Decomposing Replay Problems: A Case Study

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

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

Abstract. Conformance checking is an important field in the process mining area. In brief, conformance checking provides us with insights how well a given process model matches a given event log. To gain these insights, we typically try to *replay* the event log on the process model. However, this replay problem may be complex and, as a result, may take considerable time. To ease the complexity of the replay, we can decompose the given process model into a collection of (smaller) submodels, and associate a (smaller) sublog to every submodel. Instead of replaying the entire event log on the entire process model, we can then replay the corresponding sublog on every submodel, and combine the results. This paper tests this *divide-and-conquer* approach on a number of process models and event logs while using existing replay techniques. Results show that the decomposed replay may indeed be faster by orders of magnitude, but that success is not guaranteed, as in some cases a smaller model and a smaller log yield a more complex replay problem.

1 Introduction

In the area of process mining, we typically distinguish three different fields: *discovery*, *conformance checking*, and *enhancement* [1]. The discovery field is concerned with mining an event log for a process model (for the remainder of this paper, we assume that these process models are represented using Petri nets). We are given an event log, and we try to come up with some process model that nicely represents the observed behavior in the event log. The conformance checking field is concerned with checking whether an event log matches a process model. We are given an event log and a process model, and we try to provide insights how well these two match. The enhancement field is concerned with enhancing a process model using an event log. We are given an event log and a process model, and we copy data as found in the event log into the process model. A popular enhancement technique is to add durations to a process model based on timestamps in the log. This allows us to detect bottlenecks in a process model. This paper focuses on the conformance checking field [10].

As mentioned, for conformance checking, it is vital that we can match the event log with the process model. A popular approach for matching both is to replay the entire event log on the process model. Every trace of the event log is replayed in the process model in such a way that mismatches are minimized in

some way. Of course, our aim is to match every event in the trace to a possible action in the process model. However, this is not always possible. In some cases, we cannot match an event to any possible action in the process model, or vice versa, we cannot match an necessary action according to the model to an event in the log. The result of this matching process is an *alignment* [4] between every trace of the event log and the process model. A situation where the alignment cannot match an event to any action can be considered as a situation where we decided to skip a position in the log during replay. This is referred to as a *log move*: We handle the event in the log (by skipping it) but do not reflect this move in the model. Vice versa, a situation where the alignment cannot match an action to any event can be considered as a situation where we decided to skip the action. Such a situation is referred to as a *model move*. The remaining situation where an event and an action are matched is referred to as a *synchronous move*. By associating *costs* to these situations, we can obtain a best alignment by minimizing these costs. Based on this idea of associating costs, replay techniques have been implemented in the process mining tool ProM 6 [3].

Earlier work [5, 2] has shown that we can apply a *divide-and-conquer* approach to these cost-based replay techniques. Instead of replaying the entire event log on the entire process model, we first decompose the process model into a collection of submodels. Second, we create a sublog from the entire event log for every submodel. Behavior not captured by a submodel will be filtered out of the corresponding sublog, only behavior captured by the submodel will be filtered in. Third, we compute subcosts for every combination of submodel and sublog. Fourth and lasts, we replay every sublog on the corresponding submodel in such a way that the subcosts are minimized. It has been proven that the weighted accumulated subcosts are a lower bound for the actual overall costs [11]. Moreover, the fraction of fitting cases is the same with or without decomposition [11]. As a result, the accumulated subcosts is a lower bound for the actual costs.

Conceptually, the entire model can be regarded as a decomposition of itself into a single submodel, with a single sublog. Nevertheless, proper decompositions may exist as well. An example of a (most likely) proper decomposition is the maximal decomposition as described in [2, 11]. This maximal decomposition uses an equivalence class on the arcs in the process model to decompose the model into submodels. This equivalence class corresponds to the smallest relation for which the incident arcs of places, silent transitions, and visible transitions with a non-unique label are equivalent. As a result, the maximal decomposition results in submodels that only have visible transitions with unique labels in common. Nevertheless, other proper decompositions of a process model may exist (like [7]) which may be of interest for the decomposed replay problem.

Replaying all sublogs on the corresponding submodels has two possible advantages:

1. It may highlight problematic areas (submodels) in the model, that is, submodels with a bad fitness. This information can hence be used for diagnostic purposes.

2. It may be much faster than replaying the entire log on the entire model, while the fraction of fitting cases is the same for both. Obviously, this may save time while maintaining quality.

This paper focuses on the second advantage, that is on the possible speed-up of the entire replay when using divide-and-conquer techniques. As the overall model is bigger than any of its submodels, it is expected that the replay problem of the entire log is more complex than the replay problem of all sublogs. As a result, determining the actual costs may take simply way too much time, while determining a decent lower limit may take an acceptable amount of time. This paper takes four different process models with six corresponding different event logs, and several ways to decompose the process models into submodels, and checks whether determining the costs of more fine-grained decomposed replay problems are indeed faster than that of more coarse-grained decomposed replay problems. Furthermore, if it is indeed faster, it checks whether the obtained costs are still acceptable. Note that a technique that return a trivial lower bound (like 0) for the costs is very fast, but is not acceptable as we do not gain any insights by such a technique.

2 Preliminaries

This section introduces the basic concepts of process models and event logs as used in this paper. Furthermore, it details how we determine the cost associated by replaying an event log on a process model. For the remainder of this paper, we use $\mathcal{U}$ to denote the universe of labels.

A process model contains a labeled Petri net [9, 8] (P, T, F, l), where P a set of places, T is a set of transitions such that $P \cap T = \emptyset$, $F \subseteq (T \times P) \cup (P \times T)$ a set of arcs, and $l \in (T \not\to \mathcal{U})$ a function that maps a transition onto its label. A marking M of a net (P, T, F, l) is a multiset of places, denoted $M \in \mathcal{B}(P)$. The input set of a place or transition $n \in P \cup T$ is denoted $\bullet n$ and corresponds to the set of all nodes that have an arc going to n, that is, $\bullet n = \{n' | n' F n\}$. In a similar way, the output set is defined: $n\bullet = \{n' | n F n'\}$. Transition $t \in T$ is *enabled* by marking M in net $N = (P, T, F, l)$, denoted as $(N, M)[t\rangle$, if and only if M contains a token for every place in the input set of t, that is, if and only if $M \leq \bullet t$. An enabled transition $t \in T$ may *fire*, which results in a new marking M' where a token is removed from every place in the input set of t and a token is added for every place in its output set, that is, $M' = M - \bullet t + t\bullet$. We use $(N, M)[t\rangle(N, M')$ to denote that transition t is enabled by marking M, and that firing transition t in marking M results in marking M'. Let $\sigma = \langle t_1, t_2, \ldots, t_n \rangle \in T^*$ be a sequence of transitions. $(N, M)[\sigma\rangle(N, M')$ denotes that there is a set of markings $M_0, M_1, \ldots, M_n$ such that $M_0 = M$, $M_n = M'$, and $(N, M_i)[t_i\rangle(N, M_{i+1})$ for $0 \leq i < n$. A marking M' is *reachable* from a marking M if there exists a σ such that $(N, M)[\sigma\rangle(N, M')$. A transition $t \in \text{dom}(l)$ is called *visible*, a transition $t \notin \text{dom}(l)$ is called *invisible*. An occurrence of a visible transition t corresponds to an observable activity $l(t)$. We can project a transition sequence $\sigma \in T^*$ onto its sequence of observable activities $\sigma_v \in \mathcal{U}^*$ in

a straightforward way, where all visible activities are mapped onto their labels and all invisible transitions are ignored. We use $(N, M)[\sigma_v \rhd (N, M')$ to denote that there exists a $\sigma \in T^*$ such that $(N, M)[\sigma\rangle(N, M')$ and σ is mapped onto σ_v.

Furthermore, a process contains an initial state and a final state, that is a process model $\mathcal{P}$ is a triple (N, M_0, M_n), where $N = (P, T, F, l)$ is a labeled Petri net, $M_0 \in \mathcal{B}(P)$ is its initial marking, and $M_n \in \mathcal{B}(P)$ is its final marking. The set of *visible* traces for a process model $\mathcal{P} = (N, M_0, M_n)$, denoted $\phi(\mathcal{P})$, corresponds to the set of sequences of observable activities that start in the initial marking and end in the final marking, that is, $\phi(\mathcal{P}) = \{\sigma_v | (N, M_0)[\sigma_v \rhd (N, M_n)\}$.

An event log [1] L is a multiset of *traces*, where a trace is a sequence of activities. Thus, if $A \subseteq \mathcal{U}$ is the set of activities, then $\sigma \in A^*$ is a *trace* and $L \in \mathcal{B}(A^*)$ is an *event log*. An event log L can be projected onto some set of activities A', denoted $L\restriction_{A'}$, in a straightforward way: All events not in A' are filtered out, and all events in A' are filtered in.

The *replay problem* [4] for an event log L and a process model $\mathcal{P}$ can now be described as finding, for every trace $\sigma_L \in L$, the transition sequence $\sigma_\mathcal{P} \in T^*$ for which its corresponding sequence of observable activities $\sigma_{\mathcal{P},v} \in \phi(\mathcal{P})$ matches σ_L *best*. To determine which transition fits a trace best, we *align* σ_L and $\sigma_{\mathcal{P},v}$ and associate *costs* to every misaligned transition and/or event. A misaligned transition means that we need to execute a visible transition in the process model that is not reflected in the event log, and is called a *model move*. A misaligned event means that an activity has been executed and logged that is not reflected by a corresponding visible transition in the process model, and is called a *log move*. An aligned transition-event pair means that both the event log and the process model agree on the next activity, and is called a *synchronous move*. By associating costs to model moves and log moves, and by minimizing the costs, we can determine the transition sequence from the process model that best fits a trace from the event log.

For the *decomposed replay problem* for an event log L and a process model $\mathcal{P}$, we first decompose the process model into a collection of smaller process models $\{\mathcal{P}_1, \ldots, \mathcal{P}_M\}$. Second, we map the costs for replaying the entire log on the entire net to costs for every smaller process model $\mathcal{P}_i = ((P_i, T_i, F_i, l_i), M_{0,i}, M_{n,i})$. In earlier work [11], we have shown that these costs can be mapped in such away that the accumulated costs for the decomposed replay problem is a lower bound for the costs of the original replay problem. Third, we filter the log for every smaller process model into a smaller log $L_i = L\restriction_{\mathrm{rng}(l_i)}$. Fourth, we replay every smaller log L_i on the corresponding smaller process model $\mathcal{P}_i$, using the adapted costs. Fifth and last, we accumulate the resulting costs into a single costs, which is a lower bound for the actual costs.

In [11], we have shown how to decompose a process model $\mathcal{P} = ((P, T, F, l), M_0, M_n)$ in such a way that the decomposition is maximal. For this maximal decomposition, we introduce an equivalence class on the arcs of F. Arcs will end up in the same submodel $\mathcal{P}_i$ if and only if they are equivalent, where this equivalence is defined as the smallest relation for which the following rules hold:

Table 1. Characteristics of process models

Process model	Transitions	Places	Arcs	Labels
RepairExample	12	12	26	8
A32	32	32	74	32
Bpic2012A	11	14	28	10
Bpic2012	58	44	124	36

Place An incident (input or output) arc of a place is equivalent to all incident arcs of that place.

Invisible transition An incident arc of an invisible transition is equivalent to all incident arcs of that transition.

Visible transition with non-unique label An incident arc of a visible transition with a non-unique label (that is, there exist other transitions that have the same label) is equivalent to all incident arcs of all transitions with that label.

As a result of these rules, any place, any invisible transition, and any visible transition with non-unique label will be part of a single submodel only, whereas visible transitions with unique labels may be split over different submodels. As a result, only these visible transitions with unique label interface between the different submodels. In [11], we have proved that this decomposition preserves perfect replay: The entire event log L can be replayed perfectly (that is, no costs and/or mismatches) on process model $\mathcal{P}$ if and only if every sublog L_i can be replayed perfectly on submodel $\mathcal{P}_i$. Moreover, a trace perfectly fits the overall model if and only if its projection fits each of the submodels. Hence, the fraction of fitting traces can be computed precisely using any decomposition.

3 Case Study Setting

For the case study, we will use four different process models (RepairExample, A32, Bpic2012A, and Bpic2012) with six corresponding different logs (RepairExample, A32f1n00, A32f1n10, A32f1n50, Bpic2012A, and Bpic2012). Table 1 shows the characteristics of these models, whereas Table 2 shows the characteristics of the corresponding logs.

The RepairExample model comes with a single event log, which is typically used for demonstration (or tutorial) purposes. The A32 model comes with three event logs, which contain a varying amount of noise. The first log, A32f1n00, contains no noise ('0%'), the second log, A32f1n10, contains some noise ('10%'), and the third log, A32f1n50, contains much noise ('50%'). This model and these logs were used to test genetic mining algorithms on their ability to recreate the original model from noisy event logs. The Bpic2012A model and event log stem from the BPI 2012 Challenge log and originate from [6]. Note that this log is a real-life log, which was obtained from a company. The Bpic2012A event log is obtained by filtering out all events that start with either "O" or "W", that

Table 2. Characteristics of event logs

Event log	Cases	Events	Event Classes
RepairExample	1104	11,855	12
A32f1n00	1000	24,510	32
A32f1n10	1000	24,120	32
A32f1n50	1000	22,794	32
Bpic2012A	13,087	60,849	10
Bpic2012	13,087	262,200	36

is, it contains only the events that start with "A". The corresponding model was hand-made based on results of running the Transition System Miner on this filtered log (cf. [13]). The Bpic2012 model and log also stem from this challenge log. In contrast with the Bpic2012A log, the Bpic2012 log contains all events from the Challenge event log, and some more. These extra events allowed us to mine this log using a passage-based technique, of which the Bpic2012 model is a result [2].

At the moment of conducting this case study, two cost-based replayers were available in ProM 6: One which uses ILP (Integer Linear Programming) techniques, and one which does not. To understand the difference between both, please recall that we need to find a best match for every trace in the log. To do so, the replayer typically starts with the initial marking for the process model, and an empty trace for the replayed trace, and tries to match the first event in the remaining trace to any enabled visible transition in the model. If both match, then the visible transition is fired and the event is moved to the replayed trace. If both do not match, then the replayer has to choose between different possibilities:

Model move Fire the visible transition but keep the event in the remaining trace.

Log move Move the event to the replayed trace but do not fire any transition.

Invisible transition Fire some invisible transition and keep the event in the remaining trace (in the hope that some matching visible transition will become enabled by doing so).

Given the fact that the process model is a Petri net, and given the fact that we know the final marking which we should reach, we can use Petri-net-related techniques to remove some of these possibilities. This is exactly what the ILP-based replayer does: It uses information on the structure of the Petri net to rule out certain paths in the search space. However, this comes at a price, as we need to construct an ILP problem first, and then solve it. As the overhead of constructing and solving the ILP problem may outweigh the speed-up that results from pruning the search space, a replayer that does not use ILP techniques is also available. Based on this description, we will use the ILP-based replayer for replaying the entire logs as well as for replaying sublogs on submodels that

Table 3. Replay results of event logs

Event log	**Running time** (in seconds)	**Costs**
RepairExample	0.25	0.197
A32f1n00	11	0.000
A32f1n10	17	0.993
A32f1n50	32	4.521
Bpic2012A	0.59	1.293
Bpic2012	480	14.228

exceed a certain size (in this case, 8 transitions). So, if the corresponding submodel contains at least 8 transitions, then the ILP-based replayer will be used. Otherwise, the non-ILP-based replayer will be used.

In a similar way, two event log implementations were available: One which uses disk-based buffering ("Buffered") and one which uses main memory only ("Naive"). After doing some tests, we concluded that the "Naive" implementation used less memory and was faster than the "Buffered" implementation. As a result, we will use the "Naive" implementation throughout the case study.

By default, the log filter as implemented in ProM 6 works by first cloning the entire log, and then removing all events that have to be filtered out. Furthermore, this log filter effectively filters out any trace for which all events are filtered out. This latter effect is unwanted in our situation, as we need to replay the empty traces as well in the sublogs. Especially if only a few activities should be filtered in for some sublog, cloning-and-filtering-out seems to be an expensive solution. For this reason, we created two new filters: One which clones and filters out the undesired events but does not remove empty traces, and one which creates a new log and filters in the desired events and also does not remove empty traces. If more than half of the activities need to be filtered out of the sublog, then the first filter (clone-and-filter-out) will be used. Otherwise, the second filter (empty-and-filter-in) will be used.

For the decomposed replay problem, we need to replay sublogs on submodels. As these sublogs and submodels are all different, we can compute the costs of replaying these sublogs on these submodels in parallel. Given the fact that we will use a machine with four physical cores and four virtual cores, we allow the decomposed replay to replay upto four sublogs in parallel (using four different threads), which can then all be scheduled on the physical cores. Furthermore, we will start by replaying the sublogs that correspond to the largest submodels, as we expect these replays to take the longest time.

For the costs during the replay, we accept the defaults costs as provided by the replay plug-ins. That is, a model move costs 2, a log move costs 5, and firing an invisible transition costs 0.

Table 3 shows the results of running the ILP-based replayer on the six event logs, where running times have been rounded to the two most significant digits (this is done throughout the paper). Note that the absolute values for these

Table 4. Decomposed replay results of event logs using near-maximal decomposition

Event log	**Submodels**	**Running time** (in seconds)	**Costs**
RepairExample	6	(171%, –) 0.43	(99%, ++) 0.196
A32f1n00	30	(12%, ++) 1.3	(100%, ++) 0.000
A32f1n10	30	(7%, ++) 1.1	(45%, o) 0.444
A32f1n50	30	(4%, ++) 1.2	(48%, o) 2.155
Bpic2012A	8	(378%, –) 2.2	(49%, o) 0.629
Bpic2012	12	(—) DNF	

running times are not that important for this paper, as we only want to *compare* them. In the next section, we will compare these results to the results of running the decomposed replayer on these logs. Our goal will be to check whether the decomposed replayer returns still-acceptable costs in better times for certain decompositions.

4 Case Study Results

The first decomposition we will check for the results, is a *near-maximal* decomposition. In Section 2, we have detailed a maximal decomposition based on an equivalence class between arcs. In this maximal decomposition, it is possible to have a submodel $((P_i, T_i, F_i, l_i), M_{0,i}, M_{n,i})$ that subsumes some other submodel $((P_j, T_j, F_j, l_j), M_{0,j}, M_{n,j})$ when it comes down to the labels, that is, $\text{rng}(l_i) \subseteq \text{rng}(l_j)$. A typical submodel where this might happen is the submodel that contains the source (or sink) place in case of a workflow net [12]. As every label of the subsumed submodel is contained in the subsuming model, every event of the 'subsumed' event log will be contained in the 'subsuming' event log. As a result, we can replay (almost for free) the 'subsumed' log while replaying the 'subsuming' log. For this reason, we add the subsumed submodel to every subsuming submodel (there may be more subsuming submodels), and do not replay the subsumed submodel on its own. The resulting decomposition is called a near-maximal decomposition.

Table 4 shows the results of replaying the event logs on the submodels as obtained by the near-maximal decomposition. These results show that the A32 logs are replayed much faster than with the regular replay (though the costs are significantly lower), but that the other replays got worse, with the Bpic2012 log as most negative example. For this log, the decomposed replay simply did not finish ("DNF"), as it eventually ran out of resources (4 GB of memory). From these results, we conclude that decomposing a replay problem need not improve the time required for replay.

Figure 1 shows the submodel that was the bottleneck for the decomposed replay of the Bpic2012 log. This submodel contains 12 invisible transitions (the entire net contains 22, so more than half of these invisible transitions ended up in

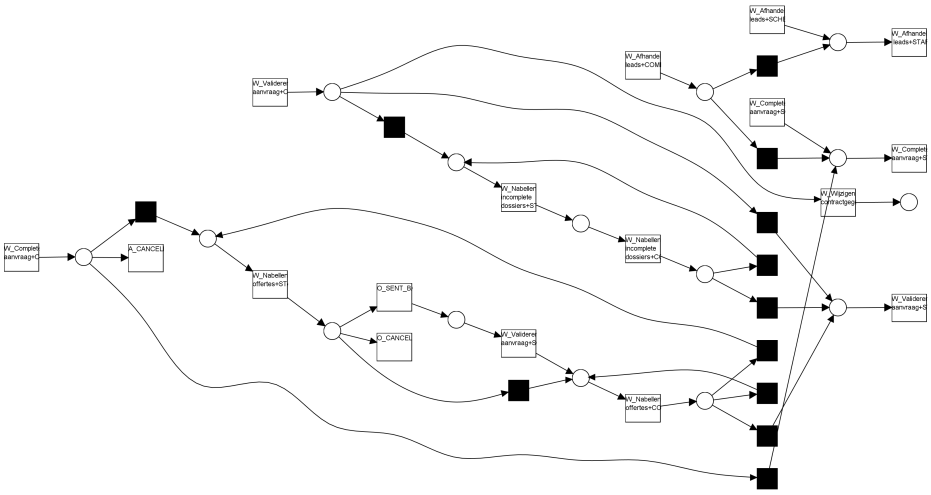

Fig. 1. Problematic submodel that results from near-maximal decomposition of Bpic2012

this submodel), of which 3 loop back, 5 source transitions, and 5 sink transitions. As a result, the replay of the corresponding sublog on this specific submodel is very complex: In any reachable state, there are at least 5 enabled transitions, and in many states there will be some invisible transitions enabled as well). It might help the replay if we restricted the unlimited freedom of these source transitions in some way. For this reason, we introduced a place that effectively restricts the number of tokens, say n, that may be present in the submodel. Initially, this place p contains n tokens. Any transition t in the original submodel is changed in the following way:

More incoming arcs If t has more incoming arcs than outgoing arcs, that is, if $|\bullet t| > |t\bullet|$, then x outgoing arcs to place p are added, where $x = |\bullet t| - |t\bullet|$.

More outgoing arcs If t has more outgoing arcs then incoming arcs, that is, if $|t\bullet| > |\bullet t|$, then x incoming arcs from place p are added, where $x = |t\bullet| - |\bullet t|$.

As many incoming arcs as outgoing arcs If t has as many incoming arcs as outgoing arcs, that is, if $|t\bullet| = |\bullet t|$, then nothing is added.

The exceptions to these rules are the arcs from source places and the arcs to sink places, as these arcs will not be counted. Reason for doing so is that otherwise all n tokens may end up in a sink place, which results in a dead submodel. As a result of applying these changes, every transition in the resulting net has as many incoming arcs as outgoing arcs (excluding source and sink places), and hence the number of tokens in every reachable marking (again excluding source and sink places) is constant, that is, equal to n. This amount of tokens

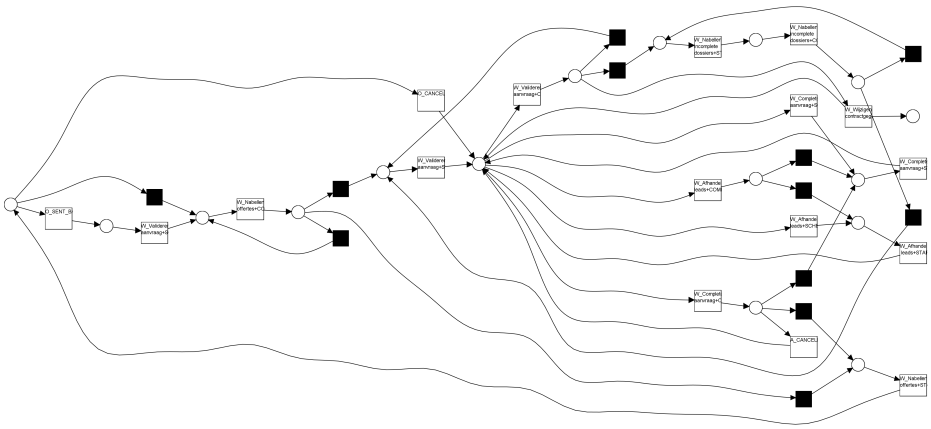

Fig. 2. Restricted problematic submodel

n is determined in such a way that the execution of a source transition can be followed by the execution of a sink transition. Therefore, we will put a single token into the place for a source transition, and as many tokens as needed to fire (once) all other (non-source) transitions in the preset of this place. Figure 2 shows the resulting submodel, where the place in the middle (the one with the many incident arcs) is the place that restricts the otherwise unlimited behavior of the former source transitions. Given the structure of this submodel, this place contains initially a single token, which allows for only a single thread in this submodel during replay.

With this change made to the decomposition of the entire net (note that only the behavior of the problematic submodel was restricted in the way as described, other submodels were unaffected), the decomposed replay finished in 470 seconds (98%), with costs of 8.722 (61%). The replay of the problematic submodel took 460 seconds (costs 5.210), which explains the better part of the replay time (please recall that for the decomposed replay we allowed 4 threads to be run simultaneously). Although the decomposed replay now finished, it finished in almost the same time as the regular replay with significant lower costs. Furthermore, by restricting the behavior of this submodel, we cannot claim anymore that the resulting costs are a lower bound for the actual costs. For this reason, we tried to reduce the number of invisible transitions instead.

An invisible transition with single input and single output can be reduced in such a way, that the behavior of the resulting model is not a restriction, but an extension of the behavior of the original model. In general, the single input place of the invisible transition will have input transitions and additional (not counting this invisible transition) output transitions. Likewise, its single output place will have additional input transitions (not counting this invisible transition) and output transitions. In general, a path from such an additional

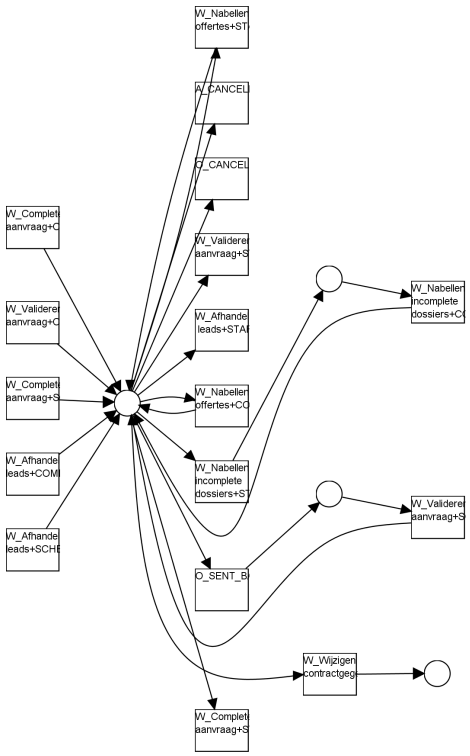

Fig. 3. Reduced problematic submodel

input transition to such an additional output transition will not be possible in the model, as the invisible transition works 'the wrong way'. However, all other paths are possible, like a path from an input transition to an output transition, and a path from an input transition to an additional output transition. Thus, if we merge the single input place and the single output place, and remove the invisible transition, than we only add behavior in the model. As a result, the replay costs might go down (which is safe for a lower bound), but cannot go up. Figure 3 shows the resulting submodel.

With this change made, the decomposed replay finished in 190 seconds (40%), with costs of 6.676 (47%). The replay of the problematic submodel took 140 seconds (costs 3.163). The bottleneck submodel is now a different submodel, which requires a replay time of 180 seconds.

Obviously, the reduction of invisible transitions has led to a 60% decrease in running time (from 480 seconds to 190 seconds), but also to a 53% decrease in the computed costs (from 14.228 to 6.676). As we know that the decomposed replay returns a lower bound for the costs, this decrease in costs is okay, but perhaps we can do better by not reducing all invisible transitions. Possibly, there is some trade-off somewhere between the invisible transitions to reduce and the

Table 5. Effects of reducing invisible transitions on running time and computed costs

Threshold	**Running time** (in seconds)	**Costs**
32	(42%, +) 200	(47%, o) 6.676
16	(58%, +) 280	(50%, o) 7.091
8	(386%, –) 1900	(55%, o) 7.849

decrease in computed costs. For this reason, we again focus on the reduction of an invisible transition. Clearly, if (1) the single input place of an invisible transition has no additional output transitions and (2) the single output place of that invisible transition has no additional input transitions, then the reduction of this invisible transitions leads to no extra behavior. In contrast, if there would be 3 additional output transitions and 4 additional input places, then this reduction would lead to $3 \times 4 = 12$ possible new paths. If we want to retain as much of the original behavior in the reduced net, then we want to reduce invisible transitions for which the product of the number of additional output transitions and the number of additional input transitions is below some threshold. If this threshold equals 0, then no new paths are allowed, if it is sufficiently large (say, 100), then any reduction is allowed. To check the effect of this threshold, we have run the decomposed replay for a number of possible thresholds. Table 5 shows the results. Please note that the result of the reduction with some fixed threshold needs not be unique, the resulting net might be non-deterministic. We are aware of this, but want to check its effect anyway.

This table indicates that if we set the threshold to 32, that then all invisible transitions will be reduced (running time approx. the same and costs the same). Furthermore, this table indicates that, to get a good lower bound for the costs (if possible at all), we need to spend more running time than we would need for the original (not decomposed) replay problem: To obtain a lower bound for the costs that is slightly above half of the actual costs, we need to spend more than three times as much running time. As a result, we conclude that this reduction technique has its merits (it actually finishes with costs that are reasonable for the running time required), but that there is no significant gain in fine-tuning this technique.

A third option to cope with the problematic submodel could be to organize the submodels into submodels that are better suited for the replay at hand. Apparently, the replay techniques we are using have problems with certain submodels (many source transitions, many invisible transitions), so we should try to avoid generating such submodels. For this reason, we propose a slightly changed notion of equivalence:

Place An incident (input or output) arc of a place is equivalent to all incident arcs of that place.

Invisible transition An incident arc of an invisible transition is equivalent to all incident arcs of that transition.

Table 6. Decomposed replay results of event logs using near-minimal decomposition

Event log	Submodels	Running time (in seconds)	Costs
RepairExample	2	(136%, –) 0.31	(100%, ++) 0.197
A32f1n00	4	(18%, ++) 1.9	(100%, ++) 0.000
A32f1n10	4	(13%, ++) 2.1	(94%, +) 0.929
A32f1n50	4	(10%, ++) 3.1	(96%, +) 4.322
Bpic2012A	1	(125%, –) 0.74	(100%, ++) 1.293
Bpic2012	1	(101%, –) 480	(100%, ++) 14.228

Visible transition with non-unique label An incident arc of a visible transition with a non-unique label (that is, there exist other transitions that have the same label) is equivalent to all incident arcs of all transitions with that label.

Visible transition with unique label The i-th incoming arc of a visible transition with unique label is equivalent to the i-th outgoing arc of that transition, if both arcs exist.

Note that only the fourth rule on visible transitions with unique labels has been added. Also note that we now assume that some order exists among the incident arcs of such a transition, as we link the i-th incoming arc to the i-th outgoing arc, if possible. As a result, again, the result may be non-deterministic. Nevertheless, this approach may help in suppressing submodels with many source transitions, as the result of the fourth rule may be that a submodel with a visible sink transition t is merged with a submodel with a visible source transition t. We use the term *near-minimal* decomposition to refer to the decomposition that results from these adapted equivalence rules, as many submodels will be linked together using these new rules. Table 6 shows the results for this decomposition.

This table shows that we typically obtain good results (costs-wise) using the near-minimal decomposition, but that we sometimes use slightly more time. Furthermore, this table shows that for the Bpic2012A and Bpic2012 event logs the near-minimal decomposition turns out to be the minimal decomposition, as both resulted in only a single submodel. In both cases this is caused because every parallel construct includes invisible transitions (either as split or as join). To be able to split these constructs, we can relax the equivalence rules in such a way that we also allow the decomposition to split invisible transitions in the same way as it splits visible transitions with unique labels:

Place An incident (input or output) arc of a place is equivalent to all incident arcs of that place.

Invisible transition The i-th incoming arc of an invisible transition is equivalent to the i-th outgoing arc of that transition, if both arcs exist.

Visible transition with non-unique label An incident arc of a visible transition with a non-unique label (that is, there exist other transitions that have

Table 7. Decomposed replay results of event logs using near-minimal (with invisible transitions) decomposition

Event log	Submodels	Running time (in seconds)	Costs
BPIC2012A	3	(391%, –) 2.3	(98%, ++) 1.272
BPIC2012	3	(83%, o) 400	(100%, ++) 14.227

the same label) is equivalent to all incident arcs of all transitions with that label.

Visible transition with unique label The i-th incoming arc of a visible transition with unique label is equivalent to the i-th outgoing arc of that transition, if both arcs exist.

This relaxation maintains the property that any perfectly fitting trace in the overall model is a perfectly fitting trace in the submodels (as we are still possible to replay such a perfectly fitting trace in the submodels). However, it may break the property that a perfectly fitting in the submodels is a perfectly fitting trace in the overall model, as the invisible transitions on the border of the submodels may not agree. When replaying a trace on the submodels, such a disagreement will not be noticed, but when replaying it on the overall model, it will be noticed, which leads to a mismatch and extra costs. Table 7 shows the results of this decomposition for the two event logs. Both computed costs are quite good, and the running time of the BPIC2012 log is better than that of the regular replay, but the running time of the BPIC2012A log is worse.

Another way to split up the near-minimal decomposition a bit further is to define a set of (visible) transitions for which the relaxed equivalence does not hold, that is, that assume that all incident arcs are not equivalent. In this paper, we will use the term *milestone transitions* for such transitions. As a result, we then have the following rules for equivalence:

Place An incident (input or output) arc of a place is equivalent to all incident arcs of that place.

Invisible non-milestone transition The i-th incoming arc of an invisible non-milestone transition is equivalent to the i-th outgoing arc of that transition, if both arcs exist.

Visible transition with non-unique label An incident arc of a visible transition with a non-unique label (that is, there exist other transitions that have the same label) is equivalent to all incident arcs of all transitions with that label.

Visible non-milestone transition with unique label The i-th incoming arc of a visible non-milestone transition with unique label is equivalent to the i-th outgoing arc of that transition, if both arcs exist.

Note that the equivalence of milestone transitions is derived from these rules. Initially, incident arcs of milestone transitions are not equivalent, but they may

become equivalent if this equivalence is the result of any combination of the four rules mentioned above. As such, it is still possible that multiple incident arcs for a milestone transition are equivalent.

As an example of this *milestone* decomposition, we have selected six such *milestone* transitions in the BPIC2012 model:

1. t14402
2. t16523
3. W_Beoordelen fraude+SCHEDULE
4. W_Nabellen incomplete dossiers+SCHEDULE
5. W_Valideren aanvraag+COMPLETE
6. W_Valideren aanvraag+START

Please note that the first two milestone transitions are invisible transitions, which have no counterpart in the event log, whereas the remaining four are visible. Together, the first four transitions form a sparsest cut in the model, where the middle two transitions link a submodel containing parallelism to a submodel containing an invisible transitions that allows the former submodel to be started over and over again during replay. Figure 4 shows the resulting decomposition. Using this decomposition, the decomposed replay takes only 100 seconds (22%) and results in a costs of 11.722 (82%), which is quite acceptable. The left submodel (the large one) took 100 seconds to replay (costs 7.102), the top-right submodel took 97 seconds (costs 2.806), the middle-right submodel took 66 seconds (costs 0.951), and the bottom-right submodel (the parallel one) took 2.8 seconds to replay (costs 0.863). Obviously, the fact that we could run the decomposed replay on 4 different threads helped a lot in reducing the running time, as the total running time would have taken not 100 but $100 + 96 + 66 + 2.8 \approx 260$ seconds.

5 Tool implementation

The decomposed replay has been implemented in the ProM 6[1] *Passage* package, which depends on the *PNetReplayer* package for the ILP-based replayer and non-ILP-based replayer. For sake of completeness, we mention that for this case study we have used version 6.1.122 of the former and version 6.1.160 of the latter package. The *Passage* package contains a number of plug-ins that are relevant for the decomposed replay:

Create Decomposed Replay Problem Parameters Using this plug-in, the user can set the parameters which are to be used for both decomposing a model and a log into submodels and sublogs, and the decomposed replay that may follow this decomposition. Both an automated and a manual version exist for this plug-in. The automated version takes a model (a Petri net) and generates default parameters for this model. The manual version takes

[1] A nightly build version of ProM 6 containing the required packages can be downloaded from http://www.promtools.org/prom6/nightly.

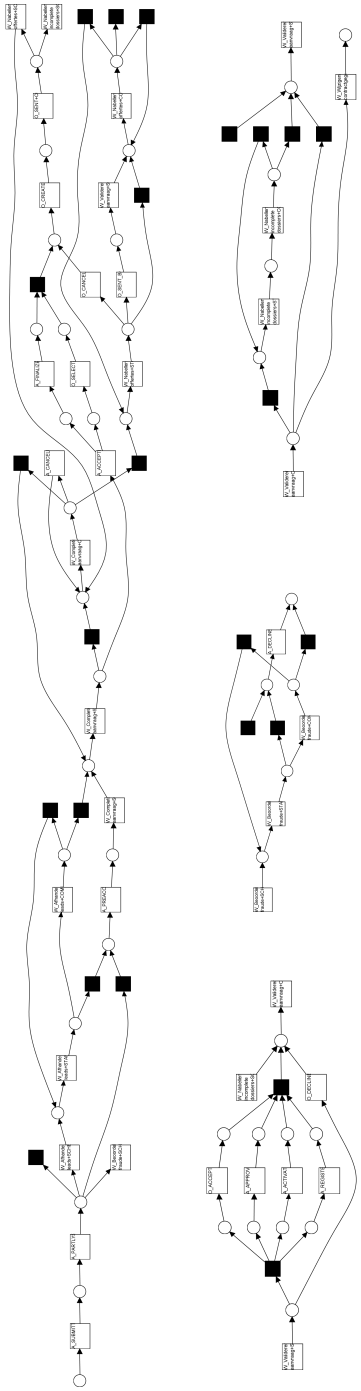

Fig. 4. Milestone decomposition of BPIC2012

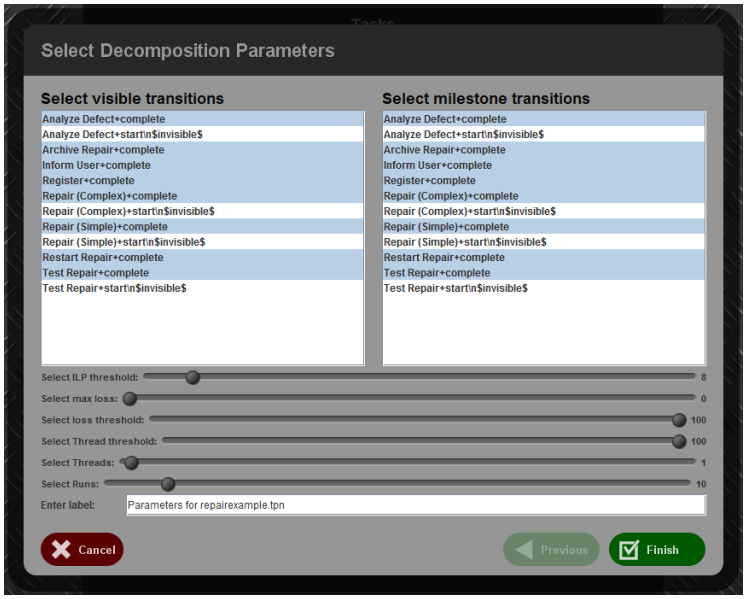

Fig. 5. Dialog for setting decomposed replay parameters

either a model with default parameters or existing parameters, and allows the user to change these parameters through a dialog. Figure 5 shows this dialog.

Select visible transitions Using this list, the user can select the transitions that are allowed to be present in multiple submodels. Only selected transitions can appear on the interface between different submodels. By default, the visible transitions will be selected.

Select milestone transitions Using this list, the user can select the milestone transitions. When determining the equivalence classes between the arcs, only the non-selected transitions will be taken into account. By default, the visible transitions will be selected.

Select ILP threshold Using this slider, the user can influence for which submodels the ILP-based replay will be used. This replayer will be used for all submodels for which the number of transitions exceeds this threshold. The non-ILP-based replayer will be used otherwise. By default, this threshold is set to 8.

Select max loss Using this slider, the user can influence to what extend singe-input-single-output invisible transitions are to be reduced. Such a transition will be reduced if the product of its number of additional inputs and its number of additional outputs (that is, the number of new paths) is below this threshold. By default, this threshold is set to 0.

Select loss threshold Using this slider, the user can influence for which submodels single-input-single-output invisible transitions are to be reduced. These transitions are only reduced if the number of transitions in

the submodel exceeds this threshold. By default, this threshold is set to 100 (the maximal value), which means that these transitions will only be reduced for very large submodels.

Select Thread threshold Using this slider, the user can influence for which submodels the behavior of the source transitions will be restricted by adding a place. The submodel will only be restricted if the number of transitions in the submodel exceeds this threshold. By default, this threshold is set to 100 (the maximal value), which means that this restricting place will only be added for very large submodels.

Select Threads Using this slider, the user can influence the number of tokens in the place restricting the behavior of (otherwise) source transitions. The number of tokens equals the number set by this slider plus the number of outgoing arcs from this place to non-source transitions. By default, this number is set to 1.

Select Runs With this slider, the user can influence how many times the regular replay and decomposed replay are to be run. The regular replay and decomposed replay will be run as many times as indicated by this slider.

Enter label With this textbox, the user can provide a meaningful name to the selected parameter setting. This name will be used by ProM 6 to identify this parameters setting. By default, this name equals the name of the process model prefixed by "Parameters for ".

Show Replay Costs Using this plug-in, the user can run the regular replay and the decomposed replay as many times as indicated by the parameters that are required as input. Additional inputs are the process model (a Petri net) and the event log. This plug-in will result in an overview that contains the parameter settings (for sake of reference) and the results (both running times and costs for both the regular replay and the decomposed replay, including the running times and costs for the replay of every sublog on the corresponding submodel).

Create Decomposed Replay Problem Using this problem, the user can construct a decomposed replay problem. Required inputs are the parameters, the process model (a Petri net), and an event log. Please note that this plug-in only constructs the decomposed replay problem, it does not solve it by actually replaying it. To actually do this replay, the user can select the "Visualize Replay" visualization, which first performs the replay and then shows the results.

6 Concluding Remarks

In this paper, we have applied our decomposed replay techniques on six event logs and four corresponding process models. For two out of six event logs any decomposed replay we tried took longer than the regular replay. However, for exactly these two event logs the regular replay takes less than a second. The four remaining event logs all take longer than second to replay, where the actual time

Table 8. Result of (decomposed) replay for event logs that take longer than a second to replay

Event log	**Submodels**	**Running time** (in seconds)	**Costs**
A32F1N00	1	11	0.000
	4	(18%, ++) 1.9	(100%, ++) 0.000
A32F1N10	1	17	0.993
	4	(13%, ++) 2.1	(94%, +) 0.929
A32F1N50	1	32	4.521
	4	(10%, ++) 3.1	(96%, +) 4.322
BPIC2012	1	480	14.228
	4	(22%, +) 100	(82%, +) 11.722

needed varies from 11 seconds to 480 seconds. For all of these four event logs, we could achieve better running times at acceptable decreases in costs. Table 8 shows the result of the decomposed replay for these event logs, and compares these results to the results of the regular replay (where there is only a single submodel).

Although this shows that we can use decomposed replay to achieve better running times at acceptable costs, there is an issue with the actual decomposition. Especially the BPIC2012 event log has been a hard nut to crack, as the replay of this event log turns out to be very sensitive to the actual decomposition. The replay simply fails to finish if we decompose the process model and event log in as many submodels and sublogs as possible. Especially the replay of one of the sublogs on its corresponding submodel caused the replay to not finish.

In the paper, we have shown that we can take several approaches to this replay problem. First of all, we can restrict the possible behavior of the problematic submodel. This leads to a replay that finishes, but not to a costs that is by definition a lower bound for the actual costs, which is what we need. Second, we can reduce the single-input single-output invisible transitions such that none remains. As a result of this reduction, we possibly introduce new paths (new behaviors) in the resulting submodel, but this is safe as this only lowers the costs. We also tried to fine-tune this reduction of invisible transitions, but that did not help much: The costs did not improve more than the running time of the replay did increase. Third, we can aim for submodels that suit the actual replay techniques we are using in some way. Doing so leads to a decomposed replay that finishes, although it might decompose the model in a single submodel.

To be able to use the decomposed replay technique effectively in practice, we need to be able to come up with a good decomposition that decreases running times while keeping the estimated costs at an acceptable level. We have shown that even for the hard BPIC2012 log this was possible, but we haven't shown that we can do this for any log. For this, we need more research on what these

good decompositions are, how they look like, and how we can derive them from a given model (possibly, given the corresponding event log). As shown, for some logs, the default (*near-maximal*) decomposition works fine, but other logs require more sophisticated decompositions.

References

1. W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. 2011.
2. W.M.P. van der Aalst. Decomposing Process Mining Problems Using Passages. In S. Haddad and L. Pomello, editors, *Applications and Theory of Petri Nets 2012*, volume 7347, pages 72–91, 2012.
3. W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. ProM 4.0: Comprehensive Support for Real Process Analysis. In J. Kleijn and A. Yakovlev, editors, *Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007)*, volume 4546, pages 484–494, 2007.
4. A. Adriansyah, B. van Dongen, and W.M.P. van der Aalst. Conformance Checking using Cost-Based Fitness Analysis. In C.H. Chi and P. Johnson, editors, *IEEE International Enterprise Computing Conference (EDOC 2011)*, pages 55–64. IEEE Computer Society, 2011.
5. J. Carmona, J. Cortadella, and M. Kishinevsky. Divide-and-Conquer Strategies for Process Mining. In U. Dayal, J. Eder, J. Koehler, and H. Reijers, editors, *Business Process Management (BPM 2009)*, volume 5701, pages 327–343, 2009.
6. B. F. van Dongen. BPI challenge 2012. Dataset. http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f, 2012.
7. J. Munoz-Gama, J. Carmon, and W. M. P. van der Aalst. Hierarchical conformance checking of process models based on event logs. In *ATPN 2013*, LNCS. Springer, 2013. Accepted for publication.
8. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
9. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491, 1998.
10. A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008.
11. W. M. P. van der Aalst. Decomposing Petri Nets for Process Mining: A Generic Approach. Technical Report BPM-12-20, BPMcenter.org, 2012.
12. Wil van der Aalst, Arya Adriansyah, and Boudewijn van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.
13. H. M. W. Verbeek. BPI Challenge 2012: The Transition System Case. In M. La Rosa and P. Soffer, editors, *BPM 2012 Workshops*, volume 132 of *LNBIP*, pages 225–226. Springer, 2013.