_1. Every column corresponds to a move, where the top row contains the activity, the middle row the transitions, and the bottom row the costs of this activity-transition pair.

Definition 3 (Petri net). *A Petri net is a 3-tuple (P, T, F) where P is a set of places, T is a set of transitions such that $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs.*

For our purposes, we extend Petri nets with labels, an initial marking, and a set of final markings, yielding an *accepting* Petri net. Figure 2 shows an accepting Petri net based on the example Petri net, with labels (like a_1 and a_8), an initial marking ($[p_1]$), and one final marking ($[p_{10}]$).

The labels are used to link transitions in the Petri net to activities in an activity log. As an example, transition t_1 is linked to activity a_1. Transitions that are linked to log activities are called *visible* transitions. Transitions that are not linked to a log activity, like transition t_2, are called *invisible* transitions. These invisible transitions are linked to a dummy activity named τ. Note that $\tau \notin \mathcal{A}$.

Definition 4 (Accepting Petri net). *Let $A \subseteq \mathcal{A}$ be a set of activities. An accepting Petri net over the set of activities A is a 6-tuple (P, T, F, l, I, O) where (P, T, F) is a Petri net, $l \in T \to (A \cup \{\tau\})$ is a labeling function that links every transition onto an activity (possibly the dummy activity τ), $I \in \mathcal{B}(P)$ is an initial marking, and $O \subseteq \mathcal{B}(P)$ is a set of final markings.*

As a result of the labeling, we can obtain an *activity sequence* from a transition sequence by removing all invisible transitions while replacing every visible transition with its label. For example, the example transition sequence $\langle t_1, t_2, t_3 \rangle$ yields $\langle a_1, a_2 \rangle$ as activity sequence.

The initial marking and final markings are included because process mining considers complete traces and not a prefix-closed language. When replaying an activity log on a Petri net, the Petri net needs to have an initial marking to start with, and final markings to conclude whether the replay has reached a proper final marking. In the example, a replay of some trace starts from marking $[p_1]$, and the replay will be successful if and only if the marking $[p_{10}]$ is reached.

2.3 Alignments

A *trace alignment* [2, 3] links activities in a trace onto transitions in a Petri net. As an example, Fig. 3 shows a possible trace alignment for the trace $\langle a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8 \rangle$ and the accepting Petri net N_1. We use $\gg$ to denote the *lack* of a visible transition in an alignment. For example, the occurrence of activity a_3 was not matched by a firing

of transition t_4 in the net. In such a case, we use $\gg$ to indicate that no corresponding transition was fired. Likewise, although not present in this example, it is possible that transition t_4 was fired but that this was not matched by an occurrence of activity a_3 in the log. In such a case, we also use $\gg$ but now to indicate that no corresponding activity occurred.

The occurrence of a_1 matches the firing of transition t_1 (this is a so-called *synchronous move*, as both the log and the net can advance), then we need to fire the transition t_2 in the net which has no counterpart in the log (a so-called *invisible model move*), followed by another synchronous move for a_2 and t_3, after which we need to skip the activity a_3 (a so-called *log move*) as there is no transition enabled in the net that matches this activity, etc. Note that we require the transition sequence in the middle row of the alignment to lead from the initial marking of the net to some final marking.

Definition 5 (Legal moves). *Let $A \subseteq \mathcal{A}$ be a set of activities, let $\sigma \in A^*$ be an activity trace over A, and let $N = (P, T, F, l, I, O)$ be an accepting Petri net over A. The set of legal moves of A and N is the union of the sets $\{(a,t)|a \in A \wedge t \in T \wedge l(t) = a\}$ (synchronous moves), $\{(a, \gg)|a \in A\}$ (log moves), $\{(\gg, t)|t \in T \wedge l(t) \in A\}$ (visible model moves), and $\{(\tau, t)|t \in T \wedge l(t) = \tau\}$ (invisible model moves).*

Definition 6 (Trace alignment). *Let $A \subseteq \mathcal{A}$ be a set of activities, let $\sigma \in A^*$ be an activity trace over A, and let $N = (P, T, F, l, I, O)$ be an accepting Petri net over A. A trace alignment h for trace σ on net N is a sequence of legal moves $(a, t) \in ((A \cup \{\tau, \gg\}) \times (T \cup \{\gg\}))$ such that:*

- *$\sigma = h\restriction^1_A$ and*
- *For some $o \in O$ it holds that $I[h\restriction^2_T\rangle o$,*

where

$$h\restriction^1_A = \begin{cases} \langle\rangle & \text{if } h = \langle\rangle; \\ \langle a\rangle \cdot \overline{h}\restriction^1_A & \text{if } h = \langle(a,t)\rangle \cdot \overline{h} \text{ and } a \in A; \\ \overline{h}\restriction^1_A & \text{if } h = \langle(a,t)\rangle \cdot \overline{h} \text{ and } a \notin A; \end{cases}$$

and

$$h\restriction^2_T = \begin{cases} \langle\rangle & \text{if } h = \langle\rangle; \\ \langle t\rangle \cdot \overline{h}\restriction^2_T & \text{if } h = \langle(a,t)\rangle \cdot \overline{h} \text{ and } t \in T; \\ \overline{h}\restriction^2_T & \text{if } h = \langle(a,t)\rangle \cdot \overline{h} \text{ and } t \notin T; \end{cases}$$

The bottom row in Fig. 3 shows the possible *costs* of every move. In this example, a synchronous move costs 0, a visible model move costs 10, an invisible model move costs 0, and a log move also costs 10. The total costs for the example alignment is 30.

Definition 7 (Costs structure). *Let $A \subseteq \mathcal{A}$ be a set of activities, and let $N = (P, T, F, l, I, O)$ be an accepting Petri net over A. A cost structure $\$$ for A and N is a function that maps every legal move of A and N onto a (non-negative) natural number.*

Typically, the costs of all synchronous moves and all invisible model moves are set to 0, as then a perfect match has costs 0. The user then only needs to set the costs for the log moves and the visible model moves. If both these costs would be set to 10 for all transitions and activities, then we would have that $\$(\gg, t_3) = 10$, $\$(a_2, \gg) = 10$, and $\$(a_2, t_3) = 0$.

a_1	τ	a_2	a_3	a_4	τ	a_5	a_6	τ	a_7	a_8
t_1	t_2	$\gg$	t_4	t_5	t_6	t_7	$\gg$	t_9	$\gg$	t_{11}
0	0	10	0	0	0	0	10	0	10	0

Fig. 4. Another optimal trace alignment for the trace $\langle a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8 \rangle$ and Petri net N_1 (now t_4 and t_{11} fire rather than t_3 and t_{10}).

Definition 8 (Costs of trace alignment). *Let $A \subseteq \mathcal{A}$ be a set of activities, let $\sigma \in A^*$ be an activity trace over A, let $N = (P, T, F, l, I, O)$ be an accepting Petri net over A, let $h = \langle (a_1, t_1), \ldots, (a_n, t_n) \rangle$ be a trace alignment (of length n) for σ and N, and let $\$$ be a cost structure for A and N. The costs of trace alignment h, denoted $\$h$, is defined as the sum of the costs of all legal moves in the alignment, that is, $\$h = \sum_{i \in \{1,\ldots,n\}} \$(a_i, t_i)$.*

If no other alignment results in lower costs, the alignment is called *optimal*. There may exist multiple optimal alignments for a single trace. For example, the alignment as shown in Fig. 3 is optimal, but the alignment as shown in Fig. 4 is also optimal.

Definition 9 (Optimal trace alignment). *Let $A \subseteq \mathcal{A}$ be a set of activities, let $\sigma \in A^*$ be an activity trace over A, let $N = (P, T, F, l, I, O)$ be an accepting Petri net over A, let h be a trace alignment for σ and N, and let $\$$ be a cost structure for A and N. The trace alignment h is called optimal if there exists no other trace alignment h' such that $\$h' < \$h$.*

A *log alignment* is an optimal trace alignment for every trace in the activity log. As a result of a log alignment, any trace in the log can be mapped to the transition sequence that best matches this trace. As an example, an optimal trace alignment for the trace $\langle a_1, a_2, a_4, a_5, a_6, a_2, a_4, a_5, a_6, a_4, a_2, a_5, a_7 \rangle$ from log L_1 could include the transition sequence $\langle t_1, t_2, t_3, t_5, t_6, t_7, t_8, t_2, t_3, t_5, t_6, t_7, t_8, t_5, t_2, t_3, t_6, t_7, t_9, t_{10} \rangle$ in net N_1, and the costs of this alignment would be 0.

Definition 10 ((Optimal) log alignment). *Let $A \subseteq \mathcal{A}$ be a set of activities, let $L \in \mathcal{B}(A^*)$ be an activity log over A, and let $N = (P, T, F, l, I, O)$ be an accepting Petri net over A. A log alignment H for log L and net N is a function that maps every possible trace $\sigma \in L$ to a trace alignment. A log alignment is called optimal if and only if all its trace alignments are optimal.*

Clearly, log L_1 can be perfectly aligned to net N_1, as there exists an alignment where all trace alignments have costs 0. Using such a log alignment, it is possible project the date and information that is present in a log onto the net, and obtain average durations between activities, an animation with the token replay, etc.

2.4 Decomposition

The overall net and the overall log can be decomposed in a number of decomposed nets and decomposed logs, in such a way that (1) the costs of the optimal overall alignment

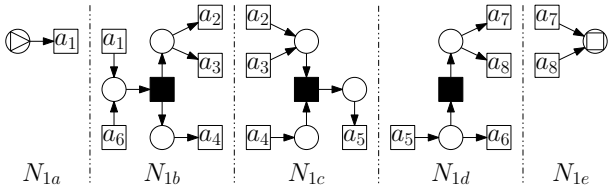

Fig. 5. Decomposed nets obtained by decomposing the net N_1. Nets N_{1b}, N_{1c}, N_{1d} , and N_{1e} have the empty marking as initial marking, while the nets N_{1a}, N_{1b}, N_{1c} , and N_{1d} have the empty marking as the only final marking.

Table 2. Decomposed log for activities $\{a_1, a_2, a_3, a_4, a_6\}$ in tabular form. This is the log that would be replayed on the decomposed net N_{1b} as shown in Fig. 5.

Trace	**Frequency**
$\langle a_1, a_2, a_4, a_6, a_2, a_4, a_6, a_4, a_2 \rangle$	1
$\langle a_1, a_2, a_4, a_6, a_3, a_4, a_6, a_4, a_3, a_6, a_2, a_4 \rangle$	1
$\langle a_1, a_2, a_4, a_6, a_3, a_4 \rangle$	3
$\langle a_1, a_2, a_4, a_6, a_4, a_3 \rangle$	1
$\langle a_1, a_2, a_4 \rangle$	4
$\langle a_1, a_3, a_4, a_6, a_4, a_3 \rangle$	2
$\langle a_1, a_3, a_4 \rangle$	1
$\langle a_1, a_4, a_2, a_6, a_4, a_2, a_6, a_3, a_4, a_6, a_2, a_4 \rangle$	1
$\langle a_1, a_4, a_2 \rangle$	4
$\langle a_1, a_4, a_3 \rangle$	2

is 0 if and only if the costs of every optimal decomposed alignment is 0, and (2) the accumulated costs of the decomposed alignments are a lower bound for the costs of the overall alignment [1]. This allows us to decompose an overall alignment problem into a number of decomposed alignment problems, which can possibly be solved much faster, while still providing certain guarantees.

Figure 5 shows the five decomposed nets that result from decomposing the net N_1. For an in-depth description of such decompositions, we refer to [1]. For this paper, it is sufficient to know that (1) every visible transition occurs in one or more decomposed nets, (2) for every label all different transitions sharing that label occur in a single decomposed net, and (3) that every place, invisible transition, and arc occurs in only a single decomposed net. As such, a token can only flow from one decomposed net to another decomposed net through firing some visible transition, which is hence observable in the log.

Based on the activity sets as present in these decomposed nets, five decomposed logs will be created. Every decomposed log contains as many traces as the overall log, but every trace in a decomposed log contains only those activities that are present in the corresponding net. As an example, Table 2 shows the decomposed log resulting from filtering log L_1 using the decomposed net N_{1b}, that is, the net that corresponds to the set of activities $\{a_1, a_2, a_3, a_4, a_6\}$.

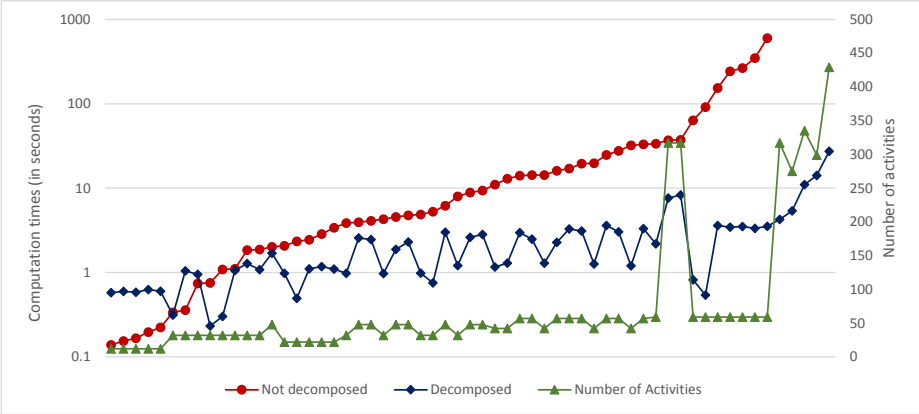

Fig. 6. Computation times and numbers of activities for the replay on some data sets. The non-decomposed replay did not finish in 10 minutes for some cases, typically the cases with many different activities.

By replaying every decomposed log on the corresponding decomposed net, we obtain optimal decomposed alignments, say h_1 to h_n. From [1] we know that the costs of these optimal decomposed alignments are guaranteed to be 0 if and only if the costs of an optimal overall alignment (of the overall log on the overall net) is 0. Furthermore, we know from [1] that we can use these decomposed alignments to obtain a lower bound for the costs of the overall alignment.

Figure 6 shows the usefulness of the decomposition approach. It shows the required computation times[1] and numbers of activities for the *DMKD 2006*, *BPM 2013*, and *IS 2014* data sets [12, 14, 15]. These data sets contain in total 59 cases of varying size, ranging from 12 to 335 activities, from 500 to 2000 traces, with varying numbers of mismatching traces (from 0% to 50%). Obviously, the decomposed replay is more robust when it comes to computation times, while it provides the same guarantee as mentioned earlier. For larger events logs and models, computing optimal alignments can take days and is often intractable. For example, for some of the cases that did not finish in 10 minutes it is known that they also do not finish within 12 hours [15], or that they do not finish because they run out of memory. If the net has a suitable decomposition, replay time may thus be reduced from more than 12 hours or infeasible to 10 seconds.

As a result of this replay, we will have a collection of decomposed alignments and a lower bound for the costs. To be able to diagnose the mismatches between the log and the net, we want to be able to merge the decomposed alignments into an overall alignment, and project that overall alignment onto the log or the net. Note that as a result of the guarantees of the decomposition, a trace has no costs in the overall alignment if and only if it has costs in the decomposed alignments.

[1] All tests are performed on a desktop computer with an Intel Core-i7-4770 CPU at 3.40 GHz, 16 GB of RAM, running Windows 7 Enterprise (64-bit), and using a 64-bit version of Java 7 where 4 GB of RAM was allocated to the Java VM.

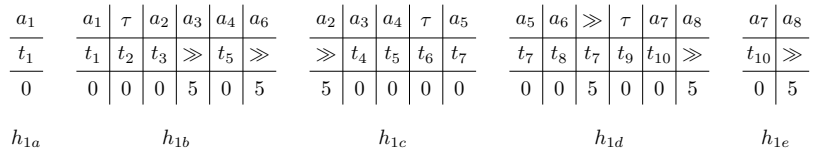

Fig. 7. Possible optimal decomposed alignments. h_{1b} is an optimal alignment for the decomposed log shown in Table 2 and the decomposed net N_{1b}, etc.

3 Merging Alignments

This section introduces an algorithm to merge a coherent set of decomposed alignments, that is, a set of alignments that result from replaying decomposed logs on corresponding decomposed nets.

To explain issues at hand for this step, we assume that we need to replay the trace $\langle a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8 \rangle$ on the net N_1 (see Fig. 2). Figure 3 shows an optimal overall alignment for this trace, which shows that the optimal costs for replaying this trace are 30. Figure 7 shows a set of possible optimal decomposed alignments, obtained by replaying the decomposed logs on the decomposed nets (see Fig. 5). Accumulating the costs from these decomposed alignments yields costs 30, which is in accordance with the costs of the optimal overall alignment mentioned earlier.

The algorithm needs to merge these five decomposed alignments into one overall alignment, if possible. To do so, the algorithm takes the trace, an empty alignment, and the five decomposed alignments, and works its way through the trace and the decomposed alignment while building up the overall alignment:

- The algorithm first encounters the activity a_1 in the trace, which is covered by two decomposed alignments (h_{1a} and h_{1b}). Fortunately, both decomposed alignments agree on a synchronous move on a_1 and transition t_1, so the algorithm adds the legal move (a_1, t_1) to the overall alignment and advances both the trace and the decomposed alignments.
- The algorithm then encounters the activity a_2, which is also covered by two decomposed fragments (h_{1b} and h_{1c}). However, h_{1b} is not yet ready to accept a_2 as it first needs to do an invisible model move on the transition t_2. Therefore, the algorithm first adds the invisible model move $(\gg, t_2)$ to the overall alignment and advances the state of h_{1b}. Then, unfortunately, the algorithm notices that h_{1b} and h_{1c} disagree on the move on a_2, as h_{1b} suggests a synchronous move (on transition t_3) while h_{1c} suggests a log move. In case of such a conflict, to ensure that the costs of the resulting alignment is lower than the accumulated costs of the decomposed alignment, the algorithm takes the cheapest option of the two, which in this case is the synchronous move. So, the algorithm adds the synchronous move (a_2, t_3) to the overall alignment and advance the state of the trace and both h_{1b} and h_{1c}.
- Activity a_3 is handled by the algorithm in a similar way as a_2, as h_{1b} and h_{1c} again disagree. Note that as a result, the algorithm now has added two synchronous moves for a_2 and a_3 to the overall alignment, which leads to a transition sequence

that is not executable in the overall net, and to an overall alignment which is (by definition) not a proper alignment. For this reason, we introduce so-called pseudo-alignments, which are alignments except for the fact that the trace does not need to be executable in the net. The result of merging the decomposed alignments at hand would then be a pseudo-alignment instead of an alignment.

– Etc.

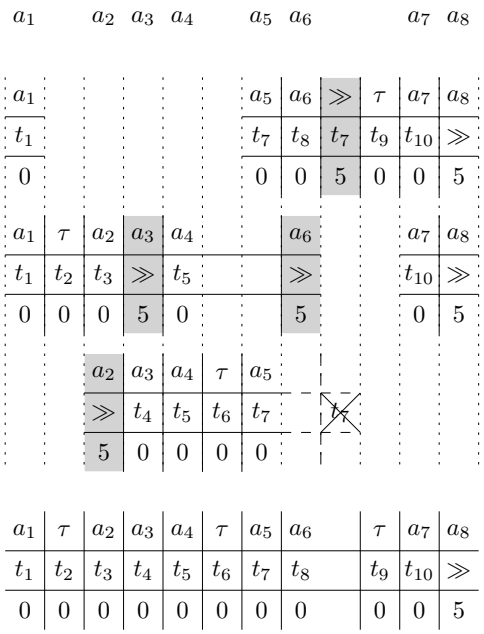

Fig. 8. Alignment of the decomposed alignments for obtaining the overall pseudo-alignment. The top row shows the trace that we need to align in the net. The middle row shows the optimal decomposed alignments of Fig. 7 laid out in such a way that the activities match in the vertical direction. Note that the ordering in the vertical direction of these decomposed alignments is of no importance. The bottom row shows the pseudo-alignment that results from merging these decomposed alignments.

This result of this merging is visualized in Fig. 8. In this figure, the middle row shows the *alignment of alignments* that results from merging the decomposed alignments. In this row, the legal moves that have been ignored (because they were in conflict and were more expensive, like t_3, or because they lacked a required counterpart, like t_7) are indicated with a grey background. The overall pseudo-alignment (see the bottom row in Fig. 8) follows directly from this alignment of decomposed alignments by taking in every column a legal move that has no grey background.

To avoid the introduction of the same concepts in the formal definitions over and over again, we first define a base setting for these formal definitions. This definition is to be used as a preamble for all other definitions in this section.

Definition 11 (Base setting). *The base setting for this section is as follows:*

– $A \subseteq \mathcal{A}$ *denotes a set of activities,*
– L *denotes an activity log over* A*,*
– σ *denotes an activity trace from* L*,*

- $N = (P, T, F, l, I, O)$ *denotes an accepting Petri net over* A*,*
- $\$$ *denotes a cost structure for* A *and* N*,*
- h *denotes an optimal trace alignment of* σ *for* A *and* N*, given* $\$$*,*
- n *equals the number of decomposed nets obtained by decomposing* N*,*
- *for every* $i \in \{1, \ldots, n\}$*:*
  - A^i *denotes the set of activities of the* i*-th decomposed net,*
  - L^i *denotes the log obtained from* L *by filtering out all activities not in* A^i*,*
  - σ^i *denotes the activity trace from* L^i *obtained from* L *by filtering out all activities not in* A^i*,*
  - $N^i = (P^i, T^i, F^i, l^i, I^i, O^i)$ *denotes the* i*-th decomposed net obtained by decomposing* N *as explained in [1], where* $P^i \subseteq P$*,* $T^i \subseteq T$*,* $F^i \subseteq F$*,* $\forall_{t \in T^i} l^i(t) = l(t)$*,* $I^i = I \cap P^i$*, and* $O^i = \{o \cap P^i | o \in O\}$*,*
  - h^i *denotes an optimal trace alignment of* σ^i *for* L^i *and* N^i*.*

First, we introduce the space of alignments that we are interested in, that is, the collection of overall alignments that fit the decomposed alignments. Later on, we introduce the algorithm that returns an alignment from this space.

Definition 12 (Merged trace alignment). *The trace alignment* h' *is a* merged trace alignment *of* $h^1, \ldots, h^n$ *if and only if* $\forall_{i \in \{1,\ldots,n\}} h'{\downarrow_{A^i}} = h^i$*, where* $h'{\downarrow_{A^i}}$ *is defined as follows:*

$$h'{\downarrow_{A^i}} = \begin{cases} \langle\rangle & \text{if } h' = \langle\rangle; \\ \langle (a,t) \rangle \cdot \overline{h'}{\downarrow_{A^i}} & \text{if } h' = \langle (a,t) \rangle \cdot \overline{h'} \text{ and } a \in A^i \vee t \in T^i; \\ \overline{h'}{\downarrow_{A^i}} & \text{if } h' = \langle (a,t) \rangle \cdot \overline{h'} \text{ and } a \notin A^i \wedge t \notin T^i; \end{cases}$$

For decomposed alignments that conflict on some legal move no such merged trace alignment exists.

Theorem 1. *(A merged trace alignment may not exist) For arbitrary decomposed alignments* $h^1, \ldots, h^n$*, a merged trace alignment* h' *may not exist.*

Proof. See Fig. 8: Either we have the legal move (a_2, t_3) or the legal move $(a_2, \gg)$. In the former case, the decomposed alignment that contains the legal move $(a_2, \gg)$ cannot be matched. In the latter case, the decomposed alignment that contains the legal move (a_2, t_3) cannot be matched.

In such cases, to get any result of merging the decomposed alignments, we need to let go of the requirement that for some $o \in O$ it holds that $I[\sigma\rangle o$, that is, the result is not an alignment, but a *pseudo-alignment*.

Definition 13 (Trace pseudo-alignment). *A trace pseudo-alignment* h *for trace* σ *on net* N *is a sequence of legal moves* $(a, t) \in (A \cup \{\tau, \gg\}) \times (T \cup \{\gg\}))$ *such that* $\sigma = h{\restriction^1_A}$*, where* $h{\restriction^1_A}$ *is defined as in Definition 6.*

In a pseudo alignment we drop the second requirement in Definition 6. Using these pseudo-alignments, the algorithm can handle conflicts between the decomposed alignments. A merged pseudo alignment has only legal moves for:

– Each activity in the log trace. In case of conflicts, the cheapest legal move is selected for the overall pseudo-alignment.
– A model move that all agree on. Note that, by definition, for an invisible model move all agree, as there is only one decomposed net that contains this transition.

Definition 14 (Merged trace pseudo-alignment). *The trace pseudo-alignment h' is a* merged trace pseudo-alignment *of $h^1, \ldots, h^n$ if and only if $\forall_{i \in \{1,\ldots,n\}} h'{\downarrow_{A^i}} = h^i$, where $h'{\downarrow_{A^i}}$ is defined as follows:*

$$h'{\downarrow_{A^i}} = \begin{cases} \langle\rangle & \text{if } h' = \langle\rangle; \\ \langle (a,t') \rangle \cdot \overline{h'}{\downarrow_{A^i}} & \text{if } h' = \langle (a,t) \rangle \cdot \overline{h'} \text{ and } a \in A^i, \\ & \text{for some } t' \text{ such that } \$(a,t') \leq \$(a,t); \\ \langle (x,t) \rangle \cdot \overline{h'}{\downarrow_{A^i}} & \text{if } h' = \langle (x,t) \rangle \cdot \overline{h'},\ x \in \{\tau, \gg\}, \text{ and } t \in T^i; \\ \langle (\gg,t) \rangle \cdot h'{\downarrow_{A^i}} & \text{if } t \in T^i \text{ and } l(t) \in A^i; \\ \overline{h'}{\downarrow_{A^i}} & \text{if } h' = \langle (a,t) \rangle \cdot \overline{h'} \text{ and } a \notin A^i \text{ and } t \notin T^i; \end{cases}$$

As a result, in case of conflicting legal moves, the cheapest legal move is selected to be included in the overall pseudo-alignment (see the second item). Of course, this *selected* legal move will not match some *unselected* legal moves, which most likely *breaks* the execution of the transition sequence in the net. Furthermore, note that it is possible to *insert* at arbitrary points arbitrary visible model moves (see the fourth item). This is required for completeness as otherwise we cannot deal with a conflict for such transitions. For example, observe that the decomposed alignment h_{1d} includes the legal move $(\gg, t_7)$, which is not matched in the decomposed alignment h_{1c}. Having $(\gg, t_7)$ in the corresponding position in h' would work for h_{1d}, but not for h_{1c}. Likewise, not having it would work for h_{1c}, but not for h_{1d}. To solve this problem, we decide *not to have* $(\gg, t)$ in h' (to preserve the lower bound for costs) and to compensate for this using this fourth item.

Theorem 2. *(A merged trace pseudo-alignment always exists) For arbitrary decomposed alignments $h^1, \ldots, h^n$, a merged trace pseudo-alignment h' always exists.*

Proof. Assume that such an h' does not exist, and assume that we could find a prefix of h' such that the projection of this prefix onto every A^i does result in a prefix of h^i. Obviously, the empty prefix satisfies this requirement. Now, assume that we cannot extend this prefix anymore, that is, we cannot add a new legal move, while a final marking has not been reached yet. This can obviously not involve an invisible transition, as there can be no conflict on such transition. Likewise, this can also not involve a visible transition contained in only one decomposed alignment. However, this can also not involve a shared visible transition that is *enabled* by all sharing decomposed alignments, as then we could simply take a cheapest of the legal moves proposed by these alignments. As a result, this can only happen in case at least one one of the sharing alignments expects another shared visible transition first. Such a shared visible transition cannot correspond to a model move, as otherwise we could use the fourth item to continue, as explain above. As a result, there should be a conflict with synchronous moves. But this clearly contradicts the fact that every decomposed alignment adheres to the order of these transitions in the original trace. Therefore, it cannot be that we cannot extend the prefix, unless we have reached a final marking.

Theorem 3. *(A merged trace pseudo-alignment may not be an alignment) Let the trace pseudo-alignment h' be a merged trace pseudo-alignment of $h^1, \ldots, h^n$. Then h' may not be a trace alignment for L and N.*

Proof. See Fig. 8: We added the legal move (a_2, t_3) as this is cheaper than adding $(a_2, \gg)$. In a similar fashion, we also added (a_3, t_4). Clearly, net N_1 does not allow for transition t_4 to be fired immediately after t_3 has fired. Hence the resulting transition sequence σ does not satisfy the requirement that $I[\sigma\rangle o$, for some $o \in O$.

Based on these definitions, we first introduce three alignment stitching rules, followed by two pseudo-alignment stitching rules. The alignment stitching rules construct a merged trace alignment, if possible. If this succeeds, we know that the result is again an alignment, and that the reported costs are exact, and not just a lower bound. Otherwise, we need a pseudo-alignment stitching rule to be able to continue, but this will result in a pseudo-alignment, and the reported costs will only be a lower bound.

Definition 15 (Stitching function Y). *Let $\mathcal{H}$ be the set of all possible trace pseudo-alignments of L and N, and let $\mathcal{H}^i$ be the set of all possible trace alignments of L^i and N^i. The function $Y \in (\mathcal{H} \times A^* \times \mathcal{H}^1 \times \ldots \times \mathcal{H}^n) \to \mathcal{H}$ returns the first argument concatenated by the merged trace pseudo-alignment of the third and following arguments $(h^1, \ldots, h^n)$, where the second argument (σ) is used to guide the stitching. As a result, $Y(\langle\rangle, \sigma, h^1, \ldots, h^n)$ returns the merged trace pseudo-alignment of $h^1, \ldots, h^n$.*

The first alignment stitching rule is a simple rule that detects when the algorithm is done: If the trace and all decomposed alignments have been dealt with completely.

Alignment Stitching Rule 1 (All done)

$$Y(h', \langle\rangle, \langle\rangle, \ldots, \langle\rangle) = h'$$

The second alignment stitching rule is a rule that allows the algorithm to continue if all relevant decomposed alignments agree on the first activity in the trace. If so, this activity is now dealt with and so are the corresponding legal moves in the relevant decomposed alignments. For the irrelevant decomposed alignments, nothing changes.

Alignment Stitching Rule 2 (Activity w/o conflict)

$$\begin{array}{l} \textit{If } \sigma = \langle a \rangle \cdot \overline{\sigma} \textit{ and } t \in (T \cup \{\gg\}) \\ \textit{and } \forall_{i \in \{1,\ldots,n\}} (a \in A^i) \Rightarrow (h^i = \langle (a,t) \rangle \cdot \overline{h^i}) \\ \textit{and } \forall_{i \in \{1,\ldots,n\}} (a \notin A^i) \Rightarrow (h^i = \overline{h^i}) \\ \textit{then } Y(h', \sigma, h^1, \ldots, h^n) = Y(h' \cdot \langle (a,t) \rangle, \overline{\sigma}, \overline{h^1}, \ldots, \overline{h^n}) \end{array}$$

The third alignment stitching rule is a rule that allows the algorithm to continue if all relevant decomposed alignments agree on a next model move. If so, these legal moves are now dealt with. Note that the set of relevant decomposed alignments differs per model move, as this set comprises all decomposed alignments where the corresponding decomposed net contains the transition involved in the selected legal move. As a result, multiple legal model moves could be a candidate for applying this rule. Note that this indicates that the candidate transitions are now all enabled, that is they are concurrent.

Alignment Stitching Rule 3 (Transition w/o conflict)

$$\begin{aligned}
&\textit{If } t \in T \textit{ and } x \in \{\tau, \gg\} \\
&\textit{and } \forall_{i \in \{1,\ldots,n\}} (t \in T^i) \Rightarrow (h^i = \langle (x,t) \rangle \cdot \overline{h^i}) \\
&\textit{and } \forall_{i \in \{1,\ldots,n\}} (t \notin T^i) \Rightarrow (h^i = \overline{h^i}) \\
&\textit{then } Y(h', \sigma, h^1, \ldots, h^n) = Y(h' \cdot \langle (\tau, t) \rangle, \sigma, \overline{h^1}, \ldots, \overline{h^n})
\end{aligned}$$

As mentioned before, applying these rules will result in an alignment if the algorithm ends and Rule 1 can be applied. However, it may be that no rule is applicable before reaching the end of one or more decomposed alignments.

Theorem 4 (Alignment stitching rules result in an alignment). *Provided that the application of the alignment stitching rules ends, that is, if at the end Rule 1 is applied, then the result of applying these rules is an alignment merged trace alignment, that is, a trace alignment.*

Proof. By construction, the alignment stitching rules append h' with a legal move (a, t) if and only if all relevant decomposed alignments h^i agree on this legal move. See also [1].

If conflicts between activities and/or transitions do occur, the algorithm can use one of the following rules to continue. However, by applying these rules, we know that the end result will not be an alignment. As we favor alignments, over pseudo-alignments, we only apply the following stitching rules if the previous stitching rules can not be applied.

The first pseudo-alignment stitching rule is a rule that allows the algorithm to continue if the relevant decomposed alignments disagree on the next legal move containing the first activity in the trace, that is, a synchronous or log move. If so, the cheapest of the conflicting legal moves is added to the resulting pseudo alignment, the activity in the trace is now dealt with, and so are all the conflicting moves in the relevant decomposed alignment.

Pseudo-alignment Stitching Rule 1 (Activity w/ conflict)

$$\begin{aligned}
&\textit{If } \sigma = \langle a \rangle \cdot \overline{\sigma} \textit{ and } R = \{i \in \{1,\ldots,n\} | a \in A^i\} \textit{ and } m \in R \\
&\textit{and } \forall_{i \in R} (t^i \in (T^i \cup \{\gg\}) \wedge h^i = \langle (a, t^i) \rangle \cdot \overline{h^i} \wedge \$(a, t^m) \leq \$(a, t^i)) \\
&\textit{and } \forall_{i \in \{1,\ldots,n\} \setminus R} h^i = \overline{h^i} \\
&\textit{then } Y(h', \sigma, h^1, \ldots, h^n) = Y(h' \cdot \langle (a, t^m) \rangle, \overline{\sigma}, \overline{h^1}, \ldots, \overline{h^n})
\end{aligned}$$

In this definition, R can be interpreted as the set of (indices of) *relevant* decomposed alignments, and m can be interpreted as the (index of the) relevant decomposed alignment with *minimal* costs.

The second pseudo-alignment stitching rule is a rule that allows the algorithm to continue if the relevant decomposed alignments disagree on a next model move. If so, the corresponding model moves are now dealt with. Note that in this case no model move is added to the resulting pseudo-alignment. In the resulting pseudo-alignment, these legal moves will not be represented. Again, this is done to ensure that the costs of the resulting pseudo-alignment does not exceed the accumulated costs of the decomposed alignments.

Pseudo-alignment Stitching Rule 2 (Transition w/ conflict)

$$\begin{array}{l} \textit{If } t \in T \textit{ and } l(t) \in A \\ \textit{and } R = \{i \in \{1,\ldots,n\} | h^i = \langle (\gg, t) \rangle \cdot \ldots\} \textit{ and } R \neq \emptyset \\ \textit{and } \forall_{i \in R} h^i = \langle (\gg, t) \rangle \cdot \overline{h^i} \\ \textit{and } \forall_{i \in \{1,\ldots,n\} \setminus R} h^i = \overline{h^i} \\ \textit{then } Y(h', \sigma, h^1, \ldots, h^n) = Y(h', \sigma, \overline{h^1}, \ldots, \overline{h^n}) \end{array}$$

In this definition, again R can be interpreted as the set of (indices of) *relevant* decomposed alignments.

Note that both rules are not deterministic, as there may be more activities or multiple visible transitions that satisfy the preamble. In such a case, one can select any of these activities or transitions, apply the rule using that activity or transition, and continue.

Theorem 5 (Stitching rules are complete). *Applying all five rules ends, and the result of applying them is a merged trace pseudo-alignment.*

Proof. By construction, Pseudo-alignment Stitching Rule 1 appends h' with a legal move (a, t) if and only if it is the cheapest option, which corresponds 1-to-1 with the second item of the merged trace pseudo-alignment. As a result, this rule deals effectively with conflicting synchronous moves and conflicting log moves (as t^i is allowed to be $\gg$). Also by construction, Pseudo-alignment Stitching Rule 2 allows to *ignore* a conflicting model move, as it may add such a model move to some h^i while not adding it to h'. This correspond 1-to-1 with the fourth item of the conflicting merged trace alignment. As a result, the rule deals effectively with conflicting model moves. What remains is the proof that Alignment Stitching Rule 1 can be applied at some point in time (algorithm ends). Assume that the stitching blocks at some point in time, while σ starts with a. If all corresponding h^i's start with the same legal move, then Alignment Stitching Rule 2 can be applied. If these h^i's start with conflicting legal moves, then Pseudo-alignment Stitching Rule 1 can be applied. As a result, to have no stitching rule applicable at this moment, some of these h^i's need to start with a model move, that is, with some (τ, t^i) or $(\gg, t^i)$. In case of (τ, t^i), Alignment Stitching Rule 3 can be applied, as t^i only occurs in T^i. Otherwise, in case of $(\gg, t^i)$, Pseudo-alignment Stitching Rule 2 can be applied. Hence, the stitching cannot block.

We now have everything in place for the merge algorithm, which simply keeps on checking whether the rules as provided earlier (in the order given) can be applied. As soon as it detects a rule that it can apply, it applies that rule and starts over again by checking whether the rules can be applied. In the end, the first rule is applied, which provides us with the result: either a pseudo-alignment or a proper alignment.

4 Implementation

The alignment merge as described in the previous section has been implemented in the "LogAlignment" package of ProM 6 [19]. This "LogAlignment" package[2] is part of the

[2] See `https://svn.win.tue.nl/repos/prom/Packages/LogAlignment/Trunk`

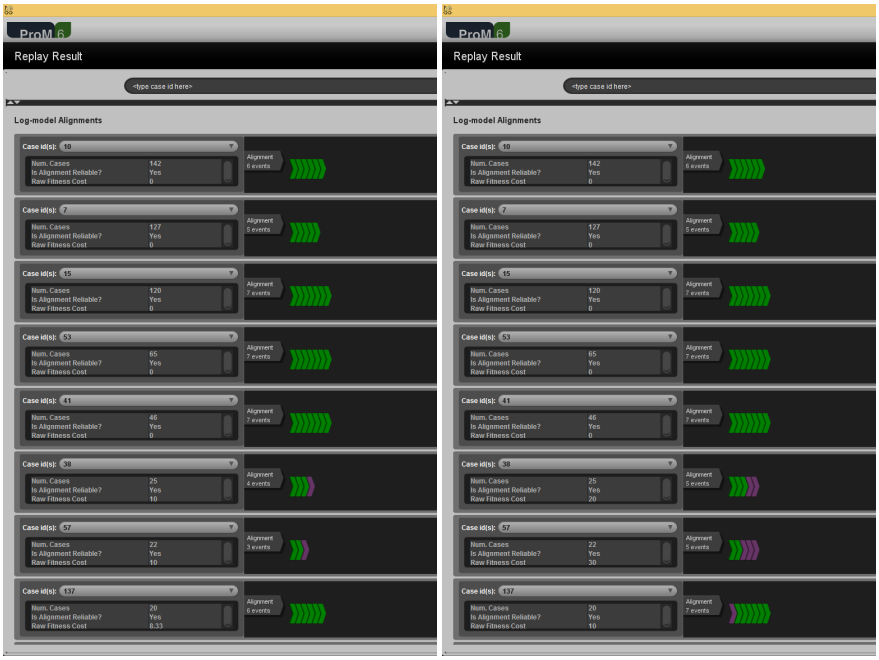

Fig. 9. Screenshot of the result of the decomposed replay (left-hand side) and of the result of the non-decomposed replay (right-hand side). Both alignments are projected onto the overall log.

"DivideAndConquer" framework [18] in ProM 6 that supports decomposed discovery and decomposed replay.

In the framework, a single replay (that is, on overall alignment of every trace in a log on a net) has a timeout of 10 minutes. In Fig. 6 we already noted that some non-decomposed replays did not finish within this time. This timeout of 10 minutes is set to prevent a single replay to take almost forever. Experiences indicated that only rarely a replay finishes successfully after more than these 10 minutes [14, 15]. In case of a decomposed replay, this timeout is enforced in a progressive way. If one of the decomposed replays has failed, then the resulting pseudo-alignment will be empty. For this reason, after one of the decomposed replays has exceeded the timeout, the timeout will be set to 0 minutes. This effectively prevents time being spent in decomposed replays that are not needed anyway as the resulting pseudo-alignment is already known to be empty.

Figure 9 shows two screenshots: one of the result of the decomposed replay and the alignment merge, and one of the result of the non-decomposed replay. The resulting (pseudo-)alignments have both been projected onto the overall log. This figure shows that the decomposed replay followed by the alignment merge successfully matches all synchronous moves (the lighter and green chevrons), and also some visible model moves (the darker and purple chevrons). Apparently, all decomposed replays agree on these visible model modes, but disagreed on some others.

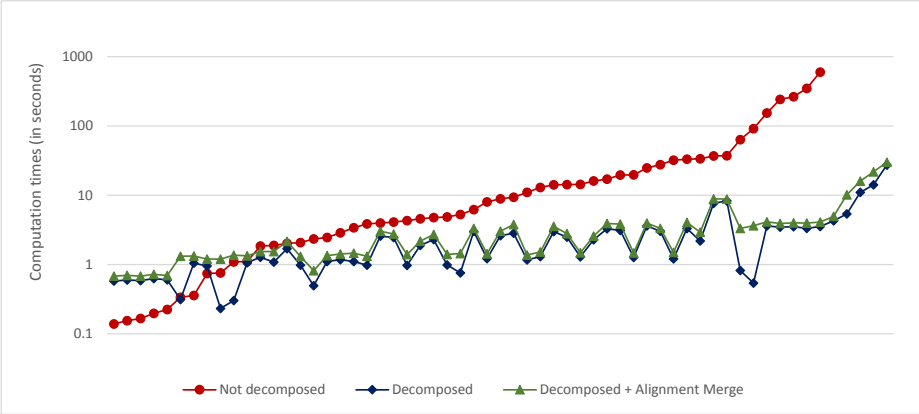

Fig. 10. Computation times for the replay and the alignment merge on the data sets.

5 Discussion

Figure 10 shows the computation time needed for the decomposed replay extended with the alignment merge for the three data sets introduced earlier. For sake of reference, this figure also includes the computation time needed for both the non-decomposed replay and the decomposed replay. Clearly, in many cases, doing a decomposed replay followed by an alignment merge is faster than doing a non-decomposed replay. As a result, a pseudo-alignment can be returned in a fraction of the time it would take to return an alignment. Note that, for every data set used, the computation of a pseudo-alignment would take up to 20 seconds, whereas the computation of an alignment might take more than 10 minutes, that is, days [14, 15].

If the decomposed alignments do no incur any costs, then there exists an alignment that is the result of the merge [1]. For this alignment, it is guaranteed that there are no conflicts when merging. As a result, in such cases, the algorithm as introduced in this paper is guaranteed to return an alignment instead of only a pseudo-alignment.

If the decomposed alignments do incur costs, then it is still possible that an optimal alignment is returned by the stitching algorithm: If at every step during the stitching all relevant decomposed alignments agree on a next legal move, then the result will be an alignment. Note that this alignment may still incur costs, as every relevant decomposed alignment may agree on the move to be a log move, or in case all misalignments are local. As a result, in certain cases if costs are incurred, the algorithm will return an alignment, and this alignment will have the same costs as the non-decomposed alignment [1].

Figure 11 shows that in general it is not possible to obtain an alignment in every case. The dashed line in this net indicates the only way this net can be decomposed into nets: The first decomposed net (N_a) contains all transitions labeled a and the second net (N_d) contains all transitions labeled d. Now assume that the trace at hand is the empty trace and that every model move costs 10. The optimal alignment for N_a contains only the legal move $(\gg, t_c)$ (with costs 5), whereas the optimal alignment for N_d contains

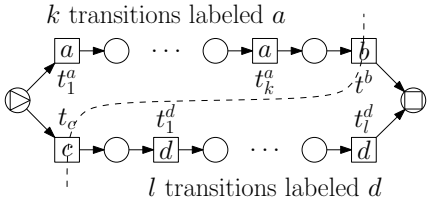

Fig. 11. An example net for which there is no proper alignment.

only $(\gg, t_b)$ (also with costs 5). Clearly, there is no alignment in the overall net that has lower costs than the costs (10) of these two model moves: The upper branch would cost $10k + 10$, the lower branch $10 + 10l$. This example also shows that we cannot give an upper bound for an overall alignment using the decomposed alignment, as k and l could be arbitrary high.

At the moment, the cost-based replay algorithm results in on overall optimal alignment. As a result, it is conceivable that for two decomposed replays two optimal decomposed alignments are returned which have conflicts, while there would also be two optimal decomposed alignments that do not conflict. In case of the conflicting decomposed alignments, the algorithm returns a pseudo-alignment, whereas it would return an alignment if the alignments do not conflict. As such, it would be beneficial for the merge algorithm if the replay algorithm would return *all* optimal alignments instead of only *one*. If so, then the merge algorithm could return an alignment in more cases, although of course this would take more time as the algorithm would have to check possibly many combinations of decomposed alignments.

6 Conclusions

This paper has introduced an algorithm to merge decomposed alignments. In general, the result of the algorithm will be a pseudo-alignment, that is, an alignment except for the fact that it may not be executable in the net. However, if the decomposed alignments agree with each other, then the algorithm returns a proper optimal alignment. In case of a perfect match between the log and the net, then these decomposed alignments will always agree with each other [1], and hence the result of the algorithm will be an alignment. In case of mismatches between the log and the net, the result many still be an alignment, but then all decomposed alignments need to agree.

This paper has also shown that, especially for logs and nets that contain many activities, the computation of a (non-decomposed) alignment takes longer than the computations of the decomposed alignments and the time it takes to merge them. As the pseudo-alignment that results from the decomposed approach can still be used effectively to diagnose mismatches, it will often be more convenient to use this approach.

In case the algorithm returns a proper alignment, then the costs of this alignment are exact, that is, it equals the costs obtained through a non-decomposed replay. In case the algorithm returns not an alignment but a pseudo-alignment, then the accumulated costs of the legal moves in the pseudo-alignment are a lower bound for the accumulated costs

of the decomposed alignments, which are in turn a lower bound for the costs obtained through a non-decomposed replay.

In the future, we want to see whether heuristics exist that help the merge into obtaining *better* (pseudo-)alignments. As is, the current algorithm contains non-determinism, in which case it has several alternatives to proceed. For example, there could be multiple visible model moves to merge next. Possibly, using some local heuristic, we can select a next merge action which *most likely* results in a (pseudo-)alignment that better matches the overall alignment one would get by doing a non-decomposed replay. Note that the better this match, the better the gain of using the non-decomposed replay.

Also, we want to extend the stitching rules with rules that allow for different ways of decomposing the net. The decomposition introduced in [1] takes care that every invisible transition ends up in a single decomposed net. As a result, there can be no conflict that involves invisible transitions. However, different decomposition approaches may not guarantee this. As an example take a decomposition approach where a decomposed net is obtained by simply hiding (that is, making invisible) those visible transitions that are not relevant for this decomposed net. For such a decomposition approach, every transition occurs in every decomposed net, only some visible transitions will be made invisible. Of course, if some visible transition has been made invisible in some decomposed net, then there will definitely be a conflict when merging the alignment. This gets even more complicated if the decomposition approach first reduces the decomposed nets, for example, by using existing Petri net reduction rules [6, 16]. Then some invisible transitions will be present in only some decomposed nets, and some of them might also have visible counterparts in other decomposed nets.

Finally, an interesting idea is to recompose decomposed nets and decomposed logs in case their decomposed alignments have many conflicts. By recomposing the nets and the logs, we remove these conflicts but obtain a slightly larger recomposed net and recomposed log. As this recomposed net and recomposed log might still be much smaller than the overall net and overall log, the decomposed replay might still be much faster than the overall replay.

References

1. Aalst, W.M.P.v.d.: Decomposing Petri nets for process mining: A generic approach. Distributed and Parallel Databases 31(4), 471–507 (2013), http://dx.doi.org/10.1007/s10619-013-7127-5
2. Aalst, W.M.P.v.d.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. 1st edn. (2011)
3. Aalst, W.M.P.v.d., Adriansyah, A., Dongen, B.F.v.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery 2(2), 182–192 (2012), http://dx.doi.org/10.1002/widm.1045
4. Adriansyah, A., Dongen, B.F.v., Aalst, W.M.P.v.d.: Conformance checking using cost-based fitness analysis. In: Proceedings of the 2011 IEEE 15th International Enterprise Distributed Object Computing Conference. pp. 55–64. EDOC '11, IEEE Computer Society, Washington, DC, USA (2011), http://dx.doi.org/10.1109/EDOC.2011.12

5. Adriansyah, A., Sidorova, N., Dongen, B.F.v.: Cost-based fitness in conformance checking. In: Application of Concurrency to System Design (ACSD), 2011 11th International Conference on. pp. 57–66 (June 2011)
6. Berthelot, G.: Transformations and decompositions of nets. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties. vol. 254, pp. 360–376 (1987)
7. vanden Broucke, S.K.L.M., De Weerdt, J., Vanthienen, J., Baesens, B.: Determining process model precision and generalization with weighted artificial negative events. Knowledge and Data Engineering, IEEE Transactions on 26(8), 1877–1889 (Aug 2014)
8. Calders, T., Günther, C.W., Pechenizkiy, M., Rozinat, A.: Using minimum description length for process mining. In: Proceedings of the 2009 ACM Symposium on Applied Computing. pp. 1451–1455. SAC '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1529282.1529606
9. Cook, J.E., Wolf, A.L.: Software process validation: Quantitatively measuring the correspondence of a process to a model. ACM Trans. Softw. Eng. Methodol. 8(2), 147–176 (Apr 1999), http://doi.acm.org/10.1145/304399.304401
10. De Weerdt, J., De Backer, M., Vanthienen, J., Baesens, B.: A robust f-measure for evaluating discovered process models. In: Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on. pp. 148–155 (April 2011)
11. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. J. Mach. Learn. Res. 10, 1305–1340 (Jun 2009), http://dl.acm.org/citation.cfm?id=1577069.1577113
12. Maruster, L., Weijters, A.J.M.M., Aalst, W.M.P.v.d., Van Den Bosch, A.: A rule-based approach for process discovery: Dealing with noise and imbalance in process logs. Data Mining and Knowledge Discovery 13(1), 67–87 (2006), http://dx.doi.org/10.1007/s10618-005-0029-z
13. Munoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. In: Hull, R., Mendling, J., Tai, S. (eds.) Business Process Management, Lecture Notes in Computer Science, vol. 6336, pp. 211–226. Springer Berlin Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-15618-2_16
14. Munoz-Gama, J., Carmona, J., Aalst, W.M.P.v.d.: Conformance checking in the large: Partitioning and topology. In: BPM. Lecture Notes in Computer Science, vol. 8094, pp. 130–145. Springer (2013)
15. Munoz-Gama, J., Carmona, J., Aalst, W.M.P.v.d.: Single-entry single-exit decomposed conformance checking. Inf. Syst. 46, 102–122 (2014), http://dx.doi.org/10.1016/j.is.2014.04.003
16. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (April 1989)
17. Rozinat, A., Aalst, W.M.P.v.d.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64 – 95 (2008), http://www.sciencedirect.com/science/article/pii/S030643790700049X
18. Verbeek, H.M.W.: Decomposed process mining with DivideAndConquer. In: BPM 2014 Demos, vol. 1295, pp. 86–90. CEUR-WS.org (2014), http://ceur-ws.org/Vol-1295/paper11.pdf
19. Verbeek, H.M.W., Buijs, J.C.A.M., Dongen, B.F.v., Aalst, W.M.P.v.d.: ProM 6: The process mining toolkit. In: Proc. of BPM Demonstration Track 2010. vol. 615, pp. 34–39. CEUR-WS.org (2010), http://ceur-ws.org/Vol-615/paper13.pdf