Quantitative Evaluation of Systems - 14th Int. Conf. (QEST 2017)
Berlin, Germany; September 5-7, 2017.

A Hybrid Multi-Trajectory Simulation Algorithm for the Performance Evaluation of Stochastic Petri Nets

Armin Zimmermann[1], Thomas Hotz[2], and Andrés Canabal Lavista[1]

[1] Department of Computer Science and Automation, Systems and Software Engineering Group, Technische Universität Ilmenau, Germany, armin.zimmermann@tu-ilmenau.de
[2] Institute for Mathematics, Stochastics Group, Technische Universität Ilmenau, Germany, thomas.hotz@tu-ilmenau.de

Abstract. Standard performance evaluation methods for discrete-state stochastic models such as Petri nets either generate the reachability graph followed by a numerical solution of equations, or use some variant of simulation. Both methods have characteristic advantages and disadvantages depending on the size of the reachability graph and type of performance measure. The paper proposes a hybrid performance evaluation algorithm for Stochastic Petri Nets that integrates elements of both methods. It automatically adapts its behavior depending on the available size of main memory and number of model states. As such, the algorithm unifies simulation and numerical analysis in a joint framework. It is proved to result in an unbiased estimator whose variance tends to zero with increasing simulation time; furthermore, its applicability is demonstrated through case studies.

Keywords: stochastic Petri nets, multi-trajectory simulation, hybrid numerical analysis / simulation method

1 Introduction

Model-based systems engineering is an important tool for complex system design, especially to predict non-functional properties in early design stages. This requires a model such as a stochastic Petri net as well as an efficient evaluation algorithm implemented in a user-friendly tool. This paper considers Stochastic Petri Nets (SPN, [2]). A variety of evaluation methods are known from the literature, all with certain individual advantages, but for a non-expert user it is not obvious which method should be used for his or her problem.

Perhaps the most significant classification of algorithms is the one between: 1) numerical analysis that explores and stores the full state space and thus suffers from large memory requirements that may exceed the given hardware; as well as 2) simulation that stores and follows just one state and trajectory of the system and thus may generate samples of interest only rarely for certain performance

measures. For the latter, there is no restriction on large state spaces thus, but some simulation problems will take unacceptable computation time.

The idea behind the algorithm presented in this paper is to find a hybrid mix of both methods: working similar to a numerical analysis as long as the memory is sufficient, but not storing all states if a certain maximum is reached and thus being able to handle any size like a simulation. We introduce a new algorithm that follows many but not all simulation trajectories, and stores them internally as particles with a certain weight. Different settings of maximum particle numbers lead to either an (adapted) standard numerical analysis or a standard simulation. This allows to explore new trade-offs between considering all or a single state in contrast to the two existing standard methods, which may be seen then as the extreme cases of the proposed algorithm. Moreover, the algorithm automatically adapts its behavior depending on whether the size of the underlying state space fits into the main memory. It thus combines the advantages of simulation and numerical analysis without a-priori in-depth knowledge of the modeler. However, the aim is not an algorithm that is faster than the existing approaches, but a method that integrates their behavior and thus avoids the decision which algorithm to use. The algorithm has been implemented as an extension of TimeNET [19]. Experiments show that it is competitive both in run time and accuracy in comparison to simulation and numerical analysis.

Considering a mathematical framework which allows treating simulation, numerical analysis as well as the proposed multi-trajectory algorithm, unbiasedness and convergence of the resulting estimator are proved. As a side effect, the possibility of configuring our algorithm with one simple numerical parameter to behave either like a simulation or a numerical analysis can be seen as a step towards a unified understanding of the two main methods in performance evaluation.

We use the name *multi-trajectory simulation* here as there are multiple trajectories of the same system model evaluated concurrently (and not multiple particles of one system as done in multi-particle simulation, for instance). This name has been coined in previous work which keeps several possible trajectories of a combat simulation [7]. However, the cited approach does not cover performance evaluation in a rigorous mathematical way. Approximation and discretization are used to decrease the number of trajectories (or particles). Heuristics merge particles that are similar, and delete trajectories viewed as being less significant.

In the literature, the term "hybrid simulation" has been used both for simulation of discrete/continuous state models as well as for methods that combine analytical and simulative methods. In the latter related work, parts of the model are evaluated by numerical analysis, and the local results are then fed into a simulation algorithm [14, 3] using a decomposition / aggregation approach quite different from our algorithm. A related method working with several internal states and trajectories (coined proxels) has been proposed in [10]. It aims at transient evaluation of non-Markovian Petri net models and uses a time discretization. Weighted ensemble simulation [9] considers simulation of trajectories that rarely pass from one attractor to another in a state space, and is applied to system

models in natural sciences. The approach is used to manage uncertainty in input data as well as probabilistic model evolutions common in weather forecasting or robotics motion planning. Importance splitting techniques in rare-event simulation [11] are another area in which multiple trajectories are being considered, which lead to the idea of the algorithm presented here. The RESTART algorithm [17], for instance, uses a depth-first-like algorithm to search for promising paths towards the state(s) of interest, and by discarding others that are assumed to be ineffective.

The paper is structured as follows: some terminology for the later explanation is introduced in Section 2, covering stochastic Petri nets and their quantitative evaluation. Section 3 introduces the multi-trajectory method for the performance evaluation of stochastic Petri nets. Subsequently, convergence results for the estimator are proved in Section 4, after which Section 5 reports numerical results obtained for a series of application examples, that were analyzed with an implementation in the TimeNET tool. Finally, conclusions and future work are pointed out.

2 Performance Evaluation of Stochastic Petri Nets

Stochastic Petri nets can be defined as $\mathsf{SPN} = (P, \mathcal{T}, \mathbf{Pre}, \mathbf{Post}, \lambda, \boldsymbol{m}_0, RV)$. We denote by P the (finite) set of places (i.e., state variables, denoted by circles), which may contain tokens. Each marking $\boldsymbol{m}$ of the Petri net is a vector of non-negative, integer numbers of tokens for each place $\boldsymbol{m} \in \mathbb{N}^{|P|}$. The initial state of the system is given by $\boldsymbol{m}_0$.

$\mathcal{T}$ specifies the (finite) set of transitions (depicted as rectangles). $\mathbf{Pre}$ describes the multiplicities of the input arcs connecting places to transitions $\mathbf{Pre} : P \times \mathcal{T} \to \mathbb{N}$. Similarly, output arcs $\mathbf{Post}$ from transitions to places are defined with their cardinality. Each transition tr has a firing rate $\lambda(tr)$ of an underlying exponential distribution. Finally, the measure(s) of interest to be computed are given by reward variables RV.

The behavior of a Petri net is defined as follows: A transition tr is enabled in a marking $\boldsymbol{m}$ if there are enough tokens available in each of its input places, i.e., $\forall p \in P : \boldsymbol{m}(p) \geq \mathbf{Pre}(p, tr)$. Whenever a transition becomes newly enabled, a remaining firing time (RFT) is randomly drawn from its associated exponential firing time distribution. The RFTs of all enabled transitions decrease with identical speed until one of them reaches zero.

The fastest transition tr is fired, changing the current marking $\boldsymbol{m}$ to a new one $\boldsymbol{m}'$ denoted as $\boldsymbol{m} \xrightarrow{tr} \boldsymbol{m}'$. The new marking is derived by removing the necessary number of tokens from the input places and adding tokens to output places with $\forall p \in P : \boldsymbol{m}'(p) = \boldsymbol{m}(p) - \mathbf{Pre}(p, tr) + \mathbf{Post}(p, tr)$.

If there is a transition tr enabled in marking $\boldsymbol{m}$ and its firing leads to marking $\boldsymbol{m}'$, we say that $\boldsymbol{m}'$ is directly reachable from $\boldsymbol{m}$. The set of all directly or indirectly reachable states from $\boldsymbol{m}_0$ is the reachability set RS or state space of the model. We assume models with a finite state space where the initial state

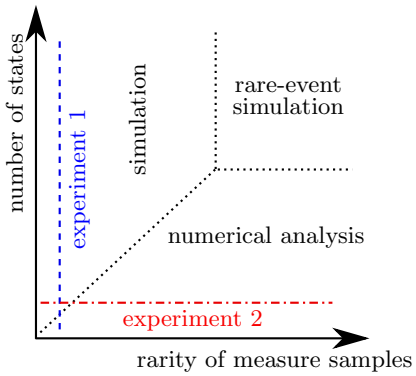

Fig. 1. Suggested application regions of evaluation methods

$\boldsymbol{m}_0$ is directly or indirectly reachable from every other state such that the state space is irreducible.

Performance measures $rvar(t) \in RV$ are defined here as functions of the stochastic process at time t and given by a rate part $RateReward^{rvar}(\boldsymbol{m}_t)$ returning the amount of reward gained in marking $\boldsymbol{m}$ per time unit, plus the impulse part $ImpulseReward^{rvar}(\boldsymbol{m}_t, tr)$ [13] specifying the amount of reward associated to firing transition tr in marking $\boldsymbol{m}_t$ if the Petri net is in state $\boldsymbol{m}_t$ at time t.

The stochastic process defined by such a model is a continuous-time, irreducible Markov chain with finite state space and transitions isomorphic to the reachability graph of the Petri net model [1, 5]. Its infinitesimal generator matrix $\boldsymbol{Q}$ with entries $q_{i,j}$ is given by the sum of all rates of exponential transitions tr for which $\boldsymbol{m}_i \xrightarrow{tr} \boldsymbol{m}_j$ (or zero if there is none). Diagonal entries of $\boldsymbol{Q}$ denote outflow rates set to $q_{j,j} = -\sum_{i \neq j} q_{j,i}$.

Stationary (steady-state) evaluation of performance measures is considered in this paper, for which there is a set of standard methods known (cf. [18]). Direct numerical analysis considers the full state space, solves for the invariant measure $\boldsymbol{\pi}$ of the underlying Markov chain via $\boldsymbol{\pi Q} = 0, 1 = \sum_i \boldsymbol{\pi}_i$, and derives the performance measure values simply from $\boldsymbol{\pi}$. The alternative is simulation, estimating results by $\lim_{T\to\infty} \frac{1}{T} \int_0^T rvar(t)\,\mathrm{d}t$. Numerical analysis is hard or impossible when the state space size becomes huge, while simulation runs into unacceptable execution times for models in which significant samples can be generated only rarely. Figure 1 depicts regions of performance evaluation problems and sketches suggested evaluation methods as well as the problem types covered by application examples in Section 5.

3 A Hybrid Multi-Trajectory Simulation Algorithm

The algorithm proposed in this paper is intended to cover the areas of simulation and numerical analysis without a-priori knowledge about the problem region.

The idea is the following: instead of the two extremes of either following just one trajectory in the state space as done by simulation versus considering all states at once as done in a numerical analysis, there should be a hybrid approach in between that stores and follows *some* trajectories. If possible, this should allow new trade-offs between available memory space, numerical accuracy and speed.

Our proposed multi-trajectory algorithm follows this scheme: we start with one simulation particle[3] for state $\boldsymbol{m}_0$ at simulation time $t = 0$ with weight 1. The weight of each particle equals the probability that a simulation would have arrived at the corresponding state until the current simulation step, given the previous probabilistic decisions. It is thus similar to trajectory weights considered in some rare-event simulation methods (e.g., [16]) to keep track of the amount of splitting.

The algorithm maintains two sets (current *Particles* and next *Particles*$'$) with elements of the form $p = (\text{marking } \boldsymbol{m}, \text{particle weight } w)$. In each step of the main simulation loop, the algorithm iterates over the current set of particles *Particles*. For each particle $p \in$ *Particles* stored, two treatments are possible:

- **Propagate**: the particle is simply followed like in a standard simulation by probabilistically choosing one of the subsequent states. The weight is kept constant.
- **Split**: all possible subsequent states are computed similarly to an iterative step in a numerical transient solution of a discrete-time Markov chain. The weight of the particle is distributed over all descendant particles resulting from a split by multiplying it with the enabled transition's firing probabilities.

All created particles are stored in the next particle set *Particles*$'$; if a particle with an identical state (marking) already exists, their weights are simply added (the particles are *merged*). For both cases, the simulation time is updated according to the average sojourn time in each particle, which can be computed from the delays of the enabled transitions.

For practical implementation reasons, the size of the particle sets needs to be bounded by a number of N particles to be stored. If the state space size of the model is larger than N, not every possible split will be executable. Obviously, the question of whether to use propagation or splitting influences the algorithm's performance. Some simple heuristics have been considered in our experiments so far [4]. The numerical results in Section 5 have been achieved with a heuristic that splits if one of the enabled transition has been fired less than average so far, if the weight of the particle is bigger than the average weight, or if the number of existing particles is less than $\frac{3}{4}N$ [4]. We see considerable room for improvements towards better heuristics in future work.

Algorithm 1 sketches the proposed program structure. It takes as input SPN, the stochastic Petri net model including performance measure definitions *rvar* as

[3] We use the term particle here to denote one simulation state as part of a larger set, not in the sense of a multi-particle simulation in physics.

MULTITRAJECTORY (SPN, N)

($*$ initializations $*$)
State($rvar$) := 0
$SimTime := Reward := 0$
$Particles := \{(\boldsymbol{m}_0, 1)\}$; $Particles' := \{\}$

repeat ($*$ main simulation loop $*$)
  while $|Particles| > 0$ **do**
    Select any $p \in Particles$; $Particles := Particles \setminus \{p\}$
    $w := p.weight$; $m := p.marking$
    $\mathcal{T}_{ena}$:= set of all transitions enabled in marking $\boldsymbol{m}$

    ($*$ rate reward and sojourn time $*$)
    $WeightSum := \sum_{tr \in \mathcal{T}_{ena}} \lambda(tr)$
    $Reward \mathrel{+}= \frac{w}{WeightSum} RateReward^{rvar}(m)$
    $SimTime \mathrel{+}= \frac{w}{WeightSum}$

    ($*$ decision heuristic, here: only split if enough space $*$)
    if $|Particles| + |Particles'| + |\mathcal{T}_{ena}| > N$ **then** ($*$ don't split $*$)
      Select any $tr \in \mathcal{T}_{ena}$ randomly
      $\mathcal{T}_{ena} := \{tr\}$; $WeightSum := \lambda(tr)$

    ($*$ fire transition(s) $*$)
    for $\forall tr \in \mathcal{T}_{ena}$ **do**
      $\boldsymbol{m}' := FireTransition(m, tr)$
      if $\boldsymbol{m}' \notin Particles'$ **then** ($*$ add new particle $*$)
        $Particles' := Particles' \cup \{(\boldsymbol{m}', 0)\}$
      ($*$ merge particle weight $*$)
      $Particles'(\boldsymbol{m}').weight \mathrel{+}= \frac{w}{WeightSum} \lambda(tr)$
      ($*$ impulse reward for fired transition $*$)
      $Reward \mathrel{+}= \frac{w}{WeightSum} ImpulseReward^{rvar}(m, tr)$

  ($*$ full particle set finished $*$)
  $Particles := Particles'$; $Particles' := \{\}$
  Collect measure sample with value $\frac{Reward}{SimTime}$

until simulation stop criterion is reached (confidence interval estimation)
return average of samples

Algorithm 1: Multi-trajectory simulation algorithm

well as an initial state $\boldsymbol{m}_0$, and the maximum number of particles N. Its output is the estimated value of performance measure *rvar*.

The behavior of the algorithm depends significantly on the number of particles and size of state space: For $N = 1$, only one particle will be considered, for which the weight will stay at 1. There will never be a split and the algorithm behaves like a normal simulation with one single trajectory. Numerically exact firing probabilities of all enabled transitions are computed instead and used to randomly select the next state.

If the algorithm is started with $|RS| < N$, particles for all markings $\boldsymbol{m} \in RS$ of the Petri net can be stored, and the algorithm works similar to a numerical algorithm variant as pointed out in Section 4. Settings of practical interest are thus $1 \ll |RS| < N$ and $|RS| > N$, which will be considered in the examples and numerical results of Section 5.

Algorithm 1 assumes only one performance measure *rvar* for simplicity, but any number can be computed concurrently. Intermediate variable *Reward* stores accumulated reward [13], i.e., the integral over the reward function.

During the evaluation, *SimTime* keeps the current simulation time (starting at 0) passed by all particles together on average; as such, marking holding (sojourn) times are not simply added to it, but weighted by the particle weights and computed for each particle using the enabled transitions' firing rates. The set $Particles'$ is organized to be rapidly searchable for a marking $\boldsymbol{m}$ to be part of it; for simplicity in the algorithm we denoted finding a marking by $\boldsymbol{m} \in Particles \longleftrightarrow \exists(\boldsymbol{m}, .) \in Particles$ and the found particle $Particles(\boldsymbol{m}) = (\boldsymbol{m}, w)$ if such a particle exists.

The decision from when on samples should be collected to avoid an initial transient bias, and how long samples have to be collected until a predefined accuracy setting given by some confidence interval and relative error is reached, is based on standard methods from the literature [12].

4 Unbiasedness and Convergence

The underlying stochastic process [5] of a well-specified stochastic Petri net model (c.f. Section 2) is a time-homogeneous and ergodic continuous-time Markov chain $MC = \{X_t\}_{t \in T}$ [15] with discrete and finite state space $\mathcal{S}$, $d = |\mathcal{S}| < \infty$. The mathematical analysis will be carried out in a discrete-time setting; this is justified since a well-known alternative to the standard way of deriving the steady-state numerical solution for the Petri net performance measures via $\boldsymbol{\pi Q} = 0, 1 = \sum_{i=1\ldots d} \boldsymbol{\pi}_i$ is to embed a discrete-time Markov chain *EMC* upon each state transition of the original *MC*. While both state spaces $\mathcal{S}$ will be identical, the one-step state transition probability matrix $\boldsymbol{P} = \{p_{i,j}\}$ of the *EMC* can be derived by

$$p_{i,j} = \begin{cases} \frac{q_{i,j}}{\sum_{k \neq i} q_{i,k}} & \text{if } i \neq j \\ 0 & \text{else} \end{cases}$$

and the steady-state solution of the *EMC* $\boldsymbol{\mu}$ will be represented as a vector in $[0,1]^d$, given by $\boldsymbol{\mu}^\top \boldsymbol{P} = \boldsymbol{\mu}^\top, 1 = \sum_{i=1\ldots d} \boldsymbol{\mu}_i$. The solution of the original

process can be computed by taking into account the state sojourn (holding) times $\boldsymbol{h}_i = \frac{1}{\sum_{k \neq i} q_{i,k}} = -\frac{1}{q_{i,i}}$ and normalization, leading to $\boldsymbol{\pi}_i = \frac{\boldsymbol{h}_i \boldsymbol{\mu}_i}{\sum_k \boldsymbol{h}_k \boldsymbol{\mu}_k}$.

In the following we consider this time-discrete, irreducible, aperiodic Markov chain $EMC = (X_t)_{t \in \mathbf{N}_0}$ on a finite state space $\mathcal{S} = \{1, \ldots, d\}$ with time-invariant transition matrix $\boldsymbol{P} \in [0,1]^d$ that results from such an embedding.

The essential aim is then to calculate $\int f \, \mathrm{d}\boldsymbol{\mu}$ for a given $f : \mathcal{S} \to \mathbf{R}$ where f corresponds to the performance measure *rvar* of the model and is expressed as a vector in $\mathbf{R}^d$ such that the integral is given by $\boldsymbol{\mu}^\top f$. If $\boldsymbol{\mu}$ can not be computed explicitly, one may resort to simulation, using that, by the ergodic theorem if $X_0 \sim \boldsymbol{\mu}$,[4]

$$F_T = \frac{1}{T} \sum_{t=1}^{T} f(X_t) \to \int f \, \mathrm{d}\boldsymbol{\mu} \text{ a.s.} \tag{1}$$

when the simulation time $T \to \infty$; note that this implies convergence in quadratic mean as well. Since $\sum_{t=1}^{T} f(X_t)$ can be updated while the Markov chain is simulated, the computation of F_T requires only constant memory.

The analysis of the (multi-particle) simulation needs a mathematical framework which allows to bridge numerical analysis and simulation. For this, recall that an alternative to solving $\boldsymbol{\mu}^\top \boldsymbol{P} = \boldsymbol{\mu}^\top$ in the numerical analysis is to start with some $\boldsymbol{\mu}_0 \in [0,1]^d$ whose entries some to 1, and iteratively compute $\boldsymbol{\mu}_{t+1}^\top = \boldsymbol{\mu}_t^\top \boldsymbol{P}$ until convergence (which is guaranteed by the Perron-Frobenius theorem). We will now describe standard simulation and multi-trajectory algorithm in a similar vector-matrix calculus for which we introduce the following notation: $e_i \in \mathbf{R}^d$, $i = 1, \ldots, d$ will denote the canonical basis vectors of $\mathbf{R}^d$ such that $\delta : \mathcal{S} \to \{e_i : i = 1, \ldots, d\}$, $x \mapsto \delta(x) = e_x$ is a bijective mapping which is interpreted as mapping a state to the Dirac measure on $\mathcal{S}$ at that state. Furthermore, we are going to denote the rows of the matrix $\boldsymbol{P}$ by $\boldsymbol{p}_i \in \mathbf{R}^d$, $i = 1, \ldots, d$ – and analogously for other matrices – such that $\boldsymbol{P} = \sum_{i=1}^{d} e_i \boldsymbol{p}_i^\top$.

Now consider independent random variables $Z_{t,i} \sim \boldsymbol{p}_i$, $t \in \mathbf{N}$, $i = 1, \ldots, d$ and form the matrices $R_t = \sum_{i=1}^{d} e_i \delta(Z_{t,i})^\top$. Then, if $X_0 \sim \boldsymbol{\mu}$ is independent of the $Z_{t,i}$, setting $X_t = \delta^{-1}\Big(\delta(X_{t-1})^\top R_t\Big)$ for $t \in \mathbf{N}$ realizes the stationary Markov chain, i.e. $\delta(X_t)^\top = \delta(X_0)^\top \prod_{s=1}^{t} R_s$ encodes the evolution of the chain through Dirac measures (where here and in the following matrix products expand from left to right). This is identical to a standard simulation that never splits, and serves as the basis for the mathematical formulation of our algorithm.

The splitting can then be modelled as follows: let $D_{t,i} \in \{0,1\}$, $t \in \mathbf{N}$, $i = 1, \ldots, d$ be random decision variables that specify if particle i will be split at time step t or not. We assume them to be random and depending only on the past, i.e., the subsequent proofs will apply to all splitting heuristics which are based only on the history of the process. $D_{t,i}$ may thus depend only on X_0 and R_s for $s < t$; formally, we can form the filtration[5] $\mathcal{F}_0 = \sigma(X_0)$, $\mathcal{F}_t = \sigma(X_0, R_s : s \leq t)$

[4] Assuming that the initial transient phase of the simulation has passed at $t = 0$.

[5] A filtration is a growing sequence of σ-algebras which may be interpreted as containing the information up to the corresponding time point.

and require that $D_{t,i}$ is $\mathcal{F}_{t-1}$-measurable (i.e. predictable given the past) for $t \in \mathbf{N}$. Then, based on the rows $r_{t,i} = \delta(Z_{t,i}) \in \mathbf{R}^d$ of $R_t = \sum_{i=1}^d e_i r_{t,i}^\top$ we define random vectors $s_{t,i} \in \mathbf{R}^d$ by[6]

$$s_{t,i} = \begin{cases} \mathbf{E}\, r_{t,i} = \boldsymbol{p}_i\,, & \text{if } D_{t,i} = 1 \\ r_{t,i}\,, & \text{if } D_{t,i} = 0 \end{cases} \tag{2}$$

for $t \in \mathbf{N}$, $i = 1, \ldots, d$. Setting $S_t = \sum_{i=1}^d e_i s_{t,i}^\top$ as well as $\rho_0 = \delta(X_0)$ and $\rho_t^\top = \rho_{t-1}^\top S_t = \delta(X_0)^\top \prod_{s=1}^t S_s$ for $t \in \mathbf{N}$, we obtain a sequence of random measures on $\mathcal{S}$, like the deterministic measures $\boldsymbol{\mu}_t$ in the numerical analysis, or the random measures $\delta(X_t)$ encoding the Markov chain in the simulation approach. In fact, S_t contains the one-step probabilities for all particles stored in ρ with their weights, including any splits. Only states with $\rho_i > 0$ are actually stored. If all states should be split, $S = \boldsymbol{P}$ as in the numerical analysis above; if no states should be split, $S = R$ as in the simulation approach. We are now interested in the convergence of $\frac{1}{T}\sum_{t=1}^T \int f \,\mathrm{d}\rho_t$ which should be compared with (1) where we have replaced $f(X_t) = \int f \,\mathrm{d}\delta(X_t)$ by $\int f \,\mathrm{d}\rho_t = \rho_t^\top f$.

Note that in case $S_t = \boldsymbol{P}$ for all $t \in \mathbf{N}$, we have convergence to the equilibrium: $\rho_t \to \boldsymbol{\mu}$ for $t \to \infty$, such that $\int f \,\mathrm{d}\rho_t \to \int f \,\mathrm{d}\boldsymbol{\mu}$ and thus also for its Cesàro means one has $\frac{1}{T}\sum_{t=1}^T \int f \,\mathrm{d}\rho_t \to \int f \,\mathrm{d}\boldsymbol{\mu}$.

The intuition regarding the performance of the multi-trajectory simulation is: for the rows $s_{t,i}$ of S_t we have either $s_{t,i} = r_{t,i}$ if we do not split or $s_{t,i} = \boldsymbol{p}_i$ if we do. The latter is deterministic and thus generates no variance, so randomly deciding whether to split or not will lead to a smaller variance than never splitting at all; since $\boldsymbol{p}_i = \mathbf{E}\, r_{t,i}$ and the decision to split depends only on the past, $\mathbf{E}\, s_{t,i} = \boldsymbol{p}_i$ as well.

Since the sequence of measures ρ_t will not be stationary in general, we will not consider almost sure convergence but we will prove $\frac{1}{T}\sum_{t=1}^T \int f \,\mathrm{d}\rho_t \to \int f \,\mathrm{d}\boldsymbol{\mu}$ in quadratic mean by showing the variance to be smaller when splitting. For this, the following lemma which is straightforward to prove provides the essential estimate.

Lemma 1 *Let $u \in \mathbf{R}^d$ be a deterministic vector, let $v, a_i \in \mathbf{R}^d$, $i = 1, \ldots, d$ be a set of independent, square-integrable vectors, for $i = 1, \ldots, d$ set either $b_i = 0 \in \mathbf{R}^d$ or $b_i = a_i$, and form the matrices $A = \sum_{i=1}^d e_i a_i^\top \in \mathbf{R}^{d\times d}$ and $B = \sum_{i=1}^d e_i b_i^\top$, respectively. Furthermore assume $\mathbf{E}\, A = 0$.*

Then $\mathbf{E}\, u^\top A v = \mathbf{E}\, u^\top B v = 0$ and $\mathbf{Var}(u^\top A v) = \mathbf{E}(u^\top A v)^2 \geq \mathbf{E}(u^\top B v)^2 = \mathbf{Var}(u^\top B v)$.

We are now in the position to state and prove our main result.

Proposition 2 *Let $\boldsymbol{P}$ be the (time-invariant) transition matrix of a time-discrete, irreducible, aperiodic Markov chain on a finite state space $\mathcal{S} = \{1, \ldots, d\}$ with steady-state solution vector $\boldsymbol{\mu}$ and fix $f : \mathcal{S} \to \mathbf{R}$. Using the notation*

[6] **E** and **Var** denote expected value and variance of the subsequent terms.

introduced above, let $X_0 \sim \boldsymbol{\mu}$, $Z_{t,i} \sim \boldsymbol{p}_i$, $t \in \mathbf{N}$, $i = 1, \dots, d$ *be independent random variables,* $R_t = \sum_{i=1}^d e_i \delta(Z_{t,i})^\top$, $X_t = \delta^{-1}\Big(\delta(X_{t-1})^\top R_t\Big)$ *such that* X_t, $t \in \mathbf{N}_0$ *realizes the Markov chain,* $\mathcal{F}_0 = \sigma(X_0)$, $\mathcal{F}_t = \sigma(X_0, R_s : s \le t)$, $D_{t,i} \in \{0,1\}$ $\mathcal{F}_{t-1}$*-measurable random variables for* $t \in \mathbf{N}$, $i = 1, \dots, d$, *and let* $S_t = \sum_{i=1}^d e_i s_{t,i}^\top$, $t \in \mathbf{N}$ *be defined via* (2) *as well as* $\rho_t^\top = \delta(X_0)^\top \prod_{s=1}^t S_s$ *for* $t \in \mathbf{N}$.

Then, considering the result $F_T = \frac{1}{T}\sum_{t=1}^T f(X_t)$ *after* $T \in \mathbf{N}$ *steps when simulating the Markov chain, and the result* $G_T = \frac{1}{T}\sum_{t=1}^T \rho_t^\top f$ *of the multi-trajectory simulation, we have* $\mathbf{E}\, F_T = \mathbf{E}\, G_T = \boldsymbol{\mu}^\top f = \int f \,\mathrm{d}\boldsymbol{\mu} = \mathbf{E}\, f(X_0)$ *and* $\mathbf{Var}\, G_T \le \mathbf{Var}\, F_T$ *for all* $T \in \mathbf{N}$. *Thus, from* $\mathbf{Var}\, F_T \to 0$ *for* $T \to \infty$ *one concludes* $G_T \to \int f \,\mathrm{d}\boldsymbol{\mu}$ *in quadratic mean, i.e.* $\mathbf{E}(G_T - \boldsymbol{\mu}^\top f)^2 \to 0$ *for* $T \to \infty$.

Proof. By stationarity of the Markov chain, we have $\mathbf{E}\, F_T = \frac{1}{T}\sum_{t=1}^T \mathbf{E}\, f(X_t) = \mathbf{E}\, f(X_0)$. Also, by independence, $\mathbf{E}(r_{t,i} \,|\, \mathcal{F}_{t-1}) = \mathbf{E}(r_{t,i}) = \boldsymbol{p}_i$, so, due to the $\mathcal{F}_{t-1}$-measurability of $D_{t,i}$,

$$\mathbf{E}(s_{t,i} \,|\, \mathcal{F}_{t-1}) = \mathbb{1}\{D_{t,i} = 1\}\, \mathbf{E}(r_{t,i}) + \mathbb{1}\{D_{t,i} = 0\}\, \mathbf{E}(r_{t,i} \,|\, \mathcal{F}_{t-1}) = \boldsymbol{p}_i,$$

thus $\mathbf{E}(S_t \,|\, \mathcal{F}_{t-1}) = \sum_{i=1}^d e_i\, \mathbf{E}(s_{t,i} \,|\, \mathcal{F}_{t-1}) = \boldsymbol{P}$ which implies

$$\begin{aligned}\mathbf{E}\, \rho_t^\top &= \mathbf{E}\Big(\delta(X_0)^\top \prod_{s=1}^t S_s\Big) = \mathbf{E}\, \mathbf{E}\Big(\delta(X_0)^\top \Big(\prod_{s=1}^{t-1} S_s\Big) S_t \,|\, \mathcal{F}_{t-1}\Big) \\ &= \mathbf{E}\Big(\delta(X_0)^\top \Big(\prod_{s=1}^{t-1} S_s\Big)\, \mathbf{E}(S_t \,|\, \mathcal{F}_{t-1})\Big) = \mathbf{E}\, \delta(X_0)^\top \Big(\prod_{s=1}^{t-1} S_s\Big) \boldsymbol{P} = (\mathbf{E}\, \rho_{t-1})^\top \boldsymbol{P}\end{aligned}$$

for all $t \in \mathbf{N}$ so that $\mathbf{E}\, \delta(X_0) = \boldsymbol{\mu}$. Induction gives $\mathbf{E}\, \rho_t^\top = \boldsymbol{\mu}^\top \boldsymbol{P}^t = \boldsymbol{\mu}^\top$ and thus $\mathbf{E}\, G_T = \frac{1}{T}\sum_{t=1}^T \mathbf{E}\, \rho_t^\top f = \boldsymbol{\mu}^\top f$ which proves that the expected value of our proposed algorithm results will be unbiased.

To derive the variance estimate, we will substitute S_k for R_k in F_T one after the other for $k = 1, \dots, T$, i.e. we set $H_0 = T\, F_T$ as well as

$$H_k = \sum_{t=1}^{k-1} \Big(\delta(X_0)^\top \prod_{s=1}^t S_s\Big) f + \Big(\delta(X_0)^\top \prod_{s=1}^{k-1} S_s\Big) S_k \Big(\mathbf{I} + \sum_{t=k+1}^T \prod_{s=k+1}^t R_s\Big) f$$

for $k = 1, \dots, T$ with the last sum being empty for $k = T$ such that $H_T = T G_T$, and prove $\mathbf{Var}\, H_k \le \mathbf{Var}\, H_{k-1}$ for $k = 1, \dots, T$. For the latter we are of course going to use Lemma 1 applied for the conditional distribution given $\mathcal{F}_{k-1}$.

Observing that (similarly as above) for $k = 1, \ldots, T$, $\mathbf{E}(H_k \,|\, \mathcal{F}_{k-1})$ equals

$$\begin{aligned}
&\sum_{t=1}^{k-1}\left(\delta(X_0)^\top \prod_{s=1}^{t} S_s\right) f + \left(\delta(X_0)^\top \prod_{s=1}^{k-1} S_s\right) \mathbf{E}(S_k \,|\, \mathcal{F}_{k-1})\, \mathbf{E}\left(\mathbf{I} + \sum_{t=k+1}^{T} \prod_{s=k+1}^{t} R_s\right) f \\
=&\sum_{t=1}^{k-1}\left(\delta(X_0)^\top \prod_{s=1}^{t} S_s\right) f + \left(\delta(X_0)^\top \prod_{s=1}^{k-1} S_s\right) \boldsymbol{P} \left(\mathbf{I} + \sum_{t=k+1}^{T} \boldsymbol{P}^{t-k}\right) f \\
=&\sum_{t=1}^{k-1}\left(\delta(X_0)^\top \prod_{s=1}^{t} S_s\right) f + \left(\delta(X_0)^\top \prod_{s=1}^{k-1} S_s\right) \mathbf{E}(R_k \,|\, \mathcal{F}_{k-1})\, \mathbf{E}\left(\mathbf{I} + \sum_{t=k+1}^{T} \prod_{s=k+1}^{t} R_s\right) f \\
=&\,\mathbf{E}(H_{k-1} \,|\, \mathcal{F}_{k-1})
\end{aligned}$$

we may set $u = \delta(X_0)^\top \prod_{s=1}^{t} S_s$, $A = R_k - \boldsymbol{P}$, $B = S_k - \boldsymbol{P}$, and $v = \big(\mathbf{I} + \sum_{t=k+1}^{T} \prod_{s=k+1}^{t} R_s\big) f$ such that A and B have zero (conditional) mean. Lemma 1 (applied conditionally given $\mathcal{F}_{k-1}$) thus gives the desired result $\mathbf{Var}(H_k \,|\, F_{k-1}) \leq \mathbf{Var}(H_{k-1} \,|\, \mathcal{F}_{k-1})$ and therefore also $\mathbf{Var}(H_k) \leq \mathbf{Var}(H_{k-1})$.

Note that Proposition 2 indeed says that both F_T and G_T are unbiased estimators of $\int f \,\mathrm{d}\boldsymbol{\mu}$ with the variance of G_T never being larger than that of F_T; however, from the proofs one may expect the variance of G_T to be considerably smaller than that of F_T if one often splits particles.

In particular, symmetric (asymptotic) confidence intervals around F_T derived from the central limit theorem are also valid if translated to be symmetric around G_T but will be quite conservative. As the sequence ρ_t will not be stationary, deriving sharp (asymptotic) confidence intervals around G_T is non-trivial and left for further research, as is a proof of almost sure convergence.

5 Application Examples and Numerical Results

The algorithm introduced in this paper has been implemented in the software tool TimeNET [6, 19] in a Master's thesis [4]. It was based on the existing simulation module for eDSPNs [8], which follows a master-slave architecture with several (6 as a standard) simulation slave processes running concurrently on the host machine. This improves the statistical independence of samples in addition to the used batch means method, and allows to exploit multi-core CPUs. All Petri net analysis parts and computation of statistical measures from the raw samples are reused, including initial transient detection as well as variance analysis for confidence interval evaluation for the stopping criterion. The latter was not changed for the current experiments; a new variance estimation method that better matches our algorithm will have to be developed in the future. Significant changes and additions to the program had to be done only to store the particles.

The goal of this section is to show exemplarily that the proposed algorithm works well both for models that can be analyzed by standard simulation or by numerical analysis methods as depicted in Fig. 1, i.e., the left and lower regions of the picture. Two example models are evaluated for this reason: the first one

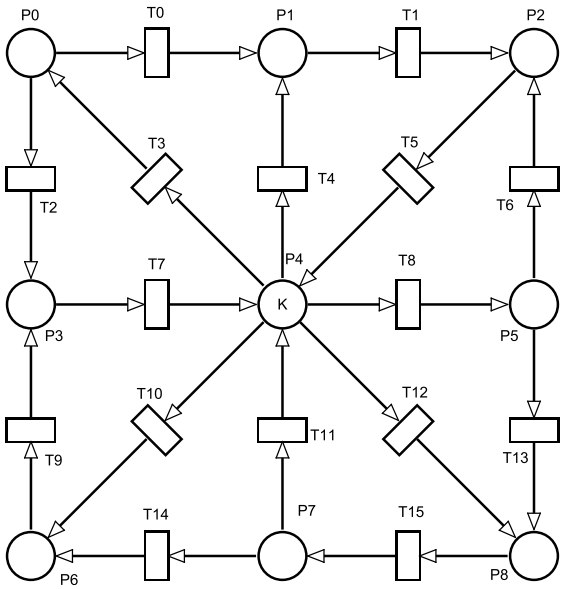

Fig. 2. Example 1: SPN with varying state space size for K

(dotted red, left side of the figure) showing that for models without rare events and differing state space size, where simulation is usually faster than a numerical analysis (at least when the state space size increases), the proposed algorithm has the same advantages as a regular simulation: the run time does not increase substantially with the state space size.The second example (chain-dotted blue, bottom of Fig. 1) explores the horizontal dimension: a model with fixed state space size is used, for which one parameter influences the "rareness" of events of interest. For the chosen case it will be shown that our proposed algorithm has the same advantage as numerical analysis: as long as the state space size is manageable (fits into the main memory), the execution time does not increase for harder problems which lead to very long simulation run times. We explicitly do not cover the area marked "rare-event simulation" in Fig. 1, i.e., models which are problematic both for numerical analysis as well as standard simulation. Our future goal is to extend the algorithm with proper heuristics to let it also solve such large rare-event models efficiently, but this is outside the scope of this paper.

All numerical results have been computed on a standard laptop computer under Windows 7 Enterprise 64 Bit on a 2013 Intel Core i7-4600U CPU running at 2.1 GHz, with 12 GByte RAM, 4MB Cache, and an SSD hard disk. This processor has two cores and four logical processors, which are all used by the simulation slave processes, while the numerical analysis (mainly spending its time during the successive over-relaxation (SOR) iterative solution of global balance equations) can only run on one core. All measured times are run times as experienced by the tool user, not pure CPU times.

Table 1. Numerical results and run times in seconds for the first example

		Numeric analysis		Simulation			Multi-Trajectory		
K	States	Result	Time	Result	Error	Time	Result	Error	Time
1	9	0.22951	1	0.230	0.4%	1	0.229	0.0%	1
3	165	0.34726	1	0.354	2.1%	1	0.347	0.0%	1
5	1 287	0.36208	1	0.361	0.2%	1	0.363	0.1%	2
7	6435	0.36403	1	0.365	0.4%	1	0.364	0.1%	2
10	43 758	0.36431	7	0.367	0.8%	1	0.360	1.3%	9
12	125970	0.36433	26	0.366	0.5%	1	0.361	1.0%	9
15	490 314	0.36433	161	0.367	0.7%	1	0.371	1.9%	9
17	1081575	0.36433	483	0.367	0.8%	1	0.372	2.2%	14
20	3 108 105	0.36433	2028	0.368	0.9%	1	0.365	0.3%	9
25	13 884 156			0.369		1	0.359		9
30	48 903 492			0.367		1	0.351		11

Figure 2 shows an abridged version of a model introduced in [4] to check the applicability of the algorithm to an arbitrary, not too simplistic stochastic Petri net. Transitions are set to either single or infinite server semantics with equal probability. The firing delays of the model have been randomly selected in the ranges $[1 \dots 9]$ and $[100 \dots 900]$ to avoid simplifying symmetries; only one parametrized model instance is considered[7]. The number of states is controlled by parameter K, the initial number of tokens in place `P4`. We assume a performance measure of interest as the probability of having at least one token in place `P1`, being expressed by `P{#P1>0}` in TimeNET 4.3 syntax.

Precision has been increased for numerical analysis in all experiments to 10^{-15} because of some small result values. For normal and multi-trajectory simulation the following settings are used: confidence level 95% and relative error 5%.

The results for the first example are shown in Table 1, all run times are given in seconds. The model has been chosen such that the state space size grows moderately with increasing number of tokens K. Run time of numerical analysis starts to grow considerably when the size exceeds 20.000 states. Several millions of states can be handled until the memory is exceeded for $K = 25$, but the run time rapidly becomes unacceptable compared to the simulation times. Standard simulation is very fast and independent of the state space size. Its results are in the desired range of accuracy. The proposed multi-trajectory algorithm is very fast and almost exact until $K = 7$, which are the cases where the state space fully fits into the chosen number of trajectories set to 10^4. After that, the speed is slower, but does not depend on the size. Accuracy is also in the desired range of 5% relative error.

[7] This and the later second experiment model file as well as more numerical results are available at www.tu-ilmenau.de/sse/timenet/data-for-the-multi-trajectory-algorithm to support reproducibility. TimeNET can be obtained from timenet.tu-ilmenau.de.

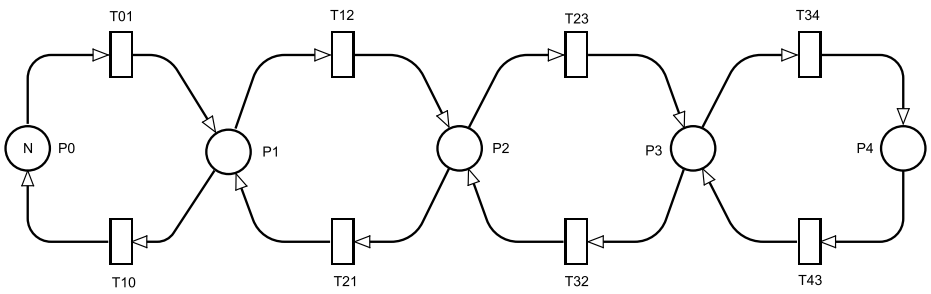

Fig. 3. Example 2: SPN with increasingly hard simulation by adapting transition delays

Table 2. Numerical results and run times in seconds for Example 2

	Numeric analysis		Simulation			Multi-Trajectory		
ϵ	Result	Time	Result	Error	Time	Result	Error	Time
1	8.621E-01	1.3	9.18E-01	6.5%	1	9.37E-01	8.7%	2
10	1.000E-04	1.3	9.95E-05	0.5%	2	1.00E-04	0.0%	3
20	6.250E-06	2.2	6.43E-06	2.9%	26	6.25E-06	0.0%	3
50	1.600E-07	1.4	1.54E-07	3.8%	296	1.60E-07	0.0%	2
70	4.165E-08	1.3	3.96E-08	4.9%	838	4.16E-08	0.0%	3
100	1.000E-08	1.2	1.01E-08	0.8%	2418	1.00E-08	0.0%	3
120	4.823E-09	1.4				4.82E-09	0.0%	2
150	2.603E-09	1.2				2.60E-09	0.0%	3
200	6.250E-10	1.2				6.25E-10	0.0%	2
500	1.600E-11	1.2				1.60E-11	0.0%	2
1000	1.000E-12	1.2				1.00E-12	0.0%	2

The measurements support our claim that the run time of the proposed algorithm does not increase substantially for a growing state space.

Figure 3 shows a second example to check how the algorithm behaves in cases where a standard simulation becomes infeasible. It is based on [4, 20], and the measure of interest is the probability of having at least one token in place `P4`. Computing this value by simulation is made increasingly hard by setting the mean delays of the lower exponential transitions (`T10`, `T21`, `T32`, `T43`) to 1 and the upper transitions' delays to $\epsilon = 1 \ldots 1000$. For greater values of ϵ this is a typical rare-event setting. The number of tokens has been chosen as $N = 25$, resulting in a state space size of 23 751.

Experimental results are shown in Figure 2 with 10^5 maximum particles for the proposed algorithm. Both numerical analysis and multi-trajectory algorithm have no problems in evaluating even hard problems, they are both very fast and the accuracy of the proposed algorithm is perfect except for the case $\epsilon = 1$ which may be due to the stopping criterion. Standard simulation has the expected accuracy, but its run time grows significantly with ϵ. The experiment thus supports the claim that the new algorithm does not suffer from the rare-

event simulation run length as long as the state space of the model is not too large, thus inheriting the advantage of numerical analysis for this case.

Other experiments showed that the proposed algorithm can be slower than standard simulation in some cases with medium-size state space. This happens when a simulation is very fast while the multi-trajectory algorithm spends unnecessary time in the particle computation. In the future we will explore algorithm variants that adapt the maximum number of particles N during run time to overcome this effect, as starting with a small N and gradually increasing it over time ought to solve this issue.

6 Conclusion

A new algorithm for the performance evaluation of Markovian stochastic Petri nets has been proposed in the paper. It uses elements of simulation as well as numerical analysis and its convergence has been proved in a unified mathematical framework. The algorithm can be applied to models for which one up to now had to choose a priori which of the standard methods to apply. The two examples show that the algorithm incorporates the advantages of simulation and numerical analysis for models that are either small enough to be handled analytically, or for which the performance measures and event generation is simple enough to result in a fast simulation.

In the future, we plan to use methods from automatic rare-event Petri net simulation to develop splitting heuristics of the algorithm. The goal is to make the algorithm useful for automated rare-event simulation as well, as a step towards a single method that should be applicable and automatically adapting to the two main cases of model evaluation complexity. We will also investigate if the advantages of the algorithm can be transferred to non-Markovian models, and explore possible parallelization of the algorithm.

7 Acknowledgements

The authors would like to thank Florian Kelma and Thomas Böhme, both from the Institute for Mathematics, Technische Universität Ilmenau, for fruitful discussions on the mathematical treatment.

References

1. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Series in parallel computing, John Wiley and Sons (1995)
2. Ajmone Marsan, M.: Stochastic Petri nets: An elementary introduction. In: Rozenberg, G. (ed.) Advances in Petri Nets 1989, Lecture Notes in Computer Science, vol. 424, pp. 1–29. Springer Verlag (1990)

3. Buchholz, P.: A new approach combining simulation and randomization for the analysis of large continuous time Markov chains. ACM Transactions on Modeling and Computer Simulation 8, 194–222 (1998)
4. Canabal Lavista, A.: Multi-Trajectory Rare-Event Simulation of Stochastic Petri Nets. Master's thesis, Technische Universität Ilmenau (2015)
5. Ciardo, G., German, R., Lindemann, C.: A characterization of the stochastic process underlying a stochastic Petri net. IEEE Transactions on Software Engineering 20, 506–515 (1994)
6. German, R., Kelling, C., Zimmermann, A., Hommel, G.: TimeNET – a toolkit for evaluating non-Markovian stochastic Petri nets. Performance Evaluation 24, 69–87 (1995)
7. Gilmer, Jr., J.B., Sullivan, F.J.: Combat simulation trajectory management. In: Chinni, M. (ed.) Proc. Military, Government, and Aerospace Simulation Conference. pp. 236–241. Society for Computer Simulation, San Diego, California (1996)
8. Kelling, C.: $\text{TimeNET}_{\text{sim}}$ – a parallel simulator for stochastic Petri nets. In: Proc. 28th Annual Simulation Symposium. pp. 250–258. Phoenix, AZ, USA (1995)
9. Kovalchuk, S.V., Boukhanovsky, A.V.: Towards ensemble simulation of complex systems. Procedia Computer Science 51, 532 – 541 (2015)
10. Lazarova-Molnar, S., Horton, G.: Proxel-based simulation of stochastic Petri nets containing immediate transitions. Electronic Notes in Theoretical Computer Science 85(4) (2003)
11. McGeoch, C.: Analysing algorithms by simulation: variance reduction techniques and simulation speedups. ACM Computing Surveys 24(2), 195–212 (1992)
12. Pawlikowski, K.: Steady-state simulation of queueing processes: Survey of problems and solutions. ACM Comput. Surv. 22(2), 123–170 (Jun 1990)
13. Sanders, W.H., Meyer, J.F.: A unified approach for specifying measures of performance, dependability, and performability. In: Avizienis, A., Laprie, J. (eds.) Dependable Computing for Critical Applications, Dependable Computing and Fault-Tolerant Systems, vol. 4, pp. 215–237. Springer Verlag (1991)
14. Schwetman, H.D.: Hybrid simulation models of computer systems. Communications of the ACM 21(9), 718–723 (Sep 1978)
15. Trivedi, K.S.: Probability and Statistics with Reliability, Queuing and Computer Science Applications. Wiley, 2nd edn. (2002)
16. Tuffin, B., Trivedi, K.S.: Implementation of importance splitting techniques in stochastic Petri net package. In: Haverkort, B.R., Bohnenkamp, H.C., Smith, C.U. (eds.) Computer Performance Evaluation, Modelling Techniques and Tools — 11th Int. Conf., TOOLS 2000. Lecture Notes in Computer Science, vol. 1786, pp. 216–229. Springer Verlag, Schaumburg, IL, USA (2000)
17. Villén-Altamirano, M., Villén-Altamirano, J.: RESTART: a method for accelerating rare event simulations. In: Queueing, Performance and Control in ATM. pp. 71–76. Elsevier Science Publishers (1991)
18. Zimmermann, A.: Stochastic Discrete Event Systems. Springer, Berlin Heidelberg New York (2007)
19. Zimmermann, A.: Modeling and evaluation of stochastic Petri nets with TimeNET 4.1. In: Proc. 6th Int. Conf on Performance Evaluation Methodologies and Tools (VALUETOOLS). pp. 54–63. Corse, France (2012)
20. Zimmermann, A., Reijsbergen, D., Wichmann, A., Canabal Lavista, A.: Numerical results for the automated rare event simulation of stochastic Petri nets. In: 11th Int. Workshop on Rare Event Simulation (RESIM 2016). pp. 1–10. Eindhoven, Netherlands (2016)