Discovering Stochastic Petri Nets with Arbitrary Delay Distributions From Event Logs

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

[1] Business Process Technology Group,
Hasso Plattner Institute, University of Potsdam, Germany
{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.
w.m.p.v.d.aalst@tue.nl

Abstract. Capturing the performance of a system or business process as accurately as possible is important, as models enriched with performance information provide valuable input for analysis, operational support, and prediction. Due to their computationally nice properties, memoryless models such as exponentially distributed stochastic Petri nets have earned much attention in research and industry. However, there are cases when the memoryless property is clearly not able to capture process behavior, e.g., when dealing with fixed time-outs.
We want to allow models to have generally distributed durations to be able to capture the behavior of the environment and resources as accurately as possible. For these more expressive process models, the execution policy has to be specified in more detail. In this paper, we present and evaluate process discovery algorithms for each of the execution policies. The introduced approach uses raw event execution data to discover various classes of stochastic Petri nets. The algorithms are based on the notion of alignments and have been implemented as a plug-in in the process mining framework ProM.

Keywords: process mining, stochastic Petri nets, generally distributed transitions

1 Introduction

Process mining has emerged as a promising technology to gain insights into the actual execution of business processes and has been successfully applied in hundreds of organizations [1]. Besides the discovery of process models, process mining can also be used to enrich existing models with information gathered from event logs. In particular, capturing activity durations and waiting times in the business process is necessary to gain insights about the performance of the process. Further, these enriched models can be used as basis for prediction algorithms to estimate the time until completion of the process [1]. Estimating the remaining run time of business processes and its activities is an important management task, since it allows to improve the allocation of resources. It also increases the quality of results when clients inquire the status and expected completion of a given business process.

Petri nets have been used widely in the business process domain, either as first class modeling languages, or as basis for verification purposes. There exist mappings for many workflow and business process modeling languages (e.g., BPMN, UML activity diagrams, BPEL, and EPCs) into Petri nets [2], as they are able to capture the most important control flow constructs.

If we have historical observations of a given process, e.g., an *event log* with timing information, it is possible to extract stochastic performance data and add it to the model. These enriched models can be used in a number of use cases. Besides answering questions such as *"How many percent of the process instances take longer than 10 days?"*, they can be used as basis for simulation, e.g., for *what-if* analysis. Moreover, they can be used to get accurate predictions of the remaining time and offer operational support.

Current state-of-the-art performance mining techniques focus only on gathering mean and variance (assuming normally distributed durations) [3,4], or the firing rate (assuming exponentially distributed durations) [5,6] of times. We are interested in automatically learning more fine grained information and want to be able to capture deterministic time-outs, or irregularities, such as multi-modal distributions. This paper investigates performance mining techniques for generally distributed transition stochastic Petri nets (GDT_SPN) that do not restrict distributions to any shape.

Multiple execution policies exist for these models that need to be taken into account [7]. In a nutshell, the problem addressed in this paper is to infer the stochastic parameters of a given Petri net, using an event log, and an execution policy. We base our algorithms on the alignment technique originally developed for conformance checking [8]. Our alignment-based approach is more robust than naïve replays of logs on the model, as it guarantees finding the globally best alignment based on a cost function that considers asynchronous parts of the replay.

The paper is organized as follows. In Section 2 preliminary definitions are provided. The main challenges and the performance mining algorithms addressing them are discussed in Section 3. A preliminary evaluation showing the capabilities to restore different kinds of models is presented in Section 4. Afterwards, related work is discussed in Section 5. Finally, conclusions are presented in Section 6.

2 Preliminaries

In order to establish a formal basis and to clarify the difficulties and solution ideas, this section introduces the concepts and techniques used throughout this paper. First, the core concepts of event logs and Petri nets are given.

Definition 1 (Event Log). *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 assigning each event to an activity,*
- *$\gamma : E \rightarrow TD$ is a function assigning each event to a timestamp,*
- *$\beta : E \rightarrow C$ is a surjective function assigning 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 sequence of events belonging to one case a "trace". We assume that $e_2 \succ e_1$ implies $\gamma(e_2) > \gamma(e_1)$, i.e., the time ordering is respected.

Definition 2 (Petri Net). *A Petri net is a tuple $PN = (P, T, F, M_0)$ where:*
- *P is a set of places,*
- *T is a set of transitions,*
- *$F \subseteq (P \times T) \cup (T \times P)$ is a set of connecting arcs representing flow relations,*
- *$M_0 \in P \to \mathbb{N}_0^+$ is an initial marking.*

Over the years, various kinds of extensions to Petri nets have been proposed in order to capture performance criteria. An overview of different important classes of stochastic Petri nets can be found in [9]. For our purposes, we extend the widely known definition of Generalized Stochastic Petri Nets (GSPNs) provided in [10], by allowing durations of the timed transitions to be generally distributed. In terms of the categorization proposed in [9], we use SPN with *generally distributed transitions*.

Definition 3 (GDT_SPN). *A generally distributed transition stochastic Petri net 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 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*
- *$\mathcal{D} : T_t \to D$ is an assignment of arbitrary probability distributions D to timed transitions, reflecting the durations of the corresponding activities.*

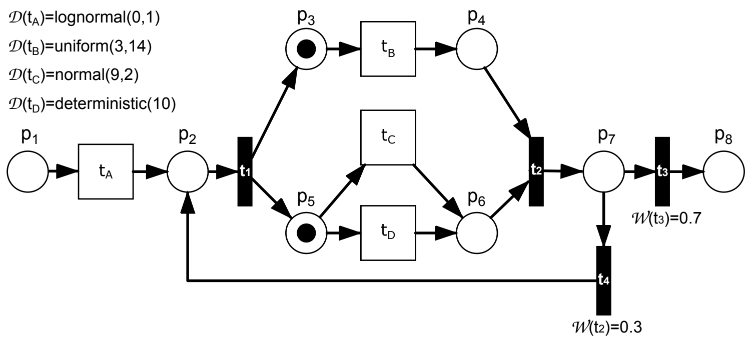


Fig. 1: Example *GDT_SPN* model with two parallel branches, and a conflict between transitions t_C, and t_D.

An example GDT_SPN model is depicted in Fig. 1. Here, all weights of transitions are 1, unless otherwise specified, e.g., the weight of the immediate transition leaving the loop t_3 is 0.7. Immediate transitions (t_1,t_2,t_3,t_4) are depicted as black bars and have priority 1. The timed transitions (t_A,t_B,t_C,t_D) are depicted as boxes and have priority 0. The distributions of the probabilistic delays of the transitions D are annotated in a legend in the top left of the figure, e.g., transition t_B has a uniform distribution in the interval $[3, 14[$. Although the transition durations depicted in this example are of parametric shape, it is also possible to specify other distributions, e.g., densities based on nonparametric regression. Note that even though the example model in Fig. 1 is structured and free-choice, the approaches presented in this paper are also applicable for non-structured and non-free-choice models.

The basic semantics of GSPN models [10] are still valid for GDT_SPN models used in this paper, i.e., only the enabled transitions of the highest priority are allowed to fire in the current marking. This ensures that if immediate transitions are enabled, no timed transition can fire. As in GSPN semantics, the choice between multiple enabled immediate transitions is resolved probabilistically in proportion of their weight parameters.

Next to the seven-tuple $GDT\_SPN = (P, T, \mathcal{P}, \mathcal{W}, F, M_0, D)$, an execution policy [7] has to be chosen to resolve conflicts between multiple enabled transitions and to decide upon the *memory* of transitions, i.e., if and how long they store the duration of time passed in enabled state. If more than one timed transition is enabled in a marking of a GDT_SPN, the selection policy defines how the next transition is chosen.

In *preselection* mode, this choice is resolved based on the weights. When using the *race* policy, each enabled transition picks a random sample of its distribution and the one with the lowest sample fires next. The memory policy defines what happens to the transitions that lose the race. There are three options, either i) *resampling*, which constitutes to losing all progress, ii) *enabling memory*, where each transition remembers its sampled duration until it becomes disabled or fires, or iii) *age memory*, where transitions remember their sampled time, even through periods of disabling, until they eventually can fire.

The most common execution policies used for business processes are either *race* with *enabling memory* or *race* with *age memory*. We do not impose restrictions upon the execution semantics in this paper, rather we provide algorithms to reconstruct GDT_SPN models, assuming a particular execution policy. Before that however, we need to introduce the notion of alignments [8,11], which we base our algorithms upon.

2.1 Cost-Based Alignments

Figure 2.a) shows two execution traces (tr_1, tr_2) of the model depicted in Fig. 1, such that each event in the trace corresponds to a transition in the net with matching subscript, e.g., event B belongs to transition t_B. For this example, we assume that immediate transitions are invisible, i.e., they do not appear in the log, and all timed transitions are visible. This must not necessarily be the case in general, as there might be observable immediate transitions or invisible timed transitions as well. Dealing with invisible timed transitions is out of scope of this paper, however. We denote invisible transitions in a model in the alignment with a τ symbol. Note that trace tr_2 does not fit well into the model, so we want to find an optimal alignment between model and log. For this purpose, we reuse the methods developed by Adriansyah et al. in [8], 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*. Figure 2.b) displays a perfect alignment for tr_1 that consists of synchronous, or invisible model moves only.

(a) a small log:

tr_1 :	⟨ A,	B,	D,	C,	B⟩
tr_2 :	⟨ B,	D,	D⟩		

(b) perfect alignment for trace tr_1:

log	A	≫	B	D	≫	≫	≫	C	B	≫	≫
model	A	τ	B	D	τ	τ	τ	C	B	τ	τ
	t_A	t_1	t_B	t_D	t_2	t_4	t_1	t_C	t_B	t_2	t_3

(c) two possible alignments for trace tr_2:

(c.1)

log	≫	≫	B	D	D	≫	≫
model	A	τ	B	D	≫	τ	τ
	t_A	t_1	t_B	t_D		t_2	t_3

(c.2)

log	≫	≫	B	D	≫	≫	≫	D	≫	≫	≫
model	A	τ	B	D	τ	τ	τ	D	B	τ	τ
	t_A	t_1	t_B	t_D	t_2	t_4	t_1	t_C	t_B	t_2	t_3

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

For trace tr_2 there exist multiple possible alignments, of which two are depicted in Fig. 2.c). The ≫ symbol represents no progress in the replay on either side, e.g., the first step in the alignment in Fig. 2.c.1) is a model move. In fact, for the model in Fig. 1 there exist infinite alignments, as the model contains a loop that could be traversed an arbitrary number of times, resulting in two additional model moves, and three invisible model moves per iteration. The cost based alignment approach in [8] makes sure that alignments containing unnecessary moves get penalized by higher cost and therefore excluded from the optimal alignments. Alignments provide a deterministic firing sequence in the model replaying the traces in an optimal way.

3 Mining GDT_SPN Models

There are multiple difficulties in mining GDT_SPN models from event logs. First, we describe how the alignment technique introduced in Section 2.1 helps dealing with noisy event logs, i.e., logs where events might be missing, be at unexpected positions, or be reflecting activities not captured in the model.

3.1 First Challenge: Dealing with Noisy Logs

In order to extract decision and timing information from logs and combine the extracted information with a Petri net to get a GDT_SPN model, each individual trace in the log needs to be *aligned* to the Petri net. That is, the path in the model that was taken in the trace has to be identified. Previous techniques to extend models with performance information, e.g. the work in [3] tries to find the path through a model in a greedy way. Typically, this is done by replaying the model and looking for the best next match(es) between enabled transitions and next events with a given look-ahead. In contrast, the cost-based alignment technique introduced in Sect. 2.1, guarantees to find one of the alignments that is optimal in the sense that it has the least number of asynchronous moves (given that all asynchronous moves are assigned equal costs).

In fact, we add a small cost δ_t based on individual event counts to each transition t in the alignment, such that less frequent events and their corresponding transitions have a slightly higher cost than more frequent ones. This ensures that the alignment algorithm always favors the most frequent option, when there are multiple options to choose a path in the model. This is a simple heuristic that may pick the wrong alignment, but the best guess that can be made based on local frequency-based information. A more accurate option would be to leverage the whole alignment approach to consider the stochastic parameters of the model, which is out of scope of this paper.

The resulting alignments are only usable for our purposes, when most of the observed behavior actually fits the model. If fitness values are very low, a lot of information contained in the event log cannot be mapped to the model, and cannot be used for performance analysis. In this case, we add a preprocessing step before eliciting performance data. In this preprocessing step, the model needs to be adjusted to the log, which can be done by *repairing the model*, cf. techniques presented by Fahland and van der Aalst [12] to add optional subprocesses to models, or alternatively the work by Buijs et al. [13] based on genetic mining.

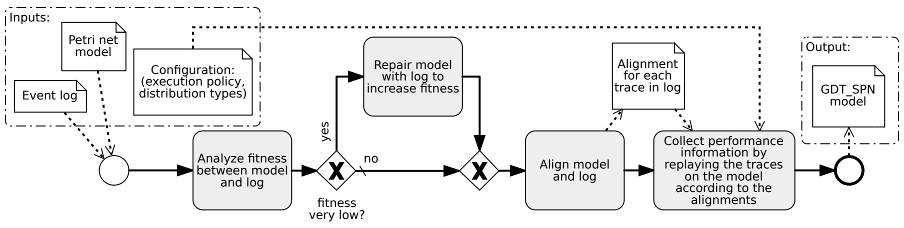

Inputs:
Petri net model
Event log
Configuration: (execution policy, distribution types)
Analyze fitness between model and log
yes
no
fitness very low?
Repair model with log to increase fitness
Align model and log
Alignment for each trace in log
Collect performance information by replaying the traces on the model according to the alignments
Output:
GDT_SPN model

Fig. 3: BPMN model showing the approach to elicit GDT_SPN models.

Fig. 3 shows an overview of the elicitation approach proposed in this paper. The inputs to the approach are the Petri net model reflecting the structural behavior of the process, and an event log containing the traces representing actual executed process cases. Further, a configuration is necessary, as GDT_SPN models are flexible in their execution semantics and transition distribution types that will be fitted to the data. The fitness between model and log can be analyzed by the technique described in [8] and implemented in the process mining framework ProM. If the fitness is under a user specified threshold, e.g., 0.7, first repair techniques available in [12,13] are executed on the model to allow for the behavior observed in the log. Then, each trace in the event log is aligned to the model to find one of the optimal paths through the model. With the alignments, the collection of stochastic execution information according to the input configuration is performed.

3.2 Second Challenge: Collecting the Stochastic Performance Information

With the alignments between model and the traces selected, the collection of performance information can proceed depending on the configuration of the elicitation algorithm, i.e., execution policies and distribution types. First, we discuss the common approach that is used regardless of the specific execution policy.

The alignment makes sure that each event is assigned to at most one transition in the model. Based on the alignments, we replay the traces on the model, as if in a simulation, but instead of sampling random values from the distributions, we use the observed values extracted from the event log and infer the most likely stochastic model that explains the observation best. This works well if we can gather all the information that would also be produced in a simulation, i.e., the sample values of the transition firings, and the firing ratios for the markings. The different execution policies (*preselection* / *race* and in case of a *race* policy also the memory policy) are complicating matters. For example, races between simultaneously enabled transitions can only be observed indirectly: the sampled values of the losing transition of the race cannot be recovered in all cases. Other reasons for missing information might be noise in the data, e.g., missing events.

Depending on the execution policy of the GDT_SPN model, different approaches are used to collect the stochastic parameters of the model.

Global preselection policy With the *global preselection* policy, only one transition can perform work at once, leading to a serialized process execution. Given this policy, we replay the traces in the model and collect in each visited marking for each enabled transition the number of times, the transition was picked. These numbers, when normalized, give us a ratio of the weights of the transitions in each marking. Note that one transition can be enabled in multiple markings, i.e., a transition weight needs to fulfill multiple equations and there may be dependencies. Hence, we solve an optimization problem to find the joint assignment to the transition weights that minimizes the error in these equations of each marking. To achieve this, we implemented a gradient descent algorithm that finds the weight vector that minimizes the individual errors in the process mining toolkit ProM[3]. The algorithm is guaranteed to converge (if the learning rate is sufficiently low) to a local optimum. Since the cost function of the errors is convex by nature and has, similarly to linear regression, no local optima, it finds the best global assignment. Note that if we would extend our model to capture marking dependent weights, we would not need to average the errors out, but could directly estimate the weights as the observed ratio of transition firings in each marking.

The time differences between events in the trace represent the duration of transition firings. Since there is no parallel execution in global preselection, these transition durations can be read from the trace directly. However, special attention needs to be devoted to asynchronous moves in the alignment. The time difference between transitions should only be collected, if the current move is synchronous and the previous is either also synchronous, or a log move. In other cases the difference between the times of events are spanning multiple transitions in the model. If more than one of these transitions is timed, we can use this delay as upper bounds for all involved timed transitions on the path in the alignment between the two events.

Race selection policy In the race selection policy, we also replay the traces in the model according to the path dictated by the alignment. The challenge in the race selection policy is that we can only read the duration sample of the winning transition directly. That duration can however serve as a lower bound for the other transitions that lose their progress. Depending on the memory policy this issue of non-retrievable samples is more or less severe.

With the *resampling* memory policy, we only get exact samples for winning transitions. In the worst case, when a transition loses every race, we have only a lower bound on it's distribution which makes inference on the actual shape or range of it's values impossible. However, this is only a problem for transitions that rarely happen. Note also that this policy is rarely used in practice, as it does introduce dependencies between

[3] See the `StochasticPetriNet` package of `ProM` (`http://www.processmining.org`)

parallel transitions, i.e., parallel transitions have to restart their work, because another transition was faster.

Of more practical relevance is the *enabling-memory* policy, which allows for transitions that lose a race against non-conflicting transitions to keep their progress. If we have both events for the transition enabling and the one firing the transition, we can calculate the duration of the transition by simple subtraction between these two events. In this policy, we still cannot recover sample durations for transitions that lose a race against a conflicting transition, i.e., a transition that disables it. Thus, we also have to deal with censored data for timed transitions in conflict with other transitions.

Last, the *age-memory* policy even allows to reconstruct the sample duration of timed transitions in conflict, if they are re-enabled, e.g., by another iteration of a loop. We need to collect all enabled periods and take their sum to reconstruct the originally sampled firing duration. This is straightforward by using age variables for transitions and incrementing them by the time spent in enabled state during replay. Note that even in the *age-memory* policy not all duration samples of conflicting transitions can be collected from the event log in general. Recall that a transition does not necessarily become enabled again after being disabled by a conflicting transition.

Thus, we can treat all memory policies for the race policy equally, besides from subtle difference in gathering the samples, i.e., points in time when transition clocks will be reset. In all cases we gather both accurate samples of the unknown distribution that we want to infer, and also lower bounds on the remaining samples, when another transition fired first. In general, we face the problem to infer the probability distribution of a random variable with randomly right-censored data. Approaches to this problem are well-known in statistics, cf. the overview by Padgett and McNichols [14]. We use the method of Kooperberg and Stone [15] that fits log-splines to censored data, and is available in R.[4]

3.3 Third Challenge: Dealing with Special Cases

As claimed in the introduction, allowing for *timeouts* is one important aspect of business process modeling. Therefore, the mining algorithm needs to be capable to detect deterministic durations in a model. In technical terms, once we have gathered the observed samples and censored samples, we can check, whether a transition is deterministic or not, by comparing the observed samples. If the samples are sufficiently close, we define the transition to be deterministic. This can be made more robust against noise, by removing outliers before applying these rules. In the mined model, the deterministic value is estimated as the mean of the observed values.

Another quite important modeling construct are immediate transitions, which fire *immediately* after they have been enabled, provided that they are selected amidst competing immediate transitions. We assumed immediate transitions to be invisible. But if corresponding events exist in the event log, the rule to identify these transitions is to set a threshold and check, whether all observed values are within 0 and that threshold. Note that we cannot give a general rule, as it depends on the domain, e.g., the response time of systems, how large these thresholds should be.

[4] See package *logspline* in R. (`http://www.r-project.org/`)

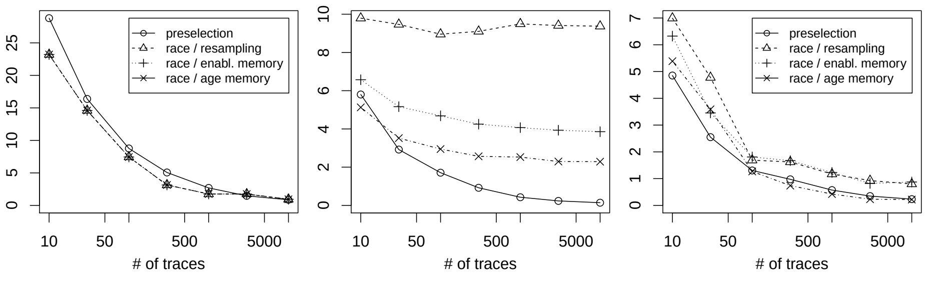
4 Evaluation

To evaluate how well the algorithm works on data, it is necessary to compare its output, i.e., the discovered model, with the model that produced the behavior in the event log. In general however, the theoretical model behind the observable phenomena is not known. Therefore, we rely on a simulation based approach. First, we need a GDT_SPN model. There exist already algorithms that can discover less expressive performance models from data [5,4], which can serve as a starting point, or a hand-made model can be used. In this evaluation, we focus on the following two questions:

– How many traces do we need to get reasonably accurate results?
– How tolerant is the algorithm towards noise in the log?

To gain insights into these questions, we ran the following experiment. First, multiple logs are simulated from the GDT_SPN model depicted in Fig. 1 with increasing trace count from 10 traces to 10000 traces. The simulated event logs, the underlying Petri net (P, T, F, M_0) of the GDT_SPN model, and the given execution policy are passed as input to the discovery algorithm. The algorithm produces another GDT_SPN model, which we compare with the original model.

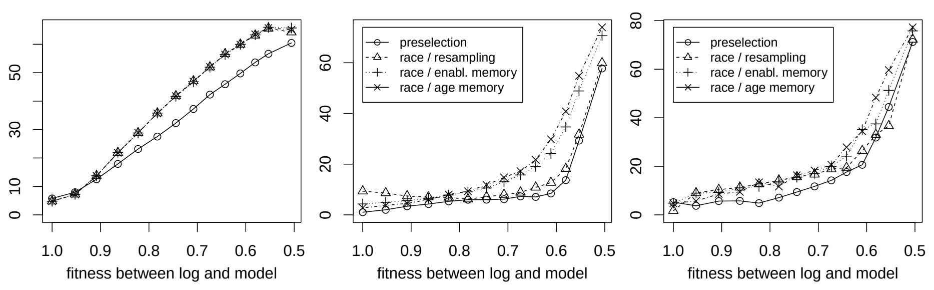
There are several ways to assess the accuracy of a model. To test for the bias that our method introduces in the model parameters, we calculate the mean absolute percentage error (MAPE) of the estimated 1.moment and the original 1.moment of each timed transition's distribution. Note that we omitted the first transition t_A from the calculation, because we cannot calculate its duration, as there is no previous event with a timestamp in the log. Weights are evaluated relatively to each other when selecting an immediate transition, and additionally in *preselection* mode also when selecting the next timed transition. Therefore, we need to compare the weight ratios in each marking of the original and discovered model, where selection of the next transition is based on weights. Because weights are evaluated relatively to each other, we normalize them first, before we calculate the MAPE of the weights in each relevant marking.



(a) MAPE of weights (b) MAPE of 1.moments (ignoring censored data) (c) MAPE of 1.moments (using censored data)

Fig. 4: Effects of trace size on restored model accuracy. Mean average percentage error (MAPE) of weights and MAPE of 1.moments of inferred distributions for timed transitions of the model in Fig. 1. Number of traces drawn in log-scale.

Figure 4.a) shows the error of the transition weights between the original model and the inferred model from the log. Note that weight errors of all *race policies* collapse, as their weights are computed in the same way. However, the *preselection* policy has more constraints on the weights, and random behavior of small event logs prohibits discovering the true weights accurately. Fig. 4.b) shows the mean average percentage error of the 1.moments, when a simple kernel density estimation is used for estimating the duration of timed transitions. As expected, the *preselection* execution policy does not suffer from bias due to censored data. The *race* with *resampling* method is the most difficult to reconstruct, as many of the samples are discarded. The *enabling memory* policy has less bias, and in the *age memory* policy, the algorithm can restore most of the original sample durations. Fig. 4.c) depicts the error that remains, when the log-spline density estimator [15] is used. Note that this method considers censored data and can correct the bias well. It reduces the biases of the *race* execution policies significantly.



(a) MAPE of weights (b) MAPE of 1.moments (ignoring censored data) (c) MAPE of 1.moments (using censored data)

Fig. 5: Mean average percentage errors between the model in Fig. 1 and the reconstructed model with increasing amount of noise, i.e., reduced fitness.

For the second experiment, we keep the trace size at 1000 and run the discovery algorithms with logs of varying degrees of artificial noise, i.e., random addition and deletion of events. Fig. 5 depicts the same measures as before, i.e., the MAPE of relative weights in the markings and the MAPE of the 1.moments of the distributions. Observe, how in Fig. 5.b) the MAPE of the 1.moments increases non-linearly with lower fitness values. The quality starts dropping rapidly below a fitness of 0.8 in this example. When dealing with noisy logs, the Petri net models should be repaired first in a preprocessing step, as described in Sect. 3.

As a concluding remark, we caution against drawing general conclusions from these preliminary evaluations. Larger errors are expected for models with bigger state spaces.

5 Related Work

There exists already work on obtaining Petri net models with stochastic performance characteristics from data. Hu et al. propose a method to mine exponentially distributed SPN models from workflow logs in [6] focusing on firing rates of transitions. In contrast, our work allows for generally distributed firing times. Another, quite different approach was proposed by Anastasiou et al. [5] and uses location data to elicit generalized stochastic Petri net (GSPN) [10] models for modeling flows of customers. They fit hyper-erlang distributions to transition durations representing waiting and service times and replace the corresponding transitions with a GSPN subnet exhibiting the same characteristics of the hyper-erlang distribution. They consider every transition in isolation though, which poses no problems in serial processes but parallelism in the processes, especially multiple parallel transitions in conflict, are not covered in that approach.

Also attempts at eliciting non-Markovian stochastic Petri nets exist. Leclercq et al. investigate how to extract models of normally distributed data in [4]. Their work is based on an expectation maximization algorithm that they run until convergence. In comparison to our approach, they are not able to deal with missing data and do not consider different execution policies. Reconstructing model parameters for stochastic systems has also been investigated by Buchholz et al. in [16]. They address the problem to find fixed model parameters of a partially observable underlying stochastic process. In contrast to our work, the underlying process's transition distributions need to be specified beforehand, while our aim is to infer also transition distributions of a GDT_SPN model. In a similar setting, i.e., with incomplete information, Wombacher and Iacob estimate distributions of activities and missing starting times of processes in [17].

In [3], Rozinat et al. investigate how to gather information for simulation models, but rather try to identify data dependencies for decisions and mean durations and standard deviations and do manual replay, which is not guaranteed to find an optimal alignment between model and log. The approach proposed in this paper is capable to deal with noise in a more robust way, by building on the notion of alignments [8,11], which identify an optimal path through the model for a noisy trace. In conclusion, we are not aware of existing work that allows for generally distributed duration distributions, and different execution policies. Moreover, unlike existing approaches our approach is supported by a publicly available ProM plug-in and can thus be combined with a wide range of control-flow discovery approaches.

6 Conclusion

This paper addresses the challenges that arise when mining performance characteristics for models that can capture distributions other than the memoryless distributions. Unlike earlier work, the paper makes very little assumptions on the event data and underlying process to be mined. Accurate models of process performance are crucial for what-if analysis, predictions, recommendations, etc.

The stochastic model used in this paper extends the popular GSPN modeling technique. To analyze discovered GDT_SPN models, we need to resort to simulation due to the expressiveness of the class of models considered. We discussed different execution policies of GDT_SPN models and have shown how these different semantics can be taken into account when eliciting models. The *preselection* firing policy is the simplest case, which can be learned without problems. All other cases need sophisticated density estimation techniques that are able to cope with randomly right censored data. An implementation producing initial results is made available open source in the ProM framework.

Our next steps include comparing these mined models with other approaches and compare the model accuracy based on a specific use case, such as predicting the duration of a process. Future work also includes extending the alignment approach to align event logs probabilistically to a given GDT_SPN model, such that we can find the alignment with the highest probability.

References

1. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Lohmann, N., Verbeek, E., Dijkman, R.: Petri net transformations for business processes - a survey. In: Transactions on Petri Nets and Other Models of Concurrency II. Volume 5460 of LNCS. Springer Berlin Heidelberg (2009) 46–63
3. Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.: Discovering simulation models. Information Systems **34**(3) (May 2009) 305–327
4. Leclercq, E., Lefebvre, D., Ould El Mehdi, S.: Identification of timed stochastic petri net models with normal distributions of firing periods. In: Information Control Problems in Manufacturing. Volume 13. (2009) 948–953
5. 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
6. 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
7. 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
8. Adriansyah, A., van Dongen, B., van der Aalst, W.: Conformance Checking using Cost-Based Fitness Analysis. In: EDOC 2011, IEEE (2011) 55–64
9. 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
10. Marsan, M., 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
11. van der Aalst, W., 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
12. Fahland, D., van der Aalst, W.: Repairing Process Models to Reflect Reality. In: BPM. Volume 7481 of LNCS., Springer (2012) 229–245
13. Buijs, J.C., La Rosa, M., Reijers, H., van Dongen, B., van der Aalst, W.: Improving business process models using observed behavior. In: Post-Proceedings of SIMPDA 2012. LNBIP, Springer (2013) (to appear)
14. Padgett, W., McNichols, D.T.: Nonparametric density estimation from censored data. Communications in Statistics-Theory and Methods **13**(13) (1984) 1581–1611
15. Kooperberg, C., Stone, C.J.: Logspline density estimation for censored data. Journal of Computational and Graphical Statistics **1**(4) (1992) 301–328
16. Buchholz, R., Krull, C., Horton, G.: Reconstructing model parameters in partially-observable discrete stochastic systems. In: Analytical and Stochastic Modeling Techniques and Applications. Springer (2011) 159–174
17. Wombacher, A., Iacob, M.E.: Start time and duration distribution estimation in semi-structured processes. Technical report, Centre for Telematics and Information Technology, University of Twente (2012)