Repairing Event Logs Using Stochastic Process Models

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

[1] Hasso Plattner Institute at the 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

Abstract. Companies strive to improve their business processes in order to remain competitive. Process mining aims to infer meaningful insights from process-related data and attracted the attention of practitioners, tool-vendors, and researchers in recent years. Traditionally, event logs are assumed to describe the as-is situation. But this is not necessarily the case in environments where logging may be compromised due to manual logging. For example, hospital staff may need to manually enter information regarding the patient's treatment. As a result, events or timestamps may be missing or incorrect.
In this report, we make use of process knowledge captured in process models, and provide a method to repair missing events in the logs. This way, we facilitate analysis of incomplete logs. 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.

Keywords: process mining, missing data, stochastic Petri nets, Bayesian networks

1 Introduction

Many information systems record detailed information concerning the processes they support. Typically, the start and completion of process activities together with related context data, e.g., actors and resources, are recorded. In business process management, such event data can be gathered into logs. Subsequently, these logs can be analyzed to gain insights into the *performance* of a process. In many cases, information systems do not force the process participants to perform tasks according to rigid paths, as specified by process models. Rather, the process participants are responsible to track their manual work which is sometimes not reflected in the system. In other words, the event logs might be *incomplete* or noisy [1]. These data quality issues affect process mining methods and often lead to unsatisfactory results.

One example can be found in a clinical setting describing a surgery process, which is performed in a sequence of tasks, e.g., induce coma, enter operating room (OR), perform surgery, leave OR, etc. These tasks are depending on each other causally, e.g., it is impossible for a patient to leave the OR, without having entered it before.

Consider the case, where for each activity in this process, an event is documented in the system. However, due to some reason, the event entry for *enter OR* is missing from the documentation. If we observe that there is an entry for the patient to leave the OR, we can safely conclude that the patient must have also entered it at some point before, since these activities depend on each other. In this case, it is likely that a documentation error occurred and it might be of interest, when the patient entered the OR *most likely* to aid with root cause analysis of the documentation error.

Typically, analysis methods do not take *missing data* in process oriented systems into account. Matters are complicated by several potential reasons that cause data to be missing. One reason might be that documentation was forgotten, or lost, but in reality the activities happened. Another reason might be that the activity was not executed. In general, it is not easy to differentiate these cases, and the usual assumption is that the event logs contain the truth, i.e., only the recorded activities happened in reality. However, as the example in the surgery process shows, this is not necessarily the case. Depending on the domain, it can be reasonable to assume that the models contain the truth, and the documentation is faulty.

Existing approaches can be used to *repair* the model based on event data [2]. However, if steps are recorded manually this may lead to misleading results as little weight is given to a priori domain knowledge. Therefore, we adopt a stochastic approach to modeling process behavior and introduce a novel approach to *repair event logs* according to a given stochastically enriched process model [3]. We discuss different ways to deal with the issue of incomplete event data and propose a probabilistic approach for inserting missing entries into an event log based on a given as-is process. To model the as-is process we use Petri nets enhanced with timing information and path probabilities.

In fact, we use a variant of the well-known Generalized Stochastic Petri nets (GSPNs) defined in [4]. As a first step, using path probabilities, it is determined which are the most likely missing events. Next, Bayesian networks [5] capturing both initial beliefs of the as-is process and real observations are used to compute the most likely timestamp for each inserted entry.

The repaired event logs can be used in further analysis, e.g., to assist in locating responsible persons to determine the reason of the missing entry, or in further process mining methods assuming complete event logs. Note that the uncertainty in the restored events needs to be accounted for in the latter case. To our knowledge, this is the first work towards using statistical techniques for missing data in the BPM/process mining domain.

The remainder of this report 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 process mining. Statisticians collect surveys from samples of a population to make inferences about parameters of the whole population. Often, values are missing from these surveys due to a number of reasons. Consider a survey asking people for their income. In this survey people might not respond, if they do not understand the question due to language reasons, or they might not want to respond if their income is very high. Thus, when dealing with missing data, it is important to know the mechanisms causing such phenomena.

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. The example survey about the income deals with data that is NMAR, as people with higher income are more likely not to respond. 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 report. 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.

2.1 How to Deal with Missing Data?

Since usual statistical methods assume data to be complete, the easiest method to deal with missing data is to only use the subset of whole data that has all entries present. This is called *listwise deletion*. The main merit in using listwise deletion is that it is simple and honest. If the data is MCAR, and only few entries in the data have to be discarded due to missing events, this is a good solution. However, if a large portion of the samples have to be discarded, the efficiency of the estimate suffers from this technique. Furthermore, if the data is not MCAR, but MAR, this method fails to take dependencies into account and produces biased parameter estimates.

More sophisticated methods that are able to deal with data that is MAR, and are recommended as state of the art [6], are *maximum likelihood* estimation, and Bayesian *multiple imputation*. Both these methods are efficient, i.e., they make use of all observed data to increase confidence in the estimations, and produce unbiased estimators in the MAR case. The seminal work by Dempster et al. [7] describes the EM algorithm that finds the *maximum likelihood* of the parameters by iterating an estimation and maximization step until convergence. Later, Rubin introduced the idea to replace missing values with multiple simulated values (imputations) and average results of each data set in order to account for the variability of the missing data [8]. The technical details of these techniques remain out of scope of this report, and the interested reader is referred to the book by Little and Rubin [9] on this topic.

Over the years, further methods have been proposed to perform missing data imputation, 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.

2.2 Missing and Noisy Data in Business Process Event Logs

Related work on missing data in process logs is scarce. Nevertheless, in a recent technical report, Bertoli et al. [10] 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, and 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 [11] and the heuristics miner [12], 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 [13]. 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 v.d. Aalst [2] that uses alignments to repair a process model to decrease inconcistency between model and log and the work by Buijs et al. [14], 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 concepts used in order to describe the approach for 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),*
- $\alpha : E \rightarrow A$ *is a function relating each event to an activity,*
- $\gamma : E \rightarrow TD$ *is a function relating each event to a timestamp,*
- $\beta : E \rightarrow C$ *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 [15], Ciardo et al. give an overview of different classes. We use the definition of Generalized Stochastic Petri Nets (GSPNs) provided in [4], but do not limit the durations of the timed transitions to be exponentially distributed. In terms of the categorization proposed in [15], we use SPN with generally distributed firing times. In the remainder of the report, we simply refer to this class as SPN.

Definition 3 (SPN). *A stochastic Petri Net with generally distributed firing times (SPN) is a seven-tuple: $SPN = (P, T, \mathcal{P}, \mathcal{W}, F, M_0, \mathcal{D})$, where (P, T, F, M_0) is the basic 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 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.

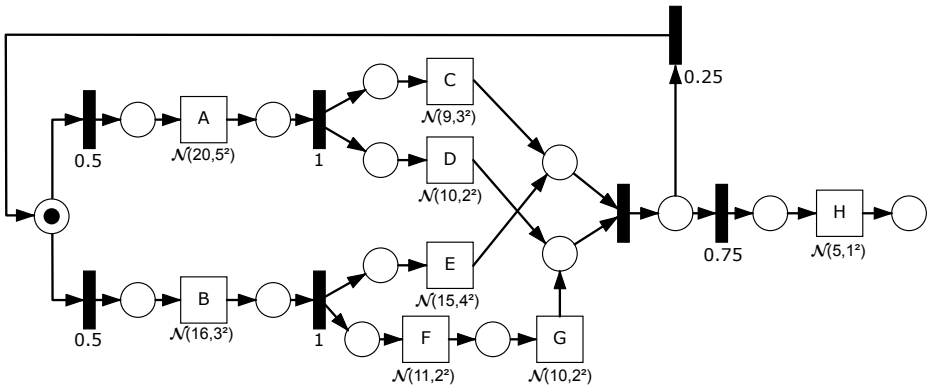

Fig. 1: Example unstructured free-choice Petri net model.

An example 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 and a standard deviation of 5. Note that the model is sound and free-choice, and contains parallelism, choices, and a loop.

We use the *race semantics with enabling memory* as described in [16]. This means that concurrently enabled transitions race for the right to fire, and transitions will be reset only if they get disabled by the firing of another transition. Concurrent transitions that remain enabled after a transition firing do not lose their progress.

In the remainder of this section, we introduce techniques used in our repair algorithm: Elicitation techniques to get an SPN model, cost-based alignments between a Petri net and a log, and inference in a Bayesian network.

3.1 Elicitation of SPN models

There exists already work on obtaining stochastic Petri net models from data. Hu et al. propose a method to mine these models from workflow logs in [17]. Another, quite different approach was proposed by Anastasiou et al. [18] and uses location data to elicit GSPN models.

For our purposes, we reuse the existing work in the ProM framework that extracts performance information of activities from the log [19] and enrich the modeled Petri nets with that performance information [3]. First, we need to elicit the probabilities to follow different paths in the model, and second, the duration distributions of timed transitions. For decisions in the model, we look at the relative number of traces following either branch in the model and store them as weights of the immediate transitions. Transition duration distributions are specified by the mean and the variance of the differences between event times in the observed traces that correspond to subsequent activities.

If no data is missing from the logs, it is straight forward to compute the differences between timestamps of subsequent events using the parallelism information encoded in the Petri net model and count relative decisions on alternative paths. However, when facing missing data, the presented approach in this report can be used to estimate parameters of the model iteratively. That procedure is similar to the EM algorithm for maximum likelihood estimation. In this report, we assume the model is given and present one iteration of the algorithm to focus on how we can repair missing entries according to the given model.

3.2 Cost-Based Fitness Alignment

Consider the example log in Fig. 2.a) consisting of two traces t_1, and t_2. In order to check, whether the trace fits to the model, we need to align both. We reuse the technique described by Adriansyah et al. in [13], which results in a sequence of movements that *replay* the trace in the model. These movements are either *synchronous moves*, *model*

(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.

moves, or *log moves*. A formal description of the alignment technique remains out of scope of this report, but we want to 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 and in this case both the model and the log progress one step. 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 similarly, 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 movements for each activity separately.

Fig. 2 shows some example alignments between the model in Fig. 1 and log in Fig. 2.a. In Fig. 2.b), a perfect alignment is depicted for t_1 above, 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. As activity *F* is parallel to *E*, there exist two candidate alignments for t_2, as shown in Fig. 2 c). The ⊥ symbol denotes a step that is used to show non-synchronous 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 [13,19] can be used to find the cost-optimal matches between a trace in a log and a model in terms of structure. 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 report, we enhance the alignment technique to also take path probabilities into account.

3.3 Bayesian Networks

Our SPN model also captures probabilistic information about the durations of each activity in the process. We use Bayesian networks [5,20] to capture the dependencies between the random durations given by the SPN model structure. Fig. 3 shows an example Bayesian network that captures the relations for a fraction of the process model in Fig. 1. The arcs between activities *B*, *F*, and *G*, and between *B* and *E*, are

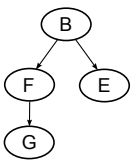

Fig. 3: Bayesian network for a fragment of Fig. 1.

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 [20]. An arc from a parent node to a child node indicates that the child's probability distribution depends on the parent's values. There are many useful applications for probabilistic graphical models, e.g. classification, voice-recognition, spam filtering, and others.

Definition 4 (Bayesian Network). *Let $X_1, \ldots, X_N$ be a set of random variables. A Bayesian network BN is a directed acyclic graph (N, F), where N is the set of nodes $n_1, n_2, \ldots, n_k$ assigned each to a random variable $X_1, \ldots, X_k$, and $F \subset N \times N$ is the set of directed edges. Let $(n_i, n_j) \in F$ be an edge from parent node n_i to child node n_j. The edges reflect conditional dependencies between the corresponding random variables X_i and X_j, s.t., each random variable is independent from its predecessors given the values of its parents.*

Let π_i denote the set of parents of X_i. A Bayesian network is fully defined by the probability distributions of the nodes n_i in as $\mathrm{P}(X_i \mid \pi_i)$ and the conditional dependency relations encoded in the graph. Then, the joint probability distribution of the whole network factorizes according to the chain rule as follows:
$\mathrm{P}(X_1, \ldots, X_N) = \prod_{i=1}^{N} \mathrm{P}(X_i \mid \pi_i)$

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. Assume that we observe trace t_2 in the log in Fig. 2.a, with times $\gamma(E) = 30$, $\gamma(G) = 35$, and $\gamma(H) = 40$. Initially, the random variable of node B in the example had 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 of B is the normal distribution $\mathcal{N}(14.58, 1.83^2)$. Note that by inserting evidence, i.e., constraining the variables in a Bayesian network, the resulting 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. A complete explanation of Bayesian networks is not the aim in this report, and the interested reader is referred to the original work by Pearl [5] and the more recent text book by Koller and Friedman [20].

4 Repairing Events in Timed Event Logs

In this report, we propose a method to probabilistically restore events in logs which contain missing events. We allow two modes of repair: restoring the *most likely* events and their corresponding times, and a *Monte Carlo simulation* repair mode that picks alignments and times randomly according to their posterior probabilities. Latter mode is a useful component in a multiple imputation approach and allows to reason about

variability in process measures involving missing events. We will focus on the first mode in this report and discuss differences to the random mode where applicable. 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 might have happened, that a whole iteration of the loop happened in reality, but was forgotten to be documented. in which case, the path $\langle B, E, F, G, A, C, D, H \rangle$ would also be an option to repair trace t_2. Equally likely, the second iteration could have gone the other way in the model: $\langle B, E, F, G, B, F, E, G, H \rangle$, in which case different assignments of the events E, and G in t_2 to the path in the model exist. In general, there are infinitely many possible traces for a model that contains loops.

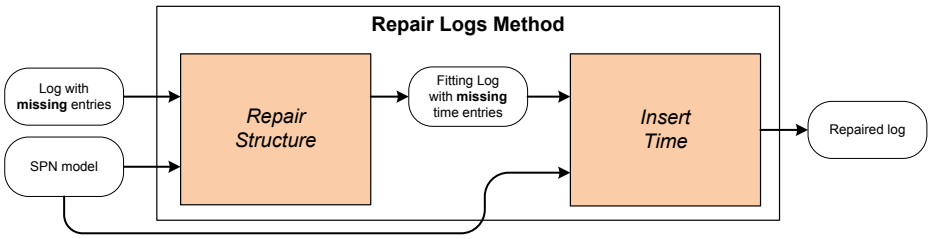

Fig. 4: We divide the problem into two subproblems: repairing the control flow and repairing the timestamps.

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, 1) repair structure and 2) insert time, as sketched in Fig. 4. The method uses as input a log that should be repaired and an SPN model specifying the as-is process.

Note that by 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. We discuss this limitation in the evaluation section.

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 SPN models, are *sound*, cf. [21], and *free-choice*, cf. [22], but do not necessarily need to be (block-)structured. This class of models

captures a fairly large class of process models and does not impose unnecessary constraints.
- The stochastic Petri net 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 look at 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.

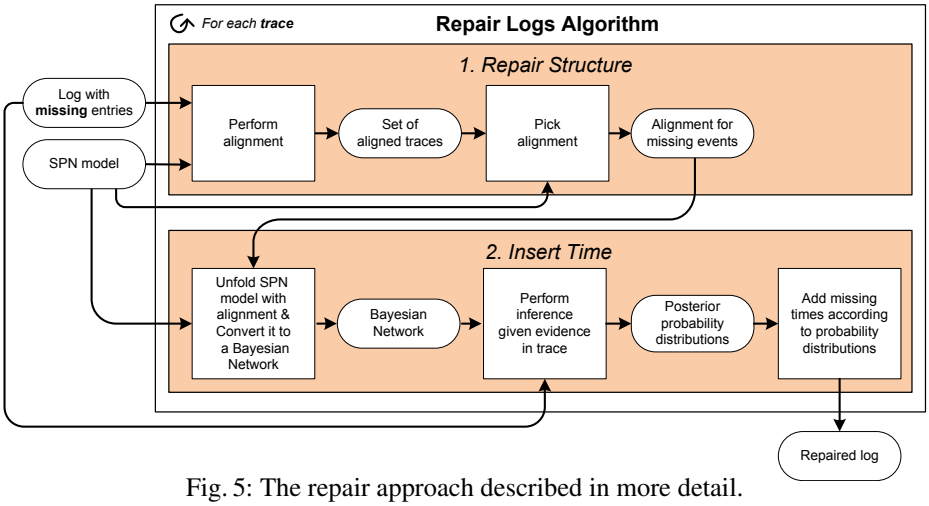

Fig. 5: The repair approach described in more detail.

5.1 Repairing the Structure

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 cost-based fitness alignment [13] introduced in Sect. 3, can be 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.

We set the costs of synchronous and model moves to 0, the cost of log moves to a very 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 some probabilistic information into account when picking an alignment and enhances the alignment approach introduced in [13]. Also, we take into account 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 how much to penalize possible paths in the model that are not reflected in a log by any event.

For example, consider a loop in a 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 are missing. Extreme probability values, such as 0, or 1 should be avoided, as the probabilities of all alignments containing both synchronous and model moves will be 0.

We can select candidate alignments using various policies depending on the intended use. If we want to find the most probable missing events only, we choose the alignment with the highest probability. But we can also choose randomly according the probabilities to follow a multiple imputation approach. Once we decided on the structure of how our repaired trace will look like, we can continue and insert the times of the missing events in the trace, i.e., the identified *model moves*.

5.2 Inserting Time

To repair the timing information it is not enough to look at the 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 Sect. 3, there exists a solution for this task: *Inference* in Bayesian networks. Thus, we can convert the SPN model into a Bayesian network and then insert the evidence given by the observations to be able to perform the inference.

In the previous step, we identified the path through the SPN model. With the path given, we can eliminate choices from the model by removing branches in the process that were not taken. We unfold the net from the initial marking along the chosen path. Note that loops are but a special type of choices and will be eliminated from the model for any given trace. 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 and is directly converted 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.

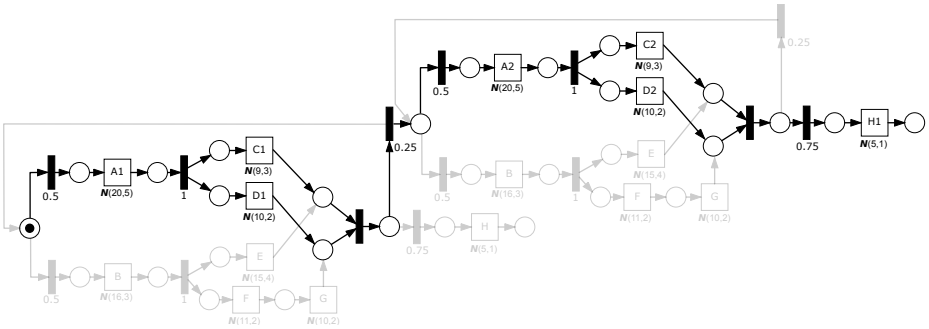

Fig. 6: Unfolded model in Fig. 1 for path $\langle A, D, C, A, C, D, H \rangle$.

We transform the resulted 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.

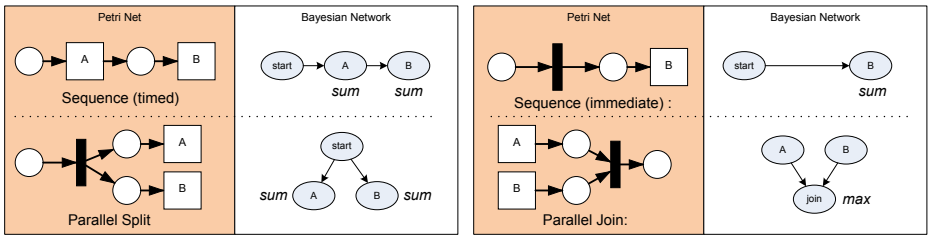

Fig. 7: Transformation of SPN models to Bayesian Networks.

Fig. 7 shows the transformation of sequences, parallel splits, and synchronizing joins. These are the only constructs remaining in the unfolded form of the 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 a timed transition t_i is followed by another 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 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 the conditional probability distribution $\mathrm{P}(X_j \leq x \mid X_i)$. For each possible value of the parent $x_i \in X_i$, the probability is defined as $\mathrm{P}(X_j \leq x \mid X_i = x_i) = D_j(x - x_i)$. This means that 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. lower right 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, $\mathrm{P}(X_k \leq x \mid X_i, X_j) = \mathrm{P}(X_i \leq x) \cdot \mathrm{P}(X_j \leq x)$, i.e., the probability distribution functions are multiplied. This proves to be a challenge, as the maximum of two normally distributed random variables is no longer normally distributed. We use a linear approximation, as described in [23]. 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 it degrades when they strongly diverge, cf. [23]. 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 $O(n^3)$ where n is the number of nodes [20]. Otherwise, inference in Bayesian networks is an NP-hard problem [24].

Once we have constructed the Bayesian network, we need to set the values for the observed events for their corresponding random variables, i.e., 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 [25], 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. One repair mode 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 could be used by process participants as clues when performing root cause analysis. For example, in order to find the responsible person for the task in question, an estimation of when it happened might prove helpful. Then obvious documentation errors could be corrected. 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. If one is interested in the variance of duration values rather than the mean durations, the random repair mode is preferable. Then, instead of the most likely values, random samples from the posterior probability distributions should be used.

We need to be careful when sampling from normal distributions, as they are defined in both the positive and the negative domain, i.e., a sample time could be negative. Therefore, our assumption is that most of the probability mass is in the positive domain, such that the error introduced by a simple correction mechanism like re-sampling negative values is acceptable in these cases.

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[3]. 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 were just not recorded. Therefore, we need to 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 remove a percentage of the events randomly from these fitting traces to get the first input to the algorithm. We pass the log with missing entries to the repair algorithm, along with the model, according which we want to do the repair.

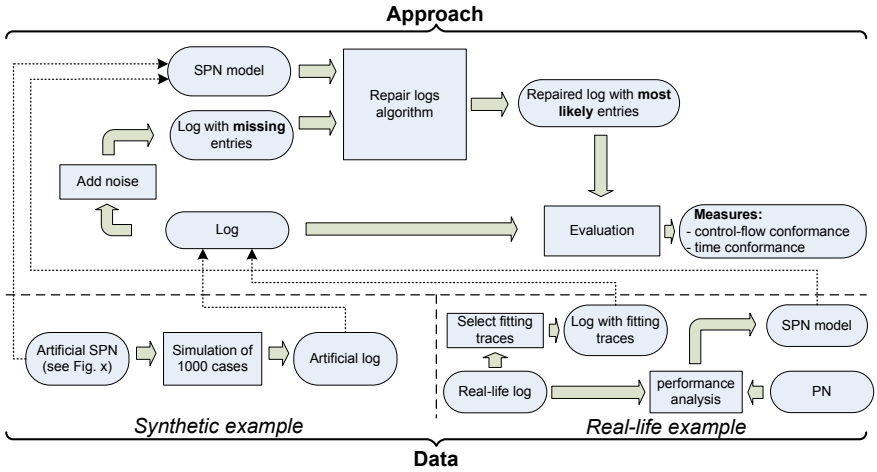

Fig. 8: Approach used to evaluate repair quality.

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 [13] compares how well a model fits a log. Here, we use it to compare the traces pairwise in the two logs. We re-use the existing calculation methods and 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 use a simple measure by comparing the real event's time with the repaired event's time. This makes sense if we chose the correct event to be inserted. 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.

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

6.1 Artificial Example

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

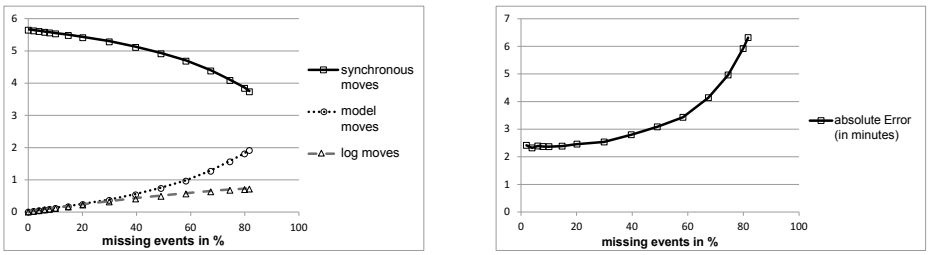

Fig. 9: Evaluation results for repairing 1000 traces of model in Fig. 1.

Figure 9 displays the resulting measures of the quality which we achieved by repairing traces. The experiment was done with a log of 1000 simulated 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 can be explained by incorrect ordering of parallel activities, while at higher noise levels the number of model moves increase further. At higher noise levels, 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 deterministic algorithm (repairing 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 can see the mean absolute error in the 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. From these results, we conclude that the repair algorithm can be used to repair logs, if the amount of noise is relatively low.

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 implemented the common 10-fold cross-validation method. 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 [26]. The process is depicted as an 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. After that, 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.

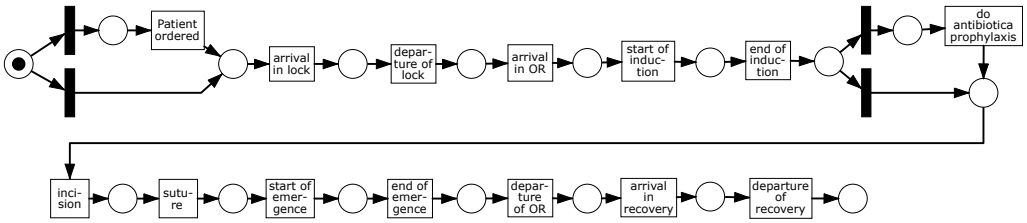

Fig. 10: Real surgery model for a surgical procedure in a Dutch hospital.

The log of this treatment process contains missing entries, which motivated the research work in this report. 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. We use the 570 fitting cases to evaluate, how well we can repair them after randomly removing events.

Figure 11 shows the evaluation results with the real log. Observe that the structure can be repaired quite well in comparison to the artificial example in Fig. 9. This is due to the sequential nature of the model with twelve events in sequence and two activities that are optional. The number of synchronous moves gradually approaches twelve synchronous moves, when the noise is very high. This is due to the algorithm being unable to repair single undetected events that are optional.

The mean absolute error in the restored events is higher than the artificial example. This value greatly depends on the variance in the activity durations. In the evaluation example, the variance of certain activity durations in the model is quite high, e.g., the duration of the *start of emergence* activity has a mean of 4.69 and variance of 18.29^2. The

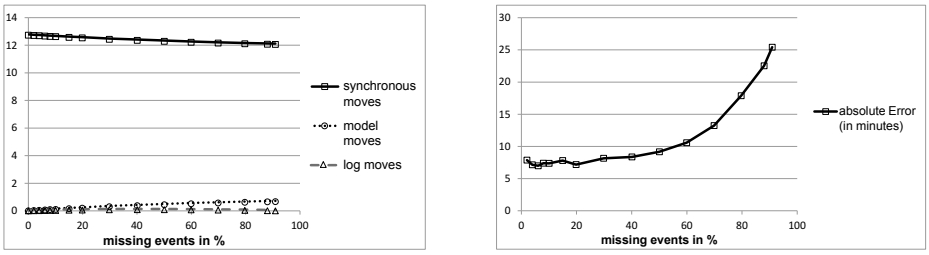

Fig. 11: Evaluation results for model in Fig. 10.

data underlying this model is right-skewed with most values small, and some outliers. Here, the normal distribution is inappropriate, and other distributions might fit better.

Obviously, the ability to repair a log highly depends on the information content of observed events in the trace and the remaining variability in the model. For instance, we can repair a pure sequential model always with fitness 1.0 of the repaired log. Even, if we observe just 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 deterministic repair mode, which we evaluated, is unable to restore optional branches without structural hints, i.e., at least one activity on that optional branch needs to be recorded. This affects single optional activities most, as their absence will not be repaired. However, many real-life processes are sequential, and can be repaired correctly.

7 Conclusion

In this report, we presented a method to repair timed event logs in order to make them available for further analysis, e.g., with process mining tools. The method works by decomposing 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 [13] 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. Both parts can be done deterministically (to get the most likely result), as well as randomly by Monte Carlo simulation (to account for variability). The algorithm can deal with a large and representative class of process models (any free-choice workflow net).

Our preliminary evaluations indicate that we can repair the structure and the time quite well, if noise is limited. Models exhibiting a high degree of parallelism are less likely to be repaired correctly than models with more dependencies between the 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 influence the path probabilities, too.
2. The normal distribution, though having nice computational properties, is of limited suitability to model activity durations, because its support covers the negative domain, too.

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 SPN model contains the truth, and deviations in the log are caused by documentation errors, than by deviations from the process. This assumption is only feasible for standardized processes with few deviations that are reflected in the model. Therefore, we advise to use this approach with care and try to correct documentation errors using repaired logs as assistance.

With this report, we paved the way for missing data techniques from the statistics community to be used in the business process domain, too. This work can also be considered as the first step towards eliciting a SPN model from logs with *missing data* in a *maximum likelihood* or *multiple imputation* fashion. This way, allowing to take all the observed data into account and get efficient estimations for the activity durations and path probabilities.

In future work, we want to relax the assumption of normal distributions and broaden the supported distributions to the class of Gaussian mixtures as described in [27]. Any distribution can be approximated by a mixture of Gaussians. We also want to investigate improvement possibilities in the structural part, i.e., look at ways to integrate temporal information into performing the alignment.

Future work needs also 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 or not. An integration with the proposed technique in [10], 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. Fahland, D., Aalst, W.v.d.: Repairing Process Models to Reflect Reality. In: BPM. Volume 7481 of LNCS., Springer (2012) 229–245
3. Rogge-Solti, A., Aalst, W.v.d., Weske, M.: Discovering Stochastic Petri Nets with Arbitrary Delay Distributions From Event Logs. In: Business Process Management Workshops, Springer (2014) (to appear)
4. Marsan, M.A., Conte, G., Balbo, G.: A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. ACM TOCS **2**(2) (1984) 93–122
5. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)
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. Dempster, A., Laird, N., Rubin, D.: Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society. Series B **39**(1) (1977) 1–38
8. Rubin, D.B.: Multiple Imputation for Nonresponse in Surveys. Wiley (1987)
9. Little, R.J., Rubin, D.B.: Statistical Analysis with Missing Data. 2nd edn. Wiley (2002)
10. Bertoli, P., Dragoni, M., Ghidini, C., Francescomarino, C., D.: Reasoning-based Techniques for Dealing with Incomplete Business Process Execution Traces. Technical report, Fundazione

Bruno Kessler, Data & Knowledge Management (2013) https://dkm.fbk.eu/images/9/96/TR-FBK-DKM-2013-1.pdf.
11. Günther, C.W., Aalst, W.v.d.: Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In: BPM. Volume 4714 of LNCS., Springer (2007) 328–343
12. Aalst, W.v.d.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
13. Adriansyah, A., Dongen, B.v., Aalst, W.v.d.: Conformance Checking using Cost-Based Fitness Analysis. In: EDOC 2011, IEEE (2011) 55–64
14. Buijs, J.C., La Rosa, M., Reijers, H., Dongen, B.v., Aalst, W.v.d.: Improving Business Process Models using Observed Behavior. In: Post-Proceedings of SIMPDA 2012. LNBIP, Springer (2013) (to appear)
15. 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
16. 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
17. Hu, H., Xie, J., Hu, H.: A Novel Approach for Mining Stochastic Process Model from Workflow Logs. Journal of Computational Information Systems **7**(9) (2011) 3113–3126
18. Anastasiou, N., Horng, T., Knottenbelt, W.: Deriving Generalised Stochastic Petri Net Performance Models from High-Precision Location Tracking Data. In: VALUETOOLS'11, ICST (2011) 91–100
19. Aalst, W.v.d., Adriansyah, A., Dongen, B.v.: 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
20. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT press (2009)
21. Aalst, W.v.d.: Verification of Workflow Nets. In: ICATPN'97. Volume 1248 of LNCS., Springer (1997) 407–426
22. 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
23. Zhang, L., Chen, W., Hu, Y., Chen, C.: Statistical Static Timing Analysis With Conditional Linear MAX/MIN Approximation and Extended Canonical Timing Model. In: TCAD. Volume 25., IEEE (2006) 1183–1191
24. Cooper, G.: The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks. Artificial intelligence **42**(2) (1990) 393–405
25. Murphy, K.: The Bayes Net Toolbox for Matlab. In: Interface'01. Volume 33 of Computing Science and Statistics. (2001) 1024–1034
26. 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
27. Shenoy, P.: Inference in Hybrid Bayesian Networks Using Mixtures of Gaussians. In: UAI'06, AUAI Press (2006) 428–436