Decomposed Replay Using Hiding and Reduction

H.M.W. Verbeek

Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, The Netherlands
h.m.w.verbeek@tue.nl

Abstract. In the area of process mining, decomposed replay has been proposed to be able to deal with nets and logs containing many different activities. The main assumption behind this decomposition is that replaying many subnets and sublogs containing only some activities is faster then replaying a single net and log containing many activities. Although for many nets and logs this assumption does hold, there are also nets and logs for which it does not hold. This paper shows an example net and log for which the decomposed replay may take way more time, and provides an explanation why this is the case. Next, to mitigate this problem, this paper proposes an alternative decomposed replay, and shows that this alternative decomposed replay is faster than the monolithic replay even for the problematic cases as identified earlier. However, the alternative decomposed replay is often slower than the original decomposed approach. An advantage of the alternative decomposed approach over the original approach is that its cost estimates are typically better.

1 Introduction

The area of *process mining* [1] is typically divided into *process discovery*, *process conformance*, and *process enhancement*. In the context of *big data*, a decomposition approach [2] has been introduced that includes both process discovery (decomposed discovery) and process conformance (decomposed replay). In this paper, we take this decomposed replay for process conformance as a starting point. The main assumptions for this decomposed replay are that (1) checking conformance using a monolithic replay (that is, by replaying a single net and log, which contain many different activities) takes prohibitively much time, and that (2) checking conformance using a decomposed replay (that is, by replaying a series of subnets and sublogs, which each contain far less different activities than the single net and log) takes far less time. The decomposition approach as introduced in [2] splits up a single Petri net in to a collection of Petri nets with shared transitions on their borders, and guarantees that (1) the result of the decomposed replay is perfect if and only if the result of the monolithic replay is perfect, and (2) the result of the decomposed replay provides a lower bound for the result of the monolithic replay otherwise.

Figure 1 supports this assumption by showing typical computation times for the nets and logs as found in a number of data sets [7–9]. These data sets contain

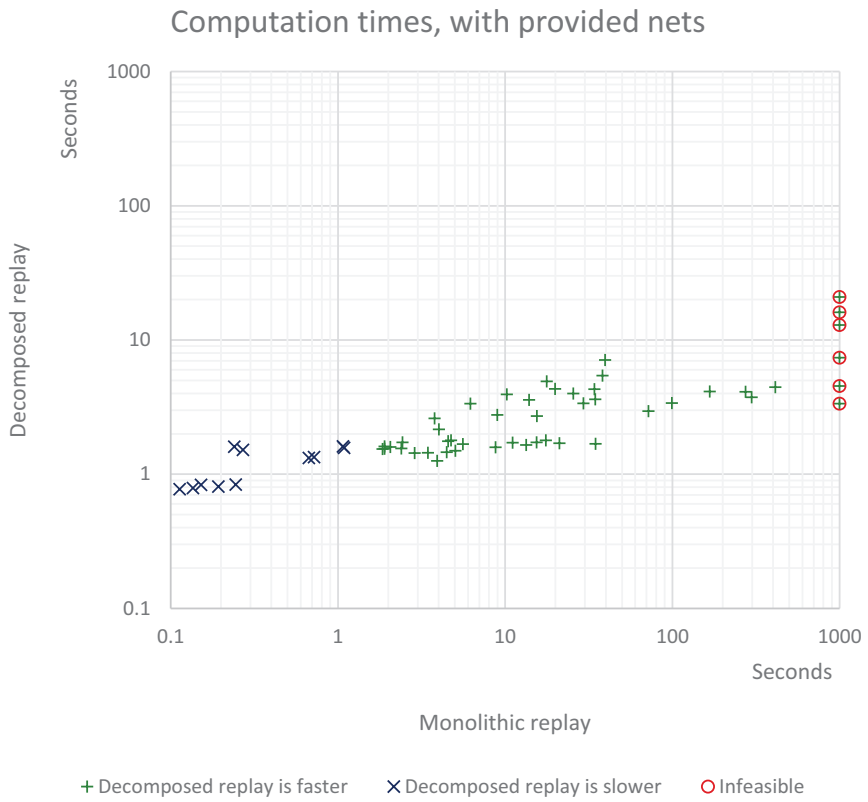

Fig. 1. Computation times for the monolithic and decomposed replay with the nets as provided by the data sets.

in total 59 cases of varying size, ranging from 12 to 429 activities, from 500 to 2000 traces, with varying numbers of mismatching traces (from 0% to 50%). This figure shows that if the monolithic replay would take more than a second, the decomposed replay would be faster. Furthermore, it shows that some replays do not finish within 10 minutes (see [9] why we use a timeout here of 10 minutes) using the monolithic replay, whereas all replays finish within 10 minutes using the decomposed replay.

However, Figure 2 paints a different picture. It shows the typical computation times for replaying the log as found in the data sets on the net *as discovered using the Inductive miner* [6] from that log. As a result, instead of checking the conformance of some log on some *designed* net, we now check the conformance of this log on some *discovered* net. As for this replay we only require a log (and not a net), we included some additional data sets [4, 5] for this figure. This figure shows that in some cases the decomposed replay misbehaves by requiring much more time than the monolithic replay. For example, the monolithic replay requires less

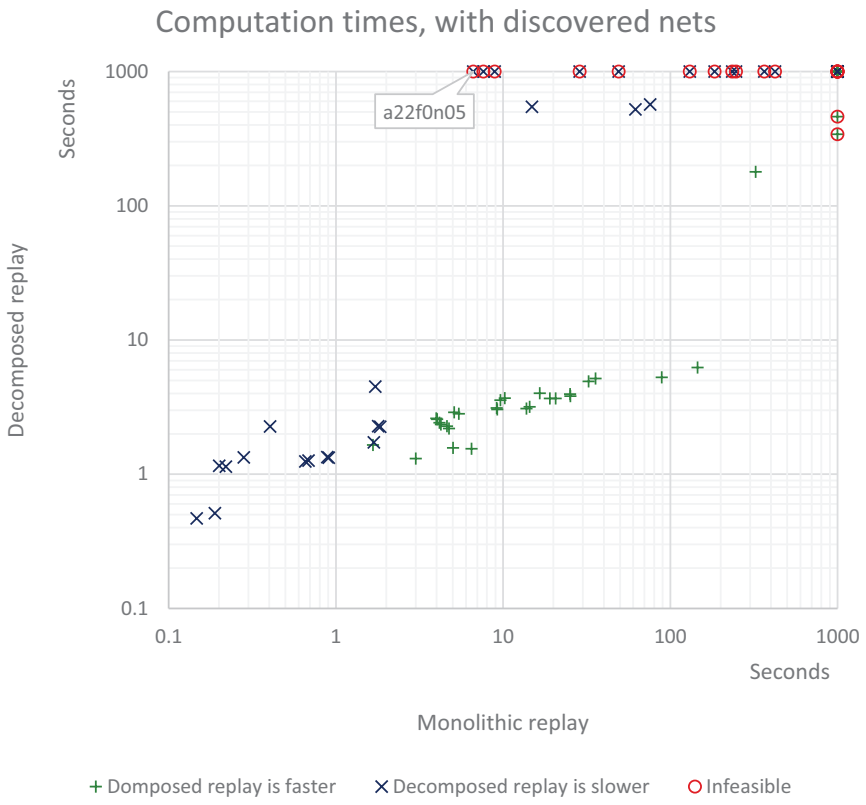

Fig. 2. Computation times for the monolithic and decomposed replay with the nets discovered using the *Inductive Miner* from the logs as provided by the data sets.

than 10 seconds for the *a22f0n05* case, where the decomposed replay takes more than 10 minutes.

In this paper, we investigate the root cause of this misbehavior. Based on these findings, we propose an *alternative decomposed replay* to mitigate the misbehavior problem. This alternative decomposition approach is based on the well known concepts of hiding transitions and reducing nets. We show that, in almost all cases that take more than 10 seconds for the monolithic replay, either using designed nets or discovered nets, the hide-and-reduce replay is indeed faster. However, we will also show that the (original) decomposed replay is often faster than the hide-and-reduce replay. But, whereas the decomposed replay does misbehave for some cases, the hide-and-reduce replay does not. Finally, we show that the hide-and-reduce replay has an extra advantage, as it provides a better estimate for the replay costs.

The remainder of this paper is organized as follows. First, Section 2 introduces the necessary concepts, like accepting Petri nets and alignments. Second,

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

Section 3 shows that there is a possible problem with the decomposed replay, and proposes the hide-and-reduce replay to mitigate this problem. Third, Section 4 evaluates the hide-and-reduce replay using the existing data sets, which shows that it can handle more cases than the monolithic and/or decomposed replay, and that it typically returns a better lower bound for the costs as the decomposed replay. Last, Section 5 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 sequence of *activities* [1]. 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

In this paper, we assume that a net is an accepting Petri net, that is, a labeled Petri net with an initial marking and a set of final markings. The transition labels are used to denote the activity a transition corresponds to. Transitions that do not correspond to an activity are labeled τ, and are henceforth called invisible. The other (activity-labeled) transitions are henceforth called visible transitions. The initial marking and final markings are needed for the replay, which needs to start at the initial marking and needs to end in a final marking.

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.*

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, $l \in T \rightarrow (A \cup \{\tau\})$ 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.*

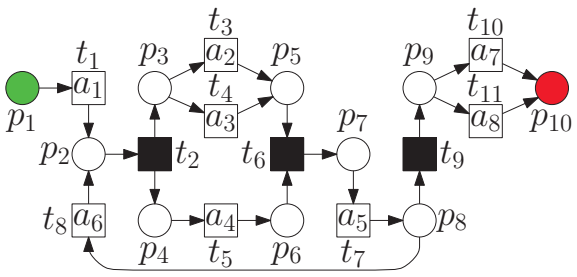

Fig. 3. An accepting Petri net N_1.

Figure 3 shows an accepting Petri net containing 10 places $\{p_1, \ldots, p_{10}\}$, 11 transitions $\{t_1, \ldots, t_{11}\}$ of which 8 labeled with activities $\{a_1, \ldots, a_8\}$ and 3 invisible, and 24 arcs. The initial marking consists of a single token in place p_1 (indicated by the light green color), and the only final marking consists of a single token in place p_{10} (indicated by the darker red color). Please note that, although this example does not show this, multiple transitions can be labeled by the same activity.

2.3 Alignments

Assume that on the example net N_1 we want to replay the trace $\langle a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8 \rangle$, that is, we want to find the *best* transition sequence that starts in the initial marking, ends in a final marking, and that has the given trace as *activity*

sequence. Basically, the activity sequence corresponds to the transitions sequence which every transitions replaced by its label and all τ labels removed afterwards. Finding the best transition sequence is done by first finding a transition sequence, and second to impose some notion of costs to every found transition sequence. By keeping these costs minimal while finding a transition sequence, we then obtain the best transition sequence.

To find a transition sequence, the replayer [3] creates a number of possible *moves*, and assigns *costs* to every move. A move can be a synchronous move, a log move, a visible model move, or an invisible model move. A synchronous move consists of a transitions and its label, that is, its corresponding activity. As an example, the move (t_3, a_2) is a synchronous move for net N_1, indicating that transition t_3 was fired which matched activity a_2 in the trace. A log move consists of a dummy transition (indicated by $\gg$) and an activity. As an example, the move $(\gg, a_2)$ is a log move for net N_1, indicating that activity a_2 could not be matched to any enabled transition. A visible model move consists of a visible transition and a dummy activity (also indicated by $\gg$). As an example, the move $(t_3, \gg)$ is a visible model move for net N_1, indicating that visible transition t_3 was fired but could not be matched to any activity in the trace. An invisible model move consists of an invisible transition and the τ label. As an example, the move (t_2, τ) is an invisible model move for net N_1, indicating that invisible transition t_2 was fired.

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).*

Log moves and visible model moves hint at mismatches, as these indicate that either some activity could not be matched by a proper enabled transition, or that the firing of the transition could not be matched to its activity in the trace. For this reason, the costs of these moves are typically set to positive values, whereas the costs for the other moves are set to 0. In this paper, we will use the costs 10 for every log move, and 4 for every visible model move.

Definition 6 (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.*

Using these moves and these costs, the question for the replayer is then to find a sequence of moves such that (1) the activities correspond to the given activity sequence, (2) the transitions correspond to a possible transition sequence in the net that starts in the initial marking and ends in the final marking, and (3) has minimal costs. This sequence of moves is henceforth called an *optimal trace alignment*.

Definition 7 (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

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

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, $\$h = \sum_{i \in \{1,\ldots,n\}} \$(a_i, t_i)$.*

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_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. 4. A trace alignment 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 activities, the middle row the transitions, and the bottom row the costs of every move.

Figure 4 shows an optimal trace alignment for the trace $\langle a_1, \ldots, a_8\rangle$ and net N_1. Please note that an optimal trace alignment may not be unique. In the example, we could also have chosen to do a synchronous move on a_3 and a log move on a_2. The resulting alignment would also be optimal.

2.4 Decomposed replay

For small number of activities (like 8 as in the example), computing an optimal trace alignment on the entire net may be possible within 10 minutes, but for larger numbers (say 100 or more), the replay will take considerable more time. To alleviate this, decomposed replay has been proposed in [2]. In decomposed replay, we first decompose the net into subnets, where the following restrictions are taken into account:

- Every place ends up in a single subnet.
- Every invisible transition ends up in a single subnet.
- Every arc ends up in a single subnet.
- If multiple visible transitions share the same label, then these transitions end up in a single subnet.

As a result, only visible transitions that have a unique label can be distributed over the subnets. Figure 5 shows how the example net N_1 can be decomposed

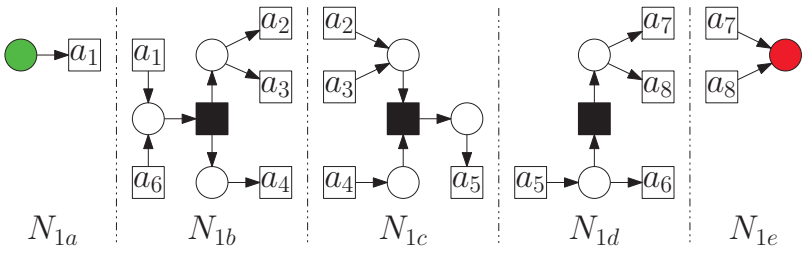

Fig. 5. The decomposed nets obtained by decomposing the net N_1. Subnets N_{1b}, N_{1c}, N_{1d}, and N_{1e} have the empty marking as initial marking, while the subnets N_{1a}, N_{1b}, N_{1c}, and N_{1d} have the empty marking as the only final marking.

into 5 subnets.

The costs associated to the various moves are also decomposed (see also [2]). For example, the activity a_1 appears in two subnets (N_{1a} and N_{1b}). As a result, a log move on a_1 in N_{1a} costs $10/2 = 5$ and a log move on a_1 in N_{1b} costs 5 as well. Note that, by coincidence, all activities appear in two subnets, therefore, all log moves in all subnets now cost 5. If some activity would have appeared in three subnets, then the costs for a log move for every subnet would be 10/3. As a result of this decomposition, if all subalignments in the decomposed replay agree on a log move for a_1, then the costs are identical to a log move in the monolithic replay. Likewise, a visible model move on transition t_1 in N_{1a} costs $4/2 = 2$ and a visible model move in N_{1b} costs 2 as well. In theory, the costs for the synchronous moves and invisible model moves are also decomposed in a similar way, but as both are typically 0, we typically ignore them.

Second, we decompose the trace into subtraces, and replay every subtrace on the corresponding subnet.

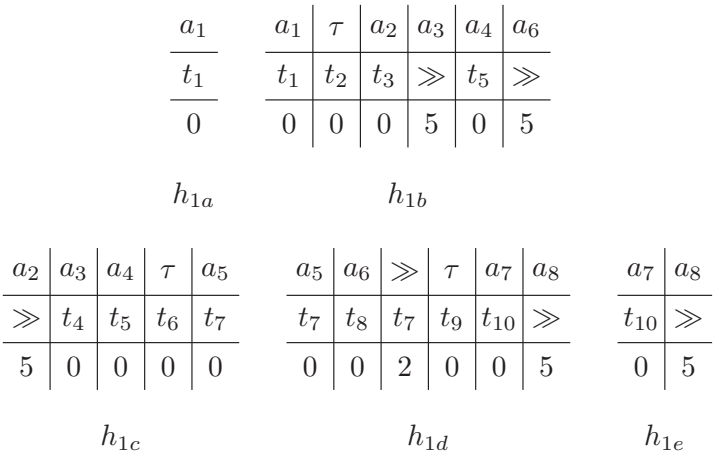

Fig. 6. Possible optimal decomposed alignments. h_{1b} is an optimal alignment for subtrace $\langle a_1, a_2, a_3, a_4, a_6 \rangle$ and the subnet N_{1b}, etc.

Third, we accumulate the costs of these subalignments, which are the costs as reported by the decomposed replay. Note that the optimal alignment h_{1d} includes a model move on transition t_7 instead of a log move on a_7. As the model move is cheaper than the log move, doing the log move would not be optimal. This shows that the costs as reported by the decomposed replay ($5 + 5 + 5 + 2 + 5 + 5 = 27$) can indeed be lower than the (correct) costs of the monolithic alignment ($10 + 10 + 10 = 30$).

As indicated in the Introduction, the decomposed replay is often much faster than the monolithic replay. Because of the formal guarantees as provided by [2], we know that the decomposed replay only returns costs 0 if and only if the monolithic replay returns costs 0, and that otherwise the decomposed replay returns less costs than the monolithic replay (that is, the decomposed replay provides a lower bound for the correct costs of the monolithic replay). As such, we can use the decomposed replay as a fast way to check whether there are any costs involved, and to obtain a lower bound for the correct costs.

3 Hide and Reduce

In the Introduction, we have shown that for some cases, the decomposed replay actually misbehaves. As an example, if we take the log from the *a22f0n05* case [7], discover a net for it using the *Inductive Miner*, and replay the log on the discovered net, then the monolithic replay requires less than 10 seconds, whereas the decomposed replay fails. In this section, we use the *a22f0n05* case to find the root cause of this failure. After having found that root cause, this section proposes an alternative decomposed replay to mitigate the root cause.

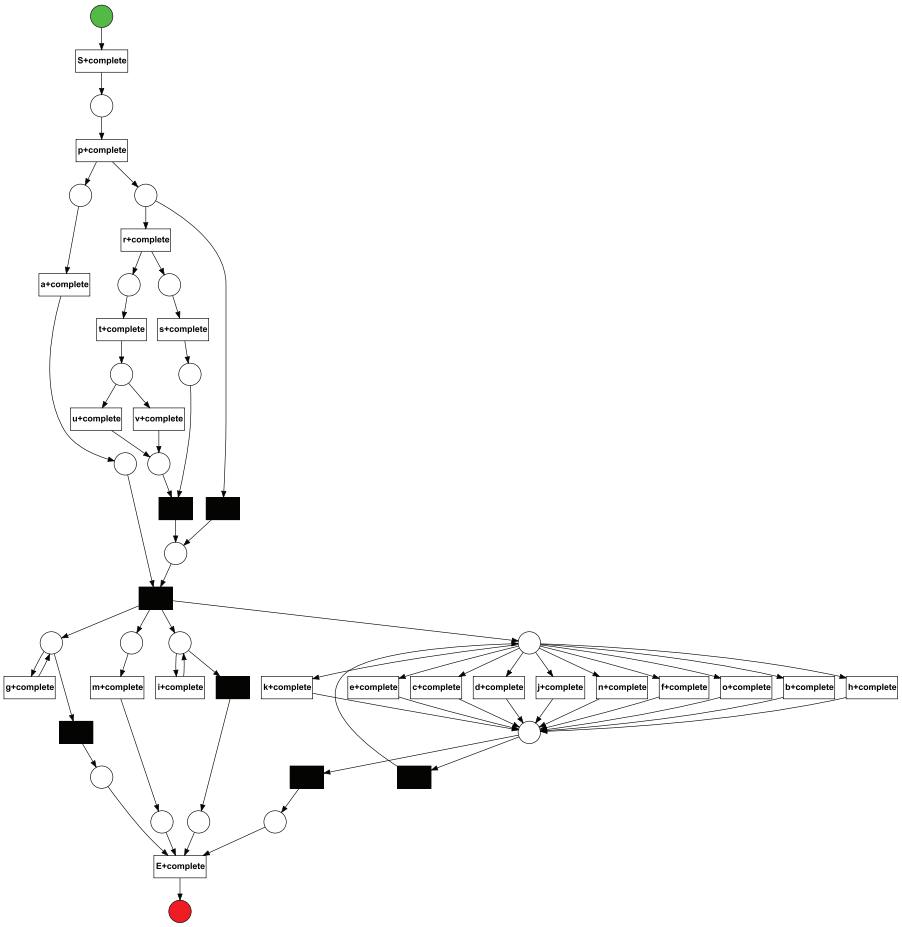

Fig. 7. The net as discovered by the *Inductive Miner* from the *a22f0n05* log.

Figure 7 shows the net as discovered by the *Inductive Miner* from the *a22f0n05* log. This net can be decomposed into 9 subnets. Two of these subnets contain a single activity (*S+complete*, *E+complete*), 6 contain two activities, and the last subnet contains 20 activities (only the activities *S+complete* and *t+complete* are not included in this subnet). The time to replay the first 8 sublogs on the subnets range from 9 to 65 milliseconds, but the replay on this last subnet turns out to require more than 10 minutes.

Figure 8 shows this problematic subnet, which immediately shows the root cause of the problem: This subnet contains five source transitions (transitions without incoming arcs), and a fair number (13) of places. Because the source transitions are always enabled, they can be fired at any possible reachable state. While checking for the optimal path through the search space of alignments, the replayer needs to investigate all enabled transitions in every state. For this

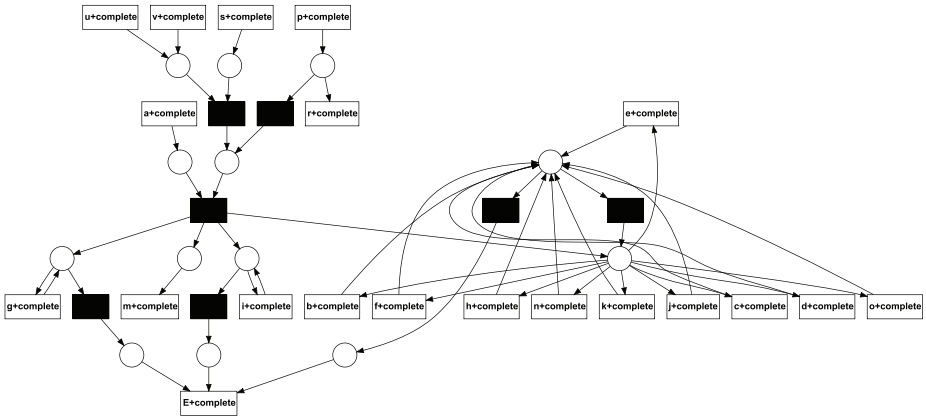

Fig. 8. Problematic subnet for *a22f0n05* case. The five source transitions (*a+complete*, *p+complete*, *s+complete*, *u+complete*, and *v+complete*) lead to a huge search space for the replay.

subnet, the fact that 5 transitions are enabled in every state is too much for the replayer.

Apparently, by removing parts of the net in the subnet, information that was vital for the replayer may have been lost. As an example, in the net, the transition labeled *s+complete* can only fire after the transition labeled *p+complete* has been fired, and both have to fire exactly once. In the subnet, both transitions can be fired any number of times, and in any order, which leads to the replay to require more than 10 minutes.

For this reason, this paper presents an alternative decomposed replay. Instead of removing places, transitions, and arcs that do not belong to a subnet, this decomposition simply keeps these places and arcs while hiding (making invisible) all visible transitions.

Definition 10 (Hidden net). *Let $A \subseteq \mathcal{A}$ be a set of activities, let $N = (P, T, F, l, I, O)$ be an accepting Petri net over A, and let $A_i \subseteq A$ be a subset of the set of activities. Then the hidden net for N and A_i, denoted $N(A_i)$, is the accepting Petri net $(P, T, F, l(A_i), I, O)$ such that*

$$l(A_i)(t) = \begin{cases} l(t) & \text{if } l(t) \in A_i; \\ \tau & \text{if } l(t) \notin A_i. \end{cases}$$

As a result, the structure of the otherwise-removed parts is maintained, and hence the inter-transition relations as described before are maintained.

Theorem 1 (Hiding preserves having costs 0). *Let $A \subseteq \mathcal{A}$ be a set of activities, let $N = (P, T, F, l, I, O)$ be an accepting Petri net over A, let $A_1, \ldots, A_n$ be the activities of the subnets $N_1, \ldots, N_n$ that result from decomposing net N (see [2] for details), and let $\sigma \in A^*$ be an activity trace over A. Then*

an optimal trace alignment h for σ and N has costs 0 if and only if for every $1 \leq i \leq n$ an optimal trace alignment h_i for σ and N_i has costs 0.

Proof. We prove this by comparing the monolithical (non-decomposed) costs, say $\$^m$, the decomposed costs, say $\$^d$, and the costs as obtained by hiding, say $\$^h$. If h has costs 0, then mutatis mutandis (some activities are replaced by τ's) h also has costs 0 for every hidden subnet $N(a_i)$. As a result, if $\$^m = 0$ then $\$^h = 0$. If h_i has costs 0, then mutatis mutandis (some columns are removed) h_i also has costs 0 for the corresponding subnet N_i. As a result, if $\$^h = 0$ then $\$^d = 0$. From [2] if follows that decomposition preserves perfect fitness, that is, it preserves having costs 0. As a result, $\$^m = 0$ if and only if $\$^d = 0$. Therefore, $\$^m = 0$ if and only if $\$^h = 0$.

This hiding step is then optionally followed by a reduction step using well-known Petri-net-reduction rules [10]. This makes the net as small as possible for the replayer. Figure 9 shows an overview of these reduction rules when applied to accepting Petri nets. From this figure, it is clear that these rules preserve the branching activity behavior of the net:

- The rules cannot remove any visible transition.
- The FPT rule can only remove an invisible transition if there exists another invisible transition.
- The FPP rule can only remove an unmarked place (that is, a place not involved in the initial or any final marking) if there exists another unmarked place.
- The ESP rule can only remove a marked place which has x tokens in the initial and every final marking, where $x > 0$.
- If a marked place is removed, the initial and final markings are updated accordingly.

Definition 11 (Reduced nets). *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. Then the reduced nets for N, denoted $R(N)$, is the collection of accepting Petri nets (P', T', F', l, I', O') that can result from applying the rules as shown in Figure 9 over and over again until no rule can be applied anymore.*

Theorem 2 (Reducing preserves costs 0). *Let $A \subseteq \mathcal{A}$ be a set of activities, let $N = (P, T, F, l, I, O)$ be an accepting Petri net over A, let $N^R \in R(N)$ be a reduced net for N, and let $\sigma \in A^*$ be an activity trace over A. Then an optimal trace alignment h for σ and N has costs 0 if and only if an optimal trace alignment h' for σ and N^R has costs 0.*

Proof. As these rules preserve the branching activity behavior of the net (see Figure 9, they preserve the costs of an optimal alignment.

As an illustration, Figure 10 shows a hidden-and-reduced subnet h'_{1b} for the subnet h_{1b} as shown in Figure 5. Note that the subnet h'_{1b} requires, for example, the transition labeled a_1 to be fired exactly once, whereas it could be fired any

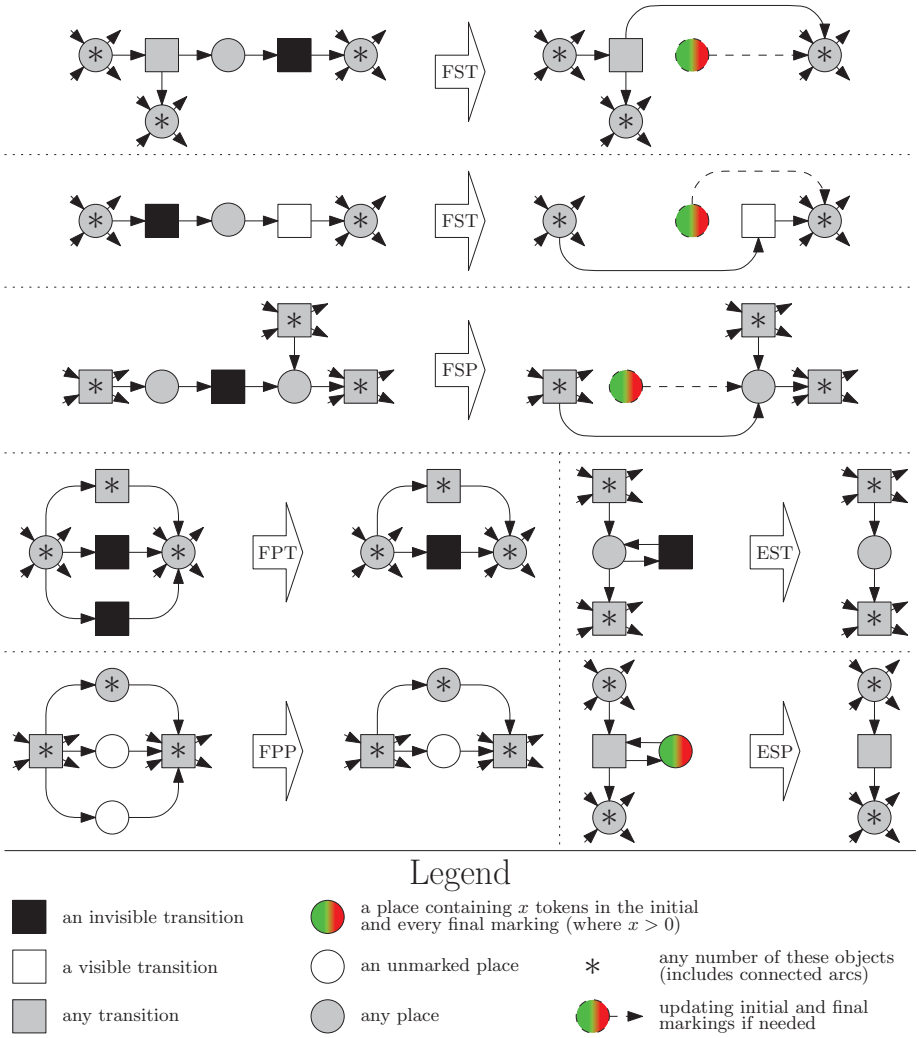

Fig. 9. Standard reduction rules extended for accepting Petri nets, and limited to preserve the branching activity behavior.

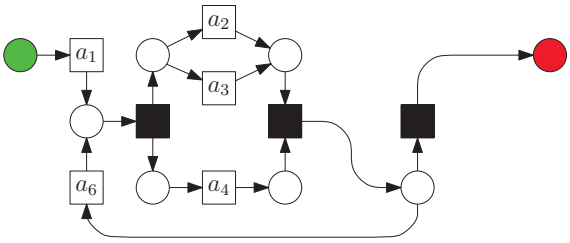

Fig. 10. Hidden-and-reduced subnet h'_{1b}.

number of times in the subnet h_{1b}. This may reduce the search space for the replayer drastically. Nevertheless, there is also a downside, as this hiding-and-reducing may result in additional invisible transitions, which need to be replayed as well.

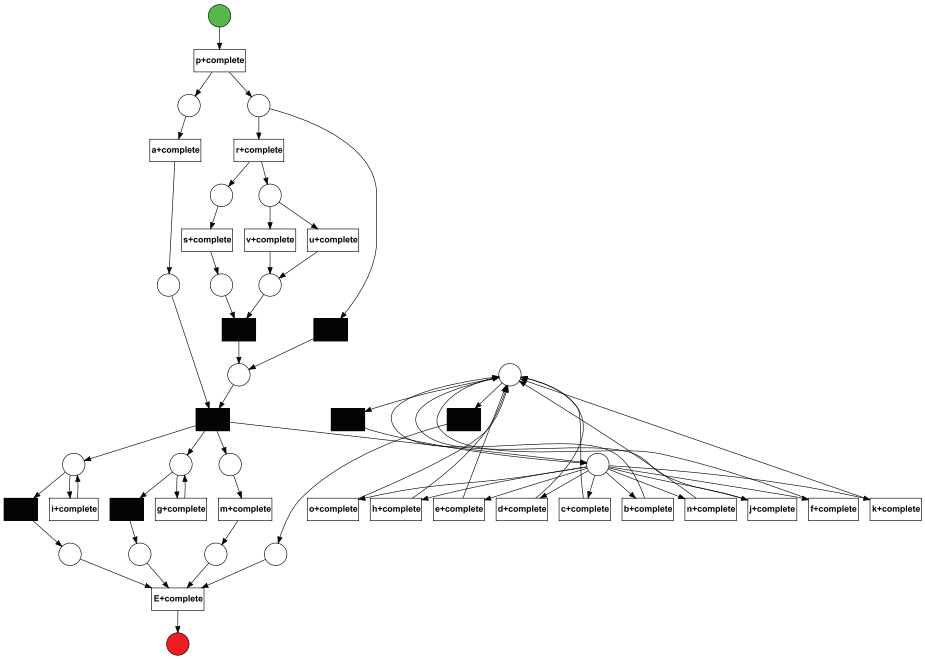

Fig. 11. Hidden-and-reduced subnet for *a22f0n05* case.

Figure 11 shows the hidden-and-reduced subnet for the subnet as shown in Figure 8.

We expect this hide-and-reduce replay to be faster than the monolithic replay, and to provide a better (higher) lower bound for the correct replay costs as the decomposed replay. In the next section, we will evaluate these assumptions.

4 Evaluation

We have evaluated the *hide-and-reduce* replay on the same data sets as we used in the Introduction. First, this section will evaluate the computation times of the hide-and-reduce replay by comparing them to the computation times of the monolithic replay. Second, it will evaluate the lower bounds for the correct replays costs as obtained by the hide-and-reduce replay by comparing them to both other replays.

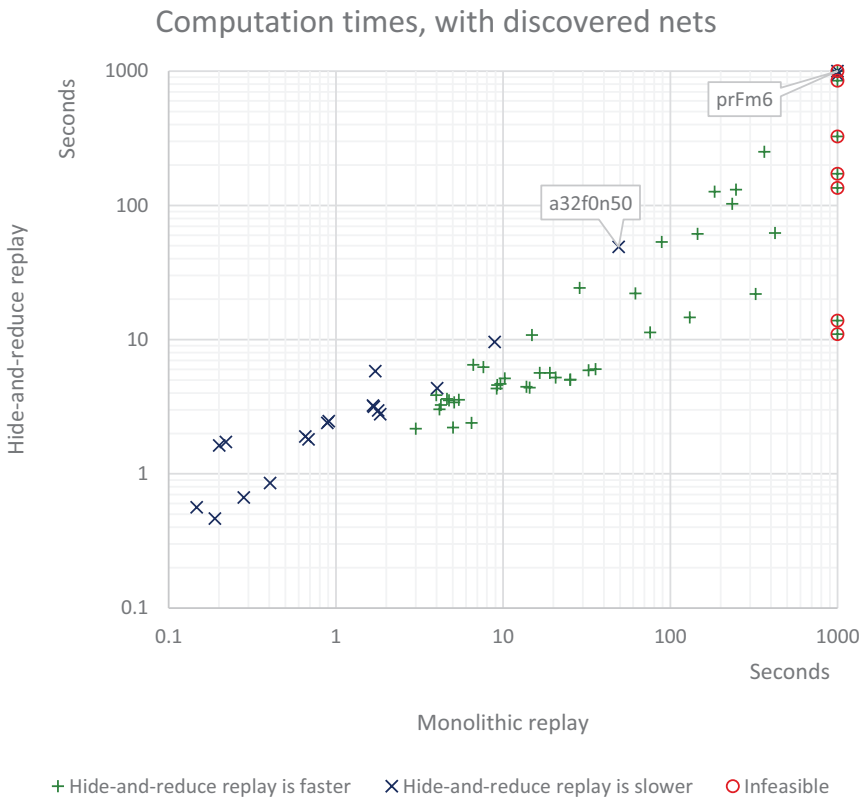

Fig. 12. Computation times for the monolithic and hide-and-reduce replay with the nets discovered using the *Inductive Miner* from the logs as provided by the data sets.

Figure 12 shows the results of the hide-and-reduce replay using the nets as discovered by the *inductive Miner*. Note the contrast with Figure 2, which shows similar times for the (original) decomposed replay.

Recall that the decomposed replay requires more than 10 minutes for a number of cases which required less than 10 minutes with the monolithic replay. The hide-and-reduce replay required more than 10 minutes for only one single case (*prFm6* [9]), which also requires more than 10 minutes with the monolithic replay. If the monolithic replay takes more than 10 seconds, then the alternative decomposed approach is likely to be faster. There is only a single case in the data sets for which this does not hold: *a32f0n50* [7]. For this case, the monolithic replay takes 49.1 seconds and the hide-and-reduce replay takes 49.2 seconds. The fact that the hide-and-reduce replay is not faster for this case is caused by a similar effect as with the *a22f0n05* case: The largest subnet contains almost all (28 of the 32) activities, and requires 48 seconds to be replayed. Of the remaining 1.2 seconds, half a second is required for replaying the other 15 subnets and 1.2 seconds are required for splitting the log and net and merging the alignments. In contrast, the decomposed replay requires more than 10 minutes for this and 13 other cases, and takes way longer for another three.

As a result we conclude that for the nets as discovered by the *Inductive Miner*, the hide-and-reduce replay requires less than 10 minutes for more cases than the other two replays. As such, it is an improvement over these other replays. Furthermore, it is typically faster than the monolithic replay when the latter takes more than 10 seconds. As such, it is an improvement over this replay.

Figure 13 shows the results of the hide-and-reduce replay using the nets as provided by the data sets. Like the decomposed replay, the hide-and-reduce replay requires less than 10 minutes for all data sets and it is typically faster than the monolithic replay. Exceptions to this are the *prAm6* and *prBm6* cases [9]. In both cases, the nets and logs are decomposed in more than 300 subnets and sublogs, the decomposed replay on these subnets and sublogs requires only 2 seconds (where the monolithic replay requires almost 40 seconds), but the reduction step takes about 60 and 40 seconds. For the other five cases for which the alternative decomposed replay takes more than 10 seconds, this reduction step is also the bottleneck. For all these cases, the decomposed replay requires at most 5 seconds, but the reduction requires from 25 to 167 seconds. Apparently, the reduction step is a possible bottleneck for the hide-and-reduce replay.

With regard to the computation times, we can conclude that both the decomposed and the hide-and-reduce replays can handle more cases than the monolithic replay, that they are typically faster than the monolithic replay, and that the net at hand largely determines whether the decomposed or the hide-and-reduce replay should be used. If the net has been designed, then the decomposed replay would be fastest, but this replay requires more than 10 minutes for many discovered nets. Therefore, in case the net has been discovered, or in case one does not know, the prudent approach would be the hide-and-reduce replay.

Apart from evaluating the differences in computation times, we also need to evaluate the reported costs of the three replays. We know that the monolithic

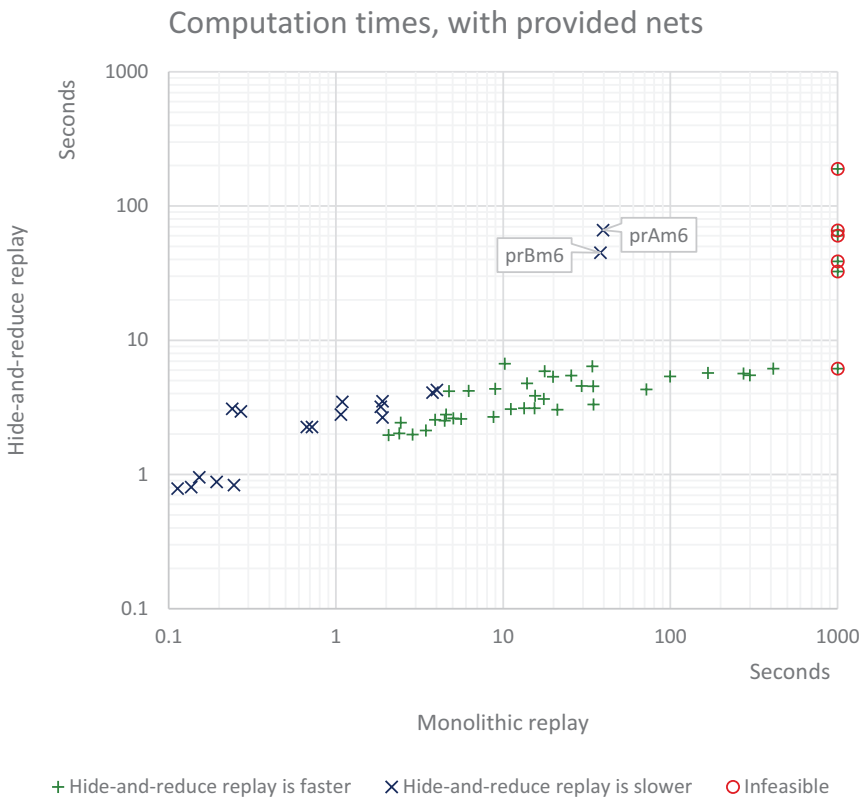

Fig. 13. Computation times for the monolithic and hide-and-reduce replay with the nets and logs as provided by the data sets.

replay provides the correct costs, and that both other approaches provide only a lower bound for these correct costs. Figure 14 shows the costs as reported by all replays on all nets (either provided or discovered) and all logs from the data sets. This figure shows that the costs as reported by hide-and-reduce approach are at least as good as the costs as provided by the decomposed replay, and that it is often better (higher). To emphasize this, we have highlighted in the figure those costs that are at least half (Above 50%) of the correct costs. Most of these highlighted costs are reported by the hide-and-reduce replay, only a few by the decomposed replay. This shows that the hide-and-reduce replay typically provides a better lower bound for the costs.

5 Conclusions

In this paper, we have shown that, for some cases, the decomposed replay [2] may take longer than the monolithic replay. Although for *designed* nets this

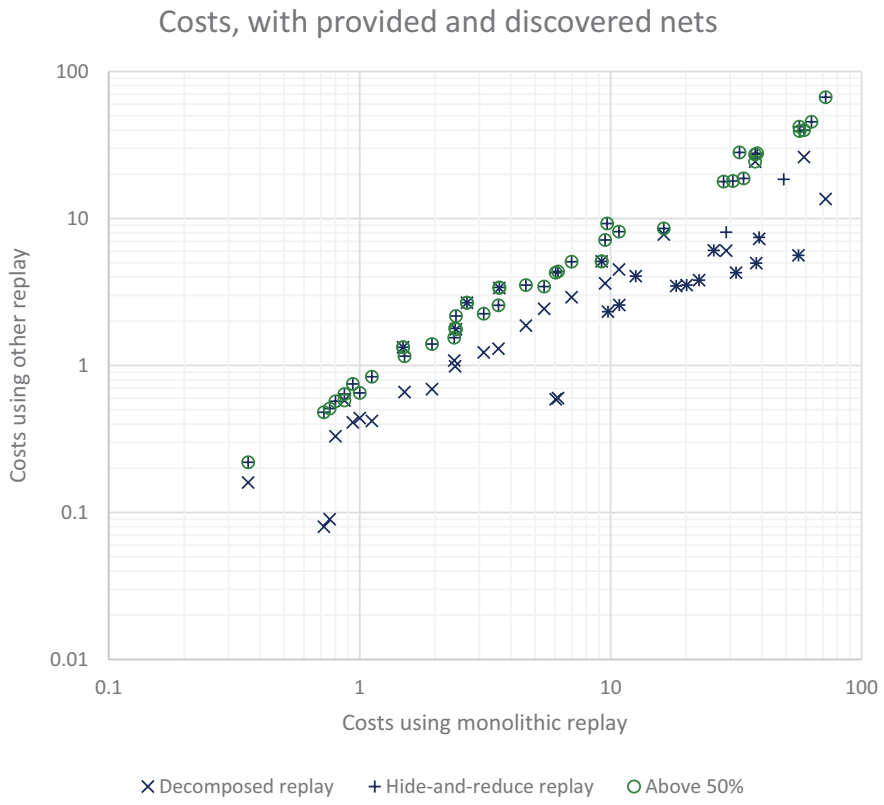

Fig. 14. Costs for all replays and all nets (provided or discovered) and logs as provided by the data sets.

is typically not the case, it may very well be the case for *discovered* nets. As such, a user who wants to check conformance on a log and a net discovered from that log may want to think twice to use the decomposed replay, as the monolithic replay may be faster (less than 10 seconds) while providing correct costs, while the decomposed replay may take longer (more than 10 minutes) while only providing a lower bound for the correct costs.

We have also shown that the root cause of this 'misbehavior' of the decomposed replay is the fact that the decomposition may result in a subnet with a fair amount of places and multiple source transitions. As a result of the fair amount of places, the state space will contain a fair amount of states as well. As a result of the multiple source transitions, the replayer needs to investigate from every reachable state at least all source transitions, which may all lead to new states that again need to be investigated. We have shown that the replay of a subnet with 13 places and 5 source transitions may require more than 10 minutes for the decomposed replayer.

To mitigate this problem with the source transitions, we have proposed a *hide-and-reduce* replay, which maintains the structure of the original net in the subnets, and hence does not introduce source transitions. We have shown that this new replay requires less than 10 minutes for all-but-one of the cases in the data sets used, and that this case required more than 10 minutes for the monolithic and decomposed replays as well. As such, the hide-and-reduce replay offers a replay that can handle more situations than either of the other replays can. For this reason, if it is possible that the net at hand was discovered from the log, then using the new hide-and-reduce replay is the best approach. Granted, it may be slower than the decomposed replay, but chances are that the case at hand simply requires more than 10 minutes for the decomposed replay.

Furthermore, we have shown that the costs as reported by the hide-and-reduce replay are at least as good as the costs as reported by the decomposed replay, but possibly better. Therefore, if it is important to the user to have an as-good-as-possible estimate for the correct costs, then using the hide-and-reduce replay is better than using the decomposed replay.

Finally, we have shown that in some cases the required reduction step in the hide-and-reduce replay is the bottleneck of the entire replay. Whereas the entire replay on the subnets would take only 2 seconds, the required reduction of these subnets may have taken 60 seconds. As a result of this bottleneck, the hide-and-reduce replay is sometimes slower than the monolithic replay. For this reason, we aim to improve in the near future on the reduction rules [10] as used by the hide-and-reduce replay. possibly, the rules may be simplified knowing that the reduction only needs to preserve the costs of any trace replayed on the net.

References

1. Aalst, W.M.P.v.d.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer-Verlag, 1st edn. (2011)
2. Aalst, W.M.P.v.d.: Decomposing Petri nets for process mining: A generic approach. Distrib. Parallel Dat. 31(4), 471–507 (2013)
3. Aalst, W.M.P.v.d., Adriansyah, A., Dongen, B.F.v.: Replaying history on process models for conformance checking and performance analysis. Data Min. Knowl. Disc. 2(2), 182–192 (2012)
4. Dongen, B.F.v.: BPI challenge 2012 data set (2012), doi: 10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
5. Dongen, B.F.v.: BPI challenge 2015 data set (2015), doi: 10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1
6. Leemans, S.J.J., Fahland, D., Aalst, W.M.P.v.d.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.M., Desel, J. (eds.) Application and Theory of Petri Nets and Concurrency. Lect. Notes Comput. Sc., vol. 7927, pp. 311–329 (2013)
7. Maruster, L., Weijters, A.J.M.M., Aalst, W.M.P.v.d., Bosch, A.v.d.: A rule-based approach for process discovery: Dealing with noise and imbalance in process logs. Data Min. Knowl. Disc. 13(1), 67–87 (2006)
8. Munoz-Gama, J., Carmona, J., Aalst, W.M.P.v.d.: Conformance checking in the large: Partitioning and topology. In: Daniel, F., Wang, J., Weber, B. (eds.) Business

Process Management: 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings. Lect. Notes Comput. Sc., vol. 8094, pp. 130–145 (2013)
9. Munoz-Gama, J., Carmona, J., Aalst, W.M.P.v.d.: Single-entry single-exit decomposed conformance checking. Inf. Syst. 46, 102–122 (2014)
10. Murata, T.: Petri nets: Properties, analysis and applications. P. IEEE 77(4), 541–580 (1989)