Automated Compositional Importance Splitting

Carlos E. Budde[a], Pedro R. D'Argenio[b,c], Arnd Hartmanns[a]

[a] *University of Twente, Enschede, Netherlands*
[b] *Universidad Nacional de Córdoba, Córdoba, Argentina*
[c] *Saarland University, Saarbrücken, Germany*

Abstract

In the formal verification of stochastic systems, statistical model checking uses simulation to overcome the state space explosion problem of probabilistic model checking. 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 present an automated method to derive the importance function. It considers both the structure of the model and of the formula characterising the rare event. It is memory-efficient by exploiting the compositional nature of formal models. We experimentally evaluate it in various combinations with two approaches to threshold selection as well as different splitting techniques for steady-state and transient properties. We find that RESTART splitting combined with thresholds determined via a new expected success method most reliably succeeds and performs very well for transient properties. It remains competitive in the steady-state case, which is however challenging to all combinations we consider. All methods are implemented in the modes tool of the MODEST TOOLSET and the FIG rare event simulator.

Keywords: rare event simulation, importance splitting, importance function, statistical model checking, transient analysis, steady-state analysis.

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 or extended unavailability 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 after the probability of failure and the expected unavailability are shown to be sufficiently low can the system design be implemented. Calculating such values—which may be on the order of 10^{-15} or lower—is challenging. For finite-state Markov chains or probabilistic timed automata (PTA [46]), *probabilistic model checking* can numerically approximate the desired values, but the state space

explosion problem limits this approach 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 [55] or merely compute probability bounds [31].

Statistical model checking (SMC [38, 72]), i.e. using Monte Carlo simulation with formal models, has become a popular alternative for large models and formalisms not amenable to (traditional) probabilistic model checking like stochastic (timed) automata [9, 18]. SMC trades memory for runtime: memory usage is constant, but the number of simulation runs which are needed to converge to a result can easily explode with the desired precision. This is exacerbated in the presence of rare events. For instance, when the true probability of an event is 10^{-15}, one may want that the error of an estimation is no larger than 10^{-16}. Such tight requirements in the precision of estimations may render traditional Monte Carlo simulation approaches infeasible [24, 65].

Rare event simulation methods (RES [58]) 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. Broadly speaking, the main RES methods are *importance sampling* and *importance splitting*. They complement each other in several application domains [57]. The former modifies the probability distributions which dictate the stochastic behaviour of the model, with the aim to make the event more likely. The challenge lies in finding a "good" *change of measure* to modify probabilities in an effective way. Importance splitting instead does not modify the model, but rather refines the simulation mechanics to perform more (partial) simulation runs, which may start from non-initial states 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.

Importance sampling requires an explicit formula for the distributions governing all state transitions [37]. This is typically insufficient on its own to allow a provably efficient change of measure, and further characterisations of the distributions are needed, like the memoryless property [4] or a rarity parameter [52]. Several specific models in the literature satisfy these assumptions, which has allowed an effective use of importance sampling in the analysis of e.g. network reliability, queueing theory, particle transport, and counting problems [6, 7, 10, 39, 56].

Importance splitting in principle poses no constraints on the distributions [24, 51]. However, splitting depends on an inherently layered state space where several transition steps govern the rarity of the event studied. If instead the rarity depends on taking very few transitions with low probabilities, then the splitting approach will not be effective. This is e.g. the case when heavy-tail distributions govern the individual steps of the rare behaviour of the model. Importance splitting enjoys a rich scope of applications, most prominently in queueing theory but also in e.g. dependability analysis, randomised algorithms, distributed systems, particle transport, and hybrid systems [8, 15, 42, 51].

Thus, to tune a particular implementation, both methods require certain knowledge of the specific setting where they are applied. Overall, importance splitting appears more amenable to automatic approaches across different modelling form-

alisms using different kinds of probability distributions due to its black-box view of the model. We thus focus on the automation of importance splitting in this paper. To achieve efficient splitting in an automated way, the main challenge lies in deriving a good importance function. However, efficiency also hinges on finding good values for further nontrivial parameters: depending on the concrete splitting method used, *thresholds* (the importance values at which to start new runs) and splitting or effort *factors* (how many new runs to generate at each threshold) need to be chosen. The performance of importance splitting for RES in a specific model can vary drastically with the choices made for these parameters [11, 51].

In general, the quality of a choice of parameters depends heavily on the structure of the model at hand; making good choices requires an expert in the system domain, who should be experienced with the modelling formalism as well as the selected RES method [65]. In this work we study ways to alleviate such a requirement, proposing combinations of techniques that enable an effective application of importance splitting on a general set of systems. We highlight that to align RES with the spirit of (statistical) model checking as a "push-button" approach, it is necessary to devise an automatic selection of parameters that perform well in most situations. Furthermore, the methods automating such selection must not negate the memory usage advantages of SMC with respect to traditional model checking. These constitute the main challenges we address here.

Contributions. In this paper, we present and experimentally evaluate a set of ingredients that, combined, allow applying robust fully automated importance splitting on general models, including non-Markovian ones, for RES in SMC. The ingredients are (i) a compositional method to automatically construct an importance function from the formal model and a temporal logic property query (Section 3), (ii) three existing splitting techniques that determine the details of how to manage the partial runs and calculate a correct estimate (Section 4), and (iii) two algorithms—one existing, one new—to derive thresholds and factors (Section 5). We consider both transient and steady-state properties. We use the splitting methods RESTART (Section 4.1), fixed effort [25] (Section 4.2), and fixed success [53, 57] (Section 4.3). While RESTART was proposed for steady-state analysis and later extended to transient properties [65, 66], the latter two are geared for estimating probabilities of transient events[1]. The two algorithms for threshold selection are a sequential Monte Carlo (SEQ) approach [17] with a single fixed splitting factor specified by the user for all thresholds (Section 5.1) and a new "expected success" (EXP) technique (Section 5.2). 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 FIG tool [11] and the modes simulator [15] of the MODEST TOOLSET [34]. The techniques can be freely combined, and work for all the formalisms supported by the two tools—including CTMC, input-output stochastic automata (IOSA [22]), and stochastic timed automata (STA [9]). We finally perform an extensive exper-

[1]For *regenerative* Markov chains, fixed effort can also be used for steady-state analysis [24].

imental evaluation (Section 7) of the various combinations on several case studies, including three new and challenging examples for steady-state measures.

Previous work. This is an extended version of our previous conference publication [13], which we combine with material from [12] and [11]. Compared to [13], we add (i) a detailed explanation of the compositional importance function approach originally introduced in [12] in Section 3, (ii) the analysis of steady-state properties throughout the paper, (iii) an explanation of the SEQ technique on the same level of detail as for EXP in Section 5, (iv) two new challenging models (*database* and *pipeline*) in the experimental evaluation in Section 7, one of them non-Markovian, and (v) a more detailed description of the simulation and RES capabilities of the modes and FIG tools in Section 6.

Related work. In [68], Villén-Altamirano et al. compared the principles behind RESTART with those of the original splitting approach [44], arguing in favour of the generality of RESTART. Garvels and Kroese [25] performed a thorough theoretical and empirical comparison of different variants of RESTART. All these works rely on the user to provide the importance function, thresholds, and splitting factors. We intend to derive this data from the model and property query provided by the user. In that sense, work by Jégourel et al. [40, 41] is closer to ours: they build an importance function (*score function*) from the temporal logic property query, and then choose the splitting points adaptively as in [16]. Their method however relies on a layered restatement of the property, resorting to approximate heuristics when this is not possible. Our proposal also differs from [40, 41] in that we additionally consider the structure of the model to measure a notion of distance from an arbitrary state to the rare event. In [73] a method similar to the monolithic (i.e. non-compositional) approach described later in [11] is introduced to build an importance function on SPN used with RESTART. This is applicable only to a restricted variant of SPN and throughput measures [74]. RES on SPN has also been approached through importance sampling [56], selecting the change of measure automatically from the structure of the model in a way that "could be adapted to importance splitting." However that method requires restricting the scope of applicability: only Markovian systems are considered and all transition intensities (i.e. rates) need to be parameterised by a rarity parameter ϵ [74]. The difficulties of automating and generalising importance sampling are also illustrated in [43]: their proposed automatic change of measure guarantees a reduction in the variance of the estimator, but this only applies to models whose stochastic behaviour is described by integrable products of random variables following exponential and uniform distributions. We argue that such strong restrictions can be dropped by automating importance *splitting* to address RES. We aim at general modelling formalisms, minimising the restrictions on the models, but still providing a significant performance boost over standard Monte Carlo. 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 stochastic models.

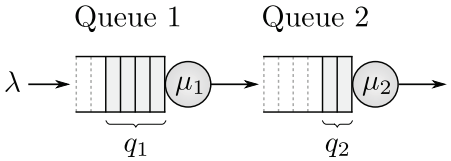

Figure 1: Tandem queue

2. Preliminaries

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

2.1. Simulation Models

Since we are interested in RES approaches that can work across several stochastic modelling formalisms with discrete and continuous time and state, we use an abstract notion of models:

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

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

Example 1. Consider a tandem Jackson network consisting of two sequentially connected queues as in Figure 1. Customers arrive at Queue 1 following a Poisson process with parameter λ; after being attended by a server at rate μ_1 they enter Queue 2, where they are attended by another server at rate μ_2; after the second service customers leave the system. This *tandem queue* system is a CTMC: time lapses between events are independent and exponentially distributed. The tandem queue has received considerable attention in the RES literature [24, 26, 27, 50, 62, 69].

2.2. Property Queries

A model M is a stochastic process $X = \{ X_t \mid t \in \mathbb{N} \}$. A probability space $(\Omega, \mathcal{F}, P)$ and a measurable space (S, Σ) are assumed so that each X_t is a random variable in Ω taking values on the state space S of M. Our algorithms require models to be Markov processes: this can be done without loss of generality, since for formalisms with memory, e.g. due to general continuous probability distributions, we encode the memory in the state space. In particular this is performed for values and expiration times of clocks in the case of IOSA and STA models.

An *event* will be a measurable subset of S, i.e. an element of Σ. In what follows we refer to $A \subsetneq S$ as the *rare event* of interest, that is a (measurable) set of states that M can enter with positive but very small probability. We call elements in A *target states*. In certain cases a *stop event* $B \subsetneq S$ denotes an end-of-simulation condition so that $B \cap A = \varnothing$ and $\lim_{t\to\infty} P(X_t \in A \uplus B) = 1$. We call elements in B *avoid states*.

We are interested in the probabilities for transient and steady-state properties. The general goal is to estimate the probability $0 < \gamma \ll 1$ of observing the rare event A in M. The manner in which γ is defined determines whether the probability is "transient" or "steady-state." For a thorough mathematical description of these concepts in the context of RES over formal models we refer the reader to [11, Sec. 2.5.1] and only summarise the fundamental notions here.

Definition 2. *For Markov process $X = \{ X_t \mid t \in \mathbb{N} \}$ of model* M*, let $E \subseteq S$ be an event observed with positive probability in X. The* entrance time into E *is the random variable describing the first time index where event E is observed:*

$$T_E \stackrel{\text{def}}{=} \inf\{ t \in \mathbb{N} \mid X_t \in E \}.$$

The transient probability *of the rare event A given the stop event B is*

$$\gamma_t = P(T_A \leqslant T_B)$$

where T_A and T_B are the entrance times into A and B, respectively. The steady-state probability *of the rare event A (also denoted* long run probability*) is*

$$\gamma_s = \lim_{t\to\infty} P(X_t \in A).$$

For models with a notion of time that is different from the process' index (e.g. for CTMC as explained above), we take the above definition of the steady-state probability on the corresponding induced (continuous-time) process with that notion of time instead. This works since our events are defined over states only.

These concepts are common in the RES literature. In our setting, transient probabilities measure the likelihood that a sample path drawn from M reaches a target state in A, before visiting an avoid state in B [24, 57]. Steady-state probabilities measure the proportion of (model) time that a sample path spends in target states once the system reaches an equilibrium [24, 65, 70]. Concretely, we estimate these values via the evaluation of *property queries* (or simply *properties*) on sample paths:

Definition 3. *A* transient property $\phi \in S \to \{\, true, false, undecided \,\}$ *maps target states to true, avoid states to false, and all other states to undecided.*

Definition 4. *A* steady-state property $\psi \in S \to \mathbb{R}_{\geqslant 0}$ *maps target states to their sojourn time*[2] *and all other states to* 0.

For transient analysis of a model `M`, standard SMC/Monte Carlo simulation generates a large number n of sample paths and estimates the transient probability as $\hat{\gamma}_t \stackrel{\text{def}}{=} \frac{n_{true}}{n}$ where n_{true} is the number of paths that satisfy the transient property ϕ. To determine whether a sample path satisfies ϕ, evaluate ϕ sequentially for every state on the path and return the first outcome $\neq$ *undecided*. Notice that, since $\lim_{t\to\infty} P(X_t \in A \uplus B) = 1$, with probability 1 all sample paths will eventually reach a state where ϕ returns *true* or *false*. This procedure corresponds to estimating the value of the until formula $\texttt{P}_{=?}\,(\neg\, avoid\, \texttt{U}\, target)$ in a logic like PCTL [32], as used in e.g. Prism [47], for state formulæ *avoid* and *target* identifying the stop and rare events in `M` respectively. Time-bounded until $\texttt{U}_{\leqslant b}$ is encoded by tracking the elapsed time t_{global} in states and including $t_{global} > b$ in *avoid*.

Example 2. For a model `M` of the tandem queue from Example 1 let q_1 denote the number of customers or packets in Queue 1 and q_2 the number in Queue 2. Let `M`.initial() $= (1, 0) = (q_1, q_2)$, i.e. the queue initially has one packet in Queue 1 and none in Queue 2. For a given maximum capacity C of the second queue, the transient property $\texttt{P}_{=?}\,(q_1 > 0\,\texttt{U}\,q_2 > C)$ queries the probability of observing an overflow in Queue 2 before the first queue becomes empty. The resulting transient probability γ_t is rare for certain service rates μ_1 and μ_2 and capacities of both queues.

For steady-state analysis it is possible to work with regenerative Markov processes [24, 59]. Alternatively, the *batch-means method* offers a practical and more general approach [23, 48, 49]. In batch-means a single "long" sample path is generated and divided into *batches* $\{\, b_i \,\}_{i=1}^{n}$ of fixed size k each[3], which are then treated similarly to the n sample paths of transient analysis [48, 60]. This favours the observation of actual steady-state behaviour because only the first batches will contain the transient phase of the model. However, models exhibiting multi-modal stochastic behaviour cannot be studied in this way [5, 54].

In the scope of definitions 1 and 4, batch-means involves drawing a sample path from `M` that visits states $s_1, s_2, \ldots, s_{m_1}$ (where e.g. $m_1 = k$ for discrete models) constituting the first batch b_1. The next batch is generated as the continuation of this initial sample path, viz. $b_2 = s_{m_1+1}, s_{m_1+2}, \ldots, s_{m_2}$, and so on. The resulting batches $\{\, b_i \,\}_{i=1}^{n}$ are used to estimate the steady-state probability $\hat{\gamma}_s \stackrel{\text{def}}{=} \frac{1}{nk} \sum \psi(b_i)$ for steady-state property ψ, where $\psi(b_i) \stackrel{\text{def}}{=} \sum_{j=m_{i-1}+1}^{m_i} \psi(s_j^i)$ for states s_j^i from

[2] In a sample path this is the sampled time of permanence in the state before performing a transition to the next state—which may be the same state as before in case of a self-loop.

[3] For discrete-time models the batch size is the number of steps, i.e. the number of states visited in a sample path; for continuous-time models it is the sum of sojourn times.

batch b_i. Thus $\hat{\gamma}_s$ is an estimate of the proportion of time spent on target states. This procedure corresponds to estimating the value of the steady-state formula $\mathtt{S}_{=?}(\mathit{target})$ in a logic like CSL [2] for state formula *target*.

Example 3. In the setting from Example 2 (and regardless of the initial state of M) the steady-state property $\mathtt{S}_{=?}\,(q_2 = C)$ queries the proportion of time that Queue 2 is saturated in the long run.

For both transient and steady-state analyses and given some confidence level selected by the user, Monte Carlo simulation usually reports a confidence interval around the point estimate $\hat{\gamma}$. In particular, the division of the sample path in batches for the batch-means method in steady-state analysis is performed specifically for this purpose: confidence intervals are computed from a *set* of measurements, thus the need to separate the (single) long sample path into several batches, each of which produces one measurement.

2.3. The Importance Function

Importance splitting increases the simulation effort for states "close" to the target set. Proximity is represented by an *importance function* $f_I \in S \to \mathbb{N}$ that maps each state to its importance in $\{\, 0, \ldots, \max f_I \,\}$. Ideally states close to the rare event A should have higher importance than those far from it, where the notion of distance is stochastic: a state s is close to the rare event if the probability of visiting some state in A after visiting s is high. To simplify our presentation we assume that $f_I(\mathtt{M}.\mathrm{initial}()) = 0$, $f_I(s_{\mathit{target}}) = \max f_I$ for all $s_{\mathit{target}} \in A$, and if $s' := \mathtt{M}.\mathrm{next}(s)$, then $|f_I(s) - f_I(s')| \leqslant 1$. These assumptions can easily be removed [11, 51].

All importance splitting methods provide unbiased estimators for the (transient or steady-state) property under study. The quality of the importance function, i.e. how well it resembles the proximity of the states to the rare event, determines the variance of the estimator. The goal is to obtain an estimator with lower variance than with the use of standard Monte Carlo simulation. This means that the performance, but not the correctness, of importance splitting hinges on the quality of the importance function f_I.

Traditionally, f_I is specified ad hoc for each model domain by a RES expert [25, 57, 65]. Methods to automatically compute an importance function are usually specialised to a specific formalism or a particular model structure, potentially providing guaranteed efficiency improvements [26, 40, 41, 73]. We use an automatic method that is applicable to any stochastic compositional model with a partly discrete state space. As a heuristic, it does not provide mathematical guarantees of performance improvements, but is aimed at generality and providing "usually good" results with minimal user input. We describe this method in detail in Section 3.

2.4. Levels, thresholds and factors

Given a model and importance function f_I, importance splitting increases the simulation effort of sample paths that visit states with growing importance. This

can be carried out in different ways, as we detail in Section 4. All techniques save and restore states from sample paths. For instance, in a typical RESTART implementation, when a *simulation run* (i.e. a path currently being sampled in the model) visits a state with higher importance than those observed before, the state is saved and new (independent) simulation runs are initiated from that state.

In principle importance splitting could spawn more simulation runs whenever the current sample path moves from a state with importance i to one with importance $j > i$. However, for certain importance functions and models, the probability of visiting a state with a higher importance could be often close to 1 for many of the i. In such scenarios splitting on every increment would lead to excessively many (partial) runs and high runtime.

It is thus common to partition the importance values into a set of intervals called *levels*, so that the saving and re-initiating of simulation runs is performed when a state in a higher level (rather than with higher importance) is visited. 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. Compositional Importance Functions

One of our main motivations is to develop methods that are versatile, e.g. whose scope of applicability includes models as general as possible. In particular we intend to scale in terms of model size, for which we resort to compositional descriptions of models. 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 these may interact, which they usually do via a synchronisation/handshaking mechanism. For instance, compositional CTMC models in PRISM or the MODEST TOOLSET use full synchronisation. This means that if the label a is shared among components $\mathtt{M}_1, \ldots, \mathtt{M}_n$, i.e. it decorates some transition in each of them, then a transition labelled a in *any* component can only take place if *each other* component can also take some transition labelled a. Instead, IOSA models in the FIG tool use broadcast communication channels where all components are input-enabled. This means that if a component "outputs" label a when taking a transition, other components may not react to it (i.e. if their current local state does not enable transitions labelled with a) even when a is part of their alphabet.

The fundamental notion behind our compositional importance function derivation method for a specific (single) model is that, regardless of its stochastic nature, the importance of a state s should reflect its (stochastic) proximity to any target state. For model $\mathtt{M}$ consider a directed graph representation where the nodes are the states of $\mathtt{M}$ and the edges represent the transitions describing its behaviour; i.e. edge $s \to s'$ indicates state s' can be reached from s with positive probability via some transition in $\mathtt{M}$. Then the rare event A is a set of nodes in

the graph, and the distance to A of any state, measured as the minimum number of edges between s and any $s' \in A$, is a rough approximation of its importance.

A breadth-first search (BFS) that starts from A and uses the reverted edges of the graph can compute these importance values. The idea can be refined e.g. to consider the (potentially) probabilistic nature of the transitions: add weights to the edges of the graph, to reflect quantitatively the likelihood of taking the transitions they represent, and employ Dijkstra or A* instead of BFS. However, even in its most simple form, the method can perform well on general models because it is embedded in a framework for importance splitting implementation—the quality of the importance function is a cornerstone, but other mechanisms like an adaptive threshold/effort selection can alleviate some poor importance approximations. Furthermore, the reason why this method is particularly amenable to the analysis of general systems models is its compositionality, on which we elaborate next.

To derive a compositional importance function one must first decompose the *global state* s of $\mathtt{M}$ into its constituent parts. For states described by valuations of (discrete) variables in a model, s is projected onto the local variables of $\mathtt{M}_i$ to reflect its *local state* $s\big|_i$ corresponding to s. For instance, consider a compositional description of the tandem queue where Queue 1 and Queue 2 are described by models $\mathtt{M}_1$ and $\mathtt{M}_2$, respectively. If the state of the tandem queue $\mathtt{M} = \mathtt{M}_1 \parallel \mathtt{M}_2$ is $s = (q_1, q_2) = (1, 0)$, then for $\mathtt{M}_1$ we have the local state $s\big|_1 = (q_1, q_2)\big|_1 = (q_1) = 1$ and for $\mathtt{M}_2$ the local state is $s\big|_2 = 0$.

The *target* formula that characterises the rare event in $\mathtt{M}$ can also be projected: to *project the formula* onto a component $\mathtt{M}_i$ we could remove from *target* all logical expressions that refer to variables which do not belong to $\mathtt{M}_i$. The resulting formula $target_i$ can then be evaluated in the states $s\big|_i$ of $\mathtt{M}_i$ independently from the other components to determine the *local rare event* A_i of component $\mathtt{M}_i$. Notice that this approach does not tolerate general formulæ that e.g. directly compare variables belonging to two different components. For example, projecting the *target* formula $5q_1 < q_2$ in the tandem queue case would produce an empty logical expression for both components, making it infeasible to compute rare events A_1 and A_2 local to components $\mathtt{M}_1$ and $\mathtt{M}_2$. Therefore our algorithms require that all literals in the formula, viz. all Boolean variables or arithmetic comparisons between numeric variables, refer to variables of a single component.

A further technical difficulty is how to evaluate a global *target* formula in each component such that the resulting local rare events A_i are in line with the global rare event A. To illustrate this consider the *target* formula $q_1 < 5 \Rightarrow q_2 < C$, where the projection would produce $target_1 \equiv q_1 < 5$ for $\mathtt{M}_1$ and $target_2 \equiv q_2 < C$ for $\mathtt{M}_2$. Notice that, in the global model of the tandem queue, all states where $q_1 \geqslant 5$ are indeed target states in A. However, this would not be locally identified in $\mathtt{M}_1$ if we use $target_1$ to construct A_1. The general issue here is whether to take positively or negatively the occurrence of a variable in the *target* formula: the expression may have nested logical operations, whose removal during the projection of the formula obliterates the semantics of *target*. A solution is to employ a normal form of the logical formula to produce the desired projections. We use negation normal form (NNF), where all logical nesting is resolved but for the resulting

(disjunctive or conjunctive) clauses, which can be considered as the fundamental building blocks of the formula. In particular using NNF does not restrict the type of *target* formulæ that can be considered, and all negations must occur at the level of the literals of the formula.

We can now describe the construction of a compositional importance function, given the compositional model $\mathtt{M} = \mathtt{M}_1 \parallel \mathtt{M}_2 \parallel \ldots \parallel \mathtt{M}_n$ and a global *target* formula characterising the rare event:

1. Convert *target* to NNF and associate each literal $target^j$ with the component $\mathtt{M}(target^j) \in \{\, \mathtt{M}_i \,\}_{i=1}^n$ 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 BFS to compute the local minimum distance of each state $s\big|_i$ to any state satisfying $target^j$.
   ▷ The resulting function $f_i^j : S \to \mathbb{N}$, bound to component $\mathtt{M}_i$ and literal $target^j$, maps each $s \in S$ to its distance (in $\mathtt{M}_i$) to the closest state in that component that satisfies $target^j$.
   ▷ The *local importance function* f_i^j thus encodes the distance of $s\big|_i$ to A_i^j, defined as the subset of the local rare event A_i that corresponds to literal $target^j$.
3. In the NNF reformulation of *target*, replace every occurrence of the literal $target^j$ by f_i^j 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 *global importance function* $f_I : S \to \mathbb{N}$.

Further implementation details can be found in [11]. In particular, in the composition of the local importance functions f_i^j to construct the global importance function f_I, other operators can be used in place of $+$, e.g. max or multiplication. Furthermore, the distributive properties between $\wedge$ and $\vee$ in the NNF reformulation of *target* can be exploited to choose a combination of operators (rather than a single one). Some studies on the use of semirings for this purpose can also be found in [11].

In any case the most relevant characteristic of the compositional method described is that, aside from the choice of operator (for which $+$ as default has worked well for most models studied), the procedure requires no user input to compute the global importance function f_I. Moreover, it takes into account both the structure of the *target* formula and the structure of the state space of each model component. 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.

4. Importance Splitting Methods

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

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

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

Algorithm 1: The RESTART method for transient analysis

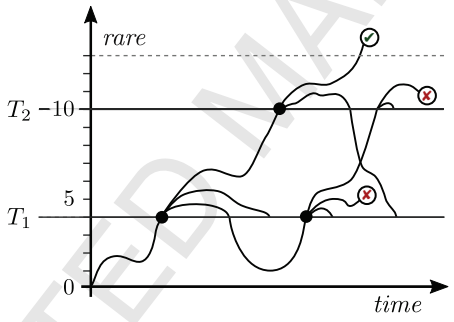

Figure 2: RESTART

4.1. RESTART

Originally discovered in 1970 [3] and popularised by J. and M. Villén-Altamirano [65], the RESTART importance splitting method was designed for steady-state measures and later extended to transient properties [66]. 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*). The number of child runs to start is given by l's splitting factor, $f_S(l) - 1$, 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 to perform importance splitting for transient analysis as Algorithm 1. Figure 2 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

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 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  // count 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 // rare event (highest level only)
12     if L(l).up = 0 then return 0       // cannot reach the target any more
13 return ∏_{i=0}^{max f_L} L(i).up/L(i).n    // multiply cond. level-up prob. estimates
```

Algorithm 2: The fixed effort method for transient analysis

marked ✓and *avoid* states are marked ✗. We have three levels with thresholds at importance values 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, as presented in Algorithm 1 for transient analysis, is carefully designed such that the mean of the results of many RESTART runs is an unbiased estimator for the true probability of the transient property [67]. In particular, 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.

The application of RESTART to steady-state analysis is a special case of the batch-means method [65]. For a single RESTART run performed from M.initial() for T (simulation) time units, say m runs visited the rare event. Let t_j^* be the total time that the j-th such run spent on a target state. Then

$$\hat{\gamma}_s \stackrel{\text{def}}{=} \frac{\sum_{i=1}^{m} t_i^*}{T \prod_{j=1}^{l} f_S(j)}$$

is an unbiased estimator of the steady-state probability of the rare event.

4.2. Fixed Effort

In contrast to RESTART, each run of the *fixed effort* method [24, 25] performs a fixed number $f_E(l)$ of partial runs on each level l. Each of these ends when it

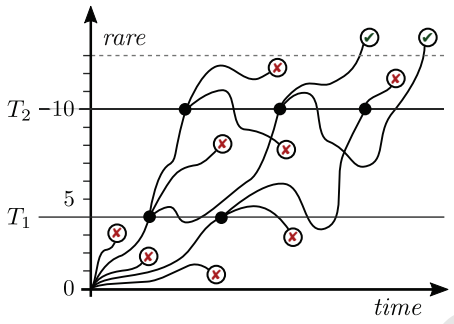

Figure 3: Fixed effort

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 Figure 3 (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 [25]. 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 [24].

The advantage of fixed effort is its predictability: each run involves at most $\sum_{l=0}^{\max f_L} f_E(l)$ partial runs, each of which will end with probability 1. The method is specifically designed for transient properties; it does not map naturally to steady-state analysis where there is no end-of-simulation condition. 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.

4.3. Fixed Success

Fixed effort intuitively controls the simulation effort by adjusting the estimator's imprecision. The fixed success method [1, 53] 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 4 (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 $L(l).up < f_E(l)$, i.e. we perform sub-runs on the current level until $f_E(l)$ sub-runs moved up to the next level (or hit the target in

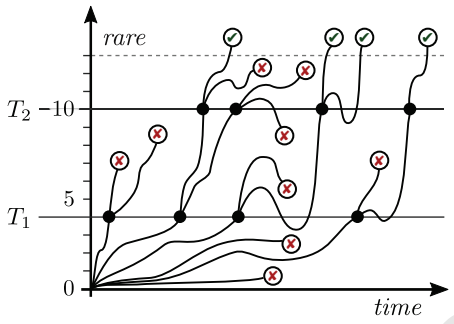

Figure 4: Fixed success

case l is the highest level). 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 $f_E(l) \geqslant 2$ for all levels l.

Like fixed effort, fixed success is designed to study transient properties. 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.

5. Thresholds and Splitting Factors

To determine the splitting levels/thresholds, we implement and compare two approaches: the sequential Monte Carlo (SEQ) method from [11] and a new technique that tries to ensure a certain expected number of runs that level up.

5.1. Sequential Monte Carlo

Our first approach is inspired by the sequential Monte Carlo splitting technique [17]. As shown in Algorithm 3, it works in two alternating phases: first, n simulation runs determine the importance values 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 as parameter k grows, the width of the levels

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
10     if T.top() ⩾ f_I(S(n − k)) then break        // 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
15             S(i) := M.step(S(i))
16         if f_I(S(i)) < T.top() then goto 13   // did not reach new threshold
17     for j from n + 1 to n + k do          // randomly select k initial states
18         i := sample uniformly from {1, ..., n}, S(j) := S(i)
19 for l from T.top() to max f_I do T.push(l)        // fill in missing thresholds
20 return T              // set of threshold importances characterising the levels
```

Algorithm 3: The sequential Monte Carlo method for threshold selection

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 [17]. We then proceed to the next round to compute the next threshold from the new initial states. Detailed pseudocode is shown Algorithm 3 and also as Algorithm 5 in [11]. The result is a sequence of thresholds—a subset of importance values from $\{0, \ldots, \max f_I\}$—characterising a level function f_L.

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 [11]. Our studies suggest this could be attributed to SEQ being originally designed for continuous importance functions [16, 17], for

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

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 (Alg. 2)
5 $\quad$ **for** l **from** 0 **to** $\max f_I$ **do**
6 $\quad\quad p_{up}(l) := p_{up}(l) + \frac{1}{m}(L(l).up/L(l).n - p_{up}(l))$
7 **for** l **from** 0 **to** $\max f_I$ **do** *// calculate splitting factors from probabilities*
8 $\quad split := 1/p_{up}(l) + e$, $F(l) := \lfloor split + 0.5 \rfloor$, $e := split - F(l)$
9 **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

which thresholds can be set infinitely close to each other [11, 13].

For the code presented in Algorithm 3 for transient analysis, SEQ may get stuck in the same way as fixed success. We encountered this with our *wlan* case study of Section 7.1. Our tool thus restarts SEQ after a 30 s timeout; on the *wlan* model, it then always succeeded with at most two retries. An alternative is that, in lines 6 and 14, a predefined max simulation run length is used (instead of the transient property ϕ) to determine the end of a simulation. This is also precisely what is done to perform steady-state analysis using SEQ, where the property query does not specify an end-of-simulation condition.

5.2. 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 [24]. 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 probabilities of a sample path moving up by one level (the *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 different from 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, which implies that the number of partial runs grows—by those factors on average—as simulations reach higher values of f_L. In practice this always led to dismal performance or timeouts, most often due to too many splits in our experiments. This indicates a simulation overhead, an "unbalanced growth" to put it in Garvel's terms, and thus we only consider the original single (1) expected successful run for ES in the sequel. We further note that the property is not an input to Algorithm 4; the algorithm can thus be employed for both transient and steady-state analysis as-is.

6. Tools, Languages and Models

We implemented the compositional importance function generation of Section 3, the splitting methods described in Section 4, and the threshold calculation methods of Section 5 in both the modes simulator [15] of the Modest Toolset [31] and the Fig tool [11] for input-output stochastic automata.

6.1. The modes Tool

modes is the statistical model checker of the Modest Toolset. It implements all of the methods described in sections 3 to 5, however for transient properties only. It uses the toolset's infrastructure to transform various input languages into an internal metamodel corresponding to a network of stochastic hybrid automata (SHA [30]) with discrete variables. The following input languages are currently supported:

- Modest [30], a process algebra-based modelling language for stochastic timed systems featuring high-level constructs such as recursive process calls, loops, and exception handling;
- xSADF [35], an extension of scenario-aware dataflow with continuous probability distributions and nondeterminism, a formalism particularly suited to the study of embedded streaming applications; and
- Jani [14], a model exchange format designed to improve the interoperation of quantitative verification tools. Other tools provide converters to Jani from various Petri net formats or the Prism language [47]. Jani closely corresponds to the Modest Toolset's internal metamodel.

Due to the mapping to a single internal representation, modes naturally provides RES capabilities for all of these input languages. The complexity of generating simulation traces—for RES or Monte Carlo simulation—however inherently depends on the underlying mathematical modelling formalism. An input language may support multiple such formalisms. modes contains simulation algorithms specifically optimised for the following cases [15]:

- For **DTMC** (discrete-time Markov chains), simulation is simple and efficient: obtain the current state's probability distribution over successors, randomly select one of them (using the distribution's probabilities), and continue from that state.
- For **CTMC**, the situation is similar: obtain the set of enabled outgoing transitions, randomly select a delay from the exponential distribution parameterised by the sum of their rates, then make a random selection of one transition weighted by the transitions' rates.
- **PTA** extend Markov decision processes with clocks, transition guards, and location invariants as in timed automata. PTA explicitly keep a memory of elapsed times in the clocks. They admit finite-state abstractions that preserve reachability probabilities and allow them to essentially be simulated as DTMC. modes implements region graph- and zone-based simulation of PTA as DTMC [21, 36]. With fewer restrictions, they can also be treated as STA.
- **STA** extend PTA with general continuous probability distributions. The STA simulator needs to keep track of the values of all clocks. For each transition, it has to compute the set of time points at which the transition is enabled. These sets can be unions of several disjoint intervals. Overall, STA simulation thus requires relatively higher computational effort.

6.2. The Fig Tool

The compositional importance function generation was first developed for and implemented in the Fig simulator [11]. Fig implements the splitting and threshold selection methods described in sections 4 and 5 for transient properties. Additionally, it supports steady-state properties using Restart in combination with all of the other techniques. Fig was developed specifically for input/output stochastic automata (IOSA [22]), originally designed in an extension of the Prism language [47] to consider arbitrary (continuous) probability distributions as in stochastic automata [18]. It recently added support for the fragment of Jani corresponding to a standard encoding of IOSA. Fig supports two variants of the IOSA formalism and modelling language:

- **IOSA** [22] allows the description of stochastic automata, where clocks variables control and observe the passage of time. In [22] a list of constraints ensures that models written in IOSA cannot exhibit nondeterministic behaviour. Input/output communication semantics are used, where all modules are input enabled and each output action can only be produced by a single module.
- **IOSA-U** or *IOSA with urgency* [19] is an extension of the original language where actions can be *urgent*. An enabled transition decorated with an urgent (output) action must be taken immediately, viz. without the passage of time. Urgent inputs can only communicate with urgent outputs. As in IOSA, a series of rules in IOSA-U ensures that models cannot show nondeterminism.

Similar to modes' STA simulation engine, Fig needs to keep track of the values and expiration times of individual clocks. For each transition, it needs to compute the (single) point in time when it becomes enabled. While slightly more efficient than STA simulation, using Fig for CTMC (which are IOSA that only use the

exponential distribution) will incur an overhead compared to a dedicated CTMC simulator as in modes.

7. Experimental Evaluation

The goal of our work was to find a RES approach that provides consistently good performance at a maximal degree of automation. Aside from the compositional importance function generation, we have three splitting methods and two approaches to threshold selection, all implemented in two different tools, at our disposal now. To find out whether there is a combination of all of these that consistently works well, and that could thus be used as a fully-automated default setting in modes and Fig, we perform an experimental evaluation of all method combinations on a number of benchmarks and case studies from the literature. For transient properties, we use modes in order to cover a wide variety of modelling formalisms from CTMC to STA and exploit its more efficient specialised simulation engines. For steady-state properties, we use Fig on both CTMC encoded as IOSA and true IOSA models that make use of non-Markovian continuous probability distributions.

7.1. Transient Properties

For transient properties, we use modes to evaluate the performance of all relevant combinations of the implemented RES methods on CTMC queueing models, network protocols modelled as PTA, and a more complex file server setting modelled as STA.

7.1.1. Case Studies

We considered the following models, which are classic RES benchmarks as well as existing case studies that had previously been analysed with model checkers:

tandem: tandem queueing networks are standard benchmarks in probabilistic model checking and RES [24, 26, 27, 50, 62]. We consider the case from [12] with all exponentially distributed interarrival times, i.e. 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. the probability of the second queue becoming full without having been empty before.

openclosed: our second CTMC has two *parallel* queues [28], 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 [11] 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 [45] 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 [12], 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 [33]. 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 $\langle N, MAX, TD\rangle = \langle 16 \cdot 2^M, 4 \cdot M, 4 \cdot 2^M\rangle$. 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 [33] 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 $\mathtt{P}_{=?}(\mathit{true}\ \mathtt{U}\ bc_1 = bc_2 = K)$, 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 [31], 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: $\mathtt{P}_{=?}(\mathit{true}\ \mathtt{U}_{\leqslant 1000}\ \mathit{queue} = C)$.

We consider several queueing systems since these are frequently used benchmarks for RES [24, 26–28, 45, 50, 62]. The CTMC could easily be modified to use general distributions and our techniques and tools would still work the same.

7.1.2. 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 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

Table 1: Model data and performance results for transient properties

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

mean.[4] 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 Section 5.2 using base effort $e \in \{8, 16, 128\}$ since all weights are $\geqslant 2$;
- 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 of using addition to replace $\wedge$ and $\vee$ in the compositional importance function (cf. Section 3) worked well except for *wlan*, where we used max instead.

7.1.3. 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

[4] 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.

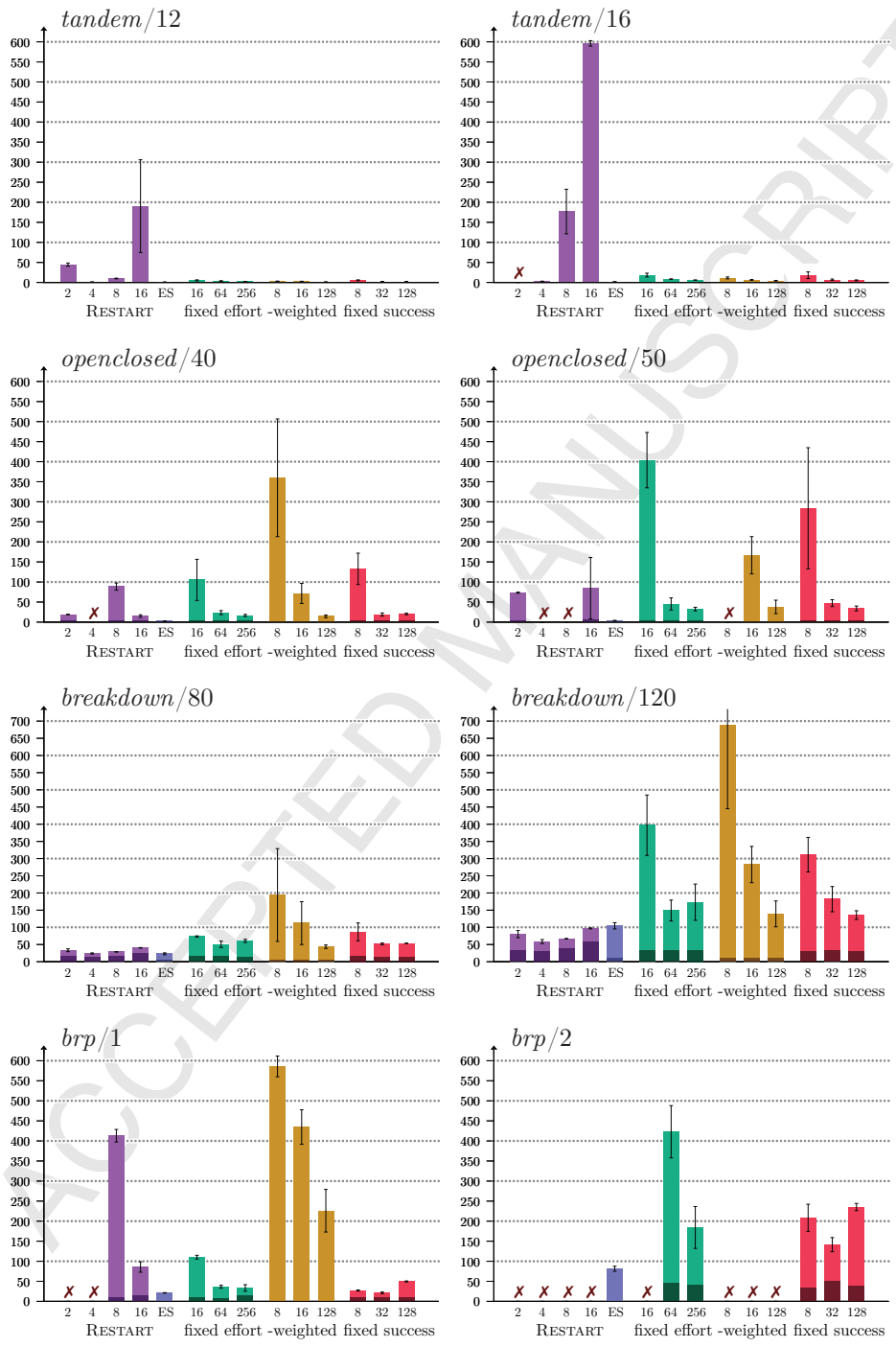

Figure 5: Selected performance results compared (runtimes in seconds)

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 Figure 5. ✗ 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. For instance in the *breakdown* case study, the darker region is practically unnoticeable for the buffer capacity $K = 80$ for RESTART with ES and weighted-fixed effort, and only marginally distinguishable for $K = 120$. The error bars show the standard deviation between the convergence times of the three runs that we performed for each experiment. A larger sample size would be needed for a thorough evaluation of this aspect, though.

Our experiments first confirm previous observations made with the first versions of 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.

7.2. Steady-State Properties

For steady-state properties, we use FIG to evaluate the performance of the presented methods. Of the splitting methods, only RESTART is designed for steady-state properties. We thus only consider the combinations of RESTART with the automatic compositional importance function generation and the two threshold and factor selection methods from Section 5.

7.2.1. Case Studies

We run FIG to estimate steady-state properties on two queueing models and two reliability evaluation examples. One of each is a CTMC. In the non-Markovian queueing model, arrival and service times follow Erlang distributions; of the non-Markovian reliability model, we consider two variants with different kinds of distributions for failures and repairs. In all, we consider the following case studies:

tandem: this is the same tandem queueing network as considered for transient properties in Section 7.1. We estimate the steady-state probability of a saturation in the second queue, that is $\mathtt{S}_{=?}\,(q_2 = C)$.

3-tandem: following the same concept as the tandem queue with an additional third queue in succession to the second one, in the triple tandem queue studied in [63] the service times for all queues follow an Erlang distribution. The shape parameter $\alpha \in \{\,2, 3\,\}$ is the same for all servers. The load at the third queue is always $1/3$, meaning that its scale parameter is $\mu_3 = 1/6, 1/9$ when $\alpha = 2, 3$, respectively. The scale parameters μ_1, μ_2 of the first and second queues, as well as the capacity C in all queues, are chosen so that the steady-state probability of a saturation in the third queue is of the same order of magnitude for all variants of the model. Here we study cases $A, \ldots, F$ defined by $(\alpha, \mu_1, \mu_2, C) = (2, 1/3, 1/4, 12)$, $(3, 2/3, 1/6, 9)$, $(2, 1/6, 1/4, 13)$, $(3, 1/9, 1/6, 10)$, $(2, 1/10, 1/8, 15)$, and $(3, 1/15, 1/12, 13)$. We estimate $\mathtt{S}_{=?}\,(q_3 = C)$, the steady-state probability of a saturation in the third queue, which is around $5.0\text{E}{-10}$ for all cases.

database: our first (Markovian) reliability evaluation example models a database computer facility consisting of disks arranged in clusters, disk controllers, and processors. Originally studied in [29] and later using RESTART in [61], for redundancy R the system is composed of two types of processors and two types of controllers, each with R copies of each type, and six disk clusters, with $R+2$ disks each. The lifetime of these units is exponentially distributed with failure rates μ_D, μ_C, and μ_P for disks, controllers, and processors, respectively. A unit can fail, with equal probability, in mode 1 or 2: repair rates are 1 and 0.5 per time unit for failure modes 1 and 2 respectively. The system is operational as long as fewer than R processors of each type, R controllers of each type, and R disks on each cluster, have failed. The number of components in the database system grows rapidly as R increases: in spite of its Markovian nature, analyses with standard model checking techniques become infeasible for redundacy values as low as $R = 4$ due to state space explosion [12]. We study system unavailability (i.e. the proportion of time the system is not operational in the long run) for $R \in \{\,2, \ldots, 5\,\}$ and failure rates $(\mu_D, \mu_C, \mu_P) = (1/75, 1/25, 1/25)$. The corresponding steady-state property e.g. for $R = 2$ is $\mathtt{S}_{=?}\left((d_1^1 \wedge d_2^1) \vee (d_2^1 \wedge d_3^1) \vee \cdots \vee (p_1^2 \wedge p_2^2)\right)$.

pipeline: the final and most challenging case studies we consider are *consecutive-k-out-of-n: F systems*, consisting of a sequence of n nodes ordered sequentially, where the whole system fails if k or more consecutive nodes fail. This resembles a pipeline where fluid is pushed through via homogeneously distributed pumps: redundancy is built-in so that if less than k consecutive pumps fail, the fluid can still be pushed by the ones remaining. We study the non-Markovian and repairable systems analysed in e.g. [64, 71], with a single repairman whose repair

times follow a log-normal distribution with parameters $\mu = 1.21$ and $\sigma = 0.8$. For $n = 12$ nodes and $k \in \{2, \ldots, 5\}$ we analyse two variants: *pipeline (exp)* has exponentially distributed failures with rate $\mu = 0.001$, and *pipeline (ray)* has Rayleigh failures (i.e. Weibull with shape parameter $k = 2$) with parameter $\beta = 0.00000157$. Notably, the single-repairman setup required the use of the IOSA-U model syntax. We study system unavailability, which e.g. for $k = 2$ corresponds to the steady-state property $\mathsf{S}_{=?}((n_1 \wedge n_2) \vee \cdots \vee (n_{11} \wedge n_{12}))$.

7.2.2. Experimental Setup

All experiments were performed on dual-processor 16-core Intel Xeon E5-2683-v4 (2.1/3.0 GHz) systems running 64-bit Ubuntu with Linux kernel v4.4.0-116. FIG is currently a single-threaded tool, and we ran one instance of FIG per CPU core to perform multiple experiments in parallel. For this reason, and since steady-state simulations using FIG's more generic IOSA engine take more time than transient properties in modes, we used a timeout of 30 minutes for the *tandem* and *3-tandem* models and 60 minutes for the reliability evaluation models. To achieve a meaningful comparison that is more robust to timeouts, we adopt a different approach than the 10 % relative confidence interval as used for transient properties: for our steady-state runs, we instead let all experiments run up to the timeout and report the relative half-width of the confidence interval attained at that point in percent. Thus again, lower numbers indicate better performance. 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");
- RESTART with thresholds and splitting factors determined by the ES method (labelled "RESTART ES").

As we did for transient properties, here we also performed Monte Carlo simulation (SMC) as a comparison baseline; FIG uses Agresti-Coull binomial confidence intervals with Student's-t quantiles.

In all cases, we used addition to replace $\wedge$ and $\vee$ in the compositional importance function (cf. Section 3).

7.2.3. Results

We provide an overview of the performance results for all model instances in Table 2. For the *database* and *pipeline* models, we report the averages of three runs of each experiment again. Column $\hat{p}$ lists the average of all (up to 18) individual estimates for each instance, which were again consistent. Column t/o recalls the timeout used for the respective models. The remaining columns report the half-width of the confidence interval, in percent of $\hat{p}$, obtained at the timeout. Dashes mark cases where the rare event was not encountered even once.

Our experiments show that steady-state properties on the considered models are truly challenging for automated importance splitting. In most cases, one of the method combinations is still noticeably better than plain SMC. However, unlike for transient properties, there is no combination that performs consistently best. Notwithstanding, the combination of RESTART with expected success was

Table 2: Model data and performance results for steady-state properties

model/param		$\hat{p}$	t/o	SMC	RESTART 2	4	8	16	ES
tandem	8	6.2E−5	30	1	1	1	1	1	1
	12	8.4E−7		8	3	3	2	2	2
	16	1.7E−8		106	10	12	12	3	6
	20	1.9E−10		—	23	36	6	8	39
3-tandem	*A*	2.5E−10	30	—	5	62	49	56	78
	B	3.4E−10		—	38	44	79	51	39
	C	5.1E−10		—	22	54	52	56	66
	D	9.9E−10		—	46	36	57	39	71
	E	9.5E−10		66	26	78	43	32	56
	F	1.5E−9		—	64	49	37	45	54
data-base	2	3.8E−2	60	0	0	0	0	0	0
	3	5.9E−4		1	1	1	2	1	1
	4	6.6E−6		8	10	12	14	12	8
	5	6.1E−8		104	65	101	128	96	107
pipe-line (exp)	2	4.8E−4	60	0	0	0	0	0	0
	3	7.4E−6		3	3	3	3	3	3
	4	2.1E−7		22	19	21	24	25	25
	5	1.1E−8		189	169	88	43	188	61
pipe-line (ray)	2	4.8E−4	60	0	0	0	0	0	0
	3	7.4E−6		3	3	3	3	3	3
	4	2.1E−7		23	23	20	30	21	20
	5	1.2E−8		48	91	220	119	67	96

always competitive, and it remains reasonable to keep it as a safe—if not always optimal—default. It is in any case preferable as the only method combination that provides *fully* automated RES.

8. Conclusion

We investigated ways to automate and improve the performance of importance splitting to perform rare event simulation for general classes of stochastic models. For this purpose, we provided a memory-efficient method to automatically derive an importance function from a compositional formal model [12]. The method takes into account the structure of the model's state space as well as the structure of the logical formula that identifies the rare event. Any method to derive importance functions is necessarily a heuristic, but this one appears to work well for diverse case studies. We further studied and implemented three existing splitting methods and two threshold selection algorithms, one of them new. The modes tool, which contains our implementation of all methods for transient properties, is publicly available as part of the MODEST TOOLSET at `www.modestchecker.net`. The FIG simulator provides complementary support for steady-state properties. Using both tools, 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 for transient properties: automatic compositional importance functions together with RESTART

and the expected success method. It pushes automated importance splitting for general models into the realm of very rare events with probabilities down to the order of 10^{-23}. For steady-state properties, however, the picture is not so clear: different methods and method parameterisations work best for different model instances. Still, the fully automated combination of RESTART with expected success shows competitive performance and thus appears as a reasonable default. Further research will be necessary to find out what the key differences are in the behaviour of transient and steady-state analysis to cause such distinct results.

As future work, we would also 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. We note that our methods have already successfully been combined with the lightweight scheduler sampling techniques of [20, 21, 36] to properly handle models that include nondeterminism, as reported in [15].

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, NWO VENI grant no. 639.021.754, and SeCyT-UNC projects 05/BP12 and 05/B497.

References

[1] Michael Amrein and Hans R. Künsch. A variant of importance splitting for rare event estimation: Fixed number of successes. *ACM Transactions on Modeling and Computer Simulation*, 21(2):13:1–13:20, 2011.

[2] Christel Baier, Joost-Pieter Katoen, and Holger Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 146–161. Springer, 1999.

[3] Anthony J. Bayes. Statistical techniques for simulation models. *Australian Computer Journal*, 2(4):180–184, 1970.

[4] Anthony J. Bayes. A minimum variance sampling technique for simulation models. *Journal of the ACM*, 19(4):734–741, 1972.

[5] Denis Benasciutti and Roberto Tovo. On fatigue damage assessment in bimodal random processes. *International Journal of Fatigue*, 29(2):232–244, 2007.

[6] José Blanchet and Michel Mandjes. *Rare Event Simulation for Queues*, chapter 5, pages 87–124. In Rubino and Tuffin [57], 2009.

[7] José Blanchet and Daniel Rudoy. *Rare Event Simulation and Counting Problems*, chapter 8, pages 171–192. In Rubino and Tuffin [57], 2009.

[8] Henk A. P. Blom, G. J. (Bert) Bakker, and Jaroslav Krystul. *Rare Event Estimation for a Large-Scale Stochastic Hybrid System with Air Traffic Application*, chapter 9, pages 193–214. In Rubino and Tuffin [57], 2009.

[9] Henrik C. Bohnenkamp, Pedro R. D'Argenio, Holger Hermanns, and Joost-Pieter Katoen. MoDeST: A compositional modeling formalism for hard and softly timed systems. *IEEE Transactions on Software Engineering*, 32(10): 812–830, 2006.

[10] Thomas Booth. *Particle Transport Applications*, chapter 10, pages 215–242. In Rubino and Tuffin [57], 2009.

[11] Carlos E. Budde. *Automation of Importance Splitting Techniques for Rare Event Simulation*. PhD thesis, Universidad Nacional de Córdoba, Córdoba, Argentina, 2017.

[12] Carlos E. Budde, Pedro R. D'Argenio, and Raúl E. Monti. Compositional construction of importance functions in fully automated importance splitting. In *VALUETOOLS*. ACM, 2016.

[13] Carlos E. Budde, Pedro R. D'Argenio, and Arnd Hartmanns. Better automated importance splitting for transient rare events. In *SETTA*, volume 10606 of *Lecture Notes in Computer Science*, pages 42–58. Springer, 2017.

[14] Carlos E. Budde, Christian Dehnert, Ernst Moritz Hahn, Arnd Hartmanns, Sebastian Junges, and Andrea Turrini. JANI: Quantitative model and tool interaction. In *TACAS*, volume 10206 of *Lecture Notes in Computer Science*, pages 151–168. Springer, 2017.

[15] Carlos E. Budde, Pedro R. D'Argenio, Arnd Hartmanns, and Sean Sedwards. A statistical model checker for nondeterminism and rare events. In *TACAS*, volume 10806 of *Lecture Notes in Computer Science*, pages 340–358. Springer, 2018.

[16] Frédéric Cérou and Arnaud Guyader. Adaptive multilevel splitting for rare event analysis. *Stochastic Analysis and Applications*, 25(2):417–443, 2007.

[17] Frédéric Cérou, Pierre Del Moral, Teddy Furon, and Arnaud Guyader. Sequential Monte Carlo for rare event estimation. *Statistics and Computing*, 22 (3):795–808, 2012.

[18] Pedro R. D'Argenio and Joost-Pieter Katoen. A theory of stochastic systems part I: stochastic automata. *Information and Computation*, 203(1): 1–38, 2005.

[19] Pedro R. D'Argenio and Raúl E. Monti. Input/output stochastic automata with urgency: Confluence and weak determinism. In *ICTAC*, volume 11187 of *Lecture Notes in Computer Science*, pages 132–152. Springer, 2018.

[20] Pedro R. D'Argenio, Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Smart sampling for lightweight verification of Markov decision processes. *Software Tools for Technology Transfer*, 17(4):469–484, 2015.

[21] Pedro R. D'Argenio, Arnd Hartmanns, Axel Legay, and Sean Sedwards. Statistical approximation of optimal schedulers for probabilistic timed automata. In *iFM*, volume 9681 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2016.

[22] Pedro R. D'Argenio, Matias David Lee, and Raúl E. Monti. Input/output stochastic automata – compositionality and determinism. In *FORMATS*, volume 9884 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 2016.

[23] George S. Fishman and L. Stephen Yarberry. An implementation of the batch means method. *INFORMS Journal on Computing*, 9(3):296–310, 1997.

[24] Marnix J. J. Garvels. *The splitting method in rare event simulation*. PhD thesis, University of Twente, Enschede, The Netherlands, 2000.

[25] Marnix J. J. Garvels and Dirk P. Kroese. A comparison of RESTART implementations. In *Winter Simulation Conference*, pages 601–608, 1998.

[26] Marnix J. J. Garvels, Jan-Kees C. W. van Ommeren, and Dirk P. Kroese. On the importance function in splitting simulation. *European Transactions on Telecommunications*, 13(4):363–371, 2002.

[27] Paul Glasserman, Philip Heidelberger, Perwez Shahabuddin, and Tim Zajic. A large deviations perspective on the efficiency of multilevel splitting. *IEEE Transactions on Automatic Control*, 43(12):1666–1679, 1998.

[28] Paul Glasserman, Philip Heidelberger, Perwez Shahabuddin, and Tim Zajic. Multilevel splitting for estimating rare event probabilities. *Operations Research*, 47(4):585–600, 1999.

[29] Ambuj Goyal, Perwez Shahabuddin, Philip Heidelberger, Victor F. Nicola, and Peter W. Glynn. A unified framework for simulating Markovian models of highly dependable systems. *IEEE Transactions on Computers*, 41(1):36–51, 1992.

[30] Ernst Moritz Hahn, Arnd Hartmanns, Holger Hermanns, and Joost-Pieter Katoen. A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods in System Design*, 43(2):191–232, 2013.

[31] Ernst Moritz Hahn, Arnd Hartmanns, and Holger Hermanns. Reachability and reward checking for stochastic timed automata. *Electronic Communications of the EASST*, 70, 2014.

[32] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[33] Arnd Hartmanns and Holger Hermanns. A Modest approach to checking probabilistic timed automata. In *QEST*, pages 187–196. IEEE Computer Society, 2009.

[34] Arnd Hartmanns and Holger Hermanns. The Modest Toolset: An integrated environment for quantitative modelling and verification. In *TACAS*, volume 8413 of *Lecture Notes in Computer Science*, pages 593–598. Springer, 2014.

[35] Arnd Hartmanns, Holger Hermanns, and Michael Bungert. Flexible support for time and costs in scenario-aware dataflow. In *EMSOFT*, pages 3:1–3:10. ACM, 2016.

[36] Arnd Hartmanns, Sean Sedwards, and Pedro R. D'Argenio. Efficient simulation-based verification of probabilistic timed automata. In *Winter Simulation Conference*, pages 1419–1430, 2017.

[37] Philip Heidelberger. Fast simulation of rare events in queueing and reliability models. *ACM Transactions on Modeling and Computer Simulation*, 5(1):43–85, 1995.

[38] Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. Approximate probabilistic model checking. In *VMCAI*, volume 2937 of *Lecture Notes in Computer Science*, pages 73–84. Springer, 2004.

[39] Kin-Ping Hui, Nigel Bean, Miro Kraetzl, and Dirk P. Kroese. The cross-entropy method for network reliability estimation. *Annals of Operations Research*, 134(1):101–118, 2005.

[40] Cyrille Jégourel, Axel Legay, and Sean Sedwards. Importance splitting for statistical model checking rare properties. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 576–591. Springer, 2013.

[41] Cyrille Jégourel, Axel Legay, and Sean Sedwards. An effective heuristic for adaptive importance splitting in statistical model checking. In *ISoLA*, volume 8803 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2014.

[42] Cyrille Jégourel, Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Distributed verification of rare properties using importance splitting observers. *Electronic Communications of the EASST*, 72, 2015.

[43] Cyrille Jégourel, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, and Sean Sedwards. Importance sampling for stochastic timed automata. In *SETTA*, volume 9984 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 2016.

[44] Herman Kahn and Ted E. Harris. Estimation of particle transmission by random sampling. *National Bureau of Standards applied mathematics series*, 12:27–30, 1951.

[45] Dirk P. Kroese and Victor F. Nicola. Efficient estimation of overflow probabilities in queues with breakdowns. *Performance Evaluation*, 36:471–484, 1999.

[46] Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.

[47] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.

[48] Averill M. Law. *Simulation modeling and analysis*. McGraw-Hill Education, 2014.

[49] Averill M. Law and John S. Carson. A sequential procedure for determining the length of a steady-state simulation. *Operations Research*, 27(5):1011–1025, 1979.

[50] Pierre L'Ecuyer, Valérie Demers, and Bruno Tuffin. Rare events, splitting, and quasi-Monte Carlo. *ACM Transactions on Modeling and Computer Simulation*, 17(2), 2007.

[51] Pierre L'Ecuyer, François Le Gland, Pascal Lezaud, and Bruno Tuffin. *Splitting Techniques*, chapter 3, pages 39–61. In Rubino and Tuffin [57], 2009.

[52] Pierre L'Ecuyer, Michel Mandjes, and Bruno Tuffin. *Importance Sampling in Rare Event Simulation*, chapter 2, pages 17–38. In Rubino and Tuffin [57], 2009.

[53] François LeGland and Nadia Oudjane. A sequential particle algorithm that keeps the particle system alive. In *EUSIPCO*, pages 1–4. IEEE, 2005.

[54] Loren D. Lutes and Curtis E. Larsen. Improved spectral method for variable amplitude fatigue prediction. *Journal of Structural Engineering (United States)*, 116(4):1149–1164, 1990.

[55] Marco Paolieri, András Horváth, and Enrico Vicario. Probabilistic model checking of regenerative concurrent systems. *IEEE Transactions on Software Engineering*, 42(2):153–169, 2016.

[56] Daniël Reijsbergen, Pieter-Tjerk de Boer, Werner R. W. Scheinhardt, and Boudewijn R. Haverkort. Automated rare event simulation for stochastic Petri nets. In *QEST*, volume 8054 of *Lecture Notes in Computer Science*, pages 372–388. Springer, 2013.

[57] Gerardo Rubino and Bruno Tuffin, editors. *Rare Event Simulation Using Monte Carlo Methods*. John Wiley & Sons, Ltd, 2009.

[58] Gerardo Rubino and Bruno Tuffin. *Introduction to Rare Event Simulation*, chapter 1, pages 1–13. In Rubino and Tuffin [57], 2009.

[59] Walter L. Smith. Regenerative stochastic processes. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 232 (1188):6–31, 1955.

[60] Natalie M. Steiger and James R. Wilson. Convergence properties of the batch means method for simulation output analysis. *INFORMS Journal on Computing*, 13(4):277–293, 2001.

[61] José Villén-Altamirano. RESTART method for the case where rare events can occur in retrials from any threshold. *International Journal of Electronics and Communications*, 52:183–189, 1998.

[62] José Villén-Altamirano. Rare event RESTART simulation of two-stage networks. *European Journal of Operational Research*, 179(1):148–159, 2007.

[63] José Villén-Altamirano. RESTART simulation of networks of queues with erlang service times. In *Winter Simulation Conference*, pages 1146–1154, 2009.

[64] José Villén-Altamirano. RESTART simulation of non-Markov consecutive-k-out-of-n: F repairable systems. *Reliability Engineering & System Safety*, 95 (3):247–254, 2010.

[65] Manuel Villén-Altamirano and José Villén-Altamirano. RESTART: a method for accelerating rare event simulations. In *Queueing, Performance and Control in ATM (ITC-13)*, pages 71–76. Elsevier, 1991.

[66] Manuel Villén-Altamirano and José Villén-Altamirano. RESTART: a straightforward method for fast simulation of rare events. In *Winter Simulation Conference*, pages 282–289, 1994.

[67] Manuel Villén-Altamirano and José Villén-Altamirano. Analysis of restart simulation: Theoretical basis and sensitivity study. *European Transactions on Telecommunications*, 13(4):373–385, 2002.

[68] Manuel Villén-Altamirano and José Villén-Altamirano. On the efficiency of RESTART for multidimensional state systems. *ACM Transactions on Modeling and Computer Simulation*, 16(3):251–279, 2006.

[69] Manuel Villén-Altamirano and José Villén-Altamirano. The rare event simulation method RESTART: efficiency analysis and guidelines for its application. In *Network Performance Engineering*, volume 5233 of *Lecture Notes in Computer Science*, pages 509–547. Springer, 2011.

[70] Manuel Villén-Altamirano, A. Martínez-Marrón, J. Gamo, and F. Fernández-Cuesta. Enhancement of the accelerated simulation method RESTART by considering multiple thresholds. In *Proc. 14th Int. Teletraffic Congress*, pages 797–810, 1994.

[71] Gang Xiao, Zhizhong Li, and Ting Li. Dependability estimation for non-Markov consecutive-k-out-of-n: F repairable systems by fast simulation. *Reliability Engineering & System Safety*, 92(3):293 – 299, 2007.

[72] Håkan L. S. Younes and Reid G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 223–235. Springer, 2002.

[73] Armin Zimmermann and Paulo Maciel. Importance function derivation for RESTART simulations of Petri nets. In *RESIM*, pages 8–15, 2012.

[74] Armin Zimmermann, Daniël Reijsbergen, Alexander Wichmann, and Andres Canabal Lavista. Numerical results for the automated rare event simulation of stochastic Petri nets. In *RESIM*, pages 1–10, 2016.