Optimized Smart Sampling*

Maxime Parmentier[1], Axel Legay[1], Firmin Chenoy[1]

[1]ICTEAM, UCLouvain, Louvain-la-Neuve, 1341, Belgium.

Abstract

We revisit the principle of Smart Sampling which makes it possible to apply Statistical Model Checking on stochastic and non-deterministic systems. We point out difficulties in the design of the initial algorithm and we propose effective solutions to solve them. Our contributions are implemented in the Plasma tool.

Keywords: Statistical Model Checking, Sampling, Markov Decision Process, Non-Determinism, Implementation

1 Introduction

Computer systems occupy an increasingly predominant position in our daily lives. Such systems find themselves embedded in strategic applications ranging from the connected home to assisted driving. In view of this situation, any failure of the system can have serious consequences both from an economic and human point of view. It is therefore necessary to deploy techniques whose purpose is to ensure that the system satisfies a set of security/safety requirements.

A first way to validate the requirements of a computer system is to test the conformity of its outputs with respect to predefined inputs. These techniques are known as very effective and they made it possible to detect numerous safety and security bugs both at code and specification models [3, 15]. Unfortunately, this technology does not generally cover all the behaviors of the system or even quantify the degree of confidence that can be given to it [26]. Another problem is that testing relies on predefined requirements, which makes it hard to detect cyber security issues [21]. This means that they cannot be applied in critical certification processes where the calculation of the degree of certainty is necessary to obtain the certificate [13].

Another approach consists in modeling the system by means of a mathematical object such as (extensions of) transition systems [8]. Such representations make it

*M. Parmentier is funded by a FNRS PhD Grant and A. Legay by a FNRS PDR - T013721.

possible to model the failures of the system as well as quantitative information or even interactions with the environment [8]. In this paper, we consider Markov Decision Processes (MDP), an extension of Markov Chains (MC) with non determinism features. The model allows us to represent the failures by means of probabilities and the choices of environment thanks to nondeterministic actions. In this context, the validation process consists in finding the environment (aka the scheduler) which maximizes or minimizes the probability of satisfying a requirement [30, 20]. To do so, one can then explore all the behaviors of the system by means of formal verification techniques such as (quantitative) model checking [8]. These exhaustive approaches make it possible to detect all errors in the system and obtain full guarantee. However, they suffer from the problem of the so-called "state-space explosion", which makes them inapplicable in many strategic and complex cases.

Another approach is to use Statistical Model Checking (SMC) to solve this problem [31, 25, 22]. A SMC algorithm consists in monitoring a finite number of executions of the system and using theoretical results from statistics such as the Chernoff bound to obtain certainty on the probability of satisfying the property. SMC, which is a compromise between testing and formal verification, has been applied in a wide variety of fields ranging from the validation of complex railway and aviation systems to the analysis of medical and space components (see [9, 29, 4, 5] for illustrations). One of the main challenges of SMC is to minimize the number of simulations to be performed [17]. Another difficulty is to extend the approach to systems that are not purely probabilistic, such as Markov Decision processes or Timed Automata [14, 16].

A series of recent works [10, 16] has made it possible to extend the SMC algorithms to the case of MDPs. Most of these algorithms use deep learning techniques that are very effective but require knowledge of the system and often complex modeling of the schedulers to be analyzed. It is for this reason that we have proposed another approach called "Smart Sampling" [10]. The idea of Smart Sampling is to represent schedulers in a simple way, with seeds and hash functions. We give ourselves an initial budget of schedulers and we apply the SMC algorithm on the MC which results from each scheduler's choices. The fifty percents of the best schedulers are then kept and the operation is repeated until the scheduler for which SMC minimizes or maximizes the requirement to be validated is found. Smart Sampling has been implemented in Plasma [24]. The approach is known to be simpler and faster than its competitors. It has been applied to various extensions of MC and requirements, including cost estimation. Unfortunately, this pure simulation approach suffers from a major preciseness problem. Indeed, it generally does not find the best scheduler but rather a scheduler which calculates the average probability of satisfying the property. This makes the approach inapplicable in certain strategic areas where extremes must be known.

In this article we revisit the Smart Sampling algorithm and we propose practical improvements. In particular, we observe that different reduction factors improve accuracy without impacting performance. Moreover, we point out an operating error of the Chernoff bound in the initial algorithm and we show empirically that the latter has important consequences on the result of the algorithm. All of our contributions have been implemented in Plasma [24] and validated through examples that illustrate the problems and the advantages in a concrete way.

2 Background

Markov Decision Processes allow to model both controllable non-determinism and uncontrollable non-determinism, which is necessary to modelize situations where an agent-based system operates in a dynamical environment with uncertainty. A *Markov Decision Process* (MDP) is a tuple $(S, s_0, A, \{P_a\}_{a \in A})$ where S is a finite set of *states*, s_0 is the initial state, A is a finite set of (labelled) *actions* and $\{P_a\}_{a \in A})$ is a set of probabilities over pairs of states such as for all $P_a \in A$, there exists a unique state $s \in S$ for which $P_a(s, s')$ can be greater than zero. A MDP can be seen as a directed graph whose edges start from a single vertex but can point to multiple vertices, with a probability associated with each of those possible destinations. In any state (vertex) s, a state s' is *reachable* if there exists an action (edge) a from s to s' with $P_a(s, s') > 0$. A (finite) *path* is a (finite) sequence of states $(s^1, s^2, ...)$ such that s^{i+1} is reachable from s^i for all $i \in \mathbb{N}_0$. A (finite) *trace* is a (finite) path such that $s^1 = s_0$. We'll use the notations T^n and T^∞ for the set of all finite traces of length n and the set of infinite traces respectively. For each simulation of a MDP, a trace is produced. To resolve the controllable non-determinism of a MDP, i.e. the necessity to select the actions which will be chosen during one simulation, one can refine a MDP into a Markov chain with the choice of a scheduler. In practice, a scheduler can be interpreted as a possible interaction with the environment (a possible choice of a user for instance) or as a chosen strategy for the execution of the system to achieve a specific goal. A *scheduler*, also called *policy*, is a function which indicates at any point during the generation of a trace which is the action that needs to be considered to determine the next state of the trace. The most basic kind of schedulers are *memoryless schedulers* whose choices only depends on the current state, i.e. functions of the form $\pi : S \to A$. For reachability problems, memoryless schedulers suffice, but they form a strict subset of *history-dependent schedulers*. *Finite-memory schedulers* take as inputs finite paths of a predetermined maximum length n, i.e. are functions of the for $\pi : (s^i)_{1 \leq i \leq n} \to A$. *Infinite-memory schedulers* take as inputs finite traces with no predetermined maximum length, i.e. are functions of the for $\pi : T^\infty \to A$. Schedulers that do not truly resolve the controllable non-determinism are also possible and are called *probabilistic schedulers*, i.e. functions of the form $\pi : S \to \mathit{Dist}(A)$, where $\mathit{Dist}(A)$ is the set of probability distributions over A. A standard probabilistic scheduler is one that choose the uniform distribution over all the available actions in each state. In what follows, unless specified otherwise, scheduler is used as a shortcut for memoryless scheduler.

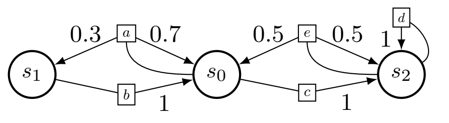
The choice of a scheduler turns a MDP into a Markov Chain. A *Markov chain* (MC) is a Markov decision process with exactly one available action for each state. Once a MDP is specified as a Markov chain, we can automatically define the associated probability space $(\Omega, \mathcal{F}, \mu)$. $\Omega = T^\infty$ is the set of all infinite traces of the Markov chain, $\mathcal{F}$ is the σ-algebra generated by subsets Ω of the form $\Omega_{(s^1,...,s^n)} = \{(t^1, ..., t^n, ...) \in \Omega \mid t_1 = s_1, ..., t_n = s_n\}$, and $\mu : \mathcal{F} \to [0, 1]$ is the distribution over $\mathcal{F}$ which verifies for any finite trace $(t_0, ..., t_n) \in T^n$: $\mu(\Omega_{(t_0,...,t_n)}) = \prod_{i=0}^{n-1} P_{a_{t_i}}(t_i, t_{i+1})$, with a_{t_i} being the unique action available in the Markov chain at state t_i. Such a distribution μ_π can be built for each scheduler π of a MDP, and for any (finite) trace T of the MDP the probability $\mu_\pi(T)$ is the probability that a simulation of the MDP under the policy π will produce the trace T.

In this work we use *Bounded Linear Temporal Logic* (BLTL) to specify the properties of a system. BLTL is a variant of Linear Temporal Logic (LTL), which is itself a modal extension of propositional logic. The addition of only one modal operator is sufficient to fully define BLTL. A minimal BNF grammar for the syntax of BLTL is:

$$\phi, \psi ::= \bot \mid \alpha \mid \neg\phi \mid \phi \wedge \psi \mid \phi\, U^k\, \psi$$

α denotes an atomic proposition. In the context of statistical model checking, those can be simply defined as a list of logical variables $\{\alpha_s\}_{s\in S}$ with α_s true if and only if the current state is s, but more complex atomic proposition can be defined, for example with fluents. $\neg$ and $\wedge$ are the basic operators of propositional logic for the negation and the conjunction. U^k (with $k \in \mathbb{N}$) is the (bounded) *until* operator. The semantics of BLTL operators are defined in a similar way to what is done for linear temporal logic. In the context of SMC, the models for the formulas of BLTL are usually taken as pairs (T, z) with T being a (infinite) trace of a Markov Chain and z being a starting index for the sequence T. The semantics of the $\neg$ and $\wedge$ operators are the same than with propositional logic, while the semantic of the U^k operator formalizes the intuitive idea of "true if the first formula is true until the second one becomes true, in a a time interval of length k". More formally, for a trace T: $(T, z) \models \phi\, U^k\, \psi$ iff there exists $j \in \mathbb{N}$ with $z \leq j \leq z + k$ such that $(T, i) \models \phi$ for all $z \leq i \leq j$ and $(T, i) \models \psi$ for all $j < i \leq k$. Just as the $\phi \vee \psi$ operator can be defined as $\neg(\neg\phi \wedge \neg\psi)$ within propositional logic, additional operators can be defined for (bounded) linear temporal logic. The most common ones are $X\,\phi$, $\phi\, F^k\, \psi$ and $\phi\, G^k\, \psi$, the *next*, the *finally/eventually* and the *globally/always* operators, respectively defined as $\top U^1 \phi$, $\top U^k \phi$, $\neg(\top U^k \neg\phi)$.

Given a BLTL formula ϕ associated with a MDP, a *score* can be given to any scheduler π by considering the quantity $\int_\Omega \mathbb{1}_{T\models\phi}\ d\mu_\pi$, i.e. the exact probability that the property of the system formalized by the formula ϕ is verified for a random possible execution of the MDP under the policy π.



A MDP with 3 states and 5 actions. The schedulers that choose the action c in state s_0 have a score of 0 for the formula $X(\neg\alpha_{s_0}) \Rightarrow (\alpha_{s_0} U^3 \alpha_{s_1})$, while any scheduler which picks action a has a score of $1 - (0.7)^3 = 0.657$. No memoryless scheduler can produce a trace that satisfies $G^8(F^4\alpha_{s_1} \wedge F^4\alpha_{s_2})$, but it is possible with a history-dependent scheduler.

Problem 1. *Given a MDP $(S, s_0, A, \{P_a\}_{a\in A})$, a BLTL formula ϕ and a specific class of schedulers Π, how to find an* optimal *schedulers π^+ and π^-, defined as:*

$$\pi^+ \in \operatorname*{arg\,max}_{\pi\in\Pi} \int_\Omega \mathbb{1}_{T\models\phi}\ d\mu_\pi \qquad\qquad \pi^- \in \operatorname*{arg\,min}_{\pi\in\Pi} \int_\Omega \mathbb{1}_{T\models\phi}\ d\mu_\pi$$

That is, a scheduler (not necessarily unique) with the greatest/lowest possible score, i.e. the greatest/lowest probability to generate traces for which the property ϕ is true.

3 The Lightweight Scheduler Approach

The most straightforward method to solve Problem 1 with SMC is to first build a large number of schedulers, compute the score of each of those schedulers, then output one with the highest score [10]. However, despite its simplicity, this approach calls for multiple remarks. First, the right representation of schedulers must be chosen. For very big models representing all schedulers as full maps does not scale well in terms of memory space. As showed in [10], one efficient alternatives is to represent schedulers as seeds for a pseudo-random number generator, whose size can be adapted in function of the expected theoretical number of schedulers.

Second, the size and complexity of the models of real-world systems may prohibit the computation of the exact theoretical scores of the schedulers [25], and those must then be estimated with SMC. Note that the non-exhaustive nature of the evaluation process for the schedulers implies that statistical guarantees are necessary for those estimations. Indeed, depending on how complex, sensitive or critical the system which needs to be verified is, those estimations must have relatively high or low precisions and confidence levels. To characterise those estimations, the notion of (ϵ, δ)-estimation is useful: $\hat{p}$ is a (ϵ, δ)-*estimation* of the true score p of a scheduler π if $P(|\hat{p} - p| > \epsilon p) < \delta$. All SMC algorithms require values for the hyperparameters ϵ and δ, explicitly or implicitly. A smaller ϵ means that the error of the estimations of the scores of the schedulers will be smaller, while a smaller δ means that the probability that one such estimation is not within the error bound is lower. In both cases, decreasing the values of those hyperparameters implies that more simulations must be performed.

Lastly, the performances of such an elementary SMC algorithm are extremely dependant of the number of schedulers which are generated and whose scores are estimated. If that number is low with respect to the total number of possible schedulers for the system under scrutiny, then the chances that an optimal scheduler can be found among the schedulers generated by the algorithm is low as well. This can be solved by allocating a smaller fraction of the simulation budget of the algorithm to each evaluation of the schedulers, i.e. by producing less traces for each scheduler. Unfortunately, that pseudo-solution is useless for real-world applications of SMC, since those usually demand estimations with very high precision and confidence level. Therefore, many SMC algorithms, instead of asking for a total simulation budget size, ask for values for the hyperparameters ϵ and δ and then try to minimize the simulation budget they need to guarantee those levels of precision and confidence. This can be done *a priori*, by exploiting statistical results such as the Chernoff bound [28, 10], or *a posteriori*, for instance with adaptive stopping algorithms[11, 12, 27].

3.1 The Original Smart Sampling Algorithm from [10]

The original Smart Sampling Algorithm from [10] given in Algorithm 1 is the version of the algorithm tailored to find optimal schedulers with maximal scores. The condition on the per-iteration budget is derived from the Chernoff Bound to ensure (ϵ, δ)-estimations [28, 10].

Algorithm 1 Original Smart Sampling Algorithm from [10]

```
Require:
    MDP $\mathcal{M}$ and BLTL formula $\phi$
    precision and confidence level $\epsilon$ and $\delta$
    per-iteration budget $B \geq \ln(2/\delta)/(2\epsilon^2)$
Ensure:
    $P$ and $\hat{p}_{\max}$, with $P$ final population,
    and $\hat{p}_{\max}$ an $(\epsilon, \delta)$-approximation of an optimal score

 1: $N \leftarrow \lceil\sqrt{B}\rceil$ ; $M \leftarrow \lceil\sqrt{B}\rceil$
 2: $P \leftarrow \{M$ schedulers sampled uniformly at random$\}$
 3: for $\pi \in P$ do
 4:     for $1 \leq i \leq N$ do
 5:         $T_i^\pi \leftarrow \text{Simulate}(\mathcal{M}, \phi, \pi)$
 6:     end for
 7: end for
 8: $R \leftarrow \{(\pi, \hat{n}_\pi) \mid \pi \in P,\ \hat{n}_\pi = |\{T_i^\pi \mid T_i^\pi \models \phi\}|\}$
 9: $\hat{p}_{\max} \leftarrow \max_{(\pi,\hat{n}_\pi)\in R} \hat{n}_\pi/N$

10: $N \leftarrow \lceil 1/\hat{p}_{\max}\rceil$ ; $M \leftarrow \lceil B\hat{p}_{\max}\rceil$
11: $P \leftarrow \{M$ schedulers sampled uniformly at random$\}$
12: for $\pi \in P$ do
13:     for $1 \leq i \leq N$ do
14:         $T_i^\pi \leftarrow \text{Simulate}(\mathcal{M}, \phi, \pi)$
15:     end for
16: end for
17: $R \leftarrow \{(\pi, \hat{n}_\pi) \mid \pi \in P,\ \hat{n}_\pi = |\{T_i^\pi \mid T_i^\pi \models \phi\}|\}$
18: $P \leftarrow \{\pi \in P \mid \hat{n}_\pi > 0\}$

19: $R \leftarrow \{(\pi, 0) \mid \pi \in P\}$ ; $i \leftarrow 0$ ; confidence $\leftarrow 1$
20: while confidence $> \delta \wedge |P| > 1$ do
21:     $i \leftarrow i + 1$
22:     $M_i \leftarrow |P|$
23:     $N_i \leftarrow 0$
24:     while confidence $> \delta \wedge N_i < \lceil B/M_i\rceil$ do
25:         $N_i \leftarrow N_i + 1$
26:         confidence $\leftarrow 1 - (1 - e^{-2\epsilon^2 N_i})^{M_i}$
27:         for $\pi \in P$ do
28:             $T_{N_i}^\pi \leftarrow \text{Simulate}(\mathcal{M}, \phi, \pi)$
29:         end for
30:     end while
31:     $R \leftarrow \{(\pi, \hat{n}_\pi) \mid \pi \in P,\ \hat{n}_\pi = |\{T_i^\pi \mid T_i^\pi \models \phi\}|\}$
32:     $\hat{p}_{\max} \leftarrow \max_{(\pi,\hat{n}_\pi)\in R} \hat{n}_\pi/N$
33:     $P \leftarrow \{\pi \in P \mid \hat{n}_\pi$ is one of the greatest $\lfloor |P|/2 \rfloor$ values in $R\}$
34: end while
35: return $P, \hat{p}_{\max}$
```

In Step 1 (lines 1 to 9), the per-iteration budget B is used once to generate $\lceil\sqrt{B}\rceil$ schedulers (line 2) and produce $\lceil\sqrt{B}\rceil$ traces for each of those schedulers (line 5). The traces which satisfy the property ϕ are counted to compute a rough estimation of the schedulers' scores (line 8), and the greatest value among those estimations is selected as a first estimation $\hat{p}_{\max}$ for the score of a (near-)optimal scheduler. In Step 2 (lines 10 to 18), the naive estimation $\hat{p}_{\max}$ is used to balance the simulation budget so to maximize the probability of producing traces with near-optimal schedulers (line 10) [10]. Then, the initial population of the algorithm is generated (line 11). A preliminary iteration is then performed to abandon the schedulers (line 14) that don't produce any trace which satisfies the property ϕ at least once (line 18). Finally, in Step 3 (lines 19 to 34), the main loop of the algorithm happens here, which doesn't stop until the population has been reduced to one scheduler or until the confidence level that has been reached, as approximated with the Chernoff Bound (line 26), is smaller than δ (line 20). First, the scores of the schedulers are reset (line 19). At each step of the loop, the iteration population is updated (line 22) and as long as the the number of simulations which have been realized at this step is smaller than $\lceil B/M_i\rceil$ (line 24), an additional trace is generated for each scheduler (line 28). At the end of each loop, the results are updated (line 31), the approximation of the optimal probability is updated and the population is cut in a way that only half of the population of schedulers, those with the best scores, are kept.

3.2 Beyond the First Implementation of the Smart Sampling Algorithm

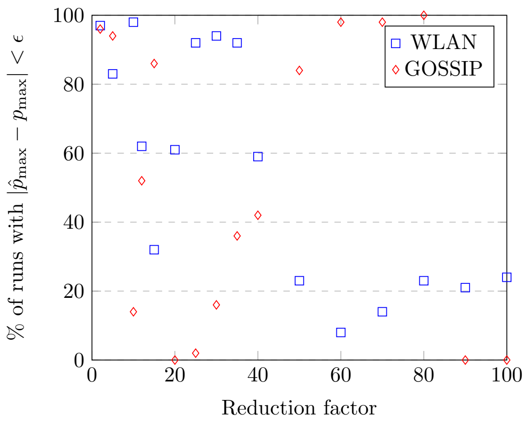
We reproduced two of the experiments of the original paper of the Smart Sampling algorithm [10], the analyses of the IEEE 802.11 Wireless LAN Protocol (WLAN) and the Gossip Protocol (GOSSIP)[1].

We were interested in observing how a greater or smaller *reduction factor*, defined as the denominator of the fraction of the population which is preserved from step to step, could impact the performance of the algorithm. On one hand, increasing the reduction factor should *a priori* lead to poorer performances but a faster execution time for the algorithm: a higher number of schedulers discarded at each step means that near optimal schedulers which can be present in the initial population might be abandoned by mistake (especially at the first step), but it also means that there are fewer steps to the algorithm and that the total simulation budget will be a smaller multiple of the per-iteration budget. On the other hand, decreasing the reduction factor should *a priori* lead to better performances but a slower execution time for the algorithm: a smaller number of schedulers discarded at each step means that near-optimal schedulers which can be present in the initial population have a smaller chance to be abandoned by mistake, but it also means that there are more steps to the algorithm and that the total simulation budget will be a greater multiple of the per-iteration budget.

[1]See https://www.prismmodelchecker.org/download.php for a description

Since the WLAN and GOSSIP models used have already been evaluated thoroughly with exhaustive model checking techniques [20, 19, 18], the theoretical score of an optimal scheduler is known for both of those experiments. Figure 1 shows the proportion of runs for the Smart Sampling algorithm that managed to output an estimation which was within the error bound of the actual optimal score in function of a varying reduction factor, with $\epsilon = 0.01$, $\delta = 0.1$ and a per-iteration budget of 15000.

Fig. 1: Original Smart Sampling algorithm performance



Surprisingly, the expected correlations can not be found. Moreover, a performance drop phenomenon can be seen around specific values. The location of those values depends of the values of the parameters of the Smart Sampling algorithm. After analysis of the implementation and individual executions of the algorithm, our conclusion is that the original implementation of the Smart Sampling algorithm is flawed in three ways. First, because it is designed to immediately stop as soon as the per-iteration budget divided by the number of remaining schedulers is greater than the Chernoff Bound, the algorithm can terminate much earlier than anticipated with a still diverse population of schedulers. Second, because barely enough traces are at that point produced so that the estimations of the schedulers of the final population are valid (ϵ, δ)-estimations, most of them are badly estimated relatively to each others. This results in individual outputs for the original Smart Sampling algorithm with great variance. Those two first flaws make for drastic changes in performance when the ratios between the size of the initial sample and powers of the reduction factor are close to integers, up to the last step of the algorithm. A slight change in one of those ratios can severely impact how soon the algorithm stops and how well the schedulers are evaluated at the last step. We hypothesize that this potential problem was not noticed with the first version of the Smart Sampling because the fixed value of 2 which was originally chosen for the reduction factor was low, making the algorithm's progress smooth in the sense that the population size decreases from step to step relatively slowly. Third, additional tailor-made experiments showed that the original implementation of the Smart Sampling algorithm suffers from another critical weakness: noise. For models with in-built noise, i.e. for which all schedulers have a fixed low chance to produce failing traces, performance is often extremely poor with non-generous per-iteration budget. Indeed, when the ratio between the initial population size and the per-iteration is close to 1, the first evaluation of the schedulers is so vulnerable to the noise that the first wave of discarded schedulers can be essentially regarded as random. Even if (near-)optimal schedulers are present in the sample at the start, the probability they are abandoned before the second step of the algorithm is high.

4 New Lightweight Algorithms for Statistical Model Checking

To improve the Smart Sampling algorithm, we modified how the Chernoff bound test dictates the termination of the algorithm, and we explored three ideas: artificially inflating the simulation budget for the first step, applying a simulated annealing strategy, and reintroducing new random schedulers at specific steps. Those modifications were implemented in *PLASMA* (Platform for Learning and Advanced Statistical Model checking Algorithms), a plug-in based interface for statistical model checking [24].

Unless specified otherwise, all experiments were realized for the WLAN and the GOSSIP models, with $\epsilon = 0.01$, $\delta = 0.1$ and a per-iteration budget of 15000. For both models, since the theoretical optimal scheduler and its score are known, the goal was to compute the probability that the algorithm produces an estimation which deviates from the theoretical optimal solution by at most ϵ and check if the probability is indeed lower or equal to δ. That is, to verify if the algorithm actually produces (ϵ, δ)-estimations. The results, which include information about the execution times as well, are averaged over up to 100 runs (depending on the experiment). The experiments were performed on a 8 GB machine with 4 cores running at 3.2 GHz.

4.1 Modification of the Chernoff Bound Test

Three approaches were considered to deal with the problems arising from the potential early termination of the Smart Sampling algorithm due to the Chernoff bound test: keep the early termination but with a better evaluation of the schedulers of the final population (option 1), ignore the early termination but scale down the simulation budget with respect to the accumulated confidence level for the remaining iterations, (option 2), or completely ignore the early termination (option 3).

Option 1 leaves the main loop of Algorithm 1 intact, but adds a last loop of x simulations for each scheduler once the Chernoff bound test makes the main loop end. Then, the modified algorithm reevaluate the schedulers and order them by their scores one last time before returning the result. Option 2 requires to remove the Chernoff bound test at line 20, and to alter the condition for the simulation loop from "confidence $> \delta \wedge N_i < \lceil B/M_i \rceil$" to "$N_i < F(\text{confidence}, \lceil B/M_i \rceil)$", with F a predetermined fixed function. Option 3 completely removes the Chernoff bound test both at line 20 and at line 24.

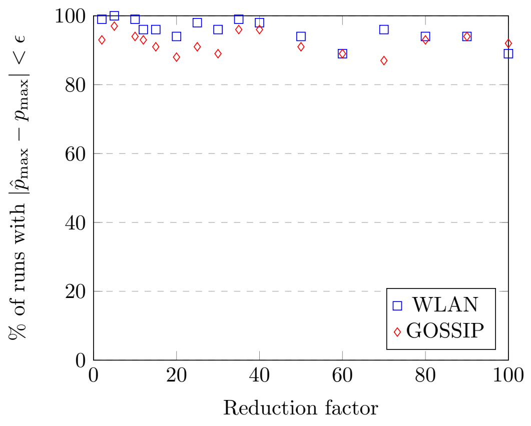
All three options were initially compared to each other fairly in the sense that the design choices (number x and function F) were made so that the number of additional simulations was identical between the options. For option 2, any test function F which scales linearly with confidence quickly leads to insignificant per-scheduler simulation budgets for the subsequent steps of the algorithm. Therefore, the type of function which was the most experimented with is a function which returns $\lceil B/M_i \rceil$ as long as confidence is greater than δ, then returns (a fraction or a multiple of) the per-scheduler simulation budget of the last iteration of the main loop for which confidence was still greater than δ.

Option 1 is not only the option that doesn't fit the original philosophy of the algorithm, but it is also the option which produced the worse results. Because option

1 allocates all the additional budget evenly between the remaining schedulers of the final population, the sampling process ceases to prioritize the most promising schedulers. Option 3 produced the best results consistently, whereas option 2 produced good results as well when the multiple of the test function F was large enough. However, when the multiple of the test function was taken as 1, option 3 was always a better choice in terms of results while the difference in total simulation budgets between option 2 and option 3 was systematically small. Indeed, unless the per-iteration budget parameter is unnecessary large with respect to ϵ and δ, the Smart Sampling algorithm generally completes most of its potential steps before the Chernoff bound condition is verified. Therefore, option 3 is the option we suggest and have chosen for the modification of the Smart Sampling algorithm. A small verification can be added at the level of the preconditions to warn the user or ensure that the per-iteration budget cannot be excessively large with respect to ϵ and δ when the user specifies a per-iteration budget instead of letting the algorithm fix it by itself.

Figure 2 is the equivalent of Figure 1 for the Smart Sampling algorithm modified with option 3. Not only the modification improves the performances of the algorithm noticeably, but it eliminates the performance drop phenomenon as well. The modification is particularly impactful with high values for the reduction factor, to the point that the algorithm's performance barely diminishes as the the reduction factor increases.

Fig. 2: Modified Smart Sampling algorithm performance



Figures 3 and 4 show the execution times for 100 runs of the Smart Sampling algorithm for the WLAN and GOSSIP experiments, for both versions of the algorithm.

Fig. 3: Execution times (WLAN)

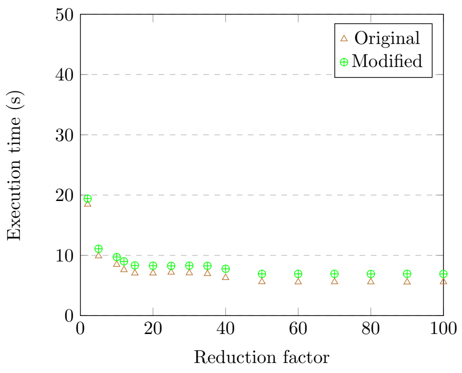
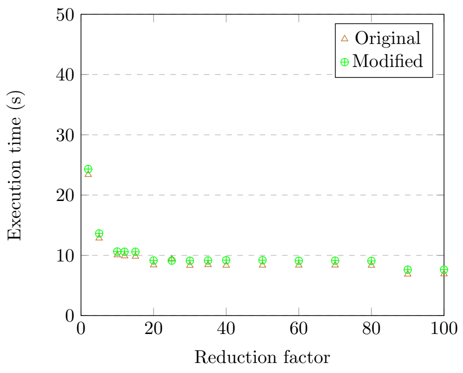


Fig. 4: Execution times (GOSSIP)



For any given reduction factor, the execution times of the modified version of the algorithm are on average 13% greater than those for the original version of the algorithm. However, since it allows for a much larger reduction factor without a loss in performance, the modified version applied with a large reduction factor outperforms the original version (with a reduction factor of 2) both in terms of results and speed.

In conclusion, we hypothesize that the ideal new version of the Smart Sampling algorithm is the third suggested modification, applied with a reduction factor larger than 2. Since the trade-off between performance and speed is inevitable as the reduction factor increases, we cannot give a perfect universal value for the reduction factor. Nevertheless, our recommendation is to take it at least as large as 5 or even 10, for we could never observe noticeable loss in performance at those values in any of our experiments (including those not discussed in this paper), while already providing a significant speed-up to the algorithm. With a reduction factor of 5, the algorithm is accelerated by a factor of $\ln(5)/\ln(2) \simeq 2.32$. With a reduction factor of 10, the algorithm is accelerated by a factor of $\ln(10)/\ln(2) \simeq 3.32$.

4.2 Other optimizations

To further improve the Smart Sampling algorithm, we first looked at one of its other weaknesses discussed in section 3.2: its fragility with respect to noise. During the first step of the algorithm, it is frequent that the algorithm drops near-optimal schedulers because their evaluation is based on so few traces that those evaluations have a high probability to be unreliable. When realistic simulation budgets are taken into account, the number of times each scheduler is simulated during the first iteration can even be close (if not equal) to 1.

Algorithm 2 Additional Budget at First Step

```
20: while $|P| > 1$
21:     $i \leftarrow i + 1$
22:     $M_i \leftarrow |P|$
23:     $N_i \leftarrow 0$
24:     if $i == 1$
25:     while $N_i < F(\lceil B/M_i \rceil)$                    ▷ $F$ is a fixed function
26:         $N_i \leftarrow N_i + 1$
27:         for $\pi \in P$ do
28:             $T^{\pi}_{N_i} \leftarrow$ Simulate($\mathcal{M}, \phi, \pi$)
29:         end for
30:     end while
31:     $R \leftarrow \{(\pi, \hat{n}_\pi) \mid \pi \in P,\ \hat{n}_\pi = |\{T^{\pi}_i \mid T^{\pi}_i \models \phi\}|\}$
32:     $\hat{p}_{\max} \leftarrow \max_{(\pi,\hat{n}_\pi)\in R} \hat{n}_\pi / N$
33:     $P \leftarrow \{\pi \in P \mid \hat{n}_\pi$ is one of the greatest $\lfloor |P|/(\text{reduction factor}) \rfloor$ values in $R\}$
34: else
35:     while $N_i < \lceil B/M_i \rceil$)               ▷ New condition for the main loop
36:         $N_i \leftarrow N_i + 1$
37:         for $\pi \in P$ do                  ▷ New condition for the simulation loop
```

To counter that problem, a very simple solution was first tested: providing the algorithm with additional simulation budget for the first step. This means slightly modifying the condition of the simulation loop at line 24 of Algorithm 1 to introduce an exception for the first iteration. Algorithm 2 shows the main modifications which were brought to the code of Algorithm 1 to remove the Chernoff bound test and introduce additional sampling at the first iteration.

The kind of test function F which was mostly tested is a collection of simple multiplicative functions with a multiplicative parameter a ranging from 2 to 10. Depending on whether the chosen per-iteration budget allows for a large or small per-scheduler simulation budget at the first step, a lower or greater value should be taken for that multiplicative parameter. Indeed, if the per-scheduler simulation budget at the first step is satisfactory to begin with, picking a value as low as 2 is the right choice as to minimize the additional computing cost. However, if the per-scheduler simulation budget at the first step is close to 1, then taking it as large as 10 (or even larger) is necessary. In general, the function F can be of the form $F(x) = \min(K, ax)$, with a rather large and K being a upper bound on the per-scheduler budget for the first step, for a jack-of-all-trades solution.

On custom small scale models designed with a lot of in-built noise, providing additional simulation budget for the first iteration improved the Smart Sampling algorithm's probability of identifying a (near-)optimal scheduler by around 6% with $a = 2$, 15% with $a = 3$, 26% with $a = 5$ and 30% with $a = 10$. Those numbers are average over reduction factors ranging from 2 to 100, albeit the gains in performance were generally more important with large (> 10) reduction factors.

Fig. 5: More budget at first iteration strategy (WLAN)

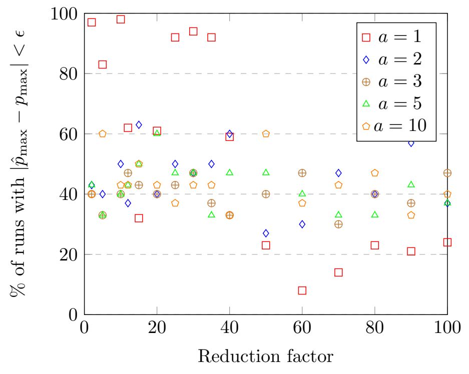
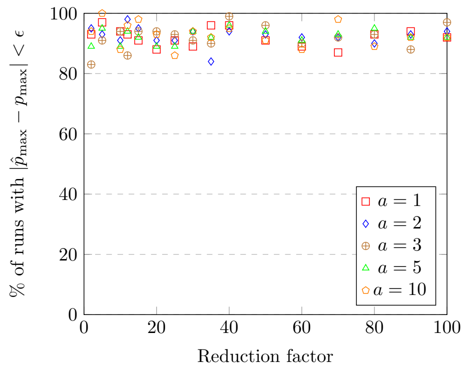


Fig. 6: More budget at first iteration strategy (GOSSIP)



Figures 5 and 6 show the impact of providing additional simulation budget for the first iteration on the probability for the Smart Sampling algorithm to output a near-optimal scheduler for the WLAN and GOSSIP models. Despite the limited importance of the noise weakness of the Smart Sampling algorithm for those models, we can observe that even though the effect of providing additional simulation budget for the first iteration is less noticeable than when noise is involved, in the case of the

GOSSIP experiment, that strategy has positive impact on the algorithm's performance nonetheless. However, as can been seen with Figure 6, which was produced with a very reduced set of experiments of 30, that strategy still suffers from the from the instability of the statistical model checking process.

Fig. 7: Execution times (WLAN)

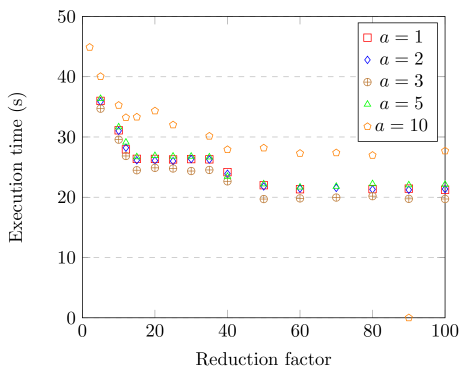
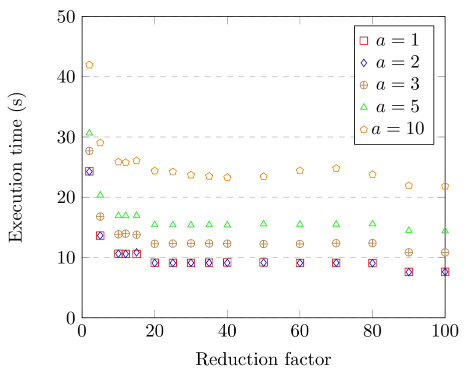


Fig. 8: Execution times (GOSSIP)



Of course, that small positive benefit is not worth if it comes with a prohibitive increase in computational costs. However, as shown with figures 7 and 8, the increase in computational costs is limited, at least for small values of the multiplicative factor a. Which is not a surprise: especially with low reduction factors, the number of iterations of the algorithm is generally large enough that adding the equivalent of a few artificial iterations at the start does not grow the total simulation budget by a significant margin. Accross all reduction factors, the increase is negligible with $a = 2$ and $a = 3$, around 25% with $a = 5$ on average and between 30% and 150% with $a = 10$. We find difficult to advocate for such an increase in computational costs in any scenario, even when the model is not expected to be "noisy". However, as the potential benefit in some situations is very valuable and the trade-off in terms of speed barely noticeable for small values of the multiplicative parameter a, we consider adopting the additional budget for first iteration strategy as the new baseline for the Smart Sampling algorithm the right choice. Fixing $a = 3$ appears to be a good starting point, with a suggestion to the user to push the value of that parameter up to 10 when working with noisy models.

Trying to extend the additional budget at first step strategy to a simulated annealing strategy was attempted. Instead of only inflating the simulation budget of the first iteration, one can try to artificially rebalance the total simulation budget between the different steps of the algorithm instead of allocating a constant per-iteration budget to all iterations. Skewing towards early iterations and towards the late iterations were both tested. The conclusion is that skewing towards the early iterations is often the best course of action, to the point that the best results were obtained when the skew was so much in favor of the first iteration that it was practically equivalent to the much simpler strategy which consists in providing additional budget at the first step.

Fig. 9: Random scheduler reintroduction strategy (WLAN)

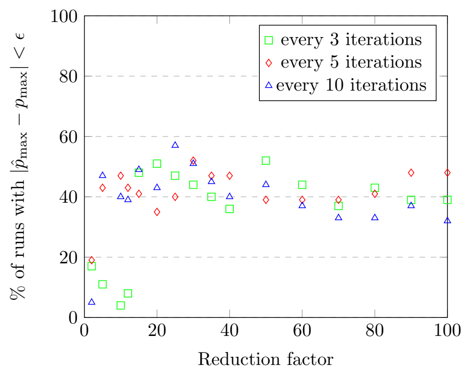
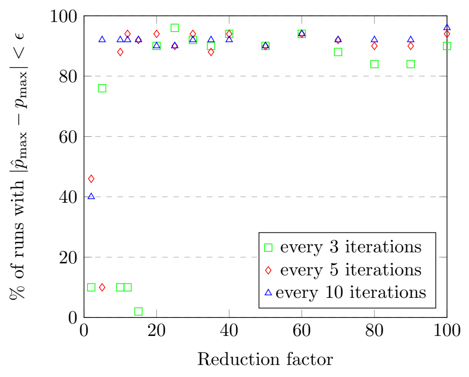


Fig. 10: Random scheduler reintroduction strategy (GOSSIP)



Lastly, the idea of reintroducing schedulers as the algorithm progresses was also experimented with. The goal was to improve the Smart Sampling algorithm with respect to another of its fundamental limitations, i.e. its inability to identify schedulers beyond those present in the initial population, but also reduces its weakness relatively to the local extremum phenomenon. Unfortunately, as shown in figures 9 and 10, the results are not conclusive. In all cases, that strategy didn't produce better results than simply starting with a larger initial population. In some cases, that strategy even made the algorithm's performance worse, by introducing badly evaluated schedulers with low scores right before the termination of the algorithm. We hypothesize that this kind of strategy is still promising nevertheless, but at the condition that the new schedulers are instead synthetized in a way that exploits the accumulated information about the discarded schedulers in order to reintroduce schedulers which are different and potentially have good scores, and not just random ones.

5 Conclusion

In this article we propose a new version of the Smart Sampling algorithm presented in [10]. Our first experimental results with Plasma show that Smart Sampling can be used to obtain schedulers that minimize/maximize a BLTL property. Aside from analyzing more examples, there are several ways to extend this work. The first is to adapt the algorithm to other systems and other requirements. One thinks, for example, of timed stochastic systems [6] or of properties that involve cost calculations [23]. A limitation of the current algorithm is that it does not exploit the knowledge between several schedulers. This means that in the case where the scheduler is rare, the efficiency of the algorithm will remain limited. One way to solve this problem would be to adapt genetic algorithms. This requires modeling seed populations and extracting information from them. Another work will consist in considering the hyperproperties which make it possible to compare sets of executions and thus to model a broader spectrum of security properties [1, 2]. Finally, using our approach to synthesize schedulers for stochastic interface theories [7] should be investigated.

References

[1] E. Ábrahám et al. “Probabilistic Hyperproperties with Nondeterminism”. In: *ATVA*. Vol. 12302. LNCS. Springer, 2020, pp. 518–534.

[2] S. Arora et al. “Statistical Model Checking for Probabilistic Hyperproperties of Real-Valued Signals”. In: *SPIN*. Vol. 13255. LNCS. Springer, 2022, pp. 61–78.

[3] V. Atlidakis, P. Godefroid, and M. Polishchuk. “RESTler: stateful REST API fuzzing”. In: *ICSE*. IEEE / ACM, 2019, pp. 748–758.

[4] E. Baranov et al. “A Secure User-Centred Healthcare System: Design and Verification”. In: *DATAMOD*. Vol. 13268. LNCS. Springer, 2021, pp. 44–60.

[5] D. Basile et al. “Exploring the ERTMS/ETCS full moving block specification: an experience with formal methods”. In: *STTT* 24.3 (2022), pp. 351–370.

[6] C. E. Budde et al. “An efficient statistical model checker for nondeterminism and rare events”. In: *Int. J. Softw. Tools Technol. Transf.* 22.6 (2020), pp. 759–780.

[7] Benoıt Caillaud et al. “Constraint Markov Chains”. In: *Theor. Comput. Sci.* 412.34 (2011), pp. 4373–4404.

[8] E. M. Clarke et al. *Model checking, 2nd Edition*. MIT Press, 2018.

[9] A. Colombo et al. “Efficient customisable dynamic motion planning for assistive robots in complex human environments”. In: *JAISE* 7.5 (2015), pp. 617–634.

[10] P. R. D’Argenio et al. “Smart sampling for lightweight verification of Markov decision processes”. In: *STTT* 17.4 (2015), pp. 469–484.

[11] P. Dagum et al. “An Optimal Algorithm for Monte Carlo Estimation”. In: *SIAM Journal on computing* 29 (2000), pp. 1484–1496.

[12] C. Domingo, R. Gavalda, and O. Watanabe. “Adaptive Sampling Methods for Scaling Up Knowledge Discovery Algorithms”. In: *Data Mining and Knowledge Discovery* 6 (2002).

[13] S. Dupont et al. “Product Incremental Security Risk Assessment Using DevSecOps Practices”. In: *SECPRE*. Vol. 13785. LNCS. 2022.

[14] O. Gadyatskaya et al. “Modelling Attack-defense Trees Using Timed Automata”. In: *FORMATS*. Vol. 9884. LNCS. Springer, 2016, pp. 35–50.

[15] P. Godefroid. “Fuzzing: hack, art, and science”. In: *Commun. ACM* 63.2 (2020), pp. 70–76.

[16] M. Jaeger et al. “Teaching Stratego to Play Ball: Optimal Synthesis for Continuous Space MDPs”. In: *ATVA*. Vol. 11781. LNCS. Springer, 2019.

[17] C. Jégourel, A. Legay, and S. Sedwards. “Importance Splitting for Statistical Model Checking Rare Properties”. In: *CAV*. Vol. 8044. LNCS. 2013.

[18] M. Kwiatkowska, G. Norman, and D. Parker. “Analysis of a Gossip Protocol in PRISM”. In: *ACM SIGMETRICS Performance Evaluation* 36 (), pp. 17–22.

[19] M. Kwiatkowska, G. Norman, and J. Sproston. “Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol”. In: *Proc. PAPM/PROBMIV’02*. Vol. 2399. LNCS. 2002.

[20] M. Z. Kwiatkowska, G. Norman, and D. Parker. “PRISM 4.0: Verification of Probabilistic Real-Time Systems”. In: *CAV*. LNCS. 2011.

[21] J. Lanet et al. “When time meets test”. In: *Int. J. Inf. Sec.* 17.4 (2018), pp. 395–409.

[22] K. G. Larsen and A. Legay. “30 Years of Statistical Model Checking”. In: *ISOLA*. Vol. 12476. LNCS. Springer, 2020, pp. 325–330.

[23] A. Legay, S. Sedwards, and L. Traonouez. “Estimating Rewards & Rare Events in Nondeterministic Systems”. In: *ECEASST* 72 (2015).

[24] A. Legay, S. Sedwards, and L. Traonouez. “Plasma Lab: A Modular Statistical Model Checking Platform”. In: *ISOLA*. Vol. 9952. LNCS. 2016, pp. 77–93.

[25] A. Legay et al. “Statistical Model Checking”. In: *Computing and Software Science - State of the Art and Perspectives*. Vol. 10000. LNCS. 2019.

[26] Y. Lin et al. “Test coverage optimization for large code problems”. In: *J. Syst. Softw.* 85.1 (2012), pp. 16–27.

[27] V. Mnih, C. Szepesvari, and J.Y. Audibert. “Empirical Bernstein Stopping”. In: *In Proc. of int. conference on Machine learning* (2008), pp. 672–679.

[28] M. Okamoto. “Some Inequalities Relating to the Partial Sum of Binomial Probabilities”. In: *Annals of the institute of Statistical Mathematics* 10 (1959).

[29] A. Paigwar et al. “Probabilistic Collision Risk Estimation for Autonomous Driving: Validation via Statistical Model Checking”. In: *IV*. IEEE, 2020.

[30] M. Y. Vardi. “Automatic Verification of Probabilistic Concurrent Finite-State Programs”. In: *FOCS*. IEEE Computer Society, 1985, pp. 327–338.

[31] H. S. Younes and R. G. Simmons. “Statistical probabilistic model checking with a focus on time-bounded properties”. In: *Inf. Comput.* 204 (2006).