POWVER

Technical Report 2017-10

Title: **Better Automated Importance Splitting for Transient Rare Events**

Authors: Carlos E. Budde, Pedro R. D'Argenio, Arnd Hartmanns

Report Number: 2017-10

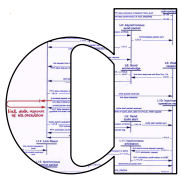
ERC Project: Power to the People. Verified.

ERC Project ID: 695614

Funded Under: H2020-EU.1.1. – EXCELLENT SCIENCE

Host Institution: Universität des Saarlandes, Dependable Systems and Software
Saarland Informatics Campus

Published In: SETTA 2017



POWER TO THE PEOPLE.
VERIFIED.
erc

Better Automated Importance Splitting for Transient Rare Events*

Carlos E. Budde[1], Pedro R. D'Argenio[2,3], and Arnd Hartmanns[1]

[1] University of Twente, Enschede, The Netherlands
[2] Universidad Nacional de Córdoba, Córdoba, Argentina
[3] Saarland University, Saarbrücken, Germany

Abstract Statistical model checking uses simulation to overcome the state space explosion problem in formal verification. Yet its runtime explodes when faced with rare events, unless a rare event simulation method like importance splitting is used. The effectiveness of importance splitting hinges on nontrivial model-specific inputs: an importance function with matching splitting thresholds. This prevents its use by non-experts for general classes of models. In this paper, we propose new method combinations with the goal of fully automating the selection of all parameters for importance splitting. We focus on transient probabilities, which particularly challenged previous techniques, and present an exhaustive practical evaluation of the new approaches on case studies from the literature. We find that using RESTART simulations with a compositionally constructed importance function and thresholds determined via a new *expected success* method most reliably succeeds and performs very well. Our implementation within the MODEST TOOLSET supports various classes of formal stochastic models and is publicly available.

1 Introduction

Nuclear reactors, smart power grids, automated storm surge barriers, networked industrial automation systems: We increasingly rely on critical technical systems and infrastructures whose failure would have drastic consequences. It is imperative to, in the design phase, perform a quantitative evaluation based on a formal stochastic model, e.g. on extensions of continuous-time Markov chains (CTMC), stochastic Petri nets (SPN), or fault trees. Only if the probability of failure can be shown to be sufficiently low can the system design be implemented. Calculating such probabilities—typically on the order of 10^{-9} to 10^{-19}—is challenging: For finite-state Markovian models or probabilistic timed automata (PTA [21]), probabilistic model checking can numerically approximate the desired probabilities, but the state space explosion problem limits it to small models. For other models, in particular those involving events governed by general continuous probability distributions, model checking techniques only exist for specific subclasses with limited scalability [23] or merely compute probability bounds [13].

* This work is supported by the 3TU.BSR project, ERC grant 695614 (POWVER), the NWO SEQUOIA project, and SeCyT-UNC projects 05/BP12 and 05/B497.

Statistical model checking (SMC [16,29]), i.e. using Monte Carlo simulation with formal models, has become a popular alternative for large models and formalisms not amenable to model checking. It trades memory usage for runtime: the number of simulation runs explodes with the desired precision. When an event's true probability is 10^{-19}, for example, we may want to be confident that the error of our estimation is at most on the order of 10^{-20}. *Rare event simulation* (RES [25]) methods have been developed to attack this problem. They increase the number of simulation runs that reach the rare event and adjust the statistical evaluation accordingly. The main RES methods are *importance sampling* and *importance splitting*. The former modifies probabilities in the model to make the event more likely. The challenge lies in finding a good such *change of measure*. Importance splitting instead performs more simulation runs, which however may start from a non-initial state and end early. Here, the challenge is to find an *importance function* that assigns to each state a value indicating how "close" it is to the rare event. More (partial) runs will be started from states with higher importance. Additionally, depending on the concrete splitting method used, *thresholds* (the subset of importance values at which to start new runs) and splitting *factors* (how many new runs to generate at each threshold) need to be chosen. The performance of RES varies drastically with the choices made for these parameters. The quality of a choice of parameters highly depends on the model at hand; making good choices requires an expert in the system domain, the modelling formalism, and the selected RES method.

Aligning RES with the spirit of (statistical) model checking as a "push-button" approach requires methods to automatically select (usually) good parameters. These methods must not negate the memory usage advantages of SMC. Between importance sampling and splitting, the latter appears more amenable to automatic approaches that work well across modelling formalisms (CTMC, PTA, etc.). We previously proposed a compositional method to automatically construct an importance function [5]. Its compositionality is the key to low memory usage. Our FIG tool [4] for RES of input-output stochastic automata (IOSA [9]) implements this method together with the RESTART splitting algorithm [26], thresholds computed via a sequential Monte Carlo (SEQ) approach [7,4], and a single fixed splitting factor specified by the user for all thresholds. Experimental results [4] show that FIG works well for steady-state measures, but less so for transient probabilities. In particular, runtime varies significantly between tool invocations due to different thresholds being computed by SEQ, and the optimal splitting factor varies significantly between different models.

Our contributions. In this paper, we investigate several alternative combinations of splitting and threshold/factor selection algorithms with the goal of improving the automation, robustness and performance of importance splitting for RES in SMC. We keep the compositional method for automatic importance function construction as implemented in FIG. Aside from RESTART, we consider the fixed effort [10] and fixed success [22,25] splitting methods (Section 3). While RESTART was proposed for steady-state measures and only later extended to transient probabilities [27], the latter two are designed for estimating transient

probabilities. For threshold selection, we specify a new "expected success" (EXP) technique as an alternative to SEQ (Section 4). EXP selects thresholds *and* individual splitting factors for each threshold, removing the need for the user to manually select a global splitting factor. We implemented all techniques in the modes simulator of the MODEST TOOLSET [15]. They can be freely combined, and work for all the formalisms supported by modes—including CTMC, IOSA, and deterministic PTA. Our final and major contribution is an extensive experimental evaluation (Section 5) of the various combinations on six case studies.

Related work. A thorough theoretical and empirical comparison of variants of RESTART is presented in [10], albeit in a non-automated setting. Approaching the issue of automation, Jégourel et al. [18,19] use a layered restatement of the formula specifying the rare event to build an importance function for use with adaptive multilevel splitting [6], the predecessor of SEQ. Garvels et al. [11] derive importance functions for finite Markov chains from knowledge about their steady-state behaviour. For SPN, Zimmermann and Maciel [30] provide a monolithic method, though limited to a restricted class of models and throughput measures [31]. Importance *sampling* has been automated for SPN [24] restricted to Markovian firing delays and a global parameterisation of the transition intensities [31]. The difficulties of automating importance sampling are also illustrated in [17]: the proposed automatic change of measure guarantees a variance reduction, yet is proved for stochastic behaviour described by integrable products of exponentials and uniforms only. We do not aim at provable improvements in specific settings, but focus on general models and empirically study which methods work best in practice. We are not aware of other practical methods for, or comparisons of, automated splitting approaches on general models.

2 Preliminaries

We write $\{| \dots |\}$ for multisets, in contrast to sets written as $\{\dots\}$. $\mathbb{N}$ is the set of natural numbers $\{0, 1, \dots\}$ and $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. In our algorithms, operation S.remove() returns and removes an element from the set S. The element may be picked according to any policy (e.g. uniformly at random, in FIFO order, etc.).

2.1 Simulation Models

We develop RES approaches that can work for any stochastic formalism combining discrete and continuous state. We thus use an abstract notion of models:

Definition 1. *A (simulation)* model $\mathtt{M}$ *is a discrete-time Markov process whose states consist of a discrete and (optionally) a continuous part. It has a fixed initial state that can be obtained as* $\mathtt{M}$.initial(). *Operation* $\mathtt{M}$.step(s) *samples a path from state s and returns the path's next state after one index time unit.*

Example 1. A CTMC M_{ctmc} is a continuous-time stochastic process. We can cast it as a simulation model $\mathtt{M}_{sim}$ by using the number of transitions taken as

the (discrete) index time of $\mathtt{M}_{sim}$. Thus, given a state s of M_{ctmc}, $\mathtt{M}_{sim}.\mathrm{next}(s)$ returns the first state s' of M_{ctmc} encountered after taking a single transition from s on a sample path. In effect, we follow the embedded discrete-time Markov chain. Only if the event of interest refers to time do we also need to keep track of the global elapsed (continuous) time as part of the states of $\mathtt{M}_{sim}$.

We require models to be Markov processes. For formalisms with memory, e.g. due to the use of general continuous probability distributions, we encode the memory (e.g. values and expiration times of clocks in the case of IOSA) in the state space. We compute transient probabilities, or more precisely the probability to reach a set of target states while avoiding another disjoint set of states:

Definition 2. *A* transient property $\phi \in S \to \{\, \mathit{true}, \mathit{false}, \mathit{undecided} \,\}$ *for a model with state space S maps target states to true, states to be avoided to false, and all other states to undecided. We require that the probability of reaching a state where ϕ returns true or false is* 1 *from a model's initial state.*

To determine whether a sample path satisfies ϕ, evaluate ϕ sequentially for every state on the path and return the first outcome that is not *undecided*. Standard SMC/Monte Carlo simulation generates a large number n of sample paths to estimate the transient probability p as $\hat{p} \stackrel{\text{def}}{=} \frac{n_{true}}{n}$ (where n_{true} is the number of paths that satisfied ϕ) and reports a confidence interval around the estimate with a specified confidence level. This corresponds to estimating the value of the until formula $\mathtt{P}_{=?}(\neg\, \mathit{avoid}\ \mathtt{U}\ \mathit{target})$ in a logic like PCTL as used in e.g. PRISM [20] for state predicates *avoid* and *target*. Time-bounded until $\mathtt{U}_{\leq b}$ is encoded by tracking the elapsed time t_{global} in states and including $t_{global} > b$ in *avoid*.

2.2 Ingredients of Importance Splitting

Importance splitting performs "more simulation" for states "close" to the target set. Closeness is represented by an *importance function* $f_I \in S \to \mathbb{N}$ that maps each state to its importance in $\{\, 0, \ldots, \max f_I \,\}$. To simplify our presentation, we assume that $f_I(\mathtt{M}.\mathrm{initial}()) = 0$, $\phi(s_{target}) \Rightarrow f_I(s_{target}) = \max f_I$, and if $s' := \mathtt{M}.\mathrm{next}(s)$, then $|f_I(s) - f_I(s')| \leq 1$. These assumptions can easily be removed. The performance, but not the correctness, of all importance splitting methods hinges on the availability of a good importance function. Traditionally, it is specified ad hoc for each model domain by a RES expert [10,25,26]. Methods to automatically compute one [11,18,19,30] are usually specialised to a specific formalism or a particular model structure, potentially providing guaranteed efficiency improvements. We build on the method of [5] that is applicable to any stochastic compositional model with a partly discrete state space. It does not provide mathematical guarantees of performance improvements, but is aimed at generality and providing "usually good" results with minimal user input.

Compositional f_I. A compositional model is a parallel composition of components $\mathtt{M} = \mathtt{M}_1 \parallel \ldots \parallel \mathtt{M}_n$. Each component can be seen as a model on its own,

but the components may interact, usually via some synchronisation/handshaking mechanism. We write the projection of state s of $\mathtt{M}$ to the discrete local variables of component $\mathtt{M}_i$ as $s|_i$. The compositional method works as follows:

1. Convert the target set formula *target* to negation normal form (NNF) and associate each literal $target^j$ with the component $\mathtt{M}(target^j)$ whose local state variables it refers to. Literals must not refer to multiple components.
2. Explore the *discrete part* of the state space of each component $\mathtt{M}_i$. For each $target^j$ with $\mathtt{M}_i = \mathtt{M}(target^j)$, use reverse breadth-first search to compute the local minimum distance $f_i^j(s|_i)$ of each state $s|_i$ to a state satisfying $target^j$.
3. In the syntax of the NNF of *target*, replace every occurrence of $target^j$ by $f_i^j(s|_i)$ with i such that $\mathtt{M}_i = \mathtt{M}(target^j)$, and every Boolean operator $\wedge$ or $\vee$ by $+$. Use the resulting formula as the importance function $f_I(s)$.

Full implementation details can be found in [4]. Other operators can be used in place of $+$, e.g. max or multiplication. Aside from the choice of operator, with $+$ as default since it works well for most models, the procedure requires no user input. It takes into account both the structure of the target set formula and the structure of the state space. Memory usage is determined by the number of discrete local states (required to be finite) over all components. Typically, component state spaces are small even when the composed state space explodes.

Levels, thresholds and factors. Given a model and importance function f_I, importance splitting could spawn more simulation runs whenever the current sample path moves from a state with importance i to one with importance $j > i$. Using the compositional approach, the probability of visiting a state with a higher importance is often close to 1 for many of the i, so splitting on every increment would lead to excessively many (partial) runs and high runtime. One thus partitions the importances into a set of sequentially numbered intervals called *levels*. This results in a *level function* $f_L \in S \to \mathbb{N}$ where, again, the initial state is on level 0 and all target states are on the highest level $\max f_L$. We also refer to the boundary between the highest importance of level $l-1$ and the lowest importance i of level l as the *threshold* T_l, identified by i. Some splitting methods can be further parameterised by the "amount of splitting" at each threshold; we will use *splitting factor* and *effort* functions f_S resp. f_E in $\mathbb{N} \to \mathbb{N} \setminus \{0\}$ that map each level to a positive natural number for this purpose.

3 Splitting Methods

We now briefly describe, from a practical perspective, the three different approaches to importance splitting that we implemented and evaluated.

3.1 Restart

Originally discovered in 1970 [2] and later popularised by J. and M. Villén-Altamirano [26], the Restart method for importance splitting was first designed for steady-state measures and later extended to transient properties [27]. Restart works by performing one main simulation run from the initial state.

Input: model `M`, level function f_L, splitting factors f_S, transient property ϕ

```
1  S := {| ⟨M.initial(), 0⟩ |}, p̂ := 0          // start with the initial state from level 0
2  while S ≠ ∅ do                               // perform main and child runs (RESTART loop)
3      ⟨s, l⟩ := S.remove(), l_create := l      // get next split and store creation level
4      while φ(s) = undecided do                // run until property decided (simulation loop)
5          s := M.step(s)                       // simulate up to next change in discrete state
6          if f_L(s) < l_create then break      // moved below creation level: kill run
7          else if f_L(s) > l then              // moved one level up: split run
8              l := f_L(s), S := S ∪ {| ⟨s, l⟩, ...(f_S(l) times)..., ⟨s, l⟩ |}
9      if φ(s) then p̂ := p̂ + 1/∏_{i=1}^{l} f_S(l)   // update result if we hit the rare event
10 return p̂
```

Algorithm 1: The RESTART method for importance splitting

As soon as any run crosses a threshold from below, a number of child runs are started from the first state in the new level l (the run is split). That number is determined by l's splitting factor: $f_S(l) - 1$ child runs are started, resulting in $f_S(l)$ runs that continue after splitting. Each run is tagged with the level on which it was started. As soon as a run crosses a threshold from above into a level below its creation level, it ends (the run is killed). A run also ends when it reaches a state satisfying *avoid* or *target*. We state RESTART formally as Algorithm 1. An illustration of its behaviour is shown in Figure 1. The horizontal axis is the model's index time while the vertical direction shows the current state's importance. Target states are marked ✓and *avoid* states are marked ✗. We have three levels with thresholds at importances 3 to 4 and 9 to 10. f_S is $\{\, 1 \mapsto 3, 2 \mapsto 2 \,\}$.

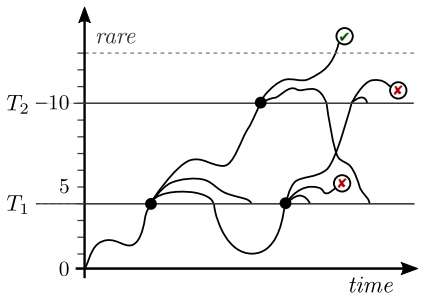


Figure 1. RESTART

The result of a RESTART run—consisting of a main and several child runs—is the weighted number of runs that reach *target*. Each run's weight is 1 divided by the product of the splitting factors of all levels. The result is thus a positive rational number. Note that this is in contrast to standard Monte Carlo simulation, where each run is a Bernoulli trial with outcome 0 or 1. This affects the statistical analysis on which the confidence interval over multiple runs is built.

While RESTART may appear simple at first glance, it is very carefully designed such that the mean of the results of many RESTART runs is actually an unbiased estimator for the true transient probability [27,28] . In particular, the weight of a successful run is the same no matter whether it went straight up through all levels to the target, or first moved up and down between levels several times, creating many new child runs on the way. Only the fact that, after splitting,

Input: model M, level function f_L, effort function f_E, transient property ϕ

```
1  L := { 0 ↦ [ S := { M.initial() }, n := 0, up := 0 ] }        // set up data for level 0
2  for l from 0 to max f_L do                                    // iterate over all levels from initial to target
3      for i from 1 to f_E(l) do                                 // perform sub-runs on level (fixed effort loop)
4          s :∈ L(l).S, L(l).n := L(l).n + 1                     // pick from the level's initial states
5          while φ(s) = undecided do                             // run until φ is decided (simulation loop)
6              s := M.step(s)                                    // simulate up to next change in discrete state
7              if f_L(s) > l then                                // moved one level up: end sub-run
8                  L(l).up := L(l).up + 1                        // level-up run for current level
9                  L(f_L(s)).S := L(f_L(s)).S ∪ { s }            // initial state for next level
10                 break
11         if φ(s) then L(l).up := L(l).up + 1 // hit rare event (highest level only)
12     if L(l).up = 0 then return 0                              // we cannot reach the target any more
13 return ∏_{i=0}^{max f_L} L(l).up/L(l).n   // multiply conditional level-up prob. estimates
```

Algorithm 2: The fixed effort method for importance splitting

exactly one of the resulting runs can survive moving a level downward results in a correct estimation: over many RESTART runs, underestimation caused by runs that die when going down is compensated by overestimation from the one that survives and is later split again.

3.2 Fixed Effort

In contrast to RESTART, each run of the *fixed effort* method [10,12] performs a fixed number $f_E(l)$ of partial runs on each level l. Each of these ends when it either crosses a threshold from below into level $l+1$, encounters a *target* state, or encounters an *avoid* state. We count the first two cases as n^l_{up}. In the first case, the new state is stored in a set of initial states for level $l+1$. When all partial runs for level l have ended, the algorithm moves to level $l+1$, starting the next round of partial runs from the previously collected initial states of the new level. This behaviour is illustrated in Figure 2 (with $f_E(l) = 5$ for all levels) and formally stated as Algorithm 2. The initial state of each partial run can be chosen randomly, or simply in a round-robin fashion among the available initial states [10]. When a fixed effort run ends, the fraction of partial runs started in level l that moved up is an approximation of the conditional probability of reaching level $l+1$ given that level l was reached. Since *target* states exist only on the highest level, the overall

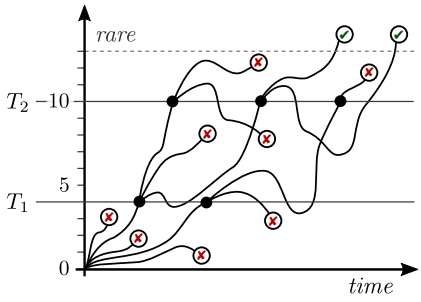


Figure 2. Fixed effort

result is thus simply the product of the fraction $n^l_{up}/f_E(l)$ for all levels l, i.e. a rational number in the interval $[0,1]$. The average of the result of many fixed effort runs is again an unbiased estimator for the transient probability [12].

The fixed effort method has been specifically designed for transient properties. Its advantage is predictability in the sense that each run involves performing (at most) $\prod_{l=0}^{\max f_L} f_E(l)$ partial runs. Like splitting levels need to be provided to RESTART via function f_S, the fixed effort method needs the effort function f_E that determines the number of partial runs to start for each level.

3.3 Fixed Success

Fixed effort intuitively controls the simulation effort by adjusting the estimator's imprecision. The fixed success method [1,22] turns this around: its parameters control the imprecision, but the effort then varies. Instead of launching a fixed number of partial runs per level, fixed success keeps launching such runs until $f_E(l)$ of them have reached the next level (or a *target* state in case of the highest level). Illustrated in Figure 3 (with $f_E(l) = 4$ for all levels), the algorithmic steps are as in Algorithm 2 except for two changes: First, the **for** loop in line 3 is replaced by a **while** loop with condition $L(l).up < f_E(l)$. Second, the final return statement in line 13 uses a different estimator: instead of $\prod_{i=0}^{\max f_L} \frac{L(l).up}{L(l).n}$, we have to return $\prod_{i=0}^{\max f_L} \frac{L(l).up-1}{L(l).n-1}$. This is due to the underlying negative binomial distribution; see [1] for details. We thus have to require $f_E(l) \geq 2$ for all levels l.

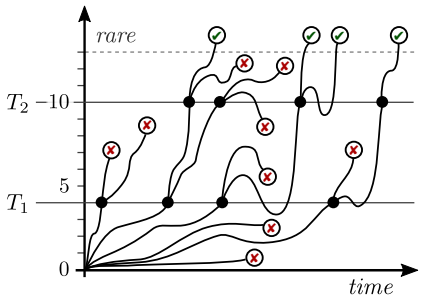


Figure 3. Fixed success

From the automation perspective, the advantage of fixed success is that it self-adapts to the (a priori unknown) probability of levelling up: if that probability is low for some level, more partial runs will be generated on it, and vice-versa. However, the desired number of successes still needs to be specified. 20 is suggested as a starting point in [1], but for a specific setting already. A disadvantage of fixed success is that it is not guaranteed to terminate: If the model, importance function and thresholds are such that, with positive probability, it may happen that all initial states of some level lie in a bottom strongly connected component without *target* states, then the (modified) loop of line 3 of the algorithm will diverge. We have not encountered this situation in our experiments, though.

4 Determining Thresholds and Factors

To determine the splitting levels/thresholds, we implement and compare two approaches: the sequential Monte Carlo (SEQ) method from [4] and a new technique that tries to ensure a certain expected number of runs that level up.

Input: model M, importance function f_I, transient property ϕ, n, $k \in \mathbb{N}^+$, $k < n$

```
1  for i from 1 to n + k do S(i) := M.initial()                  // set up state vector
2  T.push(0)                                                     // stack of selected threshold importances
3  while T.top() < max f_I do                                    // find upper threshold for every importance
4      for i from 1 to n do                                      // first set of runs: find importance distribution
5          s := S(i)
6          while φ(s) = undecided do
7              s := M.step(s)
8              if f_I(s) > f_I(S(i)) then S(i) := s           // keep most important state
9      sort S(i) for i ∈ {1, ..., n} according to f_I          // sort first n states, ascending
10     if T.top() ≥ f_I(S(n − k)) then break                   // found no more important state
11     T.push(f_I(S(n − k)))                                   // new threshold at (n−k)/n importance quantile
12     for i from 1 to n do                                    // second set of runs: initial states for next round
13         j := sample uniformly from {n + 1, ..., n + k}, S(i) := S(j)
14         while φ(S(i)) = undecided ∧ f_I(S(i)) < T.top() do S(i) := M.step(S(i))
15         if f_I(S(i)) < T.top() then goto line 13            // did not reach new threshold
16     for j from n + 1 to n + k do                            // randomly select k initial states
17         i := sample uniformly from {1, ..., n}, S(j) := S(i)
18 for l from T.top() to max f_I do T.push(l)                  // fill in missing thresholds
19 return T                                                    // set of threshold importances characterising the levels
```

Algorithm 3: The sequential Monte Carlo method for threshold selection

4.1 Sequential Monte Carlo

Our first approach is inspired by the sequential Monte Carlo splitting technique [7]. As shown in Algorithm 3, it works in two alternating phases: First, n simulation runs determine the importances that can be reached from the current level (starting with the model's initial state as level 0), keeping track of the state of maximum importance for each run. We then sort the collected states and pick the importance of the one at position $n - k$, i.e. the $\frac{n-k}{n}$ importance quantile, as the start of the next level. This means that as parameter k grows, so does the width of the levels, and the probability of moving from one level to the next decreases. In the second phase, the algorithm randomly selects k new initial states that lie just beyond the newfound threshold via more simulation runs, then proceeds to the next round to compute the next threshold from there. The result is a sequence of importances characterising a level function.

This SEQ algorithm only determines the splitting levels. It does not decide on splitting factors, which the user must select if they wish to run RESTART. The approach implemented in FIG and modes is to request a fixed splitting factor g and then run SEQ with $k = n/g$. When used with fixed effort and fixed success, we set $k = n/2$ and use a user-specified effort value e, i.e. $f_E(l) = e$ for all levels l. A value for n must also be specified; by default, we use $n = 1000$. It is clear that the degree of automation offered by SEQ is not satisfactory. Furthermore, we found in previous experiments with FIG that the levels computed by differ-

Input: model `M`, importance function f_I, transient property ϕ, $n \in \mathbb{N}^+$

1 $f_L := f_I$, $f_E := \{ l \mapsto n \mid l \in \{0, \ldots, \max f_I\} \}$
2 $m := 0$, $e := 0$, $p_{up} := \{ l \mapsto 0 \mid l \in \{0, \ldots, \max f_I\} \}$
3 **while** $p_{up}(\max f_I) = 0$ **do** *// roughly estimate the level-up probabilities*
4 $\quad m := m + 1$, $L :=$ level data computed in one fixed effort run (Algorithm 2)
5 $\quad$ **for** l **from** 0 **to** $\max f_I$ **do** $p_{up}(l) := p_{up}(l) + \frac{1}{m}(L(l).up/L(l).n - p_{up}(l))$
6 **for** l **from** 0 **to** $\max f_I$ **do** *// turn level-up probabilities into splitting factors*
7 $\quad split := 1/p_{up}(l) + e$, $F(l) := \lfloor split + 0.5 \rfloor$, $e := split - F(l)$
8 **return** F *// if $F(l) > 1$, then l is a threshold and $F(l)$ the splitting factor*

Algorithm 4: The expected success method for threshold and factor selection

ent SEQ runs could differ significantly, leading to a large variance in RESTART performance [4]. Combined with mediocre results for transient properties, this was the main trigger for the work we present in this paper.

SEQ may get stuck in the same way as fixed success, and we did encounter this with our *wlan* case study (see Section 5). Our implementation thus restarts SEQ whenever it does not terminate within 30 s; on the *wlan* model, SEQ always succeeded with at most two retries in our experiments.

4.2 Expected Success

To replace SEQ, we propose a new approach based on the rule-of-thumb that one would like the expected number of runs that move up on each level to be 1. This rule is called "balanced growth" by Garvels [12]. The resulting procedure, shown as Algorithm 4, is conceptually much simpler than SEQ: We first perform fixed effort runs, using constant effort n and each importance as a level, until the rare event is encountered. We extract the approximations of the conditional level-up probabilities computed inside the fixed effort runs, averaging the values if we need multiple runs (line 5). After that, we set the factor for each importance to one divided by the (very rough) estimate of the respective conditional probability computed in the first phase. Since splitting factors are natural numbers, we round each factor, but carry the rounding error to the next importance. In this way, even if the exact splitting factors would all be close to 1, we get a rounded splitting factor of 2 for some of the importances. The result is a mapping from importances to splitting factors, characterising both the level function f_L—every importance with a factor $\neq 1$ starts a new level—and the splitting function f_S. We call this procedure the *expected success* (ES) method. Aside from the choice of n (for which our default of $n = 256$ has worked well in all experiments), it provides full automation with RESTART. To use it with fixed effort, we need a user-specified base effort value e, and then set f_E to $\{ l \mapsto e \cdot f_S(l) \mid l \in \{0, \ldots, \max f_L\} \}$ resulting in a *weighted fixed effort* approach.

We also experimented with expected numbers of runs that move up of 2 and 4, but these almost always lead to dismal performance or timeouts due to too many splits or partial runs, so we do not consider them any further.

5 Experimental Evaluation

The purpose of our work was to find a combination of RES methods that provides consistent good performance at a maximal degree of automation. We thus implemented the compositional importance function generation, the splitting methods described in Section 3, and the threshold calculation methods of Section 4 in the modes simulator of the MODEST TOOLSET [13]. This allowed us to study CTMC queueing models, network protocols modelled as PTA, and a more complex fileserver setting modelled as a stochastic timed automaton (STA [3]) using a single tool. We evaluated the performance of all relevant combinations of the implemented RES methods on all of these models.

5.1 Case Studies

We consider the following six case studies:

tandem Tandem queueing networks are a standard benchmark in probabilistic model checking as well as RES [**?**]. We consider the case from [5] with all exponentially distributed interarrival times (a CTMC). The arrival rate into the first queue is $\lambda = 3$ and its service rate is $\mu_1 = 2$. After that, customers move into the second queue, which initially contains one customer, to be served at rate $\mu_2 = 6$. The model has one parameter C: the capacity of each queue. We estimate the value of the transient property $\texttt{P}_{=?}(q_2 > 0 \texttt{ U } q_2 = C)$, i.e. of the second queue becoming full without having been empty before.

openclosed Our second model [**?**], again a CTMC, consists of two *parallel* queues: an *open queue* q_0, receiving packets at rate $\lambda = 1$ from an external source, and a *closed queue* q_c that receives internal packets. One server processes packets from both queues: packets from q_0 are processed at rate $\mu_{11} = 4$ while q_c is empty; otherwise, packets from q_c are served at rate $\mu_{12} = 2$. The latter packets are put back into another internal queue, which are independently moved back to q_c at rate $\mu_2 = \frac{1}{2}$. We study the system as in [4] with a single packet in internal circulation, i.e. an M/M/1 queue with server breakdowns, and parameter B of the capacity of q_0. We estimate $\texttt{P}_{=?}(\neg \mathit{reset} \texttt{ U } \mathit{lost})$: the probability that q_0 overflows before a packet is processed from q_0 or q_c such that the respective queue becomes empty again.

breakdown The final queueing system that we consider [**?**] as a CTMC consists of ten sources of two types, five of each, that produce packets at rate $\lambda_1 = 3$ (type 1) or $\lambda_2 = 6$ (type 2), periodically break down with rate $\beta_1 = 2$ resp. $\beta_2 = 4$ and get repaired with rate $\alpha_1 = 3$ resp. $\alpha_2 = 3$. The produced packets are collected in a single queue, attended to by a server with service rate $\mu = 1000$, breakdown rate $\gamma = 3$ and repair rate $\delta = 4$. Again, and as in [5], we parameterise the model by the queue's capacity, here denoted K, and estimate $\texttt{P}_{=?}(\neg \mathit{reset} \texttt{ U } \mathit{buf} = K)$: starting from a single packet in the queue, what is the probability for the queue to overflow before it was empty?

brp Taking advantage of the wide modelling formalism support of the MODEST TOOLSET, we also study two PTA examples using the MODEST models ori-

ginally introduced in [14]. The first is of the bounded retransmission protocol, another classic benchmark in formal verification. We use one parameter parameter M that determines the actual parameters N (the number of chunks of the file to transmit), MAX (the retransmission bound) and TD (the transmission delay) by way of $\langle N, MAX, TD\rangle = \langle 16 \cdot 2^M, 4 \cdot M, 4 \cdot 2^M\rangle$. We consider the large instances $\langle 32, 4, 8\rangle$, $\langle 64, 8, 16\rangle$ and $\langle 128, 12, 32\rangle$. To avoid nondeterminism, TD is both lower and upper bound for the transmission delay. We estimate $\mathtt{P_{=?}}(\mathit{true}\ \mathtt{U}\ s_{nok} \wedge i > \frac{N}{2})$, i.e. the probability that the sender eventually reports an unsuccessful transmission after more than half of the chunks have been sent successfully.

wlan The second PTA model is of two stations communicating wirelessly via IEEE 802.11 wireless LAN. In contrast to [14] and the original PRISM case study the model was derived from, we use the exact timing parameters described by the standard, which lead to a model too large for standard probabilistic model checkers, and a stochastic semantics of the PTA, scheduling events as soon as possible and resolving all other nondeterminism uniformly. The model has one parameter K, the maximum value of the backoff counter in the exponential backoff procedure. We estimate $\mathtt{P_{=?}}(\mathit{true}\ \mathtt{U}\ bc_1 = bc_2 = K)$, the probability that both station's backoff counters reach K.

fileserver Our last case study combines exponentially and uniformly distributed delays plus discrete probabilistic choices. It is a STA model of a file server where a fraction of the files is archived and requires significantly more time to retrieve. Introduced in [13], we change the archive access time from nondeterministic to continuously uniform over the same interval. Model parameter C is the server's queue size. We estimate the time-bounded probability of queue overflow: $\mathtt{P_{=?}}(\mathit{true}\ \mathtt{U}_{\leq 1000}\ \mathit{queue} = C)$.

Several of our case studies are queueing systems, reflecting the fact that they are very frequently used benchmarks for RES [?]. The three CTMC models could easily be modified to use other distributions and our techniques and tools would still work the same.

5.2 Experimental Setup

We ran experiments on two different machines: Those for the *tandem* and *wlan* models were performed on a four-core Intel Core i5-6600T (2.7/3.5 GHz) system running 64-bit Windows 10 v1607 x64 using three simulation threads. All other experiments ran on a six-core Intel Xeon E5-2620v3 (2.4/3.2 GHz, 12 logical processors) system running Mono 5.2 on 64-bit Debian v4.9.25 using five simulation threads each for two separate experiments running concurrently. We used a timeout of 600 s for the *tandem*, *openclosed* and *brp* models and 1200 s for all others. Simulations were run until the half-width of the 95 % normal confidence interval was at most 10 % of the currently estimated mean. By this use of a relative width, the precision automatically adapted to the rareness of the event. We also performed SMC/Monte Carlo simulation as a comparison baseline, where modes uses the Agresti-Coull approximation of the binomial confidence interval.

Table 1. Model data and performance results

model/param		$\hat{p}$	n_I	SMC	RESTART 2	4	8	16	ES	fixed effort 16	64	256	-weighted 8	16	128	fixed success 8	32	128
tandem	8	5.6E−6	22	70	3	1	1	11	1	1	1	1	1	1	1	1	1	1
	12	1.9E−8	30	—	45	1	10	190	1	5	4	3	3	2	1	6	2	2
	16	7.1E−11	38	—	—	3	177	588	2	18	8	6	11	6	4	18	7	5
	20	3.0E−13	46	—	—	5	—	—	4	124	23	14	84	21	12	59	17	12
open-	20	3.9E−8	155	—	2	142	3	2	1	5	3	2	6	4	2	5	3	3
closed	30	8.8E−12	235	—	5	—	21	7	1	19	9	9	46	19	6	24	8	8
	40	2.0E−15	315	—	19	—	89	15	3	105	24	17	360	72	14	133	19	20
	50	4.6E−19	395	—	74	—	—	85	4	404	45	33	—	167	38	284	47	34
break-	40	4.6E−4	193	46	7	7	8	11	4	10	10	16	15	13	7	11	9	15
down	80	3.7E−7	353	—	33	24	29	40	23	73	51	61	194	112	44	87	52	54
	120	3.0E−10	513	—	80	59	67	97	104	397	149	173	687	283	139	312	182	136
	160	2.4E−13	673	—	316	109	121	175	583	794	377	290	—	—	335	999	421	313
brp	1	3.5E−7	2 k	—	—	—	413	86	21	110	36	33	856	435	226	27	21	50
	2	5.8E−13	6 k	—	—	—	—	—	81	—	423	184	—	—	—	208	141	235
	3	9.0E−19	16 k	—	—	—	—	—	216	—	—	—	—	—	—	—	420	569
wlan	4	2.2E−5	14 k	376	—	—	—	—	—	57	38	31	120	131	221	44	36	39
	5	1.6E−7	23 k	—	—	—	—	—	—	457	177	121	784	855	809	139	153	164
file-	50	3.9E−11	156	—	125	88	61	57	27	572	137	75	—	435	79	—	—	140
server	100	4.8E−23	306	—	—	—	—	229	319	—	—	765	—	—	851	—	—	—

For each case study, aside from attempting standard Monte Carlo simulation (labelled "SMC" in tables and charts) we evaluated the following approaches:

- RESTART with thresholds selected via SEQ and a fixed splitting factor $g \in \{2, 4, 8, 16\}$ (labelled "RESTART g"), using $n = 512$ and $k = n/g$ for SEQ;
- RESTART with thresholds and splitting factors determined by the ES method (labelled "RESTART ES") and the default $n = 256$ for ES;
- fixed effort with SEQ as above and effort $e \in \{16, 64, 256\}$ for every level;
- weighted fixed effort with ES (labelled "-weighted") as described in Section 4.2 using base effort $e \in \{8, 16, 128\}$ (note that all weights are ≥ 2);
- fixed success with SEQ as before and the required number of successes for each level being either 8, 32 or 128.

We did not consider ES where the splitting factors it computes would not be used (such as with "unweighted" fixed effort or fixed success). The default of using addition to replace both $\wedge$ and $\vee$ in the compositional importance function generation (cf. Section 2.2) worked well for all cases except *wlan*, for which we used max instead. Note that RESTART with SEQ and a user-specified splitting factor is the approach previously implemented in and studied with FIG [4].

5.3 Results

We provide an overview of the performance results for all model instances in Table 1. We report the averages of three runs of each experiment to account for fluctuations due to the inherent randomisation in the simulation and especially

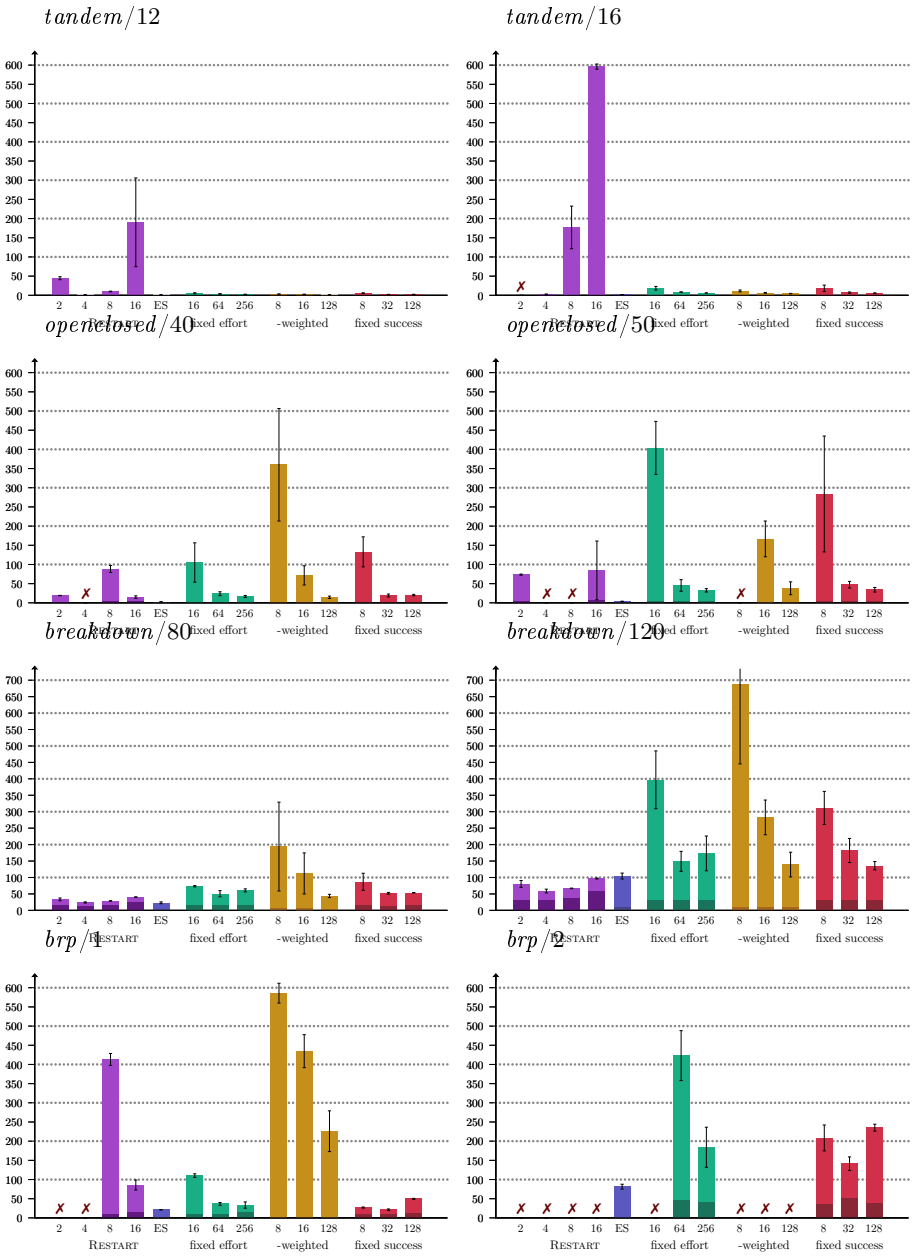


Figure 4. Some detailed performance results compared (runtimes in seconds)

in the threshold selection algorithms. Column $\hat{p}$ lists the average of all (up to 45) individual estimates for each instance. All estimates were consistent, including with SMC on the few cases where it did not time out. To verify that the compositional importance function construction does not lead to high memory usage, we list the total number of states that it needs to store in column n_I. These numbers are consistently low; even on the two PTA cases, they are far below the total number of states of the composed state spaces. The remaining columns report the total time, in seconds, that each approach took to compute the importance function, perform threshold selection, and use the respective splitting method to estimate the probability of the transient rare event. Dashes mark timeouts.

We show a selection of the most interesting cases graphically with added details in Figure 4. Each bar's darker part is the time needed to compute the importance function and the thresholds. The lighter part is the time spent in performing the actual RES. We see that the former, which is in fact almost entirely spent in threshold selection, is much lower for ES than for SEQ. The error bars show the standard deviation between the three runs that we performed for each experiment. A larger sample size would be needed for a thorough evaluation of this aspect, though.

Overall, our experimental evaluation first confirms the previous observations made with FIG: The performance of RESTART depends not only on the choice of importance function, but also very much on the selected thresholds and the splitting factor. Out of $g \in \{2, 4, 8, 16\}$, there was no single optimal splitting factor that worked well for all models. RESTART with ES usually performed best, being drastically faster than any other method in many cases. This is a very encouraging result since RESTART with ES is also the one approach that requires no more user-selected parameters. We thus selected it as the default used by the modes tool. The *wlan* case stands out as the only one where this default, and in fact none of the RESTART-based methods, terminated within our 1200 s time bound. All of the splitting methods specifically designed for transient properties, however, worked for *wlan*, with fixed success performing best. They also work reasonably well on the other cases, but we see that their performance depends on the chosen effort parameter. In contrast to the splitting factors for RESTART, though, we can make a clear recommendation for this choice: larger effort values rather consistently result in better performance.

6 Conclusion

We have investigated ways to improve the automation and performance of importance splitting to perform rare event simulation for general classes of stochastic models. For this purpose, we studied and implemented three existing splitting methods and two threshold selection algorithms, one from a previous tool and one new. Our implementation in the MODEST TOOLSET's modes simulator is publicly available at `www.modestchecker.net`.[1] We performed an extensive ex-

[1] Reviewers: The tool version used for our experiments, the model files, and the complete raw results can be accessed anonymously at www.modestchecker.net/setta17/.

perimental evaluation, reporting in this paper the only *practical* comparison of RESTART and other splitting methods that we are aware of.

Our results show that we have found a *fully* automated rare event simulation method that performs very well in the form of automatic compositional importance functions together with RESTART and the expected success method. It is also easier to implement than our previous approach, and finally pushed automated importance splitting for general models to the realm of truly rare events with probabilities down to the order of 10^{-23}.

In future work, we would like to more deeply investigate models with few points of randomisation such as the PTA examples that proved to be most challenging for our methods, and combine RES with lightweight scheduler sampling [8] to be able to handle models that include non-spurious nondeterminism.

Acknowledgements. We are grateful to José Villén-Altamirano for very fruitful discussions that led to our eventual design of the expected success method.

References

1. Amrein, M., Künsch, H.R.: A variant of importance splitting for rare event estimation: Fixed number of successes. ACM Trans. Model. Comput. Simul. 21(2), 13:1–13:20 (2011)
2. Bayes, A.J.: Statistical techniques for simulation models. Australian Computer Journal 2(4), 180–184 (1970)
3. Bohnenkamp, H.C., D'Argenio, P.R., Hermanns, H., Katoen, J.P.: MoDeST: A compositional modeling formalism for hard and softly timed systems. IEEE Trans. Software Eng. 32(10), 812–830 (2006)
4. Budde, C.E.: Automation of Importance Splitting Techniques for Rare Event Simulation. Ph.D. thesis, Universidad Nacional de Córdoba, Córdoba, Argentina (2017)
5. Budde, C.E., D'Argenio, P.R., Monti, R.E.: Compositional construction of importance functions in fully automated importance splitting. In: VALUETOOLS (2016)
6. Cérou, F., Guyader, A.: Adaptive multilevel splitting for rare event analysis. Stochastic Analysis and Applications 25(2), 417–443 (2007)
7. Cérou, F., Moral, P.D., Furon, T., Guyader, A.: Sequential Monte Carlo for rare event estimation. Statistics and Computing 22(3), 795–808 (2012)
8. D'Argenio, P.R., Hartmanns, A., Legay, A., Sedwards, S.: Statistical approximation of optimal schedulers for probabilistic timed automata. In: iFM. LNCS, vol. 9681, pp. 99–114. Springer (2016)
9. D'Argenio, P.R., Lee, M.D., Monti, R.E.: Input/output stochastic automata. In: FORMATS. LNCS, vol. 9884, pp. 53–68. Springer (2016)
10. Garvels, M.J.J., Kroese, D.P.: A comparison of RESTART implementations. In: Winter Simulation Conference. pp. 601–608. WSC (1998)
11. Garvels, M.J.J., van Ommeren, J.C.W., Kroese, D.P.: On the importance function in splitting simulation. Eur. Trans. Telecommun. 13(4), 363–371 (2002)
12. Garvels, M.J.J.: The splitting method in rare event simulation. Ph.D. thesis, University of Twente, Enschede, The Netherlands (2000)
13. Hahn, E.M., Hartmanns, A., Hermanns, H.: Reachability and reward checking for stochastic timed automata. ECEASST 70 (2014)

14. Hartmanns, A., Hermanns, H.: A Modest approach to checking probabilistic timed automata. In: QEST. pp. 187–196. IEEE Computer Society (2009)
15. Hartmanns, A., Hermanns, H.: The Modest Toolset: An integrated environment for quantitative modelling and verification. In: TACAS. LNCS, vol. 8413, pp. 593–598. Springer (2014)
16. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: VMCAI. LNCS, vol. 2937, pp. 73–84. Springer (2004)
17. Jégourel, C., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Importance sampling for stochastic timed automata. In: SETTA. LNCS, vol. 9984, pp. 163–178. Springer (2016)
18. Jégourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: CAV. LNCS, vol. 8044, pp. 576–591. Springer (2013)
19. Jégourel, C., Legay, A., Sedwards, S.: An effective heuristic for adaptive importance splitting in statistical model checking. In: ISoLA. LNCS, vol. 8803, pp. 143–159. Springer (2014)
20. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. LNCS, vol. 6806, pp. 585–591. Springer (2011)
21. Kwiatkowska, M.Z., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. Theor. Comput. Sci. 282(1), 101–150 (2002)
22. LeGland, F., Oudjane, N.: A sequential particle algorithm that keeps the particle system alive. In: EUSIPCO. pp. 1–4. IEEE (2005)
23. Paolieri, M., Horváth, A., Vicario, E.: Probabilistic model checking of regenerative concurrent systems. IEEE Trans. Software Eng. 42(2), 153–169 (2016)
24. Reijsbergen, D., de Boer, P., Scheinhardt, W.R.W., Haverkort, B.R.: Automated rare event simulation for stochastic Petri nets. In: QEST. LNCS, vol. 8054, pp. 372–388. Springer (2013)
25. Rubino, G., Tuffin, B. (eds.): Rare Event Simulation Using Monte Carlo Methods. Wiley (2009)
26. Villén-Altamirano, M., Villén-Altamirano, J.: RESTART: a method for accelerating rare event simulations. In: Queueing, Performance and Control in ATM (ITC-13). pp. 71–76. Elsevier (1991)
27. Villén-Altamirano, M., Villén-Altamirano, J.: RESTART: a straightforward method for fast simulation of rare events. In: WSC. pp. 282–289. ACM (1994)
28. Villén-Altamirano, M., Villén-Altamirano, J.: Analysis of restart simulation: Theoretical basis and sensitivity study. European Transactions on Telecommunications 13(4), 373–385 (2002)
29. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: CAV. LNCS, vol. 2404, pp. 223–235. Springer (2002)
30. Zimmermann, A., Maciel, P.: Importance function derivation for RESTART simulations of Petri nets. In: RESIM 2012. pp. 8–15 (2012)
31. Zimmermann, A., Reijsbergen, D., Wichmann, A., Canabal Lavista, A.: Numerical results for the automated rare event simulation of stochastic Petri nets. In: RESIM. pp. 1–10 (2016)