Automated Reliability Estimation over Partial Systematic Explorations

Esteban Pavese Víctor Braberman
Universidad de Buenos Aires
{epavese,vbraber}@dc.uba.ar

Sebastián Uchitel
Universidad de Buenos Aires and
Imperial College London
suchitel@dc.uba.ar

***Abstract*—Model-based reliability estimation of software systems can provide useful insights early in the development process. However, computational complexity of estimating reliability metrics such as mean time to first failure (MTTF) can be prohibitive both in time, space and precision. In this paper we present an alternative to exhaustive model exploration–as in probabilistic model checking–and partial random exploration–as in statistical model checking. Our hypothesis is that a (carefully crafted) partial systematic exploration of a system model can provide better bounds for reliability metrics at lower computation cost. We present a novel automated technique for reliability estimation that combines simulation, invariant inference and probabilistic model checking. Simulation produces a probabilistically relevant set of traces from which a state invariant is inferred. The invariant characterises a partial model which is then exhaustively explored using probabilistic model checking. We report on experiments that suggest that reliability estimation using this technique can be more effective than (full model) probabilistic and statistical model checking for system models with rare failures.**

I. Introduction

Model-based automated verification for assessing reliability of software systems aims to provide insights early in the development process that can reduce significantly both development costs, and costs associated with deploying unreliable software. However, traditional reliability metrics such as mean time to first failure (MTTF) require models that support describing probabilistic, non-deterministic and timed behaviour; models for which estimating such metrics can be prohibitive in time, space and/or precision.

Model checking is emerging as an effective software verification method. In particular, in the context of reliability, quantitative guarantees (on MTTF and other measures) can be computed for complex models using the techniques developed in the area known as probabilistic model checking [1], [2]. These techniques input probabilistic models (such as Markov Chains and Markov Decision Processes) and can assess quantitative properties through exhaustive exploration of the model state space and subsequent numerical analysis.

Applicability of probabilistic model checking for reliability assessment of complex models is threatened by the size of the model. Although state space reduction techniques exist, they may still fail to prevent state explosion on complex enough models. Further, even if the entire state space can be explored, its size typically impedes exact numerical calculation (e.g. Gaussian elimination) of reliability metrics, so iterative methods (such as Jacobi or Gauss-Seidel) that approximate metrics must be used. However, these methods do not have convergence guarantees, and when they do converge they may do so (intractably) slowly. The latter can be a problem for reliability metrics of models with unlikely failures (e.g. probability of failure in a fixed period below 10^{-5}), and can lead to iterations being cut short far from the actual value of the metric being estimated.

In summary, although probabilistic model checking may seem to promise exact calculation of quantitative reliability properties, state space explosion and numerical methods can be computationally prohibitive or result in poor approximations. Despite these limitations, probabilistic model checking can provide bounds with 100% confidence for reliability metrics even though the distance of these bounds to the real value cannot be known.

Numerical analysis and, to some extent, state explosion can be avoided using statistical techniques. These apply statistical inference over a finite set of sample executions extracted from the model. Variations of these approaches are usually referred to as Monte Carlo estimations. When using these techniques to estimate metrics such as MTTF, the population mean X is approximated through an estimator such as the sample mean $\overline{X}$ [3]. Of course, such estimation is subject to statistical error and thus it is crucial to understand how far and how likely the estimator deviates from the actual mean.

The deviations from the actual value that result from the specific samples used is usually conveyed in terms of statistical errors and confidence intervals. Bounds for statistical error and confidence intervals can be computed based on the number of samples being analysed. Although significant progress for fast generation of random walks over models has been made, sample generation can be very costly timewise even for analyses with modest guarantee requirements due to the sheer number of samples required. For Monte Carlo reliability estimations, the length of samples can be particularly problematic, since sampled executions must reach a failure in order to allow computing an estimator. This may make sample generation for high-reliability systems intractable.

In summary, statistical techniques can provide approximations with measurable confidence intervals and error bounds. However, in the presence of large models with rare failures, the number and length of samples may make such techniques intractable or lose guarantees over results.

In this paper we present an alternative to exhaustive model exploration –as in probabilistic model checking– and partial random exploration –as in statistical model checking– which may counter some of the limitations of existing model-based reliability verification techniques. Our hypothesis, inspired on the Pareto principle, is that a (carefully crafted) partial systematic exploration of system models can provide good bounds on reliability metrics with lower computation cost. More specifically, probabilistic model checking of a submodel of the system can bound the value of reliability metrics for the complete model in a cost effective manner. Furthermore, it can produce better approximations, given equal time and memory budgets, than probabilistic and statistical model checking.

We hypothesise that there is a gain to be had by identifying a small probabilistically significant portion of the state space, considering all other states as failures and performing probabilistic model checking on the resulting submodel. The intuition is that, in contrast to full-model probabilistic model checking, performing a probabilistic check on a portion of the full model allows for faster iterations of the numerical analysis methods. Consequently, more iterations can be performed within the same time budget and, for slowly converging models, a better approximation may be achieved.

More specifically, in this paper we present a novel automated technique for MTTF estimation that combines simulation, invariant inference and probabilistic model checking. We use simulation to produce a set of traces that represent likely behaviour of the full model. These traces are used to infer invariants of the state space explored during the simulation. A submodel containing all states that satisfy the invariant is constructed and its MTTF is computed using a probabilistic model checker.

The technique we propose obtains lower bounds to real MTTFs with 100% confidence (as full-model probabilistic model checking and in contrast to statistical model checking). Preliminary evidence shows that the lower bounds achieved (for a fixed budget of time and memory) are higher than those obtained by full model probabilistic and stochastic model checking for models with rare failures. Furthermore, automated invariant generation seems to perform reasonably well against domain-expert provided invariants.

The remainder of the paper is organized as follows. In section II we provide background. In Section III we describe our approach to the MTTF estimation. Section IV provides case studies illustrating the approach and comparing results to existing techniques. In Section V we present a discussion of our results and of related work. Finally we offer our conclusions and discuss future work in Section VI.

II. Background

Mean time to first failure [3] is a widely accepted metric for software reliability. The metric represents the time a client can expect to operate a software system until it experiences its first failure. In order to calculate such a measure, practitioners base their efforts on a *failure model* [3], which describes conditions under which the component is known to fail. Most often, these conditions are probabilistic in nature. This failure behaviour is usually modelled with a Markov chain.

Definition 2.1 (Discrete Time Markov Chain): A *Discrete Time Markov Chain* (DTMC) [4] is defined by a tuple $\langle S, s_0, A, R\rangle$ where $S \subseteq V \rightarrow C$ is a finite set of states, defined by mapping a finite set of variables V to values on a finite subset of $\mathbb{Z}$, C. $s_0 \in S$ is the initial state. A is a finite set of action labels. $R \subseteq S \times (A \cup \{\tau\}) \times \mathcal{D}(S)$ is the transition relation where the transition target is defined by a distribution on target states. R is such that $\forall s_i \in S, \exists! a \in (A \cup \{\tau\}), \exists! \mu \in \mathcal{D}(S)$ such that $(s_i, a, \mu) \in R$. That is, the choice of transition distributions is deterministic.

Complex DTMCs can be built compositionally using parallel composition [4] to model components that run asynchronously but synchronise on shared actions.

In order to assert intended properties of DTMC models several modal logics have been proposed. pCTL [5], a probabilistic extension of CTL, is widely used to describe model properties. pCTL formulae differ from CTL in that, instead of predicating about properties that may hold globally or for some execution paths, they aim at quantifying the probability of witnessing traces that satisfy a given property.

Definition 2.2 (Traces and Probabilities): Given a DTMC $M = \langle S, s_0, A, R\rangle$, an *execution trace* on M is a nonempty and possibly infinite sequence $\pi = s_0 \xrightarrow{p_0} s_1 \xrightarrow{p_1} s_2 \ldots$, such that for all i, $s_i \in S$ and there exists a unique $(s_i, a, \mu) \in R$ such that $\mu(s_{i+1}) = p_i > 0$. The probability induced by a finite path π is given by $Pr(\pi, M) = \prod_{0 \leq i \leq length(\pi)} p_i$, where $length(\pi)$ is the number of transitions in π. We note the existence of a finite execution trace π from s_0 to s_n by $s_0 \xrightarrow{\pi} s_n$. We will denote the infinite set of all possible traces through M as $\Pi(M)$.

Reward structures are used to convey some sense of value to DTMC traces. For example, a transition reward structure that assigns a value of 1 to each transition is a standard way of defining overall time steps cost for the traces of a DTMC.

Definition 2.3 (Reward Structures): Given a DTMC $M = \langle S, s_0, A, R\rangle$, a *transition reward structure* is a function $\rho : S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$.

Given a trace π of a DTMC M, and a reward structure ρ over M, the *path-reward* of π is the sum of the reward of each of its transitions. We will abuse notation and note $\rho(\pi)$ to note the path-reward of π based on reward structure ρ.

We will note $\Pi_{S_{end}}(M)$ (where S_{end} is a set of states) to refer to the possibly infinite set of all execution traces of M, but where they have been pruned so that the last state of each trace is one of those in S_{end}, and no other state in S_{end} exists in the trace before the end. Note that $\Pi_{S_{end}}(M)$ may contain traces of infinite length (i.e., those that never reach a state in S_{end} and therefore have not been pruned).

Definition 2.4 (Mean time to first failure): Let $M = \langle S, s_0, A, R\rangle$ be a DTMC, $S_{event} \subseteq S$ be a set of states from M representing failure states, and ρ a reward structure over M modelling time passage on transitions. Let time to first failure TF be a random variable on $\mathbb{R}_{\geq 0} \cup \{+\infty\}$ such that the probability of the time to failure being equal to x is defined as

$Pr(TF = x) = \sum_{\pi \in \Pi_{S_{end}}(M), \rho(\pi)=x} Pr(\pi, M)$. The *mean time to first failure (MTTF)* for failures S_{event} in M, noted $\overline{T}(S_{event}, M)$ is given by the *mathematical expectation* of the random variable TF.

It is generally accepted to employ *execution time* rather than *calendar time* for MTTF estimations [3]. While calendar time measures real time in terms of hours, weeks, etc., execution time is the time actually spent executing the software. This distinction is important for reactive systems which may have long idle times.

III. Approach

This section formally defines an approach to computing bounds to MTTF of software models. The approach is based on calculating MTTF for a partial systematic exploration of the model's state space. We first define what is meant by a partial exploration and show that MTTF computed over these partial explorations are guaranteed bounds to the MTTF of the entire system model. We then show how some partial explorations can be specified declaratively through invariant properties that drive the exploration. Finally, we show how these invariant-driven partial explorations can be obtained automatically from any given model, without need for human intervention.

A. Partial Explorations

We refer to a partial exploration of a system model as a submodel. Intuitively, a submodel of a DTMC M is a model that retains a subset of the states and transitions of M – including and reachable from the initial state – and in which all other states in M have been abstracted away into a new λ trap state. Formally, a DTMC submodel is defined as follows, where $supp(\mu)$ denotes the *support set* of the distribution μ, that is, the set of values x_i for which $\mu(x_i) > 0$:

Definition 3.1 (Submodels): Given a DTMC $M = \langle S, s_0, A, R \rangle$, a *submodel* of M is a DTMC $M' = \langle S' \cup \{\lambda\}, s_0, A, R' \rangle$ such that $S' \subseteq S$, $s_0 \in S'$, and $R' \subseteq (S' \cup \{\lambda\}) \times (A \cup \{\tau\}) \times \mathcal{D}(S' \cup \{\lambda\})$ is such that for all $a \in A$

1) for each $(\lambda, a, \mu_{R'}) \in R'$, it must be the case that $supp(\mu_{R'}) = \{\lambda\}$ and $a = \tau$;
2) for all $s \in S'$, for all $a \in A \cup \{\tau\}$, it must be that $\exists \mu_{R'} \in \mathcal{D}(S' \cup \{\lambda\})$ such that $(s, a, \mu_{R'}) \in R' \iff \exists \mu_R \in \mathcal{D}(S \cup \{\lambda\})$ such that $(s, a, \mu_R) \in R$;
3) for all $s_1, s_2 \in S'$ such that $s_1 \neq \lambda$, and for all $a \in A \cup \{\tau\}$ it must be that $\exists \mu_{R'} \in \mathcal{D}(S' \cup \{\lambda\})$ such that $(s_1, a, \mu_{R'}) \in R' \wedge s_2 \in supp(\mu_{R'}) \Rightarrow \exists \mu_R \in \mathcal{D}(S \cup \{\lambda\})$ such that $(s_1, a, \mu_R) \in R \wedge \mu_R(s_2) = \mu_{R'}(s_2)$;
4) for all $s_1 \in S'$ such that $s_1 \neq \lambda$, and for all $a \in A \cup \{\tau\}$ it must be that $\exists \mu_{R'} \in \mathcal{D}(S' \cup \{\lambda\})$ such that $(s_1, a, \mu_{R'}) \in R' \Rightarrow \exists \mu_R \in \mathcal{D}(S \cup \{\lambda\})$ such that $(s_1, a, \mu_R) \in R \cdot \mu_{R'}(\lambda) = 1 - \sum_{s_2 \in supp(\mu_{R'}) \setminus \{\lambda\}} \mu_R(s_2)$.

Submodels are key to our approach since they conservatively approximate MTTF. That is, the MTTF of a model M is always greater or equal to the MTTF of its submodels.

Theorem 3.1 (Submodels bound MTTF): Let M and M' be two DTMCs with state spaces S and S' and such that M' is a submodel of M. If $S_f \subseteq S$ are all the failure states in M then $\overline{T}((S_f \cap S') \cup \{\lambda\}, M') \leq \overline{T}(S_f, M)$.

Proof: Note that, for every trace in the complete model, it either exists completely in the submodel, or the submodel contains only a prefix that is extended by the λ state. Since reward structures are based on transitions, every trace in the full model accrues *at least* as much reward to the failure state (possibly ∞) as the corresponding trace (or prefix) in the submodel. Hence these prefixes contribute to $\overline{T}((S_f \cap S') \cup \{\lambda\}, M')$ at most what their extensions in M contribute to $\overline{T}(S_f, M)$ ■

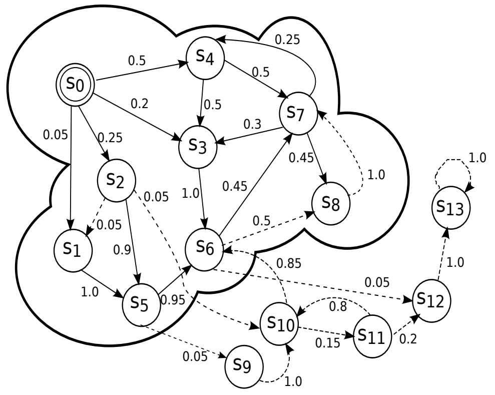

Fig. 1. Example partial exploration of a state space

The above result entails that if computing MTTF of a system model is intractable, it can be conservatively approximated on any of its submodels.

B. Automatic Submodel Generation

Although any submodel will provide a lower bound for MTTF, the key to a tractable reliability estimation technique is to identify a submodel for which its MTTF can be computed within a reasonable time budget, and for which the resulting bound is a reasonable approximation to the MTTF of the full model. Of course, and independently of the fact that the MTTF for the full model is unknown, this is a problem for which coming up with an exact solution (i.e. the "best" submodel) is intractable [6]. In this section we discuss a heuristic for automatically constructing submodels that can provide better bounds for reliability at lower computation cost than full model checking and Monte Carlo approaches.

Our approach adopts a heuristic based on the reasoning that the submodel construction strategy should aim to identify a portion of the model that is probabilistically dense, that is, a small submodel for which the probability of reaching the λ trap state in a fixed time is low. Such models will contain probabilistically likely loops that delay the traces from reaching the submodel boundary, hence contributing to a higher reliability bound.

The problem of finding a probabilistically dense submodel is NP-hard. Our alternative approach attempts to approximate such a submodel through bounded simulation. Hence, we simulate several traces over the full model. The resulting set of finite traces, if sufficiently large and consisting of sufficiently long traces, is likely to cover a good part of a probabilistically dense submodel. These traces form the basis for building our submodels. The smallest submodel that includes the set of states and transitions covered by the simulated traces can be constructed easily by simply adding any non-visited transitions between any two visited states, abstracting all non-visited states into the λ trap state. Figure 1 shows such a construction, where solid lines are transitions covered by the simulated traces, and dotted lines are transitions in the model that were not covered. States outside the boundary have not been covered, and would be abstracted away in the submodel. However, such submodels are likely to have relatively short traces that escape the submodel (see path $s_0, s_2, s_{10}, \ldots$ in the figure). These short traces contribute a relatively high probability of escaping the submodel (the shorter the prefix, the larger the probability of the set of extended traces), reducing the submodel's MTTF. Note that s_{10} falls back within the boundary to s_6 with high probability and including it would increase the submodel's MTTF. This is consistent with our experimentation in [7] where we observed that submodels generated with a breadth first search strategy tend to approximate reliability measures better.

In our approach, rather than adopting a syntactic notion of breadth first traversal for extending the submodel determined by a simulation of the full model, we take a more semantic approach based on the attributes of states visited during the simulation. We compute state invariants based on the states visited during the simulation and then add to the submodel any states and transitions that satisfy the invariant. In this way, we expect to add behaviour that, although different to what was simulated, represents variations in terms of symmetries, race conditions, and independent events [8], and contributes significantly to the probabilistic weight of the submodel.

We now formally define our submodel construction method. We start with the notion of invariant of a set of traces.

Definition 3.2 (Invariant): Given a DTMC $M = \langle S, s_0, A, R\rangle$, and a set of finite execution traces T obtained from said model, an *invariant* of M through T is a state predicate ψ on the variables of M such that for every execution trace $t = s_0 \xrightarrow{p_0} s_1 \xrightarrow{p_1} s_2 \ldots s_n \in T$, it holds that $\forall 0 \leq i \leq n, s_i \models \psi$.

An invariant then induces a unique submodel as follows:

Definition 3.3 (Invariant-driven submodels): Let $M = \langle S, s_0, A, R\rangle$ be a DTMC and ψ an invariant; an *invariant-driven submodel* induced by ψ is a submodel $M' = \langle S' \cup \{\lambda\}, s_0, A', R'\rangle$ of M such that *a)* each state $s' \in S'$ is such that $s' \models \psi$; *b)* for each $s'_1 \in S', s'_1 \neq s_0, s_0 \xrightarrow{\pi} s'_1$; and finally *c)* for all states $s'_2 \in S \setminus S'$ such that there exist $s'_1 \in S, (s'_1, a, \mu_R) \in R$ with $\mu_R(s'_2) > 0$, it is the case that $M, s'_2 \not\models \psi$. In other words, if a state s'_2 not in the submodel is directly reachable from a state s'_1 in the submodel, it must be the case that s'_2 violates ψ. The submodel is thus maximally connected from the initial state through the invariant ψ.

Our approach aims at automatically obtaining invariants. To this end, we produce probabilistically driven walks over the full system model, recording the states (i.e. variable valuations) traversed. We use Daikon [9], an invariant inference engine, to obtain predicates that hold over all traversed states.

IV. Validation

In this section we set out to answer three questions in order to validate our approach.

Q1: can our approach, when compared to model checking over full explorations, produce higher bounds, in less time, for MTTF of system models with rare failures?

Q2: can our approach, when compared to Monte Carlo approaches, produce higher bounds, in less time, for MTTF of system models with rare failures?

Q3: how do the MTTF for submodels compare when these submodels are generated from automatically inferred invariants as in our approach against manually generated ones?

Q1 and *Q2* aim at comparing our proposal with established approaches to reliability estimation, to evaluate if our approach can complement existing techniques. *Q3* aims at assessing the added value of automatic techniques for obtaining submodels.

A. Methodology

For each case study taken from the literature, we built DTMC system models to describe behaviour, modelled failures as state formulae, and defined an appropriate reward structure. We used the same input for all MTTF estimation techniques.

We ran our approach for all case studies for several automatically generated invariants varying the number and length of traces used for invariant inference. We used Daikon v4.6.4[9] configured to produce invariants that are conjunctions of terms of the form $x \sim y$, where x and y are either variables in the model, or integer constants, and $\sim \in \{<, \leq, =, \geq >\}$. The invariants we obtained were used to automatically build an *observer* O, a DTMC that monitors the validity of the invariant. This observer, when composed with the system model M, synchronises with all actions and forces transitioning into the λ trap state whenever the destination state of the intended transition would result in an invariant violation.

For *Q1* we used a modified version of PRISM v4.0.3 [10] to perform probabilistic model checking to estimate MTTF both for the full state space and for its invariant-driven submodels. Modifications allow for batch trace generation (used for invariant inference) and time and memory-use tracking (used for generating intermediate MTTF results and for timing out when time budget is up). Intermediate MTTF results were generated for visualising convergence rates. PRISM was deployed on an 8x Core Intel Xeon CPU @1.60 GHz with 8GB RAM.

PRISM provides different numerical methods for reward calculation. We compared computation of MTTF of the full and partial explorations for the Jacobi, Gauss-Seidel and Power methods, as well as several optimisations over the Jacobi and Gauss-Seidel methods. Due to space limitations, we only report on results obtained with the backwards variation of the Gauss-Seidel method (BGS), which proved to be the most effective method for full model probabilistic model checking in terms of bounds obtained for time budget.

PRISM runs were considered complete when any of the following criteria held: either *a)* the *absolute* difference between results of successive iterations of the numerical method was less than 0.01 (relative differences are not adequate because of slow convergence, which causes iterative methods to cut too early). Alternatively, *b)* running time reached 24 hours; or *c)* memory was exhausted. Note that the time measured includes only the execution of the numerical methods. This allows for convergence analysis and favours full-model exploration as the time spent on construction of the model state space is not considered (we comment on execution time for submodel generation later in the Experimental Results subsection).

For *Q2* Monte Carlo simulations were generated using the same version of PRISM and the same hardware as $Q1$. However, note that while our approach produces lower bounds to MTTF with 100% confidence but for which precision (percentual difference between the estimation and the actual value) is unbounded, Monte Carlo produces estimations with varying degrees of confidence but for which precision can be bounded. Consequently, we performed statistical model checks for a range of confidence and precision values.

For Monte Carlo approaches to be successful, all randomly generated traces must eventually fail, and enough traces must be generated in order to guarantee estimations with a fixed precision and confidence. Setting a trace length horizon for the simulator to ensure all traces fail is typically done based on an estimation of the MTTF. We used the MTTF estimations obtained in $Q1$ to set this horizon for each case study.

In addition to comparing probabilistic model checking of submodels against Monte Carlo simulations of the complete model, we compared probabilistic model checking against Monte Carlo simulations over the same submodels.

Finally, *Q3* uses the same setup and MTTF estimation approach based on automatically inferred invariants as in $Q1$. Manually produced invariants for submodel generation were put forth before any of the experiments were performed. Hence the manually proposed invariants were not tainted by knowledge gained from the automatic approach. The main heuristic for coming up with the invariants was the use of of necessary (and more likely) conditions for failures.

B. Case Studies

Tandem Queueing Network: The first case study is a tandem queueing network, based on [11]. Queueing systems have been extensively studied in queueing theory, and analytical solutions for some variants exist. In the case of this model, general queueing models do not apply. Generating an ad-hoc analytical formulation would require extensive expertise and time.

The system consists of two process queues C and M of given capacities. Clients queue processes for execution in the first queue while it is not full. The queue may either route a process to the second queue after a probabilistically chosen time elapses, or it might choose to deal with the request itself. The behaviour of this first queue is governed by two different *phases*; the difference between the phases is given by the probability with which it will choose to route to the second queue or deal with requests. The second can service its requests after a probabilistically chosen time elapses. A failure is observed when both queues are full. The capacity of the queues is fixed at 1200 each. Clients are less inclined (i.e., they take more time in average) to enqueue processes as the free capacity of the queues decreases.

The reliability metric to be estimated is the mean operational time until the first failure. Consequently, the reward structure ρ assigns the value 1 to every timing transition. The state predicate that captures failure is $cliC = 1200 \wedge cliM = 1200$ and computing the metric amounts to calculating the expectation of the accumulated reward before reaching a state satisfying the state predicate.

Bounded Retransmission Protocol: The second case study [12] models a robust communication protocols that attempts to ensure delivery of data, the bounded retransmission protocol (BRP) [13].

BRP is a variant of the alternating bit protocol, which allows for a bounded number of retransmissions of a given chunk (i.e., a part of a file). The protocol consists of a sender, a receiver, and two lossy channels, used for data and acknowledgements respectively. The sender transmits a file composed of a number of chunks, by way of *frames*. Each frame contains the chunk itself and three bits. The first bit indicates whether the chunk is the first one; the second one if it is the last chunk; and the third bit is the alternating one, used for avoiding data duplication.

The sender waits for acknowledgement of each frame sent. The sender may timeout if either the frame or the corresponding acknowledgement are dropped. When this happens, the sender resends the frame and does so repeatedly up to a specified retry limit. If the limit is reached and the transmission is terminated, the sender may be able to establish that the file was not sent (if some chunks were left unsent) or it may not know the outcome (if the last frame was sent but no acknowledgement was received). In any case, the sender may send a new file, resetting the retry count. A maximum of 256 retransmissions are attempted per file before the sender gives up and aborts transmission of the file. Once a file is sent successfully or its transmission fails, the system waits for another file to be sent.

Protocol clients send files one at a time. Each of these files is of a different size (in number of chunks). The size is selected probabilistically for each file, between just a few and 1500 chunks. Exceedingly large or small files are modelled to be less likely to be sent than those of average size.

We wish to estimate the mean time to the first failure, where failure is defined as the sender failing to send a complete file (*incomplete*) or not being able to establish if a file was sent successfully (*unknown*). Consequently, the state predicate describing failures is $incomplete \vee unknown$. The definition of time for this case study aims at establishing how many data packets can be expected to be sent successfully before failure.

C. Experimental Results

We now present some of the experimental results obtained for the three research questions presented above. The models used and complete experimental results can be found at http://lafhis.dc.uba.ar/~epavese/ICSE13.

Question 1: When comparing probabilistic model checking of both full and partial models we are interested in considering

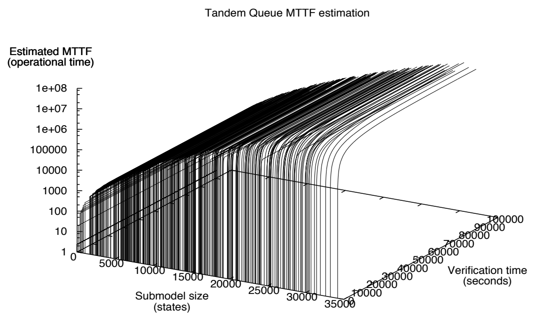

Fig. 2. Results of analysis of Tandem Queue for different sized submodels, Backwards Gauss-Seidel method.

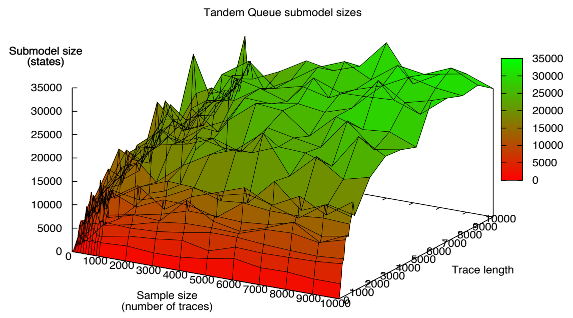

Fig. 3. Tandem Queue submodels sizes for different sample size and trace length parameters.

TABLE I
Selection of Tandem Queue submodel sizes and invariants for different parameter configurations.

Traces	Length	States	Invariant
10000	1	14	$cliC = cliM \wedge cliC \leq 0 \wedge state \leq 2 \wedge cliC \leq state$
10000	501	12690	$cliC \leq 73 \wedge cliM \leq 15 \wedge state \leq 9$
5000	1000	14134	$cliC \leq 69 \wedge cliM \leq 18 \wedge state \leq 9$
10000	1000	16086	$cliC \leq 83 \wedge cliM \leq 17 \wedge state \leq 9$
5000	2000	23388	$cliC \leq 100 \wedge cliM \leq 21 \wedge state \leq 9$
10000	2000	22486	$cliC \leq 92 \wedge cliM \leq 22 \wedge state \leq 9$
5000	3000	20932	$cliC \leq 98 \wedge cliM \leq 19 \wedge state \leq 9$
10000	3000	25228	$cliC \leq 108 \wedge cliM \leq 21 \wedge state \leq 9$
5000	4000	24538	$cliC \leq 105 \wedge cliM \leq 21 \wedge state \leq 9$
10000	4000	24882	$cliC \leq 94 \wedge cliM \leq 24 \wedge state \leq 9$
5000	5000	26424	$cliC \leq 104 \wedge cliM \leq 23 \wedge state \leq 9$
10000	5000	23686	$cliC \leq 97 \wedge cliM \leq 22 \wedge state \leq 9$
5000	6000	26182	$cliC \leq 99 \wedge cliM \leq 24 \wedge state \leq 9$
10000	6000	31902	$cliC \leq 121 \wedge cliM \leq 24 \wedge state \leq 9$
5000	7000	29926	$cliC \leq 123 \wedge cliM \leq 22 \wedge state \leq 9$
10000	7000	30674	$cliC \leq 121 \wedge cliM \leq 23 \wedge state \leq 9$
5000	8000	23910	$cliC \leq 107 \wedge cliM \leq 20 \wedge state \leq 9$
10000	8000	29424	$cliC \leq 116 \wedge cliM \leq 23 \wedge state \leq 9$
5000	9000	29924	$cliC \leq 118 \wedge cliM \leq 23 \wedge state \leq 9$
10000	9000	29926	$cliC \leq 123 \wedge cliM \leq 22 \wedge state \leq 9$
5000	10000	27174	$cliC \leq 107 \wedge cliM \leq 23 \wedge state \leq 9$
10000	10000	27460	$cliC \leq 100 \wedge cliM \leq 25 \wedge state \leq 9$

the impact between the inferred invariant, the resulting submodel's size and the value of the MTTF estimation obtained from it. We are also interested in gaining insight on combinations of trace length and number that are likely to yield the best overall result.

For the Tandem Queue case study the estimated MTTF calculated in 24 hours over the full model was 4.20×10^5. The full model comprises $\sim 1.5 \times 10^7$ states. Regarding computations over submodels, we report on MTTF estimation (Figure 2), submodel sizes (Figure 3) and invariants obtained (Table I) for various settings of numbers and lengths of traces. Note that we report here on a subset of the values obtained, however non-reported data is in-line with the trends shown.

The first figure shows, for different automatically generated sized submodels, the estimated MTTF (over a logarithmic scale) along with how much time it took for the calculation to finish. Executions that finished before the 24 hour timeout are flattened on the MTTF axis at the time the result was reached. It is noteworthy that none of the automatically obtained submodels is larger than 35000 states, comprising roughly 0.25% of the states of the complete model. Despite having explored only such a small percentage of the full model, the obtained lower bound for MTTF is quite large in some cases, possibly sufficient to argue for high system reliability – MTTF is *at least* in the order of 1.0×10^7. Although very small submodels do not provide good bounds, larger submodel MTTF estimations increase dramatically, quickly rising to the 7×10^7 maximum MTTF witnessed, which is a full two orders of magnitude beyond the estimation for the full model.

An important question is if good submodels can be obtained in a consistent fashion regarding trace quantity and length parameters of the simulation phase. Figure 3 shows that such submodels are easily attainable for this example. Experiments with trace length below 3000 do not consistently produce rich enough models that yield good MTTF estimates. Unsurprisingly, small sample sets are also inconsistent in their results. However, once the sample set size parameter is set to at least 6000 samples, the submodels produced consistently yield large MTTF estimates. In summary, for this case study a minimum of 6000 samples of traces at least 4000 steps long are necessary for consistent results. Furthermore, increasing these parameters does not yield clear advantage in terms of the final MTTF estimation. Both figures also show that results become more stable as these parameters are increased.

State space size alone is not the only important factor when evaluating the effectiveness of the approach. For a given size, many submodel of that size exist, and not all of them may be effective. Preliminary work [7] has shown that submodels obtained through depth first search (DFS) explorations yield very poor results. Although breadth first search (BFS) obtains higher MTTF lower bounds than DFS when used as a submodel generator, it performs poorly against our approach, as the state space that it explores is not as relevant. For example, our approach using 10000 traces 10000 states long (one of the best performers) obtains a 27460 state sized submodel, which is characterised by the invariant $cliC \leq 100 \wedge cliM \leq 25 \wedge state \leq 9$. Consider a similarly sized BFS generated submodel of 28000 states. The Tandem Queue model allows four different actions (push, fwd, svc_1, svc_2). Conservatively assuming at most two actions enabled at each state, an equal sized BFS submodel would explore at most $\lceil \log_2(27460) \rceil = 15$ levels deep. Such a submodel would only allow for very limited behaviour. If each transition level generated a new state, queues of no more than 15 elements could be generated by such a submodel. Of course, it is not always the case that a new state is generated. In fact, a BFS exploration that allows for 50 elements per queue results in a 32000 state submodel. The MTTF obtained through such a submodel is ~ 70000, very far from the results we obtain.

Regarding potential overhead of trace generation and invariant inference, memory consumption is negligible with respect

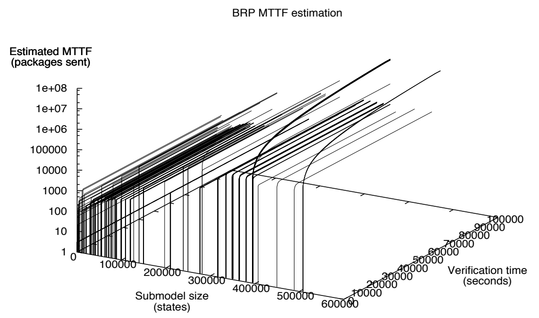

Fig. 4. Results of analysis of BRP for different sized submodels, Backwards Gauss-Seidel method.

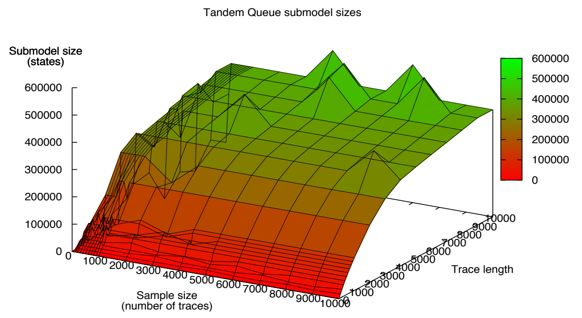

Fig. 5. BRP submodels sizes for different sample size and trace length parameters.

to representing the state space of the full model, as only one relatively short trace needs to be kept in memory at a time. Timewise, analysis of 10000 traces of length 10000 took less than an hour. Accounting for this hour in the verification time budget, the submodel that yielded the highest MTTF lower bound would have achieved a result of $\sim 6 \times 10^7$ in 23 hours.

Although not intended to be shown to developers, we report on some of the automatically inferred invariants in Table I. The discovered invariants deal with bounding the size of both queues, while the variable $state$ encodes whether the queues are full or not, and the phase the system is in at the time. It is noteworthy that although it is intuitive that an invariant should bound the queue sizes, it is unlikely that a human would come up with the particular bounding values used.

For the BRP case study similar results were obtained and are shown in Figures 4 and 5, and Table II.

In contrast to the prior case study, we were unable to obtain the MTTF for the full model due to state explosion that exhausted available memory. However, observations prior to running out of memory showed that the full model contains at least 30 million states, which means that the submodels analysed represent at most 2% of the size of the full model, still a very low percentage. Furthermore, the highest MTTF bounds were obtained for submodels starting from 400000 states (less than 1.33% of the full model) yielding an MTTF in the order of 2.5×10^7. This result is most significant, because of the impossibility of estimating MTTF for the full model.

Note that around the 400000 and 500000 states mark, there are both estimations that provide very good bounds and those that yield not so useful ones. Interestingly, those that do not perform well arise from submodels obtained through invariants inferred from sets of traces shorter than 7000 states long, while sets of longer traces perform very well. This shows that appropriate trace length is critical to the final MTTF estimation. As before, a similarly sized submodel obtained through BFS does not provide such higher MTTF lower bounds. One of our best performers, at 10000 traces 10000 states long, produces a submodel 392786 states in size which (with eight BRP actions and conservatively assuming three enabled at any time) results in a BFS submodel of depth $\lceil \log_3(392786) \rceil = 12$, which models very few frames being sent. In fact, a BFS-like submodel that allows only for 5 frames to be sent per file comprises ~ 400000 states and yields an MTTF of only 40.

Figure 5 depicts information related to the possibility of obtaining useful submodels. It can be seen that it is quite easy to obtain such submodels, without many restrictions on experiment configuration. In fact, the configurations for this case study behave much more steadily than with that of the Tandem Queue. Sets of 4000 traces of at least 7000 states seem to be enough for obtaining good estimates. Further increases of these parameters yield larger and slightly better-performing models, and this increase is much smoother (hence predictable) than is the case for the Tandem Queue submodels.

As in the other case study, trace generation and invariant inference incurs an overhead. In this case, since the model is more complex, this analysis can take up to 2 additional hours. Reducing the verification time by these 2 hours, the estimated MTTF would have been still large, about 2×10^7. Recall that this overhead was not included in measured time to allow graphs to show convergence speed of numerical analysis.

As before, we show for reference some of the inferred invariants. The variables $fileSize$, i and $nrtr$ describe the size of the file being sent, how many frames have been sent for that file, and the number of retries attempted, respectively. Other variables such as s_{ab}, r_{ab}, bs and fs encode the bit alternation in the protocol. The invariants obtained establish relationships between variables that seem unrelated, making them quite unintuitive even for a domain expert.

What both case studies and experiments indicate is that, through careful partial exploration of the model, we can obtain useful bounds for MTTF estimation with very low percentages ($< 1.5\%$) of actual state space exploration. Further, submodels that yield these results also converge very quickly (much before the 24 hour timeout) to good MTTF results. While the estimation does constantly improve during the rest of the 24 hours, it does so at a slower pace. This is good news, as even with the trace analysis overhead, good MTTF results can still be attained under the same time budget. From these results it follows that, for these case studies, effort into estimating MTTF through automatically obtained submodels through model invariants of the full model pays off.

It must be noted that it is possible that the *actual* MTTF is much larger than any of those obtained. Of course, we are always limited by the fact that the actual MTTF cannot be calculated, neither with partial nor full models. It can be

TABLE II
SELECTION OF BRP SUBMODEL SIZES AND INVARIANTS FOR DIFFERENT PARAMETER CONFIGURATIONS.

Traces	Length	States	Invariant
10000	1	35	$srep = nrtr \wedge srep = fileSize \wedge srep = r \wedge srep = rrep \wedge srep = k \wedge srep = l \wedge bs = s\_ab \wedge bs = fs \wedge bs = ls \wedge bs = fr \wedge bs = lr \wedge bs = br \wedge bs = r\_ab \wedge bs = recv \wedge s \leq 7 \wedge srep \leq 0 \wedge i \leq 1 \wedge s \geq srep \wedge s \geq i \wedge srep \leq i$
10000	501	66282	$s \leq 7 \wedge srep \leq 3 \wedge nrtr \leq 2 \wedge fileSize \leq 1500 \wedge i \leq 84 \wedge r \leq 4 \wedge rrep \leq 3 \wedge k \leq 2 \wedge l \leq 2 \wedge s \geq k \wedge s \geq l \wedge srep \leq fileSize \wedge srep \leq i \wedge srep \leq r \wedge srep \leq rrep \wedge nrtr \leq fileSize \wedge nrtr \leq i \wedge fileSize \geq r \wedge fileSize \geq rrep \wedge fileSize \geq k \wedge fileSize \geq l \wedge r \geq l$
10000	5000	333099	$s \leq 7 \wedge srep \leq 3 \wedge nrtr \leq 2 \wedge fileSize \leq 1500 \wedge i \leq 833 \wedge r \leq 4 \wedge rrep \leq 3 \wedge k \leq 2 \wedge l \leq 2 \wedge s \geq k \wedge s \geq l \wedge srep \leq fileSize \wedge srep \leq i \wedge srep \leq r \wedge srep \leq rrep \wedge nrtr \leq fileSize \wedge fileSize \geq r \wedge fileSize \geq rrep \wedge fileSize \geq k \wedge fileSize \geq l \wedge r \geq l$
10000	10000	392786	$s \leq 7 \wedge srep \leq 3 \wedge nrtr \leq 2 \wedge fileSize \leq 1500 \wedge i \leq 1500 \wedge r \leq 4 \wedge rrep \leq 3 \wedge k \leq 2 \wedge l \leq 2 \wedge s \geq k \wedge s \geq l \wedge srep \leq fileSize \wedge srep \leq i \wedge srep \leq r \wedge srep \leq rrep \wedge nrtr \leq fileSize \wedge nrtr \leq i \wedge fileSize \geq r \wedge fileSize \geq rrep \wedge fileSize \geq k \wedge fileSize \geq l \wedge r \geq l$

argued, though, that it is often the case that the exact MTTF value is not needed as such; rather, satisfying a minimum degree of reliability is a sufficient guarantee. Hence, methods which provide higher lower bounds faster are useful.

Question 2: Experimentation to answer this question is not straightforward due to the problem of generating sufficient failing simulations to ensure given precision and confidence parameters. We first aimed at performing a straightforward statistical analysis of the model. A first experiment was designed requiring a result precision of 99%. As is standard for statistical analyses, we also required a 95% confidence.

A straightforward calculation of the necessary sample size based on the Chernoff bound [14] determines that a total of ~ 60000 samples are necessary, which does not seem excessive. However recall that each sample must eventually fail. For systems with rare failures, this means that samples may be extremely long. Through trial and error, and based on the bounds obtained in *Q1*, we tried to determine the minimum length for samples to consistently reach failure states. For the Tandem Queue full model, even samples as long as 4×10^8 do not consistently fail. Considering that generating a sample of such length takes 15 minutes, generation of the full 60000 traces required leads to a 2 year period for sample generation. A similar situation is found upon analysis of the BRP model.

Relaxing the precision requirement to 95% reduces the sample generation cost to 1 month. Further relaxation to 90% still requires a week of execution. In fact, if we were to set a 24 hour budget for sample generation, the precision obtained would be 70%. That is, the MTTF estimate would be up to $\pm 30\%$ away from the true MTTF value with a 95% guarantee. Note that this is a very conservative estimate as it is unlikely that all traces of length 4×10^8 generated in the 24 hour period will consistency reach failure states, and possibly much lengthier traces will be needed.

To overcome this limitation of standard Monte Carlo verification, we tried a variation of Wald's sequential testing [15]. This procedure generates samples while at the same time it determines whether more samples are necessary or not. As a result of this online estimation, it might require less samples than those mandated by the Chernoff bound, although it cannot be stated beforehand how many samples will be needed exactly. This optimization does not eliminate the need for samples to reach property-determining states, so sample length remains a problem. We attempted to perform this analysis truncating generated samples at length 4×10^8 and treating them as *failing* samples once they reached this threshold. This is a similar strategy as the one used in our approach (anything beyond the submodel is a failure). However, this procedure yielded no results after 24 hours of execution, indicating that the sequential testing still needed more evidence.

Furthering this strategy of over-approximation of failures in Monte Carlo verification, we generated samples over the submodels with highest MTTF obtained in *Q1* rather than the full model. However, the problem of producing samples that consistently fail persisted, failing to provide an MTTF in the budgeted time. These results suggest that Monte Carlo approaches may be unsuitable to answer reliability questions in systems with high MTTF (i.e., rare failures).

Question 3: In this section, we compare the results obtained while answering *Q1* with the results a practitioner might obtain by specifying invariants herself, based on her knowledge of the model. Prior to experimenting on automatically generated invariants, we analysed the models and came up with at least one invariant for each one. These invariants were selected based on our understanding that their negation is a necessary condition for failure.

For the Tandem Queue case study, we established the invariant to be that the total number of enqueued processes globally in both queues is less than c, and ran experiments for different values of c ranging up to the total capacity of the queueing system ($2 \times C$). A failure entails that the invariant does not hold for $c < 2 \times C$, and that for $c = 2 \times C$ the resulting invariant-driven submodel is exactly the whole model. In our experiments we found that there where multiple c values for which the invariant resulted in a significantly higher MTTF than the MTTF estimated for the full model. In Table III results are presented for various invariant-driven submodel parameter values together with estimated MTTF and computation time using the BGS method.

From the table it follows that the best MTTF is obtained for the submodel which considers up to 120 processes queued (MTTF $> 5.5 * 10^7$).

In the case of the Bounded Retransmission Protocol case study, a parametric invariant chosen was that the number of retries performed while transmitting a single file was less than $max_{retries}$. We ran experiments for different values of $max_{retries}$ ranging up to the true maximum number of retries (256). A failure entails that the invariant does not hold for $max_{retries} < 256$. For $retries = max_{retries}$ the resulting invariant-driven submodel is the whole model.

Again, we show a selection of submodels, ranging from the very small upwards to almost the complete model. Results for these experiments are depicted in Table III.

Estimation results are even more significant than for the previous case study considering that analysis of the full model with 256 retries was not possible within the memory budget.

TABLE III
EXPERIMENTAL RESULTS FOR TANDEM QUEUE (2×1200 PROCESSES) AND BRP (256 RETRIES) MEAN TIMES TO FIRST FAILURE TIMES.

c	Size	BGS	
		MTTF	Time
20	2398 st 6560 tr	$0.83{\cdot}10^3$	68.75 s
40	8778 st 24280 tr	$1.12{\cdot}10^4$	82.72 s
60	19158 st 53200 tr	$1.25{\cdot}10^5$	276.69 s
80	33538 st 93320 tr	$1.36{\cdot}10^6$	64.06 m
100	51918 st 144640 tr	$1.49{\cdot}10^7$	17.93 h
120	74298 st 207160 tr	$5.50{\cdot}10^7$	TO
140	100678 st 280880 tr	$4.63{\cdot}10^7$	TO
160	131058 st 365800 tr	$3.17{\cdot}10^7$	TO
180	165438 st 461920 tr	$2.31{\cdot}10^7$	TO
200	203818 st 569240 tr	$1.66{\cdot}10^7$	TO
900	4067118 st 11381440 tr	$8.41{\cdot}10^5$	TO
1600	11219198 st 31407194 tr	$4.20{\cdot}10^5$	TO
2400	14362898 st 40213194 tr	$4.20{\cdot}10^5$	TO

retries	Size	BGS	
		MTTF	Time
1	366915 st 489574 tr	$1.50{\cdot}10^6$	21.06 h
2	480460 st 646758 tr	$1.69{\cdot}10^7$	TO
5	821095 st 1118310 tr	$1.08{\cdot}10^7$	TO
10	1388820 st 1904230 tr	$6.29{\cdot}10^6$	TO
50	5930620 st 8191590 tr	$1.39{\cdot}10^6$	TO
150	17285120 st 23909990 tr	$4.86{\cdot}10^5$	TO
250	28639620 st 39628390 tr	$2.73{\cdot}10^5$	TO
256	N/A st N/A tr	N/A	OOM

However, the trend indicates that augmenting the number of retries considered does not yield better MTTF and in fact, a very low number of retries gives a much higher MTTF.

It is interesting to note that for relatively small submodels (e.g. $c = 80$ on the Tandem Queue case study, and $max_{retries} < 2$ for BRP) the estimated MTTF is much higher than the MTTF computed over the complete model.

Still, while the manual invariant approach did provide useful bounds, it turns out that the best MTTF values generated by the automatic approach obtains slightly higher bounds for the same time budget. For the Tandem Queue study, the best automatically estimated MTTF is of $\sim 7 \times 10^7$ against $\sim 5.5 \times 10^7$. For the BRP case study the best automatic estimation is $\sim 2.5 \times 10^7$ versus $\sim 1.69 \times 10^7$ when manual intervention is applied.

V. DISCUSSION AND RELATED WORK

We have presented a fully automated technique for MTTF estimation of system models. Experimental results have shown that this approach may provide more useful MTTF estimations than both standard probabilistic model checking and Monte Carlo verification. However, two parameters exist that need to be set for the approach to work–the size of the simulation set and the length of the simulated traces. Good news is that our experimentation has shown that, at least for the examples studied, very good results can be obtained through relatively small set of short traces. Although the elaboration of guidelines on how to set the number and length of traces is beyond the scope of this paper, results show that there may be a broad combination of parameter values for which high MTTF results are obtained in reasonable time. Further, overshooting these parameters does not have an dramatic impact in the resulting submodel size. It is important to note that exploration of an appropriate parameter space can be done concurrently, taking as the final MTTF estimation the highest of the bounds obtained. Full model probabilistic checking cannot exploit concurrent computation in such a way. Monte Carlo verification can be applied concurrently, however we believe that the significant time cost for sample generation would not be outweighed by concurrent execution; further experimentation is needed to address this point.

As was previously mentioned, most probabilistic model checkers [16], [10], [17], [18] provide functionality that may either reduce the time required to obtain results, or reduce the memory footprint required for verification, such as symmetry reductions [19], *lumping* [20] and several numerical methods. All these optimizations are orthogonal to the model checking procedure itself. Our work relies on probabilistic model checking and the experiments were run on PRISM, which implements some of these optimizations.

Monte Carlo verification of models where the required size or verification time have determined exhaustive model checking to be intractable, statistical simulation has proven to be an effective technique. As was mentioned in section IV, an important issue with simulation approaches is that they tend to work well mostly in the case that the specified properties are bounded in time, i.e. when these properties can be written in the form $\psi \mathcal{U}^{\leq T} \rho$ for a fixed T. This is so because estimation of the random variable X_ϕ by means of a sample of traces σ_i requires that the question of whether $M, \sigma_i \models \phi$ or not be answered in a definite way for each trace σ_i in the sample set. If the formula ϕ is temporally bounded, then termination is guaranteed when evaluating its truth for the traces, but for temporally unbounded formulae such termination is threatened.

In such cases, generating traces with acceptable length bounds that answer the property definitively can be very unlikely. To address this problem *biased sampling* [21], [22], [23], [24] has been studied. However, bias to sampling must be done manually resulting in an impact on the analysis results that cannot be quantified in general. The result obtained by our approach is guaranteed to be a true bound to MTTF.

Recent work by Younes et. al. [25] proposes two novel Monte Carlo approaches that do not rely on biased sampling. However, one of them may require an inordinate number of samples to produce results; while the other relies on reachability analysis, which requires the full model to be constructed, relinquishing one of the key advantages of Monte Carlo model checking over probabilistic model checking

It should also be noted that Monte Carlo approaches are typically inadequate for models that exhibit non-determinism. A workaround is to transform non-determinism into a probabilistic choice, introducing bias. We believe that our proposed approach can be straightforwardly adapted to be applied to non-deterministic DTMCs. This remains future work.

The analysis of system behaviour that exhibits *rare* yet relevant (e.g. failures) events is the subject of focused study within the simulation community as well. A technique that is usually used in conjunction with stochastic processes that have rare events is that of *importance sampling* [26]. Roughly speaking, the idea of importance sampling is to replace the original process's distribution for another more likely to generate the (originally) rare event during the sample generation. The distribution replacement is chosen so that results from analyses for the new distribution can be translated back to results valid for the original distribution. Although a promising approach, finding suitable replacement distributions is a complex and ad-hoc task for which further research and expertise is necessary.

Another promising simulation technique that also focuses on rare-events is that of *sample splitting*, most notably the RESTART implementation [27] which, roughly, rather than starting each simulation from the initial state, it does so from a state s visited in a previous simulation and from which reaching a rare-event is more likely. The likelihood of reaching state s from the initial state is taken into account for producing the final analysis results. Key to the application of these techniques is making appropriate decisions on where to restart simulations; these decisions demand deep understanding of both the model and the underlying splitting technique.

Finally, common to both the Monte Carlo approach and the simulation techniques discussed is the fact that they are inherently statistical results. As such, there is always a nonzero probability that the results obtained are completely off the mark. Further reducing this error probability may require excessive amount of additional traces to be sampled in order to obtain the guarantee. Our technique, though conservative in the bounds it obtains, is definitive in its answers.

VI. Conclusions and Further Work

In this paper we have proposed an approach to reliability estimation of system models. The approach is a novel combination of simulation, invariant inference and probabilistic model checking. We report on experiments that suggest that reliability estimation using this technique can be more effective than (full model) probabilistic and statistical model checking for system models with rare failures.

We believe the notion of reliability analysis over partial yet systematic explorations offers an alternative to, and hence complements, exhaustive model exploration–as in probabilistic model checking–and partial random exploration–as in statistical model checking.

We believe the experimental results presented in this paper are promising. However further experimentation is due. In addition, this work opens up two fronts that need to be furthered to develop more effective reliability estimation approaches. One front is generalising the approach to support estimation of other properties related to stochastic behaviour. In addition, the generalisation of the kinds of models for which this technique can be applied is also of interest (e.g. Markov Decision Processes, that extend Discrete Time Markov Chains by introducing nondeterminism).

Additionally, another area that calls for future work is better understanding the relationship between the simulated set of traces (both its size as the trace length) and the submodels that result from them. This understanding should lead to heuristics for setting appropriate values to these parameters.

Acknowledgements

This work was partially supported by grants PICT 2272, PIP 0955, UBACYT W0813, ERC StG PBM-FIMBSE and MEALS 295261.

References

[1] M. Vardi, "Automatic verification of probabilistic concurrent finite state programs," in *SFCS 1985*. IEEE, Oct. 1985, pp. 327–338.

[2] A. Bianco and L. De Alfaro, "Model checking of probabilistic and nondeterministic systems," in *Foundations of Software Technology and Theoretical Computer Science*. Springer, 1995, pp. 499–513.

[3] M. R. Lyu, *Handbook of software reliability engineering*. Hightstown, NJ, USA: McGraw-Hill, Inc., 1996.

[4] R. Segala, "Modelling and verification of randomized distributed real time systems," Ph.D. dissertation, Massachusetts Institute of Technology, 1995.

[5] A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A. Sangiovanni-Vincentelli, "It Usually Works: The Temporal Logic of Stochastic Systems," *Lecture Notes in Computer Science*, pp. 155–155, 1995.

[6] L. Jamieson and B. Dean, "Weighted alliances in graphs," *Congressus Numerantium*, vol. 187, p. 76, 2007.

[7] E. Pavese, V. Braberman, and S. Uchitel, "My model checker died!: how well did it do?" in *QUOVADIS/ICSE'10*. ACM, 2010, pp. 33–40.

[8] C. Baier and J. Katoen, *Principles of model checking*. MIT press, 2008.

[9] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao, "The Daikon system for dynamic detection of likely invariants," *Sci. Comput. Program.*, vol. 69, no. 1-3, pp. 35–45, Dec. 2007.

[10] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A tool for automatic verification of probabilistic systems," in *TACAS'06 Proceedings*, vol. 3920. Springer, 2006, pp. 441–444.

[11] H. Hermanns, J. Meyer-Kayser, and M. Siegle, "Multi terminal binary decision diagrams to represent and analyse continuous time Markov chains," in *Proc. NSMC'99*. Prensas Universitarias de Zaragoza, 1999, pp. 188–207.

[12] P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen, "Reachability analysis of probabilistic systems by successive refinements," in *PAPM/PROBMIV*, ser. LNCS, vol. 2165. Springer, 2001, pp. 39–56.

[13] L. Helmink, M. Sellink, and F. Vaandrager, "Proof-checking a data link protocol," in *Proc. International Workshop on Types for Proofs and Programs (TYPES'93)*, ser. LNCS, vol. 806. Springer, 1994.

[14] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *Annals of Mathematical Statistics*, vol. 23, no. 4, pp. 493–507, 1952.

[15] V. Nimal, "Statistical approaches for probabilistic model checking," M.Sc. Dissertation, Oxford University Computing Laboratory, 2010.

[16] J. Katoen, M. Khattri, and I. Zapreevt, "A Markov reward model checker," in *QEST'05*. IEEE, 2005, pp. 243–244.

[17] K. Sen, M. Viswanathan, and G. Agha, "VESTA: A statistical model-checker and analyzer for probabilistic systems," in *QEST'05*. IEEE, 2005, pp. 251–252.

[18] H. Younes, "Ymer: A statistical model checker," in *Computer Aided Verification*. Springer, 2005, pp. 171–179.

[19] M. Kwiatkowska, G. Norman, and D. Parker, "Symmetry reduction for probabilistic model checking," in *Computer Aided Verification*. Springer, 2006, pp. 234–248.

[20] T. Dean and R. Givan, "Model minimization in Markov decision processes," in *Proceedings of the National Conference on Artificial Intelligence*, 1997, pp. 106–111.

[21] K. Sen, M. Viswanathan, and G. Agha, "On statistical model checking of stochastic systems," in *Proc. CAV'05*. Springer, 2005, pp. 266–280.

[22] D. Rabih and N. Pekergin, "Statistical model checking using perfect simulation," in *Proc. ATVA'09*. Springer-Verlag, 2009, pp. 120–134.

[23] R. Lassaigne and S. Peyronnet, "Probabilistic verification and approximation," *ENTCS*, vol. 143, pp. 101–114, 2006.

[24] S. Basu, A. Ghosh, and R. He, "Approximate model checking of PCTL involving unbounded path properties," *ICFEM'09*, pp. 326–346, 2009.

[25] H. Younes, E. Clarke, and P. Zuliani, "Statistical verification of probabilistic properties with unbounded until," *Formal Methods: Foundations and Applications*, pp. 144–160, 2011.

[26] R. Rubinstein and D. Kroese, *Simulation and the Monte Carlo method (Series in Probability and Statistics)*. Wiley, 2008, vol. 707.

[27] M. Villén-Altamirano and J. Villén-Altamirano, "RESTART: a straightforward method for fast simulation of rare events," in *Proc. WSC'94*, San Diego, USA, 1994, pp. 282–289.