Better Automated Importance Splitting for Transient Rare Events

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

[1] University of Twente, Enschede, The Netherlands
a.hartmanns@utwente.nl
[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 (reachability) properties, 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 perform a quantitative evaluation in the design phase 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—which may be on the order of 10^{-19} or lower—is challenging: For finite-state Markov chains or probabilistic timed automata (PTA [23]), 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 [26] or merely compute probability bounds [14].

K.G. Larsen et al. (Eds.): SETTA 2017, LNCS 10606, pp. 42–58, 2017.
https://doi.org/10.1007/978-3-319-69483-2_3

Statistical model checking (SMC [17,33]), 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 for runtime: memory usage is constant but 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 [28]) 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 [4]. Its compositionality is the key to low memory usage. Our Fig tool [3] for RES of input-output stochastic automata (IOSA [8]) implements this method together with the Restart splitting algorithm [30], thresholds computed via a sequential Monte Carlo (SEQ) approach [3,6], and a single fixed splitting factor specified by the user for all thresholds. Experimental results [3] show that Fig works well for steady-state measures, but less so for transient properties. 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 [9] and fixed success [25,28] splitting methods (Sect. 3). While Restart was proposed for steady-state measures and only later extended to transient properties [31], the latter two were designed for estimating probabilities of transient events in the first place. For threshold selection, we specify a new "expected success" (EXP) technique as an alternative to SEQ (Sect. 4). EXP selects thresholds *and* an individual splitting factor 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 [16]. 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 (Sect. 5) of the various combinations on six case studies.

Related work. A thorough theoretical and empirical comparison of variants of Restart is presented in [9], albeit in a non-automated setting. Approaching the issue of automation, Jégourel et al. [19,20] use a layered restatement of the formula specifying the rare event to build an importance function for use with adaptive multilevel splitting [5], the predecessor of SEQ. Garvels et al. [10] derive importance functions for finite Markov chains from knowledge about their steady-state behaviour. For SPN, Zimmermann and Maciel [34] provide a monolithic method, though limited to a restricted class of models and throughput measures [35]. Importance *sampling* has been automated for SPN [27] restricted to Markovian firing delays and a global parameterisation of the transition intensities [35]. The difficulties of automating importance sampling are also illustrated in [18]: 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 time step.*

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) time steps of $\mathtt{M}_{sim}$. Thus, given a state s of M_{ctmc}, $\mathtt{M}_{sim}.\text{step}(s)$ 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 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 probabilities for transient properties, 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 $\neq$ *undecided*. Standard SMC/Monte Carlo simulation generates a large number n of sample paths to estimate the probability p of the transient property as $\hat{p} \overset{\text{def}}{=} \frac{n_{true}}{n}$, where n_{true} is the number of paths that satisfied ϕ, and reports a confidence interval around the estimate for 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 [22]) for state formulas *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 increases the simulation effort 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 $f_I(\mathtt{M}.\text{initial}()) = 0$, $(\phi(s_{target}) = \mathit{true}) \Rightarrow f_I(s_{target}) = \max f_I$, and if $s' := \mathtt{M}.\text{next}(s)$, 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 quality of the importance function. Traditionally, it is specified ad hoc for each model domain by a RES expert [9,28,30]. Methods to automatically compute one [10,19,20,34] are usually specialised to a specific formalism or a particular model structure, potentially providing guaranteed efficiency improvements. We build on the method of [4] 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 $\mathtt{M}(\mathit{target}^j)$ 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 $\mathtt{M}_i = \mathtt{M}(\mathit{target}^j)$, 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 $\mathtt{M}_i = \mathtt{M}(\mathit{target}^j)$, 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 [3]. 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. Importances are hence partitioned into a set of 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 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 are further parameterised by the "amount of splitting" at each threshold or the "effort" at each level; we use *splitting factor* and *effort* functions f_S resp. f_E in $\mathbb{N} \to \mathbb{N}^+$ 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 popularised by J. Villén-Altamirano and M. Villén-Altamirano [30], the Restart importance splitting method was designed

```
Input: model $\mathtt{M}$, level function $f_L$, splitting factors $f_S$, transient property $\phi$
$S := \{\!|\, \langle \mathtt{M}.\text{initial}(), 0\rangle \,|\!\}$, $\hat{p} := 0$        // start with the initial state from level 0
while $S \neq \varnothing$ do                                  // perform main and child runs (RESTART loop)
    $\langle s, l\rangle := S.\text{remove}()$, $l_{create} := l$          // get next split and store creation level
    while $\phi(s) = \mathit{undecided}$ do        // run until property decided (simulation loop)
        $s := \mathtt{M}.\text{step}(s)$                     // simulate up to next change in discrete state
        if $f_L(s) < l_{create}$ then break          // moved below creation level: kill run
        else if $f_L(s) > l$ then                              // moved one level up: split run
            $l := f_L(s)$, $S := S \cup \{\!|\, \langle s, l\rangle, \ldots (f_S(l) - 1 \text{ times}) \ldots, \langle s, l\rangle \,|\!\}$
    if $\phi(s)$ then $\hat{p} := \hat{p} + 1/\prod_{i=1}^{l} f_S(l)$   // update result if we hit the rare event
return $\hat{p}$
```

Algorithm 1. The Restart method for importance splitting

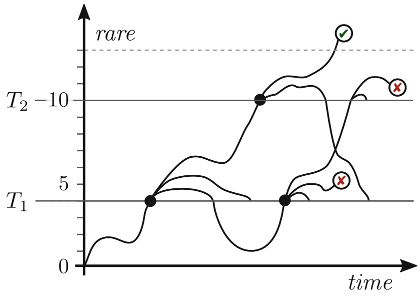

Fig. 1. Restart

for steady-state measures and later extended to transient properties [31]. It works by performing one main simulation run from the initial state. As soon as any run crosses a threshold from below, new child runs are started from the first state in the new level l (the run is split). Their number is given 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 is created. When 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 an *avoid* or *target* state. We state Restart formally as Algorithm 1. Figure 1 illustrates its behaviour. The horizontal axis is the model's time steps 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 $\{1 \mapsto 3, 2 \mapsto 2\}$.

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. Restart is carefully designed s.t. the mean of the results of many Restart runs is an unbiased estimator for the true probability of the transient property [32].

3.2 Fixed Effort

In contrast to Restart, each run of the *fixed effort* method [9,11] performs a fixed number $f_E(l)$ of partial runs on each level l. Each of these ends when it

```
Input: model M, level function $f_L$, effort function $f_E$, transient property $\phi$
$L := \{\, 0 \mapsto [\, S := \{\, \texttt{M}.\text{initial}() \,\}, n := 0, up := 0 \,] \,\}$          // set up data for level 0
for $l$ from 0 to $\max f_L$ do          // iterate over all levels from initial to target
    for $i$ from 1 to $f_E(l)$ do          // perform sub-runs on level (fixed effort loop)
        $s :\in L(l).S$, $L(l).n := L(l).n + 1$          // pick from the level's initial states
        while $\phi(s) = undecided$ do          // run until $\phi$ is decided (simulation loop)
            $s := \texttt{M}.\text{step}(s)$          // simulate up to next change in discrete state
            if $f_L(s) > l$ then          // moved one level up: end sub-run
                $L(l).up := L(l).up + 1$          // level-up run for current level
                $L(f_L(s)).S := L(f_L(s)).S \cup \{\, s \,\}$          // initial state for next level
                break
        if $\phi(s)$ then $L(l).up := L(l).up + 1$ // hit rare event (highest level only)
    if $L(l).up = 0$ then return 0          // we cannot reach the target any more
return $\prod_{i=0}^{\max f_L} L(l).up / L(l).n$   // multiply conditional level-up prob. estimates
```

Algorithm 2. The fixed effort method for importance splitting

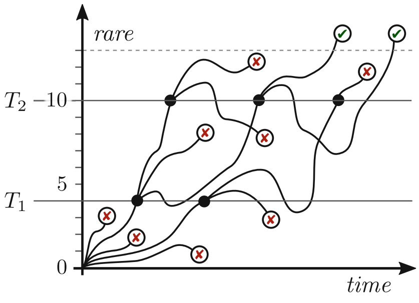

Fig. 2. Fixed effort

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 Fig. 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 in a round-robin fashion among the available initial states [9]. 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 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 probability of the transient property [11].

The fixed effort method is specifically designed for transient properties. Its advantage is predictability: each run involves at most $\sum_{l=0}^{\max f_L} f_E(l)$ partial runs. Like Restart needs splitting levels via function f_S, fixed effort needs the effort function f_E that determines the number of partial runs for each level.

3.3 Fixed Success

Fixed effort intuitively controls the simulation effort by adjusting the estimator's imprecision. The fixed success method [1,25] 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 Fig. 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. The method thus requires $f_E(l) \geq 2$ for all levels l.

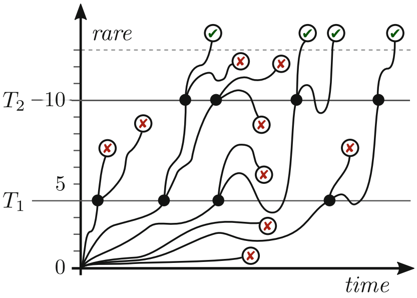

Fig. 3. Fixed success

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 found for some level lie in a bottom strongly connected component without *target* states, then the (modified) loop of line 3 of the algorithm diverges. 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 [3] and a new technique that tries to ensure a certain expected number of runs that level up.

4.1 Sequential Monte Carlo

Our first approach is inspired by the sequential Monte Carlo splitting technique [6]. It works in two alternating phases: First, n simulation runs determine the importances that can be reached from the current level , keeping track of the state of maximum importance for each run. We sort these states by ascending importance and pick the importance of the one at position $n - k$, i.e. the $(n - k)$-th n-quantile of importances, as the start of the next level. This means that

```
Input: model M, importance function $f_I$, transient property $\phi$, $n \in \mathbb{N}^+$
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     $m := m + 1$, $L :=$ level data computed in one fixed effort run (Algorithm 2)
5     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     $split := 1/p_{up}(l) + e$, $F(l) := \lfloor split + 0.5 \rfloor$, $e := split - F(l)$
8 return $F$        // if $F(l) > 1$, then $l$ is a threshold and $F(l)$ the splitting factor
```

Algorithm 3. The expected success method for threshold and factor selection

as parameter k grows, the width of the levels decreases and the probability of moving from one level to the next increases. In the second phase, the algorithm randomly selects k new initial states that lie just above the newfound threshold via more simulation runs. This extra phase is needed to obtain new *reachable* states because we cannot *generate* them directly as in the setting of [6]. We then proceed to the next round to compute the next threshold from the new initial states. Detailed pseudocode is shown as Algorithm 5 in [3]. 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. Fig and modes 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 for all levels. A value for n must also be specified; by default $n = 1000$. The degree of automation offered by SEQ is clearly not satisfactory. Furthermore, we found in previous experiments with Fig that the levels computed by different SEQ runs differed significantly, leading to large variations in Restart performance [3]. 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. We encountered this with our *wlan* case study of Sect. 5. Our tool thus restarts SEQ after a 30 s timeout; on the *wlan* model, it then always succeeded with at most two retries.

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 [11]. The resulting procedure, shown as Algorithm 3, 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 (we use a default of $n = 256$, which 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. Note that our default of $n = 256$ is much lower than the default of $n = 1000$ for SEQ. This is because SEQ performs simple simulation runs where ES performs fixed effort runs, each of which provides more information about the behaviour of the model.

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

5 Experimental Evaluation

The goal of our work was to find a RES approach that provides consistently good performance at a maximal degree of automation. We thus implemented compositional importance function generation, the splitting methods described in Sect. 3, and the threshold calculation methods of Sect. 4 in the modes simulator of the Modest Toolset [14]. This allowed us to study CTMC queueing models, network protocols modelled as PTA, and a more complex fileserver setting modelled as stochastic timed automata (STA [14]) using a single tool.

5.1 Case Studies

tandem: Tandem queueing networks are standard benchmarks in probabilistic model checking and RES [10–12,24,29]. We consider the case from [4] with all exponentially distributed interarrival times (a CTMC). The arrival rate into the first queue q_1 (initially empty) is 3 and its service rate is 2. After that, packets move into the second queue q_2 (initially containing one packet), to be processed at rate 6. The model has one parameter C, the capacity of each queue. We estimate the value of the transient property $\mathtt{P}_{=?}(q_2 > 0 \;\mathtt{U}\; q_2 = C)$, i.e. of the second queue becoming full without having been empty before.
openclosed: Our second CTMC has two *parallel* queues [13], both initially empty: an *open queue* q_o, receiving packets at rate 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_o are processed at rate 4 while q_c is empty; otherwise, packets from q_c are served at rate 2. The latter packets are put back into another internal queue, which are independently moved back to q_c at rate $\frac{1}{2}$. We study the system as in [3] with a single packet in internal circulation,

i.e. an M/M/1 queue with server breakdowns, and the capacity of q_o as parameter. We estimate $\mathtt{P_{=?}}(\neg\, \mathit{reset}\ \mathtt{U}\ \mathit{lost})$: the probability that q_o overflows before a packet is processed from q_o or q_c such that the respective queue becomes empty again.

breakdown: The final queueing system that we consider [21] 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. $\alpha_2 = 1$. The produced packets are collected in a single queue, attended to by a server with service rate $\mu = 100$, breakdown rate $\gamma = 3$ and repair rate $\delta = 4$. Again, and as in [4], we parameterise the model by the queue's capacity, here denoted K, and estimate $\mathtt{P_{=?}}(\neg\, \mathit{reset}\ \mathtt{U}\ \mathit{buf} = K)$: starting from a single packet in the queue, what is the probability for the queue to overflow before it becomes empty?

brp: We also study two PTA examples from [15]. The first is the bounded retransmission protocol, another classic benchmark in formal verification. We use parameter M to determine the actual parameters N (the number of chunks to transmit), MAX (the retransmission bound) and TD (the transmission delay) by way of $\langle N, \mathit{MAX}, \mathit{TD}\rangle = \langle 16 \cdot 2^M, 4 \cdot M, 4 \cdot 2^M\rangle$. We thus 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 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 unsuccessful transmission after more than half of the chunks have been sent successfully.

wlan: Our second PTA model is of IEEE 802.11 wireless LAN with two stations. In contrast to [15] and the original Prism case study, we use the timing parameters from the standard (leading 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 parameter is K, the maximum backoff counter value. We estimate $\mathtt{P_{=?}}(\mathit{true}\ \mathtt{U}\ \mathit{bc}_1 = \mathit{bc}_2 = K)$, the probability that both stations' backoff counters reach K.

fileserver: Our last case study combines exponentially and uniformly distributed delays. It is an STA model of a file server where some files are archived and require significantly more time to retrieve. Introduced in [14], 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: $\mathtt{P_{=?}}(\mathit{true}\ \mathtt{U}_{\leq 1000}\ \mathit{queue} = C)$.

We consider several queueing systems since these are frequently used benchmarks for RES [10–13,21,24,29]. The CTMC could easily be modified to use general distributions and our techniques and tools would still work the same.

5.2 Experimental Setup

The experiments 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

Table 1. Model data and performance results

				SMC	Restart					fixed effort			-weighted			fixed success		
model/param		$\hat{p}$	n_I		2	4	8	16	ES	16	64	256	8	16	128	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	2k	—	—	—	413	86	21	110	36	33	856	435	226	27	21	50
	2	5.8E−13	6k	—	—	—	—	—	81	—	423	184	—	—	—	208	141	235
	3	9.0E−19	16k	—	—	—	—	—	216	—	—	—	—	—	—	—	420	569
wlan	4	2.2E−5	14k	376	—	—	—	—	—	57	38	31	120	131	221	44	36	39
	5	1.6E−7	23k	—	—	—	—	—	—	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	—	—	—

E5-2620v3 (2.4/3.2 GHz, 12 logical processors) system with 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 the others. Simulations were run until the half-width of the 95 % normal confidence interval was at most 10 % of the currently estimated mean.[1] By this use of a relative width, precision automatically adapted to the rareness of the event. We also performed SMC/Monte Carlo simulation as a comparison baseline (labelled "SMC" in results), where modes uses the Agresti-Coull approximation of the binomial confidence interval. For each case study and parameterisation, we evaluated the following combinations of methods:

- 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 ($n = 512$, $k = n/2$) and effort $e \in \{ 16, 64, 256 \}$;
- weighted fixed effort with ES (labelled "-weighted") as described in Sect. 4.2 using base effort $e \in \{ 8, 16, 128 \}$ since all weights are ≥ 2;
- fixed success with SEQ as before ($n = 512$, $k = n/2$) and the required number of successes for each level being either 8, 32 or 128.

We did not consider ES in cases where the splitting factors it computes would not be used (such as with "unweighted" fixed effort or fixed success). The default

[1] We rely on the standard CLT assumption for large enough sample sizes; to this end, we do not stop before we obtain at least one sample > 0 and at least 50 samples.

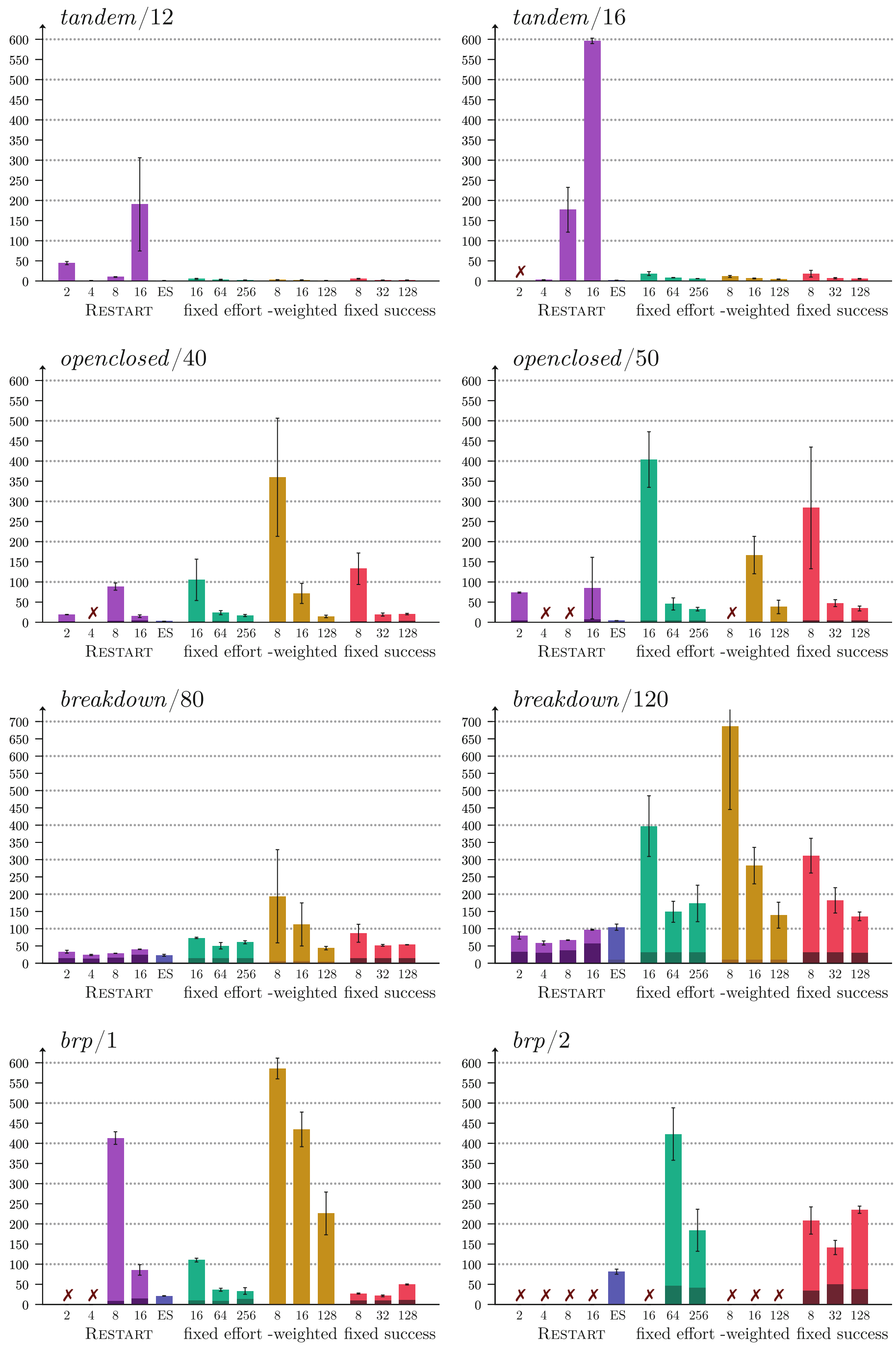

Fig. 4. Selected performance results compared (runtimes in seconds)

of using addition to replace $\land$ and $\lor$ in the compositional importance function (cf. Sect. 2.2) worked well except for *wlan*, where we used max instead. Restart with SEQ and a user-specified splitting factor is the one approach used in Fig [3].

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 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 SMC in 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 some interesting cases graphically with added details in Fig. 4. ✗ marks timeouts. Each bar's darker part is the time needed to compute the importance function and thresholds. The lighter part is the time for the actual RES. The former, which is almost entirely spent in threshold selection, is much lower for ES than for SEQ. The error bars show the standard deviation between the convergence times of the three runs that we performed for each experiment.

Our experiments first confirm previous observations made with Fig: The performance of Restart depends not only on the importance function, but also very much on the thresholds and 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 for modes. The *wlan* case is 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 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 is publicly available at www.modestchecker.net. We performed extensive experiments, resulting in the only *practical* comparison of Restart and other methods that we are aware of.

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

As future work, we would like to more deeply investigate models with few points of randomisation such as the PTA examples that proved to be the most challenging for our methods, and combine RES with the lightweight scheduler sampling technique [7] to properly handle models that include nondeterminism.

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

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.

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. Aust. Comput. J. **2**(4), 180–184 (1970)
3. Budde, C.E.: Automation of Importance Splitting Techniques for Rare Event Simulation. Ph.D. thesis, Universidad Nacional de Córdoba, Córdoba, Argentina (2017)
4. Budde, C.E., D'Argenio, P.R., Monti, R.E.: Compositional construction of importance functions in fully automated importance splitting. In: VALUETOOLS (2016)
5. Cérou, F., Guyader, A.: Adaptive multilevel splitting for rare event analysis. Stoch. Anal. Appl. **25**(2), 417–443 (2007)
6. Cérou, F., Moral, P.D., Furon, T., Guyader, A.: Sequential Monte Carlo for rare event estimation. Stat. Comput. **22**(3), 795–808 (2012)
7. D'Argenio, P.R., Hartmanns, A., Legay, A., Sedwards, S.: Statistical approximation of optimal schedulers for probabilistic timed automata. In: Ábrahám, E., Huisman, M. (eds.) IFM 2016. LNCS, vol. 9681, pp. 99–114. Springer, Cham (2016). doi:10.1007/978-3-319-33693-0_7
8. D'Argenio, P.R., Lee, M.D., Monti, R.E.: Input/Output stochastic automata. In: Fränzle, M., Markey, N. (eds.) FORMATS 2016. LNCS, vol. 9884, pp. 53–68. Springer, Cham (2016). doi:10.1007/978-3-319-44878-7_4
9. Garvels, M.J.J., Kroese, D.P.: A comparison of RESTART implementations. In: Winter Simulation Conference, WSC, pp. 601–608 (1998)
10. 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)

11. Garvels, M.J.J.: The splitting method in rare event simulation. Ph.D. thesis, University of Twente, Enschede, The Netherlands (2000)
12. Glasserman, P., Heidelberger, P., Shahabuddin, P., Zajic, T.: A large deviations perspective on the efficiency of multilevel splitting. IEEE Trans. Autom. Control **43**(12), 1666–1679 (1998)
13. Glasserman, P., Heidelberger, P., Shahabuddin, P., Zajic, T.: Multilevel splitting for estimating rare event probabilities. Oper. Res. **47**(4), 585–600 (1999)
14. Hahn, E.M., Hartmanns, A., Hermanns, H.: Reachability and reward checking for stochastic timed automata. In: ECEASST 70 (2014)
15. Hartmanns, A., Hermanns, H.: A Modest approach to checking probabilistic timed automata. In: QEST, pp. 187–196. IEEE Computer Society (2009)
16. Hartmanns, A., Hermanns, H.: The Modest Toolset: an integrated environment for quantitative modelling and verification. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 593–598. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54862-8_51
17. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 73–84. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24622-0_8
18. Jegourel, C., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Sedwards, S.: Importance sampling for stochastic timed automata. In: Fränzle, M., Kapur, D., Zhan, N. (eds.) SETTA 2016. LNCS, vol. 9984, pp. 163–178. Springer, Cham (2016). doi:10.1007/978-3-319-47677-3_11
19. Jegourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 576–591. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39799-8_38
20. Jegourel, C., Legay, A., Sedwards, S.: An effective heuristic for adaptive importance splitting in statistical model checking. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014. LNCS, vol. 8803, pp. 143–159. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45231-8_11
21. Kroese, D.P., Nicola, V.F.: Efficient estimation of overflow probabilities in queues with breakdowns. Perform. Eval. **36**, 471–484 (1999)
22. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_47
23. 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)
24. L'Ecuyer, P., Demers, V., Tuffin, B.: Rare events, splitting, and quasi-Monte Carlo. ACM Trans. Model. Comput. Simul. **17**(2) (2007)
25. LeGland, F., Oudjane, N.: A sequential particle algorithm that keeps the particle system alive. In: EUSIPCO, pp. 1–4. IEEE (2005)
26. Paolieri, M., Horváth, A., Vicario, E.: Probabilistic model checking of regenerative concurrent systems. IEEE Trans. Softw. Eng. **42**(2), 153–169 (2016)
27. Reijsbergen, D., de Boer, P.-T., Scheinhardt, W., Haverkort, B.: Automated rare event simulation for stochastic Petri nets. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 372–388. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40196-1_31
28. Rubino, G., Tuffin, B. (eds.): Rare Event Simulation Using Monte Carlo Methods. Wiley (2009)
29. Villén-Altamirano, J.: Rare event RESTART simulation of two-stage networks. Eur. J. Oper. Res. **179**(1), 148–159 (2007)

30. 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)
31. Villén-Altamirano, M., Villén-Altamirano, J.: RESTART: a straightforward method for fast simulation of rare events. In: WSC, pp. 282–289. ACM (1994)
32. Villén-Altamirano, M., Villén-Altamirano, J.: Analysis of restart simulation: theoretical basis and sensitivity study. Eur. Trans. Telecommun. **13**(4), 373–385 (2002)
33. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 223–235. Springer, Heidelberg (2002). doi:10.1007/3-540-45657-0_17
34. Zimmermann, A., Maciel, P.: Importance function derivation for RESTART simulations of Petri nets. In: RESIM 2012, pp. 8–15 (2012)
35. 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)