Improving Documentation by Repairing Event Logs

Andreas Rogge-Solti, Ronny Mans, Wil Aalst, Mathias Weske

Improving Documentation by Repairing Event Logs

Andreas Rogge-Solti[1], Ronny S. Mans[2], Wil M. P. van der Aalst[2], and Mathias Weske[1]

[1] Hasso Plattner Institute, University of Potsdam
Prof.-Dr.-Helmert-Strasse 2-3, 14482 Potsdam
{andreas.rogge-solti,mathias.weske}@hpi.uni-potsdam.de
[2] Department of Information Systems, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{r.s.mans,w.m.p.v.d.aalst}@tue.nl

Summary. In enterprises, business process models are used for capturing as-is business processes. During process enactment correct documentation is important to ensure quality, to support compliance analysis, and to allow for correct accounting. Missing documentation of performed activities can be directly translated into lost income, if accounting is based on documentation. Still, many processes are manually documented in enterprises. As a result, activities might be missing from the documentation, even though they were performed.
In this paper, we make use of process knowledge captured in process models, and provide a method to repair missing entries in the logs. The repaired logs can be used for direct feedback to ensure correct documentation, i.e., participants can be asked to check, whether they forgot to document activities that should have happened according to the process models. We realize the repair by combining stochastic Petri nets, alignments, and Bayesian networks. We evaluate the results using both synthetic data and real event data from a Dutch hospital.

Key words: documentation quality, missing data, stochastic Petri nets, Bayesian networks

1 Introduction

Enterprises invest a lot of time and money to create business process models in order to use them for various purposes: documentation and understanding, improvement, conformance checking, performance analysis, etc. The modeling goal is often to capture the as-is processes as accurately as possible. In many cases, process activities are performed and documented manually. We call the documentation of activities in a business process *event logs*. When event logs are subject to manual logging, data quality problems are common, resulting in *incorrect* or *missing* events in the event logs [1]. We focus on the latter and more frequent issue, as in our experience it is often the case that activities are performed, but their documentation is missing.

For an enterprise, it is crucial to avoid these data quality issues in the first place. Accounting requires activities to be documented, as otherwise, if documentation is missing, potential revenues are lost. In the healthcare domain, for example, we encountered the case that sometimes the activity *preassessment* of a patient is not documented, although it is done always before treatment. In this paper, we provide a technique to

automatically repair an event log that contains missing entries. The idea is to use repaired event logs to alert process participants of potential documentation errors as soon as possible after process termination. We employ probabilistic models to derive the *most likely* timestamps of missing events, i.e., when the events should have occurred based on historical observations. This novel step assists process participants in correcting missing documentation directly, or to identify the responsible persons, who performed the activities in question.

State-of-the-art conformance checking methods [2] do not consider timing aspects. In contrast, we provide *most likely* timestamps of missing events. To achieve this, we use stochastically enriched process models, which we discover from event logs [3]. As a first step, using path probabilities, it is determined which are the most likely missing events. Next, Bayesian networks [4] capturing both initial beliefs of the as-is process and real observations are used to compute the *most likely* timestamp for each inserted event. Inserted events are marked as artificial, as long as they are not corrected by the process participants. An extended version of this paper is available as a technical report [5].

The remainder of this paper is organized as follows. First, we present background on missing data methods along other related works in Section 2. Afterwards, preliminaries are given in Section 3. Our approach for repairing individual traces in an event log is described in Section 4 followed by a presentation of the algorithmic details in Section 5. An evaluation of our approach using both synthetic and real-life event data is given in Section 6. Finally, conclusions are presented in Section 7.

2 Background and Related Work

Missing data has been investigated in statistics, but not in the context of conformance checking of business processes. There are different types of missing data: missing completely at random (MCAR), missing at random (MAR), and not missing at random (NMAR), cf. the overview by Schafer and Graham in [6]. These types refer to the independence assumptions between the fact that data is missing (*missingness*) and the data values of missing and observed data. MCAR is the strongest assumption, i.e., missingness is independent of both observed and missing data. MAR allows dependencies to observed data, and NMAR assumes no independence, i.e., captures cases where the missingness is influenced by the missing values, too. Dealing with NMAR data is problematic, as it requires a dedicated model for the dependency of missingness on the missing values, and is out of scope of this paper. We assume that data is MAR, i.e., whether data is missing does not depend on the value of the missing data, but may depend on observed data values.

Over the years, multiple methods have been proposed to deal with missing data, cf. [6]. However, these techniques are focusing on missing values in surveys and are not directly applicable to event logs, as they do not consider control flow relations in process models and usually assume a fixed number of observed variables.

Related work on missing data in process logs is scarce. Nevertheless, in a recent technical report, Bertoli et al. [7] propose a technique to reconstruct missing events in process logs. The authors tackle the problem by mapping control flow constraints in BPMN models to logical formulae and use a SAT-solver to find candidates for missing

events. In contrast, we use an alignment approach based on Petri nets, allowing us to deal with loops and probabilities of different paths. We also consider the time of the missing events, which allows performance analysis on a probabilistic basis.

Some techniques developed in the field of process mining provide functionality that enables analysis of noisy or missing event data. In process mining, the quality of the event logs is crucial for the usefulness of the analysis results and low quality poses a significant challenge to the algorithms [1]. Therefore, discovery algorithms which can deal with noise, e.g., the fuzzy miner [8], and the heuristics miner [9], have been developed. Their focus is on capturing the common and frequent behavior and abstract from any exceptional behavior. These discovery algorithms take the log as granted and do not try to repair missing events.

Another example is the alignment of traces in the context of conformance checking [2]. Here, the aim is to replay the event log within a given process model in order to quantify conformance by counting skipped and inserted model activities. We build upon this technique and extend it to capture path probabilities as gathered from historical observations. Note that the lion's share of work focuses on *repairing models* based on logs, rather than logs based on models. Examples are the work by Fahland and van der Aalst [10] that uses alignments to repair a process model to decrease inconcistency between model and log, and the work by Buijs et al. [11], which uses genetic mining to find similar models to a given original model.

3 Preliminary Definitions and Used Methods

In this section, we give a formal description of the used concepts, to describe the approach to the repair of missing values in process logs. We start with event logs and Petri nets.

Definition 1 (Event logs). *An event log over a set of activities A and time domain TD is defined as* $L_{A,TD} = (E, C, \alpha, \gamma, \beta, \succ)$*, where:*

- E *is a finite set of events.*
- C *is a finite set of cases (process instances).*
- $\alpha : E \to A$ *is a function relating each event to an activity.*
- $\gamma : E \to TD$ *is a function relating each event to a timestamp.*
- $\beta : E \to C$ *is a surjective function relating each event to a case.*
- $\succ \subseteq E \times E$ *is the succession relation, which imposes a total ordering on the events in E. We use* $e_2 \succ e_1$ *as shorthand notation for* $(e_2, e_1) \in \succ$*. We call the ordered set of events belonging to one case a "trace".*

Definition 2 (Petri Net). *A Petri net is a tuple* $PN = (P, T, F, M_0)$ *where:*

- P *is the set of places.*
- T *is the set of transitions.*
- $F \subseteq (P \times T) \cup (T \times P)$ *is the set of connecting arcs representing flow relations.*
- $M_0 \in P \to \mathbb{N}_0^+$ *is the initial marking.*

There have been many extensions of Petri nets to capture time, both deterministic and stochastic. In [12], Ciardo et al. give an overview of different classes. In terms of this classification, we use stochastic Petri nets with generally distributed transition durations.

Definition 3 (GDT_SPN). *A stochastic Petri net with generally distributed transition durations is a seven-tuple: GDT_SPN* $= (P, T, \mathcal{P}, \mathcal{W}, F, M_0, \mathcal{D})$*, where* (P, T, F, M_0) *is the underlying Petri net. Additionally:*

- *The set of transitions* $T = T_i \cup T_t$ *is partitioned into immediate transitions* T_i *and timed transitions* T_t*.*
- $\mathcal{P} : T \to \mathbb{N}_0^+$ *is an assignment of priorities to transitions, where* $\forall t \in T_i : \mathcal{P}(t) \geq 1$ *and* $\forall t \in T_t : \mathcal{P}(t) = 0$*.*
- $\mathcal{W} : T_i \to \mathbb{R}^+$ *assigns probabilistic weights to the immediate transitions.*
- $\mathcal{D} : T_t \to D(x)$ *is an assignment of arbitrary probability distribution functions* $D(x)$ *to timed transitions, capturing the random durations of the corresponding activities.*

Although this definition of GDT_SPN models allows us to assign arbitrary duration distributions to timed transitions, in this work, we assume normally distributed durations. Note that normal distributions are defined also in the negative domain, which we need to avoid. Therefore, we assume that most of their probability mass is in the positive domain, such that errors introduced by correction of negative durations are negligible.

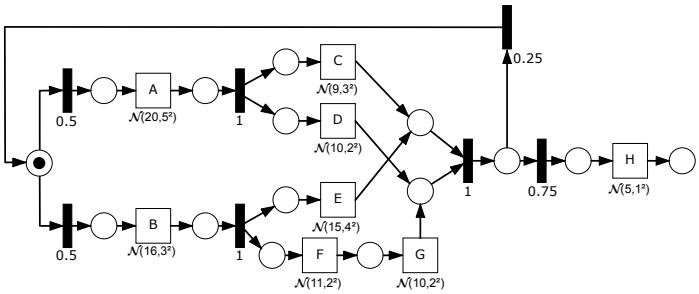

Fig. 1: Example unstructured free-choice GDT_SPN model.

An example GDT_SPN model is shown in Fig. 1 and has immediate transitions (bars), as well as timed transitions (boxes). In the figure, immediate transitions are annotated with their weights, e.g., the process will loop back with a probability of 0.25, and leave the loop with 0.75 probability. We omitted priorities and define priority 1 for all immediate transitions. The timed transitions are labeled from *A* to *H* and their durations are normally distributed with the parameters annotated underneath. In this example, activity *A*'s duration is normally distributed with a mean of 20, and a standard deviation of 5. Note that the model is sound and free-choice, and contains parallelism, choices, and a loop.

Because we allow generally distributed durations in the model, we require an execution policy [13]. We use *race semantics with enabling memory* as described in [13]. This means that concurrently enabled transitions race for the right to fire, and transitions will only be reset, if they get disabled by another transition firing.

For our purposes, we reuse the existing work in the ProM framework that extracts performance information of activities from an event log and enriches plain Petri nets to GDT_SPN models [3]. In [3], we discuss the challenges for discovering GDT_SPN models with respect to selected execution semantics of the model. The discovery algorithm uses replaying techniques, cf. [14], to gather historical performance characteristics and enriches a given Petri net to a GDT_SPN model with that performance information.

3.1 Cost-Based Fitness Alignment

(a) example log:

t_1 : ⟨A, C, D, B, E, F, G, H⟩
t_2 : ⟨E, G, H⟩

(b) alignment for trace t_1:

log	A	C	D	B	E	F	G	H
model	A	C	D	B	E	F	G	H

(c) alignments for trace t_2:

(c.1)

log	≫	E	≫	G	H
model	B	E	F	G	H

(c.2)

log	≫	≫	E	G	H
model	B	F	E	G	H

Fig. 2: Example log and possible alignments for the traces.

Consider the example log in Fig. 2a consisting of two traces t_1, and t_2. To check, whether the trace fits to the model, we need to align them. We reuse the technique described by Adriansyah et al. in [2], which results in a sequence of movements that *replay* the trace in the model. These movements are either *synchronous moves*, *model moves*, or *log moves*. A formal description of the alignment technique is provided in [2] and remains out of scope of this paper. We only give the intuition. For an alignment, the model and the log are replayed side by side to find the best mapping of events to activities in the model. Thereby, a *synchronous move* represents an event in the log that is allowed in the respective state in the model, such that both the model and the log progress one step together. However, if an activity in the model or an event in the log is observed with no counterpart, the model and log have to move asynchronously. Then, a *model move* represents an activity in the model, for which no event exists in the log at the current position and conversely, a *log move* is an event in the log that has no corresponding activity in the model that is enabled in the current state during replay. It is possible to assign costs to the different types of moves for each activity separately.

Fig. 2 shows some example alignments of the model in Fig. 1 and log in Fig. 2a. In Fig. 2b, a perfect alignment is depicted for trace t_1, i.e., the trace can be replayed completely by a sequence of *synchronous moves*. A closer look at trace t_2 and the model in Fig. 1 reveals that the two events *B*, and *F* are missing from the trace, which might have been caused by a documentation error. Because activity *F* is parallel to *E*, there exist two candidate alignments for t_2, as shown in Fig. 2c. The ≫ symbol denotes a step that is used to show empty moves, i.e., modeled and recorded behavior disagree. In this example, there are two model moves necessary to align the trace t_2 to the model.

Summarizing, the alignment technique described in [2, 14] can be used to find the cost-optimal matches between a trace in a log and a model. However, the approach only considers the *structure* of the model and the sequence of events encountered in the log without considering timestamps or probabilities. In this paper, we enhance the alignment technique to also take path probabilities into account.

3.2 Bayesian Networks

GDT_SPN models capture probabilistic information about the durations of each activity in the process. We use Bayesian networks [4, 15] to capture the dependencies between the random durations given by the GDT_SPN model structure. Fig. 3 shows an example Bayesian network that captures the relations for a part of the process model in Fig. 1. The arcs between activities *B*, *F*, and *G*, and between *B* and *E*, are sequential dependencies. Note that there is no direct dependency between *F* and *E*, since they are executed in parallel, and we assume that the durations of these activities are independent. More

generally, a Bayesian network is a directed acyclic graph and captures dependencies between random variables in a probabilistic model [15]. An arc from a parent node to a child node indicates that the child's probability distribution depends on the parent's values.

We use Bayesian networks to reason about our updated probabilistic beliefs, i.e., the posterior probability distributions in a model, once we assigned specific values to some of the random variables. Suppose that we observe trace t_2 in the log in Fig. 2a, with times $\gamma(E) = 30$, $\gamma(G) = 35$, and $\gamma(H) = 40$. Initially, the random variable of node B in the example has a duration distribution of $\mathcal{N}(16, 3^2)$, i.e., a normally distributed duration with mean 16, and standard deviation 3. However, after inserting the observed times of events E, and event G into the network in Fig. 3, we can calculate the resulting posterior probability distributions by performing *inference* in the Bayesian network. In this case, the posterior probability distribution of B is $\mathcal{N}(14.58, 1.83^2)$. Note that by inserting evidence, i.e., constraining the variables in a Bayesian network, the posterior probability distributions get more accurate. In this example, the standard deviation is reduced from 3 to 1.83. The intuition is that we narrow the possible values of the unobserved variables to be in accordance with the observations in the log. There exist algorithms for Bayesian networks automating this process [16]. A complete explanation of Bayesian networks, however, is not the aim in this paper, and the interested reader is referred to the original work by Pearl [4] and the more recent text book by Koller and Friedman [15].

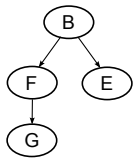

Fig. 3: Bayesian network for a fragment of Fig. 1.

4 Repairing Events in Timed Event Logs

In this paper, we propose a method to probabilistically restore events in logs which contain missing events. In particular, we are interested in knowing when things happened *most likely*. The problem that we try to solve is to identify the parts in the model that are missing from the trace (which) and also to estimate the times of the activities in those parts (when).

In theory, we need to compare the probabilities of all possible paths in the model that are conforming to the trace. Each path may allow for different assignments of events in the trace to the activities in the model. For example, for trace t_2: $\langle E, G, H \rangle$ and the model in Fig. 1 two cost-minimal paths through the model are given by the alignments in Fig. 2.c. But, there might be further possibilities. It is possible that a whole iteration of the loop happened in reality, but was not documented. In that case, the path $\langle B, E, F, G, A, C, D, H \rangle$ would also be an option to repair trace t_2. Furthermore, the second iteration could have taken another path in the model: $\langle B, E, F, G, B, F, E, G, H \rangle$. In this case it is not clear to which iteration the events E and G belong. In general, there are infinitely many possible traces for a model that contains loops.

In order to compare the probabilities of these paths, we need to compute the probability distributions of the activities on the paths and compare which model path and which assignment explains the observed events' timestamps best. To reduce the complexity, we propose to decompose the problem into two separate problems, i) repair structure

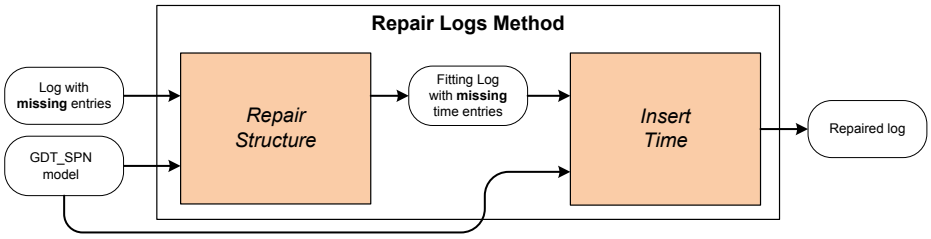

Fig. 4: We divide the problem into two subproblems: repairing the control flow, and repairing the timestamps.

and ii) insert time, as sketched in Fig. 4. The method uses as input a log that should be repaired and a GDT_SPN model specifying the as-is process.

Note that by choosing this approach, we accept the limitation that missing events on a path can only be detected, if at least one event in the log indicates that the path was chosen.

5 Realization of Repairing Logs

In this section, we explain a realization of the method described above. For this realization, we make the following assumptions:

- The supported models, i.e., the GDT_SPN models, are *sound*, cf. [17], and *free-choice*, cf. [18], but do not necessarily need to be (block-)structured. This class of models captures a large class of process models and does not impose unnecessary constraints.
- The GDT_SPN model is normative, i.e., it reflects the as-is processes in structural, behavioral and time dimension.
- Activity durations are independent and have normal probability distributions, containing most of their probability mass in the positive domain.
- The recorded timestamps in the event logs are correct.
- Each trace in the log has at least one event, and all events contain a timestamp.
- The activity durations of a case do not depend on other cases, i.e., we do not consider the resource perspective and there is no queuing.
- We assume that data is MAR, i.e., that the probability that an event is missing from the log does not depend on the time values of the missing events.

The algorithm is depicted in Fig. 5, and repairs an event log as follows.

For each trace, we start by repairing the structure. This becomes trivial, once we identified a path in the model that fits our observations in the trace best. The notion of cost-based alignments [2] that we introduced in Section 3, is used for this part. It tells us exactly:

a) when the model moves *synchronously* to the trace, i.e., where the events match
b) when the *model moves* alone, i.e., an event is missing from the trace
c) when the *log moves* alone, i.e., there is an observed event that does not fit into the model at the recorded position

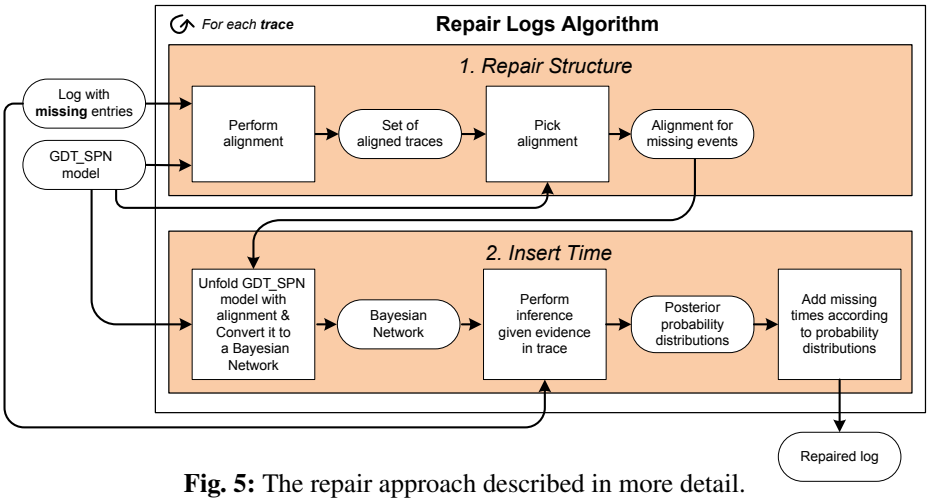

Fig. 5: The repair approach described in more detail.

We set the costs of synchronous and model moves to 0, and the cost of log moves to a high value, e.g., 1000. The alignment algorithm returns all paths through the model, where the events in the trace are mapped to a corresponding activity. This works well for acyclic models. For cyclic models, where infinite paths through a model exist, we need to assign some small costs to model moves, in order to limit the number of resulting alignments that we compare in the next step.

In the next step, cf. box *Pick alignment* in Fig. 5, we decide which of the returned cost-minimal alignments to pick for repair. The algorithm replays the path taken through the model and multiplies the probabilities of the decisions made along the path. This allows us to take probabilistic information into account when picking an alignment and enhances the alignment approach introduced in [2]. We also consider that, for one trace, paths with many forgotten activities are less likely than others. That is, we allow to specify the parameter of the missing data mechanism, i.e., the rate of missingness. We let the domain expert define the probability to forget an event. The domain expert can specify how to weigh these probabilities against each other, i.e., to give preference to paths with higher probability, i.e., determined by immediate transition weights, or to paths with less missing events that are required to be inserted into the trace. This novel post-processing step on the cost-optimal alignments allows to control the probability of paths in the model that are not reflected in a log by any event.

For example, consider a loop in a GDT_SPN model with n activities in the loop. By setting the chance of missing entries low, e.g., setting the missingness probability to 0.1 (10% chance that an event is lost), an additional iteration through the loop will become more unlikely, as its probability will be multiplied by the factor 0.1^n. This factor is the probability that all n events of an iteration are missing. We select the alignment with the highest probability. Once we decided on the structure of the repaired trace, we can continue and insert the times of the missing events in the trace, i.e., the identified *model moves*.

To insert the timing information, it is not enough to look at the GDT_SPN model alone. We need to find a way to add the information that we have for each trace, i.e., the

timestamps of the recorded events. Fortunately, as mentioned in Section 3, there exists a solution for this task: *Inference* in Bayesian networks. Therefore, we convert the GDT _SPN model into a Bayesian network to insert the evidence given by the observations to be able to perform the inference.

In the previous step, we identified a probable path through the GDT_SPN model. With the path given, we eliminate choices from the model by removing branches of the process model that were not taken. We unfold the net from the initial marking along the chosen path. Consider trace $t_3 = \langle A, D, C, C, D, H \rangle$ and assume, we picked the following alignment:

log	A	D	C	$\perp$	C	D	H
model	A	D	C	A	C	D	H

Then, the unfolded model looks like Fig. 6, where the black part marks the path taken in the model. The grey part is removed while unfolding. Note that the unfolded model still contains parallelism, but it is acyclic. Thus, we can convert it into a Bayesian network with a similar structure, where the random variables represent timed transitions. As, due to multiple iterations of loops, activities can happen multiple times, we differentiate them by adding an index of their occurrence, e.g., $A1$ and $A2$ correspond to the first and second occurrence of the transition A. The unfolding is done by traversing the model along the path dictated by the alignment and keeping track of the occurrences of the transitions.

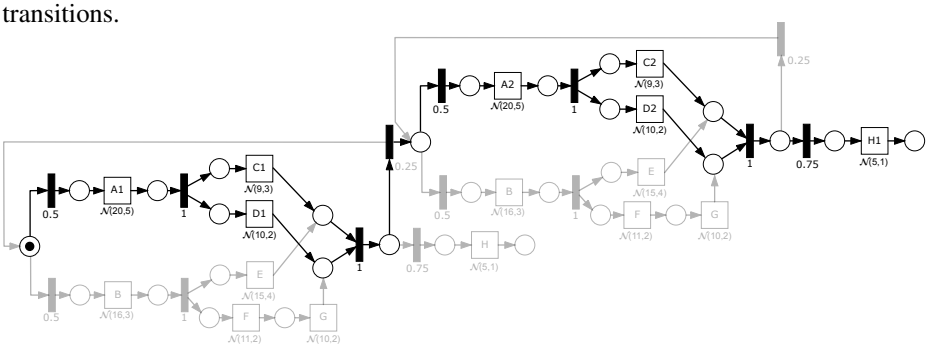

Fig. 6: Unfolded model in Fig. 1 for path $\langle A, D, C, A, C, D, H \rangle$.

We transform the unfolded model into a Bayesian network with a similar structure. Most immediate transitions are not needed in the Bayesian network, as these do not take time and no choices need to be made in the unfolded process. Only immediate transitions joining parallel branches will be kept.

Fig. 7 shows transformation patterns for sequences, parallel splits, and synchronizing joins. These are the only constructs remaining in the unfolded form of the GDT_SPN model. In the resulting Bayesian network, we use the *sum* and *max* relations to define the random variables given their parents. More concretely, if timed transition t_i is followed by timed transition t_j in a sequence, we can convert this fragment into a Bayesian network with variables X_i and X_j. From the GDT_SPN model, we use the transition duration distributions $\mathcal{D}(t_i) = D_i(x)$ and $\mathcal{D}(t_j) = D_j(x)$. Then, the parent variable X_i has the unconditional probability distribution $\mathrm{P}(X_i \leq x) = D_i(x)$ and the child variable X_j has

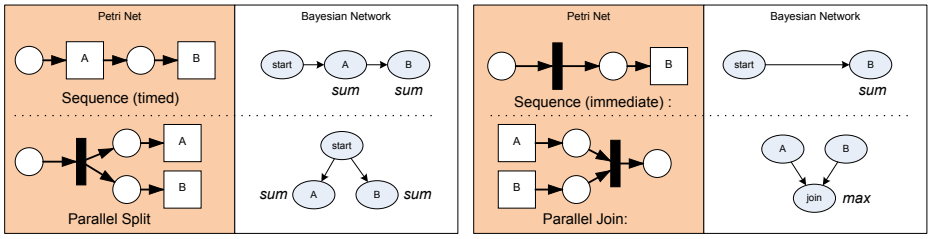

Fig. 7: Transformation of GDT_SPN models to Bayesian networks.

the probability distribution $\mathrm{P}(X_j \leq x \mid X_i) = \mathrm{P}(X_j + X_i \leq x)$. For each value of the parent $x_i \in X_i$, the probability distribution is defined as $\mathrm{P}(X_j \leq x \mid X_i = x_i) = D_j(x - x_i)$, i.e., the distribution of X_j is shifted by its parent's value to the right. A parallel split, cf. lower left part in Fig. 7, is treated as two sequences sharing the same parent node.

The *max* relation that is required for joining branches at synchronization points, cf. the lower right pattern in Fig. 7, is defined as follows. Let X_i and X_j be the parents of X_k, such that X_k is the maximum of its parents. Then, $\mathrm{P}(X_k \leq x \mid X_i, X_j) = \mathrm{P}(\max(X_i, X_j) \leq x) = \mathrm{P}(X_i \leq x) \cdot \mathrm{P}(X_j \leq x) = D_i(x) \cdot D_j(x)$, i.e., the probability distribution functions are multiplied. Note that the maximum of two normally distributed random variables is not normally distributed. Therefore, we use a linear approximation, as described in [19]. This means that we express the maximum as a normal distribution, with its parameters depending linearly on the normal distributions of the joined branches. The approximation is good, when the standard deviations of the joined distributions are similar, and degrades when they differ, cf. [19]. The resulting Bayesian network model is a linear Gaussian model, which is a class of continuous type Bayesian networks, where inference is efficiently possible. More precisely, inference can be done in $\mathrm{O}(n^3)$ where n is the number of nodes [15]. Otherwise, inference in Bayesian networks is an NP-hard problem [20].

Once we constructed the Bayesian network, we set the values for the observed events for their corresponding random variables, i.e., we insert the evidence into the network. Then, we perform inference in the form of querying the posterior probability distributions of the unobserved variables. We use the Bayesian network toolkit for Matlab [16], where these inference methods are implemented. This corresponds to the second step in the *insert time* part of Fig. 5.

The posterior probabilities of the queried variables reflect the probabilities, when the conditions are given according to the evidence. Our aim is to get the *most likely* time values for the missing events. These most likely times are good estimators for when the events occurred in reality, and thus can be used by process participants as clues during root cause analysis. For example, in order to find the responsible person for the task in question, an estimation of when it happened *most likely* can be helpful. Note that repaired values with most likely time values need to be treated with caution, as they do not capture the uncertainty in the values. Therefore, we mark repaired entries in the log as artificial.

Once we determined probable values for the timestamps of all missing events in a trace, we can proceed with the next trace starting another iteration of the algorithm.

6 Evaluation

We have implemented our approach in ProM[1]. To evaluate the quality of the algorithm, we follow the experimental setup described in Fig. 8. The problem is that in reality we do not know whether events did not happen, or only were not recorded. Therefore, we conduct a controlled experiment. In order to have actual values to compare our repaired results with, we first acquire traces that fit the model. We do this either by selecting the fitting ones from original cases, or by simulation in artificial scenarios. In a second step, we randomly remove a percentage of the events from these fitting traces. We pass the log with missing entries to the repair algorithm, along with the model, according to which we perform the repair.

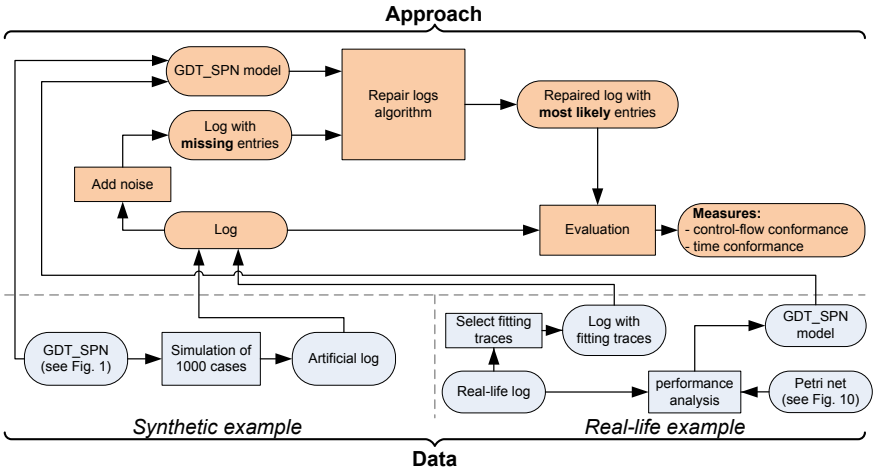

Fig. 8: Approach used to evaluate repair quality.

The repair algorithm's output is then evaluated against the original traces to see, how well we could restore the missing events. We use two measures for assessing the quality of the repaired log. The cost-based *fitness* measure as defined in [2] compares how well a model fits a log. Here, we compare the traces of the original and repaired log. Therefore, we convert each original trace into a sequential Petri net model and measure its fitness with the repaired trace.

Fitness deals with the structural quality, i.e., it is a good measure to check, whether we repaired the right events in the right order. For measuring the quality of repaired timestamps, we compare the real event's time with the repaired event's time. We use the mean absolute error (MAE) of the events that have been inserted. This is the mean of the absolute differences between repaired event times and original event times.

6.1 Artificial Example

We first evaluate the repair algorithm according to the artificial model introduced in Section 3 in Fig. 1.

[1] See package *RepairLog* in ProM `http://www.promtools.org`

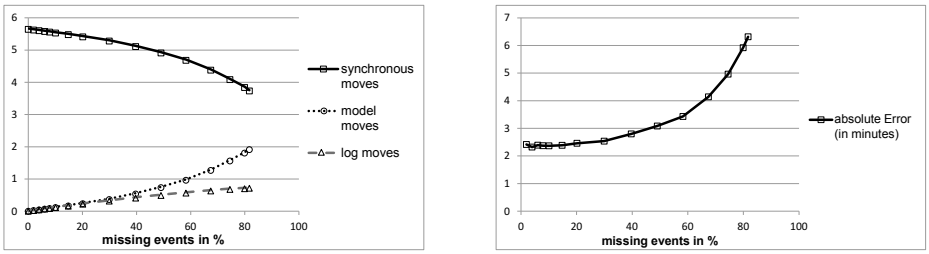

Fig. 9: Evaluation results for repairing 1000 traces of model in Fig. 1.

The experiment was done with a log of 1000 simulated traces. Figure 9 displays the resulting quality measures of the repaired traces. Each dot is based on the repair results of this log with a different percentage of randomly removed events. On the left-hand side of the figure, you can see the performance values of the alignment. The solid line with squares shows the number of *synchronous moves*. The other two lines are the number of model moves (dotted line with circles) and the number of log moves (gray dashed line with triangles) necessary to align the two traces.

Because of the structural properties of the model in Fig. 1, i.e., there is a choice between two branches containing three (upper), and four (lower) activities, we can restore the *correct* activities at low noise levels (around 30%). But we can not guarantee for their ordering due to parallelism in the model. A change in the ordering of two events in the repaired trace results in a *synchronous move* for one event, and a *log move* and a *model move* for the other (to remove it from one position and insert it in another). Note that at lower noise levels the number of *log moves* and *model moves* are equal. This indicates incorrect ordering of parallel activities. At higher noise levels the number of model moves increase further. Then, it gets more likely that there remains no single event of an iteration of the loop in Fig. 1. The size of the gap between model moves and log moves shows how much the repair quality suffers from the fact that the presented algorithm, which repairs events with the most likely values, does not restore optional paths of which no event is recorded in the trace.

On the right-hand side of Fig. 9 we see the mean absolute error in relative time units specified in the model. The graph shows that the offset between original event's time and repaired event's time increases with the amount of noise non-linearly.

6.2 Repairing a real example log of a hospital

In this second part of the evaluation, we look at the results obtained from repairing a real log of a hospital. In contrast to the experimental setup, where we used the model to generate the example log, now the log is given, and we try to estimate the model parameters. To avoid using a model that was learned from the events, which we try to repair, we use 10-fold cross-validation. That is, we divide the log into ten parts and use nine parts to learn the model parameters and one to perform the repair with.

We use the log of a Dutch clinic for the ambulant surgery process, described in [21]. The process is depicted as a GDT_SPN model in Fig. 10. It is a sequential process

that deals with both ambulant patients and ordered stationary patients. Each transition corresponds to a treatment step that a nurse records in a spread sheet with timestamps. In the process, the patient arrives in the lock to be prepared for the surgery. Once the operating room (OR) is ready, the patient leaves the lock and enters the OR. In the OR, the anesthesia team starts the induction of the anesthesia. Afterwards, the patient optionally gets an antibiotica prophylaxis treatment. The surgery starts with the incision, i.e., the first cut with the scalpel, and finishes with the suture, i.e., the closure of the tissue with stitches. Next, the anesthesia team performs the emergence from the anesthesia, which ends when the patient has regained consciousness. Finally, the patient leaves the OR and is transported to the recovery.

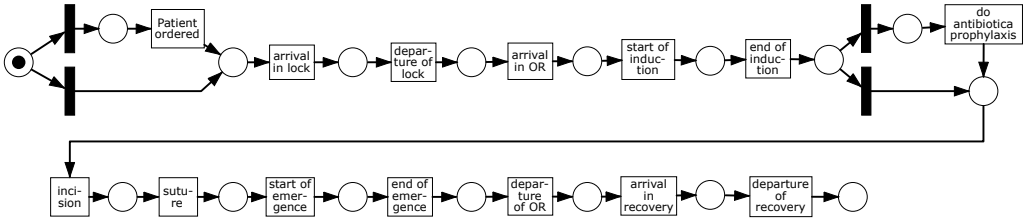

Fig. 10: Real surgery model for a surgical procedure in a Dutch hospital.

Out of 1310 patient treatment cases, only 570 fit the model shown in Fig. 10 perfectly. The other cases contain one or more missing events, which motivated our research. We use the 570 fitting cases to evaluate, how well we can repair them after randomly removing events.

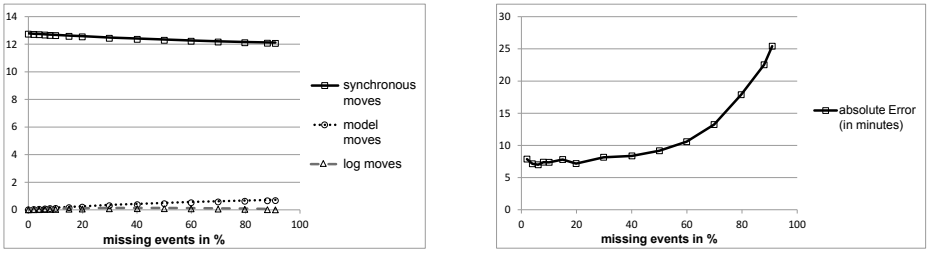

Fig. 11: Evaluation results for model in Fig. 10.

Figure 11 shows the evaluation results of the hospital event log. Observe that the structure can be repaired better than in the artificial example in Fig. 9. This is due to the sequential nature of the model—it comprises twelve sequential, and two optional activities. With increasing number of missing events, the number of correctly repaired events (synchronous moves) approaches twelve. That is, only twelve activities are restored, because the algorithm is unable to repair single undetected optional events.

The mean absolute error in the restored events is higher than the artificial example. This value depends on the variance in the activity durations. In this evaluated example, the variance of certain activity durations in the model is high, due to outliers. Latter activity durations exhibit many short durations with a few outliers, which can be better captured with other distributions than the normal distribution.

Obviously, the ability to repair a log depends on the information content of observed events in the trace and the remaining variability in the model. For instance, we can repair a sequential model always with fitness 1.0 of the repaired log—if we observe only one activity. However, the chance to pick the same path through a model composed of n parallel activities with equally distributed times is only $\frac{1}{n!}$.

The presented approach is unable to restore optional branches without structural hints, i.e., at least one activity on an optional branch needs to be recorded. This affects single optional activities most, as their absence will not be repaired. Still, many real-life processes comprise only a sequence of activities, and can be repaired correctly.

7 Conclusion

We introduced a method to repair event logs to assist timely correction of documentation errors in enterprises. Thereby, we present *which*, and also *when* activities should have happened *most likely* according to a given stochastic model. The method decomposes the problem into two sub-problems: i) repairing the structure, and ii) repairing the time.

Repairing the structure is done with a novel extension of the alignment approach [2] based on path probabilities. And repairing the time is achieved by using inference in a Bayesian network representing the structure of the individual trace in the model. The algorithm can deal with a large and representative class of process models (any sound, free-choice workflow net).

Our preliminary evaluations indicate that we can repair structure and time, if noise is limited. Models exhibiting a high degree of parallelism are less likely to be repaired in correct order than models with more dependencies between activities. However, there are some limitations that we would like to address in subsequent research:

1. Separating structure from time during repair is a heuristic to reduce the computational complexity of the problem, as timestamps of events also influence path probabilities.
2. The normal distribution, though having nice computational properties, is of limited suitability to model activity durations, since its support also covers the negative domain.
3. The independence assumption between activity durations and between traces might be too strong, as resources play an important role in processes.
4. We assumed that the GDT_SPN model contains the truth, and deviations in the log are caused by documentation errors, instead of deviations from the process model. This assumption only is feasible for standardized processes with few deviations that are captured in the model. Therefore, we advise to use this approach with care and try to correct documentation errors using repaired logs as assistance.

Future work also needs to address the question of how to model causalities of activities more directly. Thus, missing events that are very likely to be documentation errors, e.g., the missing event for *enter OR*, when *exit OR* is documented, need to be separately treated from missing events of rather optional activities, e.g., missing event of *do antibiotica prophelaxe*, where it is not clear, whether the absence of the event is caused by a documentation error. An integration with the proposed technique in [7], seems promising to address this issue.

References

1. IEEE Task Force on Process Mining: Process Mining Manifesto. In: BPM Workshops. Volume 99 of LNBIP., Springer (2012) 169–194
2. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking using Cost-Based Fitness Analysis. In: EDOC 2011, IEEE (2011) 55–64
3. Rogge-Solti, A., van der Aalst, W.M.P., Weske, M.: Discovering Stochastic Petri Nets with Arbitrary Delay Distributions From Event Logs. BPM Workshops. Springer (to appear)
4. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)
5. Rogge-Solti, A., Mans, R., van der Aalst, W.M.P., Weske, M.: Repairing Event Logs Using Stochastic Process Models. Technical Report 78, Hasso Plattner Institute (2013)
6. Schafer, J.L., Graham, J.W.: Missing Data: Our View of the State of the Art. Psychological methods **7**(2) (2002) 147–177
7. Bertoli, P., Dragoni, M., Ghidini, C., Di Francescomarino, C.: Reasoning-based Techniques for Dealing with Incomplete Business Process Execution Traces. Technical report, Fundazione Bruno Kessler, Data & Knowledge Management (2013)
8. Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In: BPM. Volume 4714 of LNCS., Springer (2007) 328–343
9. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
10. Fahland, D., van der Aalst, W.M.P.: Repairing Process Models to Reflect Reality. In: BPM. Volume 7481 of LNCS., Springer (2012) 229–245
11. Buijs, J.C.A.M., La Rosa, M., Reijers, H.A., van Dongen, B.F., van der Aalst, W.M.P.: Improving Business Process Models using Observed Behavior. In: SIMPDA 2012. Volume 162 of LNBIP., Springer (2013) 44–59
12. Ciardo, G., German, R., Lindemann, C.: A Characterization of the Stochastic Process Underlying a Stochastic Petri Net. IEEE Transactions on Software Engineering **20**(7) (1994) 506–515
13. Marsan, M.A., Balbo, G., Bobbio, A., Chiola, G., Conte, G., Cumani, A.: The Effect of Execution Policies on the Semantics and Analysis of Stochastic Petri Nets. IEEE Transactions on Software Engineering **15** (1989) 832–846
14. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.: Replaying History on Process Models for Conformance Checking and Performance Analysis. In: WIREs: Data Mining and Knowledge Discovery. Volume 2., Wiley Online Library (2012) 182–192
15. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT press (2009)
16. Murphy, K.P.: The Bayes Net Toolbox for Matlab. In: Interface'01. Volume 33 of Computing Science and Statistics. (2001) 1024–1034
17. van der Aalst, W.M.P.: Verification of Workflow Nets. In: ICATPN'97. Volume 1248 of LNCS., Springer (1997) 407–426
18. Best, E.: Structure Theory of Petri Nets: The Free Choice Hiatus. In: Petri Nets: Central Models and Their Properties. Volume 254 of LNCS., Springer (1987) 168–205
19. Zhang, L., Chen, W., Hu, Y., Chen, C.C.: Statistical Static Timing Analysis With Conditional Linear MAX/MIN Approximation and Extended Canonical Timing Model. In: TCAD. Volume 25., IEEE (2006) 1183–1191
20. Cooper, G.F.: The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks. Artificial intelligence **42**(2) (1990) 393–405
21. Kirchner, K., Herzberg, N., Rogge-Solti, A., Weske, M.: Embedding Conformance Checking in a Process Intelligence System in Hospital Environments. In: Process Support and Knowledge Representation in Health Care, Springer (2013) 126–139