Statistical Model Checking: Black or White? *

Pranav Ashok, Przemysław Daca, Jan Křetínský, and Maximilian Weininger

Technical University of Munich, Germany

Abstract. One of the advantages of statistical model checking (SMC) is its applicability to black-box systems. In this paper, we discuss the advantages gained when SMC is applied to white-box systems, utilizing the knowledge of their internals. We focus on the setting of unbounded-horizon properties such as reachability or LTL. We compare our approach to other statistical and numerical techniques both conceptually as instantiations of the same framework, and experimentally. It not only clearly preserves scalability advantages of black-box SMC compared to classical model checking (while providing high level of guarantees), but it also scales yet better than either of the two for a wide class of models.

1 Introduction

Classical *probabilistic verification* techniques rely on iterative approximation algorithms for linear equation systems and linear programs, such as *value iteration (VI)*, e.g. [Put14]. However, the scalability of such numeric analyses is severely limited, compared to standard non-quantitative (hardware or software) verification, since exact transformations, such as abstraction or partial-order reduction, are more difficult to use. Consequently, weaker guarantees such as *probably approximately correct (PAC)* results become acceptable not only in contexts where the system is executable but unknown (*black box*) and absolute guarantees are principally impossible, but also for completely known systems (*white box*).

Example 1. Consider the task of model checking a reachability property of a probabilistic communication protocol, which starts by generating a few, say k, random bits. Thus the execution immediately branches into 2^k states. If there are only few or hard-to-find symmetries in the behaviour, standard analysis quickly becomes infeasible. In the following, we discuss drawbacks of previously studied alternative approaches; then we suggest a new one that overcomes the difficulties for a wide class of models.

The exponential state-space explosion quickly renders explicit VI unable to propagate information by more than a single step. Besides, if the transition probabilities depend on the generated bits, even the symbolic variants of VI

* This research was funded in part by TUM IGSSE Grant 10.06 (PARSEC) and the German Research Foundation (DFG) project 383882557 *Statistical Unbounded Verification* (KR 4890/2-1).

[BCH+97] cannot help much. There have been two major alternatives proposed, both relying on extensive use of simulations.

• (I) For large and possibly *unknown* systems, *statistical model checking (SMC)* [YS02] reincarnates the Monte Carlo method. It runs simulations of the system; the resulting statistics then yields confidence intervals, i.e. PAC results. However, for unbounded-horizon properties, such as reachability or linear temporal logic (LTL) [Pnu77], performing simulations of finite length requires some information about the model [Kře16]:

1. Either the second eigenvalue can be bounded [LP08,YCZ10], which requires essentially the complete knowledge of the system (white box) and is as hard as solving the model checking problem, or
2. the topology is known [YCZ10,HJB+10] (sometimes called *grey box*, e.g. [AKW19]) and the whole system is preprocessed, which beats the purpose of sublinear analysis, or
3. a bound on the minimum transition probability $p_{\min}$ is known [BCC+14,DHKP17], the closest to black box, thus called *black SMC* here.

In black SMC, long enough simulations can be run to ensure the system passes from the transient to the recurrent part and reliable information on the whole infinite run is obtained. While the a-priori length is practically infeasible [BCC+14], early detection of recurrent behaviour has been proposed [DHKP17] as follows. Based on the observed part of a simulation run, a hypothesis on the topology of the system is made, answering what bottom strongly connected component (BSCC) this run ends up in. With repetitive observations of transitions over the run, the confidence grows that what currently looks as a BSCC indeed is a BSCC. Since quite a few repetitions of *all* transitions in the BSCC are required, this approach turns out practical only for systems with small BSCCs and not too small $p_{\min}$.

In this paper, assuming knowledge of the system (white-box setting), we twist the technique to a more efficient one as follows. After quickly gaining (unreliably low) confidence that the run entered a BSCC, we use the knowledge of the topology to confirm this information—again very quickly since not the whole model is built but only the *local* BSCC. Consequently, BSCCs are detected fast even in the case with larger BSCCs or small $p_{\min}$. As the information used turns out quite limited, corresponding to the grey-box setting, we call this approach *grey SMC*.

• (II) The other alternative to VI, now in the context or large but *known* systems, is the *asynchronous value iteration*, e.g. [BT89], a generalization of the Gauss-Seidel method and the core of reinforcement learning and approximate dynamic programming. There, the VI updates on states of the system are performed in varying orders, in particular possibly entirely skipping some states. The class of algorithms providing guarantees is represented by *bounded real-time dynamic programming (BRTDP)* [MLG05,BCC+14,AKW19] where the states to be updated at each moment are those appearing on a current simulation run. Consequently, states with low probability of visiting and thus low impact on the overall value are ignored. While this allows for treating very "wide" systems

with lots of unimportant branches, the scalability problem persists as soon as the branching is very uniform (see also Example 5 on Fig. 2b). From this perspective, grey SMC relaxes the rigorous approximation in the transient part and replaces it with a statistical estimate.

Overall, grey SMC fills the gap in the following spectrum:

VI	BRTDP	grey SMC	black SMC
analysis	analysis with simulation inside	simulation with analysis inside	simulation

On the one end, numeric analysis (VI) provides reliable results; in BRTDP, simulations are additionally used in the analysis to improve the performance while preserving the guarantees. On the other end, simulations (SMC) provide PAC guarantees; grey SMC then improves the performance by additional analysis in the simulation.

Our contribution can be summarized as follows:

- We modify the black SMC for unbounded properties of [DHKP17] to perform better in the white-box (and actually also in the so-called grey-box) setting.
- We compare our grey SMC to black SMC, BRTDP and VI both conceptually, illustrating advantages on examples, as well as experimentally, comparing the runtimes on standard benchmarks.
- We present all algorithms within a unified framework, which in our opinion eases understanding and comparison, provides a more systematic insight, and is pedagogically more valuable.

Outline of the paper: After recalling necessary definitions in Section 2, we describe and exemplify the algorithms in Section 3 and the respective key subprocedure in Section 4. Then we compare the algorithms and other related work in Section 5, discussing the expected implications, which we confirm experimentally in Section 6. For a broader account on related work on SMC in the context of unbounded-horizon properties, we refer the interested reader to the survey [Kře16].

2 Preliminaries

A *probability distribution* on a finite set X is a mapping $\delta : X \to [0, 1]$, such that $\sum_{x \in X} \delta(x) = 1$. The set of all probability distributions on X is denoted by $\mathcal{D}(X)$.

Definition 1 (MC). *A* Markov chain *(MC) is a tuple* $(\mathsf{S}, \mathsf{s}_0, \delta)$*, where* S *is a finite set of* states *with a designated* initial state $\mathsf{s}_0 \in \mathsf{S}$*, and* $\delta : \mathsf{S} \to \mathcal{D}(\mathsf{S})$ *is a* transition function *that given a state* s *yields a probability distribution* $\delta(\mathsf{s})$ *over* successor *states. For ease of notation, we write* $\delta(\mathsf{s}, \mathsf{t})$ *instead of* $\delta(\mathsf{s})(\mathsf{t})$ *and* $\mathsf{Post}(\mathsf{s}) := \{\mathsf{t} \mid \delta(\mathsf{s}, \mathsf{t}) > 0\}$ *to denote the set of successors of a state.*

The semantics of an MC is given in the usual way by the probability space on paths. An *infinite path* ρ is an infinite sequence $\rho = \mathsf{s}_0\mathsf{s}_1 \cdots \in (\mathsf{S})^\omega$, such that for

every $i \in \mathbb{N}$ we have $\mathsf{s}_{i+1} \in \mathsf{Post}(\mathsf{s}_i)$. A finite path is a finite prefix of an infinite path. The Markov chain together with a state s induces a unique probability distribution $\mathbb{P}_\mathsf{s}$ over measurable sets of infinite paths [BK08, Ch. 10].

Definition 2 (Reachability probability). *For a target set $\mathsf{T} \subseteq \mathsf{S}$, we write $\Diamond\mathsf{T} := \{\mathsf{s}_0\mathsf{s}_1 \cdots \mid \exists i \in \mathbb{N} : \mathsf{s}_i \in \mathsf{T}\}$ to denote the (measurable) set of all infinite paths which eventually reach T. For each $\mathsf{s} \in \mathsf{S}$, we define the* value *in s as*

$$\mathsf{V}(\mathsf{s}) := \mathbb{P}_s(\Diamond\mathsf{T}).$$

The reachability probability *is then $\mathsf{V}(\mathsf{s}_0)$.*

The value function V satisfies the following system of equations, which is referred to as the *Bellman equations*:

$$\mathsf{V}(\mathsf{s}) = \begin{cases} 1 & \text{if } \mathsf{s} \in \mathsf{T} \\ \sum_{\mathsf{s}' \in \mathsf{S}} \delta(\mathsf{s}, \mathsf{s}') \cdot \mathsf{V}(\mathsf{s}') & \text{otherwise} \end{cases} \tag{1}$$

Moreover, V is the *least* solution to the Bellman equations, see e.g. [CH08].

Certain parts of the state space are of special interest for the analysis of MC with respect to unbounded-horizon properties, such as reachability:

Definition 3 (SCC, BSCC). *A non-empty set $T \subseteq \mathsf{S}$ of states is* strongly connected *if for every pair $\mathsf{s}, \mathsf{s}' \in \mathsf{S}$ there is a path (of non-zero length) from s to s'. Such a set T is a* strongly connected component (SCC) *if it is maximal w.r.t. set inclusion, i.e. there exists no strongly connected T' with $T \subsetneq T'$. An SCC T is called* bottom (BSCC)*, if for all states $\mathsf{s} \in T$ we have $\mathsf{Post}(\mathsf{s}) \subseteq T$, i.e. no transition leaves the SCC.*

Note that the SCCs of an MC are disjoint and that, with probability 1, infinitely often reached states on a path form a BSCC.

We consider algorithms that have a limited information about the MC:

Definition 4 (Black box and grey box setting). *An algorithm inputs an MC as* black box *if it cannot access the whole tuple, but*

- *it knows the initial state,*
- *for a given state, it can sample a successor t according to $\delta(\mathsf{s})$,*[1]
- *it knows $p_{\min} \leq \min_{\mathsf{s} \in \mathsf{S}, \mathsf{t} \in \mathsf{Post}(\mathsf{s})} \delta(\mathsf{s}, \mathsf{t})$, an under-approximation of the minimum transition probability.*

When input as grey box*, it additionally knows the number $|\mathsf{Post}(\mathsf{s})|$ of successors for each state s.*[2]

[1] Up to this point, this definition conforms to black box systems in the sense of [SVA04] with sampling from the initial state, being stricter than [YS02] or [RP09], where simulations can be run from any desired state.

[2] This requirement is slightly weaker than the knowledge of the whole topology, i.e. $\mathsf{Post}(\mathsf{s})$ for each s.

3 Description of algorithms

In this section, we describe all of the algorithms that we compare in this paper. They all use the framework of Algorithm 1. The differences are in the instantiations of the functions (written in capital letters). This allows for an easy and modular comparison.

Algorithm 1 Framework for all considered algorithms

Input: MC M, reachability objective T
Output: (An estimate of) $\mathbb{P}_{\mathsf{s}_0}(\Diamond \mathsf{T})$
1: **procedure** COMPUTE REACHABILITY PROBABILITY
2: INITIALIZE
3: **repeat**
4: $X \leftarrow$ GET_STATES
5: UPDATE(X)
6: **until** TERM_CRIT

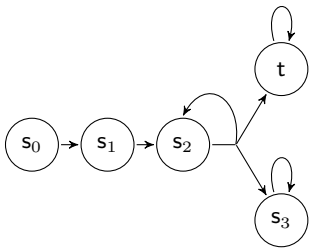
3.1 Value iteration

Value iteration (VI), e.g. [Put14], computes the value for all states in the MC. As memory, it saves a rational number (the current estimate of the value) for every state. In INITIALIZE, the estimate is set to 1 for target states in T and to 0 for all others. GET_STATES returns the whole state space, as the estimate of all values is updated simultaneously. The UPDATE works by performing a so called *Bellman backup*, i.e. given the current estimate function L_i, the next estimate L_{i+1} is computed by applying the Bellman Equation (1) as follows:

$$\mathsf{L}_{i+1}(\mathsf{s}) = \sum_{\mathsf{s}' \in \mathsf{S}} \delta(\mathsf{s}, \mathsf{s}') \cdot \mathsf{L}_i(\mathsf{s}')$$

Example 2. Consider the MC from Figure 1a, with $\delta(\mathsf{s}_2, \mathsf{s}_2) = \delta(\mathsf{s}_2, \mathsf{t}) = \delta(\mathsf{s}_2, \mathsf{s}_3) = \frac{1}{3}$ and the reachability objective $\{t\}$. The estimates that VI computes in the first 4 iterations are depicted in Figure 1b. The target state t is initialized to 1, everything else to 0. The estimate for s_3 stays at 0, as it is a BSCC with no possibility to reach the target state. Since these two states do not change, they are omitted in the figure. In every iteration, the estimates are updated and become more precise, coming closer to the true value 0.5 for s_0, s_1 and s_2. However, they converge to 0.5 only in the limit, as for any finite number of iterations there is a positive probability to remain in s_2. Note that s_0 always is two steps behind s_2, as it takes two iterations to backpropagate the current estimate.

VI converges to the true value only in the limit, hence we need some termination criterion TERM_CRIT to stop when we are close enough. However, to be



(a)

Iter.	$\mathsf{L}(\mathsf{s}_0)$	$\mathsf{L}(\mathsf{s}_1)$	$\mathsf{L}(\mathsf{s}_2)$
0	0	0	0
1	0	0	$1/3$
2	0	$1/3$	$4/9$
3	$1/3$	$4/9$	$13/27$
4	$4/9$	$13/27$	$40/81$

(b)

Iter.	$\mathsf{L}(\mathsf{s}_2)$	$\mathsf{U}(\mathsf{s}_2)$
0	0	1
1	0.01	0.99
2	0.029	0.980
3	0.039	0.970
4	0.048	0.961

(c)

Fig. 1: (a) Example Markov chain (b) Under approximations computed by value iteration, see Example 2 (c) Under- and over-approximations computed by bounded value iteration, see Example 3.

certain that the estimate is close, one has to perform an exponential number of iterations [CH08], which is infeasible. Hence, usually this version of VI does not give convergence guarantees, but instead just runs until the difference between two successive iterations is small. The result of this heuristic is guaranteed to be a lower bound, but can be arbitrarily imprecise [HM18], as we will also see in Example 3.

3.2 Bounded value iteration

To be able to give convergence guarantees, *Bounded value iteration* (BVI, also called interval iteration) was introduced more generally for Markov decision processes in [BCC+14,HM18]. In addition to the under-approximation computed by VI, this approach also computes a convergent over-approximation. For this, it stores a second rational number for every state. Dually to the under-approximation, INITIALIZE sets the estimate to 0 in states that cannot reach the target and 1 everywhere else. Note that finding the states with value 0, i.e. BSCC that do not contain the target, BVI has to perform a graph analysis, e.g. a backwards search from the targets. BVI still works on the whole state space and the update is completely analogous to VI, only this time updating both approximations. As TERM_CRIT, BVI checks that difference between the over- and under-approximation in the initial state is smaller than a given precision ε. This guarantees that the returned value is ε-precise.

Example 3. Consider the MC from Figure 1a with the same objective, but this time with $\delta(\mathsf{s}_2, \mathsf{s}_2) = 0.98$ and $\delta(\mathsf{s}_2, \mathsf{t}) = \delta(\mathsf{s}_2, \mathsf{s}_3) = 0.01$. Note that by preprocessing we set the over approximation $\mathsf{U}(\mathsf{s}_3)$ to 0, as it is a BSCC with no possibility of reaching the target. The estimates BVI computes for s_2 in the first 4 iterations are depicted in Figure 1c.

If we were running VI only from below, we might stop after iteration 4, as the lower bound changes by less than 0.01 between these iterations and hence it seems to have converged close to the value. However, the difference between

upper and lower bounds is still very high, so BVI knows that there still is a huge uncertainty in the values, as it could be anything between 0.048 and 0.961. Eventually, both estimates converge close enough to 0.5; for example the lower bound might be 0.49 and the upper bound 0.51. Then BVI can return the value 0.5 (the center of the interval) with a precision of 0.01, as this value is off by at most that.

3.3 Simulation-based asynchronous value iteration

The biggest drawback of the two variants we introduced so far is that they always work on the whole state space. Because of the state-space explosion, this is often infeasible. In contrast, asynchronous value iteration only updates parts of the state space in every iteration of the loop, i.e. GET_STATES does not return the whole state space, but instead heuristically selects the states to update next. This not only speeds up the main loop, but also allows the algorithm to reduce the memory requirements. Indeed, instead of storing estimates for all states, one stores estimates only for the partial model consisting of previously updated states. In [BBS95,MLG05,BCC+14], the heuristic for selecting the states is based on simulation: a path is sampled in the model, and only the states on that path are updated. The partial model contains all states that have been encountered during some of the simulations. If the part of the state space that is relevant for convergence of value iteration is small, this can lead to enormous speed-ups [BCC+14,KM19].

Algorithm 2 Simulation-based implementation of GET_STATES

Input: MC M, reachability objective T, s_0
Output: A set of states $X \subseteq \mathsf{S}$

procedure SIMULATE
  $\rho \leftarrow \mathsf{s}_0$
  repeat
    $\mathsf{s}' \leftarrow$ sample from $\delta(\text{last}(\rho))$ according to NEXT_STATE
    $\rho \leftarrow \rho\mathsf{s}'$
  until $\text{last}(\rho) \in \mathsf{T}$ or STUCK
  return ρ

Algorithm 2 shows how states can be sampled through simulations, as done in [BCC+14]: Starting from the initial state, in every step of the simulation a successor is chosen from the distribution of the last state on the path. Note that this choice depends on another heuristic NEXT_STATE. The successor can be chosen according to the transition probabilities δ, but it has proven to be advantageous to additionally consider the difference between the upper and lower bound in the successor states [MLG05,BCC+14]. In consequence, states where we already know a lot (under- and over-approximations are close to each other) are given less priority than states where we still need information.

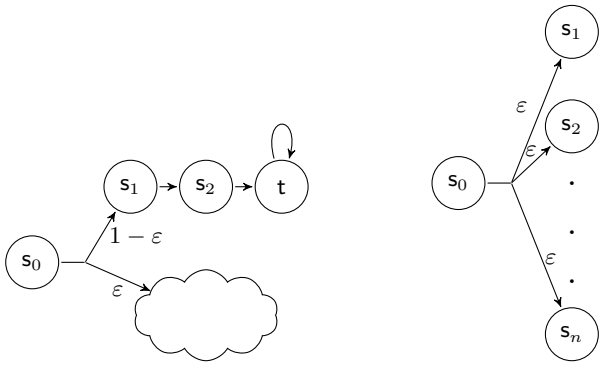


Fig. 2: (a) A Markov chain where exploring the whole state space can be avoided. ε denotes a transition probability. The cloud represents an arbitrarily large state space. (b) A Markov chain with high branching. From s_0, there is a uniform probabilistic choice with $n = \frac{1}{\varepsilon}$ successors.

The simulation is stopped in two cases: Either (i) it reaches a target state or (ii) it is stuck in a BSCC with no path to the target. Different heuristics for checking whether the simulation is stuck are discussed in depth in Section 4. Note that being able to differentiate between targets and non-target BSCCs during the simulations allows us not to do anything in INITIALIZE; we can set the value to 1 when reaching a target and 0 in the other case. The UPDATE function for simulation based asynchronous value iteration again uses the Bellman equation (1) to update the estimates of all states on the path; moreover, it can utilize additional information: Since GET_STATES returns a path, there is a notion of order of the states. Updating the states in reverse order backpropagates information faster.

Example 4. Consider the MC in Figure 2a, again with reachability objective $\{t\}$. The cloud represents an arbitrarily large state space. However, since it is only reachable with a very small probability ε, it need not be explored. Let the first sampled path be $\mathsf{s}_0\mathsf{s}_1\mathsf{s}_2\mathsf{t}$. This happens with high probability, as the only other possibility would be to select a successor from the cloud in state s_0, but since the selection process depends on the transition probabilities δ, going to s_1 has a higher probability. After the simulation reaches t, this value is backpropagated in reverse order. First the lower estimate $\mathsf{L}(\mathsf{s}_2)$ is set to 1, then $\mathsf{L}(\mathsf{s}_1)$ is set to 1, then $\mathsf{L}(\mathsf{s}_0)$ is set to $1-\varepsilon$. At this point the algorithm has converged, as difference between the lower and upper bound is ε.

So in this example, sampling the most probable path a single time gives a good approximation. The algorithm avoids exploring the large cloud and backprogagates values faster than synchronous VI.

Example 5. As an adversarial example, consider the MC in Figure 2b. Here, the model exhibits high branching, so every single path has a low probability, and only by aggregating all paths we actually get a high value. Unlike the previous example, there is no part of the state space that is clearly irrelevant. In fact, to achieve precision of ε the algorithm has to see so many paths that their cumulative probability is $1 - \varepsilon$, which in this case means seeing all but one transition from the starting state. This needs at least $\frac{1}{\varepsilon}$ simulations, but since the successors are chosen probabilistically, most likely a lot more.

Note that similarly to synchronous VI, there are versions of asynchronous VI without (RTDP [BBS95]) and with (BRTDP [MLG05,BCC+14])[3] guaranteed error bounds.

3.4 Statistical model checking

Algorithms for statistical model checking (SMC), [YS02], are different from all previously described ones in two ways, namely what they store and what they return. The VI-based algorithms store estimates for every (seen) state and they update these values to be ever more precise. Thus, the returned bounds on the values are certainly correct, although possibly quite loose. In contrast, SMC stores only a single accumulator (for the value of the initial state) and the returned value is probably approximately correct (PAC [Val84]), i.e. with high probability, the returned value is close to the true value. However, the returned confidence interval is not guaranteed to be a valid under- and over-approximation; if we are unlucky (i.e. with the remaining probability), there is no guarantee whatsoever on the returned value.

SMC does not need to do anything in INITIALIZE. It only stores a single accumulator to remember how often a target state was reached. GET_STATES works as in Algorithm 2 with NEXT_STATE typically sampling the successor according to the transition probabilities δ (in some settings, importance sampling may also be possible, e.g. [JLS12,BDH17]). UPDATE remembers whether we reached the target or not; in the end we can divide the number of reaches by the total number of samples to get the probability estimate. TERM_CRIT is a (typically low) number of samples that depends on the required probability of the guarantee and the width of the confidence interval; see [DHKP17, Section 2.2] for details or [JSD19] for more advanced techniques.

Example 6. Consider again the MC depicted in Figure 1a. Let the first sampled path be $\mathsf{s}_0\mathsf{s}_1\mathsf{s}_2\mathsf{s}_2\mathsf{t}$. At this point the simulation stops, as we have reached a target state, and we remember that we have seen a target once. Let the second path be $\mathsf{s}_0\mathsf{s}_1\mathsf{s}_2\mathsf{s}_2\mathsf{s}_2\mathsf{s}_3\mathsf{s}_3\ldots$. On the one hand, the STUCK function has to let the simulation continue, even though s_2 is seen 3 times and it looks like a cycle. On the other hand, it has to detect that the simulation will loop forever in s_3 and has to stop

[3] While all are more generally applicable to Markov decision processes, [MLG05] only ensures convergence if no end components [BK08] are present (for MC, no BSCCs without a target are present) and [BCC+14] lifts this restriction.

it. Ways to detect this are discussed in Section 4. After detecting that we are stuck, we remember that the simulation did not reach the target.

Let the required probability of the guarantee be 0.9 and the width of the confidence interval 0.1. Using Hoeffding's inequality [Hoe63] we can show that the required number of samples for this is 461. So assume that after 461 simulations we have seen the target 223 times. Then we know that with probability at least 0.9, the value is in the interval $^{223}/_{461} \pm 0.05$, i.e. $[0.434, 0.534]$. Increasing the number of simulations can both increase the confidence or decrease the width of the interval.

Note that this number of simulations is independent of the system. While 461 simulations are a lot for this small system, the number would be the same if we were considering a model with several billion states where value iteration is impossible.

4 STUCK

In this section, we discuss heuristics for detecting whether a simulation is stuck in a BSCC with no path to a target state. We also propose one new such heuristic with convenient theoretical properties.

For simulation-based asynchronous value iteration, previous work either excluded the existence of non-target BSCCs in their assumptions [BBS95,MLG05] or used a heuristic with no false negatives, but the possibility of false positives [BCC+14]. This means that if the simulation is stuck in a BSCC, the simulation definitely is stopped, which is required for termination. However, if the simulation is not stuck in a BSCC, it might still be stopped, guessing the value of the last state in the path is 0, although it might not be. The STUCK-heuristics used in previous work either depend on the path length ([BCC+14], [Ujm15, Chapter 7.5]) or simply stops exploring when any state is seen twice [AKW19, Appendix A.3].

SMC has to be sure with high probability that the simulation is stuck, as otherwise it loses the probabilistic guarantee. In [YCZ10], two approaches are described. The first approach requires knowledge of the second eigenvalue of the MC in order to guarantee asymptotic convergence. However, getting the second eigenvalue is as hard as the verification problem itself. The second approach works in the grey-box setting and pre-processes the MC so that all potentially infinite paths are eliminated. A similar transformation, using white-box information, was suggested in [HJB+10]. However, both of these approaches transform the whole model and thus face problems in the case of very large models.

An alternative was suggested in [DHKP16]. It monitors the finite path sampled during the simulation, implicitly constructing a graph with all seen states as nodes and all seen transitions as edges. The *candidate* of the current path is the (possibly empty) set of states forming the maximal BSCC of this graph. Intuitively, it is what we believe to be a BSCC given the observation of the current simulation. This candidate has to be validated, because as we saw in Example 6, a state set can look like a BSCC for several steps before being exited.

In the black-box setting, this validation works by continuing the simulation until the probability of overlooking some transition exiting the candidate becomes very small [DHKP16].

In this paper, we pinpoint that in the grey-box or white-box setting, this costly type of validation is not necessary. Instead of validating the candidate by running around in it for a huge number of steps, one can verify it using the additional information on the model. If no successor of any state in the candidate is outside of the candidate, then it indeed is a BSCC. Formally, for a candidate T, we check that $\{\mathsf{s} \mid \exists \mathsf{t} \in T : \mathsf{s} \in \mathsf{Post}(t)\} \subseteq T$ (if the topology is known), or alternatively that $\forall \mathsf{t} \in T : |\widehat{\mathsf{Post}}(t)| = |\mathsf{Post}(t)|$ (in what we defined as the grey-box setting) where $\widehat{\mathsf{Post}}$ yields the number of successors within the observed candidate.

Example 7. Consider again the MC depicted in Figure 1a. When a simulation enters s_3, STUCK should return true in order to stop the simulation, as it has reached a BSCC with no path to a target. In the black box setting of [DHKP16], this is only possible after continuing the simulation for another huge amount of steps. For example, even in a BSCC with only a single state, hundreds of further steps can be necessary to reach the required confidence. Given the grey-box information, the algorithm can determine that all successors of the states in the candidate ($\{\mathsf{s}_3\}$) have been seen and conclude that the candidate is indeed a BSCC.

However, this check stops the simulation and can incur an overhead if there are many SCCs in the transient part of the state space. Hence, we can delay it, not checking at the first occurrence of a cycle, but e.g. only when every state in the candidate has been seen twice. Alternatively, one can only allow the check every n (e.g. hundred) steps of the simulation. Depending on the model and the implementation of the algorithm, these heuristics can have some impact on the runtime.

Furthermore, one might modify this heuristic even further. If a state of the BSCC is only reached with low probability, it takes many steps for the simulation to reach it. When we check whether the current candidate is a BSCC, this state might not have occurred in the simulation yet. Instead of concluding that the information is insufficient and the simulation has to continue, one could deterministically explore the unknown successors and *compute* the BSCC. On the one hand, for small to medium sized BSCCs, this could result in a speed-up. On the other hand, it increases the overhead when transient SCCs are checked by STUCK. Consequently, in the available benchmarks, this heuristic did not prove advantageous. Hence we do not even report on it in the evaluation section.

5 Discussion

5.1 Dependency of simulation length on topology

Although the number of samples in SMC is independent of the model size, the length of the simulations is highly dependent on the model size and even more on the structure. Indeed, any kind of cyclic behaviour in the transient part of the state space increases the simulation time for two reasons. Firstly, the simulation loops in transient SCCs and does not make progress towards a target or a BSCC. Secondly, the check whether the simulation is stuck in transient SCCs incurs an overhead. An adversarial handcrafted worst-case example where simulations struggle is given in [HM18, Figure 3]. Moreover, the structure of BSCCs affects the length of the simulation. For cyclic BSCCs, the simulation easily encounters all states of the BSCC and can quickly terminate. For more complex topologies, some states are typically only seen with very low frequency and thus the simulation takes longer.

If the model exhibits many transient SCCs, using any simulation-based technique is problematic.

5.2 Black, grey and white SMC

The difference between the variants of SMC we report on are their knowledge of the transition system: $p_{\min}$ corresponds to black, the number of successors to grey and the exact successors and probabilities to the white-box setting. This information can be used in the STUCK-check; apart from that, the algorithms are the same.

Comparing grey and black box, it is apparent that simulations in grey box can be much shorter, as upon detection of a candidate that is a BSCCs the simulation is immediately stopped, whereas in the black box setting it has to continue for a number of steps. This number of steps depends on two things: (i) The size of the BSCC, as larger BSCC take longer to explore, especially since all states, no matter how improbable, need to be seen a certain amount of times, and (ii) the given under-approximation of the minimum transition probability p_{min}, as this determines how often every state in the candidate has to be seen until the probability of a false positive is small enough.

Thus, for large BSCCs or small $p_{\min}$, grey SMC is clearly better, as we also experimentally validate in Table 3 (large BSCC) and Table 2 (various $p_{\min}$) in the next section. For small BSCCs (e.g. only of size 1) and not so small $p_{\min}$, black and grey SMC become more comparable, but grey SMC still has shorter simulations. However, practically, the overhead of verifying the candidates in grey SMC can be so large that black SMC can even be slightly faster than grey SMC (see e.g., `leader6_11` in Table 1).

Heuristically reducing the number of checks in grey SMC (as described in Section 4) can make it faster again, but the effectiveness of the heuristics depends on the models. So, if it is known that the BSCC-detection is very easy for black SMC (e.g. they are of size 1 or cyclic and $p_{\min}$ is not too small), black SMC can

be a viable choice. However, as black SMC is never far better, using grey SMC is the safer variant when facing models with uncertain topology.

5.3 Comparison of algorithms

Finally, we compare the (dis-)advantages of the different algorithms, giving a practical decision guidance. If hard guarantees are required, then BVI or BRTDP are to be used. The latter is simulation based, and thus good if only a small part of the state space is relevant for convergence. Additionally, if the model is too large for BVI, BRTDP still has a chance, but quite possibly the partial model will also be too large. Conversely, if the model contains lots of transient SCCs, BVI is preferable, as simulation based approaches fail on this kind of model, see Section 5.1. Note that, if there are small probabilities present, it might take very long for BVI and BRTDP to converge, see Example 3.

For a quick estimate, or if PAC guarantees are sufficient, or if the system is too large, so that it is not possible to provide hard guarantees, SMC is to be used, if possible (white or grey box setting) in our grey variant. As both the memory and the termination criterion are independent of the size of the system, SMC always has a chance to yield an estimate, which additionally comes with a probabilistic guarantee.

There is no case in which un-guaranteed (synchronous or asynchronous) VI are preferable, as they suffer from the same drawbacks as BVI and BRTDP, but additionally do not provide guarantees. Whenever hard guarantees are not of interest and the system is not strongly connected, grey SMC should be used for a quick estimate.

5.4 Extensions to other unbounded-horizon properties

For more complex unbounded-horizon properties [BK08], such as Until (avoid-reach), LTL or long-run average reward, (B)VI pre-processes the state space to analyze the BSCCs [BK08] and BRTDP [BCC$^+$14] can either do the same or analyze the encountered BSCCs only. Black SMC of [DHKP17] is applicable through additional analysis of the BSCC candidates after they have been found likely to be BSCCs. This is directly inherited by grey SMC and makes it available for these specifications with low overhead.

6 Experimental evaluation

We implemented grey SMC in a branch of the PRISM Model Checker [KNP11] extending the implementation of black SMC [DHKP17]. We ran experiments on (both discrete- and continuous-time) Markov chains from the PRISM Benchmark Suite [KNP12a]. In addition to a comparison to black SMC, we also provide comparisons to VI and BVI of PRISM and BRTDP of [BCC$^+$14]. An interested reader may also want to refer [DHKP17, Table II] for a comparison of black SMC against two unbounded SMC techniques of [YCZ10].

Table 1: Runtime (in seconds) comparison of black and grey SMC for various benchmarks. BVI runtimes are also presented as a baseline.

Model/	Size	$p_{\min}$	BSCC	SMC		BVI
Property			(No., Max. Size)	Black	Grey	
`bluetooth(10)` `time_qual`	$> 569K$	7.81×10^{-3}	$> 5.8K, 1$	9	**7**	TO
`brp_nodl(10K,10K)` `p1_qual`	$> 40M$	1×10^{-2}	$> 4.5K, 1$	86	**84**	TO
`crowds_nodl(8,20)` `positive_qual`	68M	5×10^{-2}	$> 3K, 1$	10	**8**	TO
`egl(20,20)` `unfairA_qual`	1719T	5×10^{-1}	$1, 1$	43	**25**	TO
`gridworld(400,0.999)` `prop_qual`	384M	1×10^{-3}	$796,160K$	15	**8**	TO
`herman-17` `4tokens`	10G	4.7×10^{-7}	$1, 34$	TO	**73**	98
`leader6_11` `elected_qual`	$> 280K$	5.6×10^{-7}	$1, 1$	**106**	152	OOM
`nand(50,3)` `reliable_qual`	11M	2×10^{-2}	$51, 1$	11	**10**	455
`tandem(2K)` `reach_qual`	$> 1.7M$	2.4×10^{-5}	$1, > 501K$	**7**	**7**	62

For every run configuration, we run 5 experiments and report the median. In black SMC, the check for candidates is performed every 1000 steps during path simulations, while in grey SMC the check is performed every 100 steps. Additionally, grey SMC checks if a candidate is indeed a BSCC once every state of the candidate is seen at least twice. In all our tables, 'TO' denotes a timeout of 15 minutes and 'OOM' indicates that the tool ran out of memory restricted to 1GB RAM.

6.1 Comparison of black and grey SMC

Table 1 compares black SMC and grey SMC on multiple benchmarks. One can see that, except in the case of `leader6_11` and `brp_nodl`, grey SMC finishes atleast as soon as black SMC. In `bluetooth`, `gridworld`, `leader` and `tandem`, both the SMC methods are able to terminate without encountering any candidate

Table 2: Effect of $p_{\min}$ on black SMC runtimes (in seconds) on some of the benchmarks. Lower $p_{\min}$ demands stronger candidates, due to which black SMC has to sample longer paths.

	Black SMC/$p_{\min}$				Grey SMC
Model	1×10^{-2}	1×10^{-3}	1×10^{-4}	1×10^{-5}	
`brp_nodl(10K,10K)`	86	93	183	TO	**84**
`crowds_nodl(8,20)`	11	41	334	TO	**9**
`egl(20,20)`	47	106	875	TO	**44**

(i.e. either the target is seen or the left side of the until formula is falsified). In `brp_nodl`, `crowds_nodl` and `nand`, the SMC methods encounter a candidate, however, since the candidate has only a single state (all BSCCs are trivial), black SMC is quickly able to confidently conclude that the candidate is indeed a BSCC. The only interesting behaviour is observed on the `herman-17` benchmark. In this case, every path eventually encounters the only BSCC existing in the model. Grey SMC is able to quickly conclude that the candidate is indeed a BSCC, while black SMC has to sample for a long time in order to be sufficiently confident.

The performance of black SMC is also a consequence of the $p_{\min}$ being quite small. Table 2 shows that black SMC is very sensitive towards $p_{\min}$. Note that grey SMC is not affected by the changes in $p_{\min}$ as it always checks whether a candidate is a BSCC as soon as all the states in the candidate are seen twice.

6.2 Grey SMC vs. Black SMC/BRTDP/BVI/VI

We now look more closely at the self-stabilization protocol `herman` [KNP12b,Her90]. The protocol works as follows: `herman-N` contains N processes, each possessing a local boolean variable x_i. A token is assumed to be in place i if $x_i = x_{i-1}$. The protocol proceeds in rounds. In each round, if the current values of x_i and x_{i-1} are equal, the next value of x_i is set uniformly at random, and otherwise it is set equal to the current value of x_{i-1}. The number of states in `herman-N` is therefore 2^N. The goal of the protocol is to reach a stable state where there is exactly one token in place. For example, in case of `herman-5`, a stable state might be $(x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1)$, which indicates that there is a token in place 2. In every `herman` model, all stable states belong to the single BSCC. The number of states in the BSCC range from 10 states in `herman-5` to 2,000,000 states in `herman-21`.

For all `herman` models in Table 3, we are interested in checking if the probability of reaching an unstable state where there is a token in places 2-5, i.e. $(x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 1)$ is less than 0.05. This property, which we name `4tokens`, identifies 2^{N-5} states as target in `herman-N`. The results in Table 3 show how well grey SMC scales when compared to black SMC, BRTDP,

Table 3: Runtime (in seconds) of the various algorithms on the Herman self-stabilization protocol [KNP12b] with the property `4tokens`. The median runtimes are reported for grey SMC, black SMC [DHKP17], BRTDP [BCC+14], Bounded value iteration (BVI) and Value iteration (VI). The SMC algorithms use SPRT method with parameters $\alpha = 0.01$ and $\beta = 0.01$. BRTDP, BVI and VI run until a relative error of 0.01 is obtained.

Model	States	Grey SMC	Black SMC	BRTDP	BVI	VI
`herman-5`	32	11	15	TO	1	1
`herman-7`	128	12	57	TO	1	1
`herman-9`	512	10	775	TO	1	1
`herman-11`	2048	19	TO	TO	1	1
`herman-13`	8192	18	TO	TO	1	1
`herman-15`	33K	17	TO	TO	9	3
`herman-17`	131K	49	TO	TO	98	21
`herman-19`	524K	252	OOM	TO	602	113
`herman-21`	2M	759	OOM	OOM	TO	TO

BVI[4] and VI. Black SMC times out for all models where $N \geq 11$. This is due to the fact that the larger models have a smaller $p_{\min}$, thereby requiring black SMC to sample extremely long paths in order to confidently identify candidates as BSCCs. BVI and VI perform well on small models, but as the model sizes grow and transition probabilities become smaller, propagating values becomes extremely slow. Interestingly, we found that in both grey SMC and black SMC, approximately 95% of the time is spent in computing the next transitions, which grow exponentially in number; an improvement in the simulator implementation can possibly slow down the blow up in run time, allowing for a fairer comparison with the extremely performant symbolic value iteration algorithms.

Finally, we comment on the exceptionally poor performance of BRTDP on `herman` models. In Table 4, we run BRTDP on three different properties: (i) tokens in places 2-3 (`2tokens`); (ii) tokens in places 2-4 (`3tokens`); and (iii) tokens in places 2-5 (`4tokens`). The number of states satisfying the property decrease when going from 2 tokens to 4 tokens. The table shows that BRTDP is generally better in situations where the target set is larger.

In summary, the experiments reveal the following:

– For most benchmarks, black SMC and grey SMC perform similar, as seen in Table 1. As expected, the advantages of grey SMC do not show up in these examples, which (almost all) contain only trivial BSCCs.

[4] We refrain from comparison to other guaranteed VI techniques such as sound VI [QK18] or optimistic VI [HK19] as the implementations are not PRISM-based and hence would not be too informative in the comparison.

Table 4: Effect of restrictive properties (satisfied in fewer states) on the runtime (in seconds) of BRTDP in the herman benchmarks.

	Property		
Model	`2tokens`	`3tokens`	`4tokens`
`herman-5`	1	2	TO
`herman-7`	1	1	TO
`herman-9`	1	2	TO
`herman-11`	2	2	TO
`herman-13`	2	2	TO
`herman-15`	3	3	TO
`herman-17`	5	6	TO
`herman-19`	9	9	TO
`herman-21`	104	111	TO
`herman-23`	OOM	OOM	OOM

– The advantage of grey SMC is clearly visible on the `herman-N` benchmarks, in which there are non-trivial BSCCs. Here, black SMC quickly fails while grey SMC is extremely competitive.

– Classical techniques such as VI and BVI fail when either the model is too large or the transition probabilities are too small. However, they are still to be used for strongly connected systems, where the whole state space needs to be analysed for every run in both SMC approaches, but only once for (B)VI.

7 Conclusion

While SMC has found its use also in the white-box setting as a scalable alternative, we introduce the first approach that utilizes the knowledge in a local way, without globally processing the state space, and thus preserves the efficency advantages of black-box SMC. We call this approach grey SMC since we utilize only the topological information and not the quantitative information (sometimes referred to as grey box). On the one hand, this is useful as the quantitative information is often unavailable or imprecise w.r.t. the modelled reality. On the other hand, while the full quantitative information is irrelevant in BSCCs, it plays a major role in the transient phase and could be used to further enhance the approach. For instance, it could be used for importance sampling in order to handle rare events efficiently [JLS12,BDH17] even in the context of unbounded-horizon properties.

References

[AKW19] Pranav Ashok, Jan Kretínský, and Maximilian Weininger. PAC statistical model checking for markov decision processes and stochastic games. In *CAV (1)*, pages 497–519. Springer, 2019.

[BBS95] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artif. Intell.*, 72(1-2):81–138, January 1995.

[BCC+14] Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov decision processes using learning algorithms. In *ATVA*, pages 98–114, 2014.

[BCH+97] Christel Baier, Edmund M. Clarke, Vasiliki Hartonas-Garmhausen, Marta Z. Kwiatkowska, and Mark Ryan. Symbolic model checking for probabilistic processes. In *ICALP*, pages 430–440. Springer, 1997.

[BDH17] Carlos E. Budde, Pedro R. D'Argenio, and Arnd Hartmanns. Better automated importance splitting for transient rare events. In *SETTA*, volume 10606 of *Lecture Notes in Computer Science*, pages 42–58. Springer, 2017.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[BT89] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., USA, 1989.

[CH08] Krishnendu Chatterjee and Thomas A Henzinger. Value iteration. In *25 Years of Model Checking*, pages 107–138. Springer, 2008.

[DHKP16] Przemyslaw Daca, Thomas A. Henzinger, Jan Kretínský, and Tatjana Petrov. Faster statistical model checking for unbounded temporal properties. In *TACAS 2016*, pages 112–129, 2016.

[DHKP17] Przemyslaw Daca, Thomas A. Henzinger, Jan Kretínský, and Tatjana Petrov. Faster statistical model checking for unbounded temporal properties. *ACM Trans. Comput. Log.*, 18(2):12:1–12:25, 2017.

[Her90] Ted Herman. Probabilistic self-stabilization. *Inf. Process. Lett.*, 35(2):63–67, 1990.

[HJB+10] Ru He, Paul Jennings, Samik Basu, Arka P. Ghosh, and Huaiqing Wu. A bounded statistical approach for model checking of unbounded until properties. In *ASE*, pages 225–234, 2010.

[HK19] Arnd Hartmanns and Benjamin Lucien Kaminski. Optimistic value iteration. *CoRR*, abs/1910.01100, 2019.

[HM18] Serge Haddad and Benjamin Monmege. Interval iteration algorithm for mdps and imdps. *Theor. Comput. Sci.*, 735:111–131, 2018.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[JLS12] Cyrille Jégourel, Axel Legay, and Sean Sedwards. Cross-entropy optimisation of importance sampling parameters for statistical model checking. In *CAV*, pages 327–342, 2012.

[JSD19] Cyrille Jégourel, Jun Sun, and Jin Song Dong. Sequential schemes for frequentist estimation of properties in statistical model checking. *ACM Trans. Model. Comput. Simul.*, 29(4):25:1–25:22, 2019.

[KM19] Jan Křetínský and Tobias Meggendorfer. Of cores: A partial-exploration framework for Markov decision processes. *Submitted*, 2019.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.

[KNP12a] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. The PRISM benchmark suite. In *QEST*, pages 203–204. IEEE Computer Society, 2012.

[KNP12b] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Probabilistic verification of herman's self-stabilisation algorithm. *Formal Asp. Comput.*, 24(4-6):661–670, 2012.

[Kře16] Jan Křetínský. Survey of statistical verification of linear unbounded properties: Model checking and distances. In *ISoLA (1)*, pages 27–45, 2016.

[LP08] Richard Lassaigne and Sylvain Peyronnet. Probabilistic verification and approximation. *Ann. Pure Appl. Logic*, 152(1-3):122–131, 2008.

[MLG05] H. Brendan Mcmahan, Maxim Likhachev, and Geoffrey J. Gordon. Bounded real-time dynamic programming: Rtdp with monotone upper bounds and performance guarantees. In *In ICML05*, pages 569–576, 2005.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[Put14] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[QK18] Tim Quatmann and Joost-Pieter Katoen. Sound value iteration. In *CAV (1)*, pages 643–661. Springer, 2018.

[RP09] Diana El Rabih and Nihal Pekergin. Statistical model checking using perfect simulation. In *ATVA*, pages 120–134, 2009.

[SVA04] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical model checking of black-box probabilistic systems. In *CAV*, pages 202–215, 2004.

[Ujm15] Mateusz Ujma. *On verification and controller synthesis for probabilistic systems at runtime*. PhD thesis, University of Oxford, UK, 2015.

[Val84] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[YCZ10] Håkan L. S. Younes, Edmund M. Clarke, and Paolo Zuliani. Statistical verification of probabilistic properties with unbounded until. In *SBMF*, volume 6527 of *Lecture Notes in Computer Science*, pages 144–160. Springer, 2010.

[YS02] Håkan L. S. Younes and Reid G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *CAV*, pages 223–235. Springer, 2002.