Optimistic Optimization for Statistical Model Checking with Regret Bounds
ECE 584 Project Report

Negin Musavi[1], Dawei Sun[1], Sayan Mitra[1], Geir Dullerud[1], and Sanjay Shakkottai[2]
{nmusavi2,daweis2,mitras,dullerud}@illinois.edu
[1]University of Illinois at Urbana Champaign
sanjay.shakkottai@utexas.edu
[2]University of Texas at Austin

December 18, 2019

Abstract

We explore application of multi-armed bandit algorithms to statistical model checking (SMC) of Markov chains initialized to a set of states. We observe that model checking problems requiring maximization of probabilities of sets of execution over all choices of the initial states, can be formulated as a multi-armed bandit problem, for appropriate costs and rewards. Therefore, the problem can be solved using multi-fidelity hierarchical optimistic optimization (MFHOO). Bandit algorithms, and MFHOO in particular, give (regret) bounds on the sample efficiency which rely on the smoothness and the near-optimality dimension of the objective function, and are a new addition to the existing types of bounds in the SMC literature. We present a new SMC tool—HooVer—built on these principles and our experiments suggest that: Compared with exact probabilistic model checking tools like Storm, HooVer scales better; compared with the *statistical* model checking tool PlasmaLab, HooVer can require much less data to achieve comparable results.

1 Introduction

The multi-armed bandit problem is an idealized mathematical model for sequential decision making in unknown random environments and it has been used to study exploration-exploitation trade-offs. In the problem setup, each arm $x \in \mathcal{X}$ of the bandit is associated with a cost λ_x of playing and an unknown reward distribution M_x. In order to maximize the final reward with a given cost budget, the algorithm plays some arm, collects the stochastically generated reward, and decides on the next arm, until the cost budget is exhausted. Starting from the motivation of designing clinical trials in the 1930s [1, 2, 3], there has been major developments in the Bandit theory over the last few decades (see, for example the books [4, 5, 6]). Several different strategies have addressed this problem and strong connections have been drawn with other fields such as online learning.

In this paper, we explore how Bandit algorithms can be used for model checking of stochastic systems. A requirement R for a stochastic system $\mathcal{M}$ usually checks whether the measure of executions of $\mathcal{M}$ satisfying certain temporal formulas cross certain thresholds [7, 8]. Model checking for such requirements can be solved by calculating the exact measure of the executions that satisfy the relevant subformulas of R [9, 10, 11, 12]. In this paper, we focus on the alternative *statistical model checking (SMC)* approach which samples some executions of $\mathcal{M}$ and uses hypothesis testing to infer whether the samples provide statistical evidence for the satisfaction (or violation) of R [7, 13, 14]. Execution data is a costly resource[1], therefore, a number of SMC approaches minimize the *expected number of samples* needed for verification, for example, using sequential probability ratio tests, Chernoff bound, and Student's t-distribution.

Several SMC tools have been developed (for example, Ymer [15], VESTA [16], MultiVesta [17], PlasmaLab [18], MODES [19], UPPAAL [20], and MRMC [21]), and they have been used to verify many systems [22, 23, 24, 25, 26, 27, 28, 29]. Most SMC algorithms crucially rely on fully stochastic models that never make nondeterministic choices. Although recent progress has been made towards verifying Markov Decision Processes with restricted types of schedulers [30, 31, 32], SMC for MDPs remain a challenge problem (see [33] and [34] for recent surveys).

We will focus on stochastic models that are essentially Discrete Time Markov Chains, except that they are initialized from a (possibly very large) set of states. In other words, these are Markov Decision Processes (MDPs) where the adversary gets to initialize[2]. Further, we restrict our attention to safety requirements[3]. That is, we study problems that require maximizing (or minimizing) the probability of hitting certain unsafe states, starting from any initial state. Further, this class of models and requirements capture many practical problems like online monitoring where the initialization has to consider worst case error bounds in state estimation, for example, from sensing and perception. We observe that this optimization of a probability measure over a set of initial choices, can coincide with the multi-armed bandit problem for appropriately defined costs and rewards. By building the connection with the Bandit literature, we not only gain algorithmic ideas, but also new types of theoretical (regret) bounds on the sample efficiency of the algorithms. These bounds rely on the smoothness and the near-optimality dimension of the objective function, and are fundamentally different from the existing performance bounds in the SMC literature.

Hierarchical optimistic optimization (HOO) [5] is a bandit algorithm that builds a tree on a search space $\mathcal{X}$ by using the so called principle of optimism in the face of uncertainty. It is a black-box optimization method that applies an upper confidence bound (UCB) on a tree search method for finding the optimal points over the uncertain domain. The UCB in the tree search approach takes care of the trade-off between exploiting the most promising parts of the domain and exploring the most uncertain parts of the domain. *Multi-fidelity hierarchical optimistic optimization(MFHOO)* [35] is a multi-fidelity HOO based method that allows noisy and biased observations from the uncertain domain. The performance of MFHOO is measured by how the *regret*—the gap between the actual maximum and the computed—scales with the number of samples. A key feature of these algorithms is that they can work with black-box functions and the regret guarantees only rely on certain smoothness parameters and the near-optimality dimension of the problem.

The standard theoretical assumptions required by off-the-shelf bandit algorithms in order to get performance guarantees do not precisely fit our verification problem, and that in-depth analysis and modification is required to get these guarantees in our setting; In addition, to apply these algorithms several functions need to be judiciously determined, a priori, and are at the heart of how the algorithms will perform. These choices are non-trivial and multi-faceted, and we develop

[1]Generating execution data involves running simulations or performing tests.

[2] Finite number nondeterministic action choices can be encoded in the choice of the initial state.

[3]All the results in the paper generalize to bounded time properties of the form $P_{\geq\theta}(\psi)$ where θ is a threshold constant and ψ is a path formula. Generalizing to nested probabilistic operators and unbounded time properties will require further research.

and provide such functions explicitly in the context of our SMC problem in order to demonstrate successful application.

The key contributions of the paper are as follows.

First, we show how the MFHOO algorithm, can be used for statistical model checking with provable regret bounds. In the process, we define an appropriate notions of fidelity, bias-functions, and also modify the existing near-optimality dimension required for regret bounds of MFHOO to accommodate the non-smoothness of the typical functions we have to optimize for SMC.

Second, we have built a new SMC tool called HooVer using MFHOO [35]. We have created a practically inspired [36, 37] suite of benchmark NiMC models that can be useful for safety analysis of driver assistance features in vehicles for standards such as ISO26262 [38]. Using the benchmarks we have carried out a detailed performance analysis of HooVer and our results suggest that the proposed approach can indeed make use of simulations more effectively than existing SMC approaches. A fair comparison of HooVer with other discrete-state model checkers like Prism [39], Storm [40], and PlasmaLab [41] is complicated as it relies on a continuous state models. We created discretized models for comparison, and observed that: Compared with exact probabilistic model checking tools like Storm, HooVer is faster, more memory efficient and scales better, and thus it can be used to check models with very large initial state space; Compared with *statistical* model checking tools like PlasmaLab, HooVer requires much less data to achieve comparable results.

Finally, to our knowledge , this is the first work connecting statistical model checking with the Bandits theory; specifically, the hierarchical tree search using the principle of optimism in the face of uncertainty. Thus we believe that the exposition of these algorithms engender new applications in verification and synthesis algorithms.

2 Model and problem statement

Consider a Euclidean space $\mathcal{X} = \mathbb{R}^m$ and let $\mathbb{R}_{\geq 0}$ denote the non-negative real numbers. For any real-valued function p of $\mathcal{X}$ its support is the set $\text{supp}(p) := \{x \in \mathcal{X} \mid p(x) \neq 0\}$. A *discrete probability distribution* over $\mathcal{X}$ is a function $p : \mathcal{X} \to [0,1]$ such that $\text{supp}(p)$ is countable, and $\sum_{x\in\text{supp}(p)} p(x) = 1$. We use $\mathbb{P}(\mathcal{X})$ to denote the set of discrete probability distributions over $\mathcal{X}$. For a finite set $\mathcal{S}$, $|\mathcal{S}|$ denotes the cardinality of $\mathcal{S}$.

2.1 Nondeterministically initialized Markov chains

Definition 1. A *Nondeterministically initialized Markov chains (NiMC)* $\mathcal{M}$ is defined by a triple $(\mathcal{X}, \Theta, P)$, where: (i) $\mathcal{X} = \mathbb{R}^m$ is the state space; (ii) $\Theta \subseteq \mathcal{X}$ is the set of possible initial states; and (iii) $P : \mathcal{X} \to \mathbb{P}(\mathcal{X})$ is the *probability transition function.*

That is, from state $x \in \mathcal{X}$, the next state is chosen according to the discrete distribution $P(x)$. The probability of transitioning from state x to state $x' \in \mathcal{X}$ is $P(x)(x')$, which we write more compactly as $P_{x,x'}$. An *execution* α of length k for the NiMC $\mathcal{M}$ is a sequence of states $\alpha = \{x_0, x_1, \dots, x_k\}$, where $x_0 \in \Theta$, and for each $i > 0$, $P_{x_{i-1},x_i} > 0$. We denote the set of all length k executions of $\mathcal{M}$ starting from x_0 as $\text{Execs}_{x_0}(k)$. The probability of an execution α, given x_0, is $\prod_{i=1}^{k} P_{x_{i-1},x_i} =: p(\alpha)$.

Given a set $\mathcal{U} \subseteq \mathcal{X}$, we say an execution α *hits* $\mathcal{U}$ if there exists $x \in \alpha$ such that $x \in \mathcal{U}$. We denote the subset of executions starting from x_0, of length k, that hit $\mathcal{U}$ by $\text{Execs}_{x_0}(k, \mathcal{U})$. From a

```
if $i = 1$
    $s_i \leftarrow s_i + \mathsf{vc}$
else
    $gap_i \leftarrow s_{i-1} - s_i$
    if $gap_i \leq \mathsf{near}$
        $s_i \leftarrow s_i + \mathsf{vc}$ $w.p.$ pnear
        $s_i \leftarrow s_i + \mathsf{vb}$ $w.p.$ $1 -$ pnear
    else
        $s_i \leftarrow s_i + \mathsf{vc}$ $w.p.$ pfar
        $s_i \leftarrow s_i + \mathsf{vb}$ $w.p.$ $1 -$ pfar
```

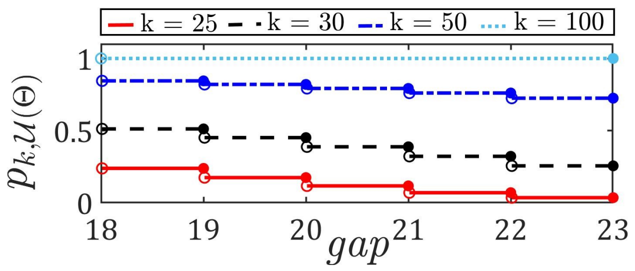

Figure 1: *Left*: Model of SLplatoon2. *Right*: Probability of hitting $\mathcal{U}$ vs. the initial *gap* with different execution lengths k. Here, initial $s_1 \in (20, 23), s_2 \in (0, 2)$, $\mathsf{vb} = 1$, $\mathsf{vc} = 2$, $\mathsf{pfar} = 0.85$, $\mathsf{pnear} = 0.15$, $\mathsf{near} = 3$ and $\delta = 1$

given initial state $x_0 \in \Theta$ the probability of $\mathcal{M}$ hitting an unsafe state within k steps is given by:

$$p_{k,\mathcal{U}}(x_0) = \sum_{\alpha \in \mathrm{Execs}_{x_0}(k,\mathcal{U})} p(\alpha). \tag{1}$$

Note that if $x_0 \in \mathcal{U}$ then $\mathrm{Execs}_{x_0}(k,\mathcal{U}) = \mathrm{Execs}_{x_0}(k)$ and $p_{k,\mathcal{U}}(x_0) = 1$. We are interested in finding the *worst case* probability of hitting unsafe states from any initial state of $\mathcal{M}$. This can be regarded as determining, for each k, the value

$$\max_{x_0 \in \Theta} p_{k,\mathcal{U}}(x_0). \tag{2}$$

Importantly, we would like to solve this optimization problem without relying on detailed information about the probability transition function P. Further, our solution should not rely on precisely computing $p_{k,\mathcal{U}}(x_0)$ for a given $x_0 \in \Theta$, but instead only the use of *noisy observations*.

2.2 Example: Single-lane platoon with two speeds (SLplatoon2)

We present an NiMC of a platoon of m cars on a single lane (SLplatoon2). Variations of this model are used in all our experiments later in Section 4. Each car probabilistically decides to "cruise" or "brake" based on its current gap with the predecessor. These types of models are used for risk analysis of Automatic Emergency Braking (AEB) systems [36, 37]. The probabilistic parameters of the model are derived from data collected from overhead traffic enforcement cameras on roads. The uncertainty in the initial positions (and gaps) arise from perception inaccuracies, which are modeled as worst-case ranges.

Let s_i be the position of i^{th} car in the sequence. Initially, s_i takes a value in an interval on the x-axis such that $s_1 > s_2 > \ldots > s_m$. The pseudocode in Figure 1 specifies the probabilistic transition rule that updates the position of all the cars synchronously. Car 1 always moves at a constant breaking speed of vc. The variable gap_i is the distance of i to the predecessor $i-1$, for each $i = 2, ..., m$. If gap_i is less than the constant threshold near, then i continues to cruise with probability pnear and it brakes with probability $1 - \mathsf{pnear}$. Similarly, if gap_i greater than near, then i continues to cruise with probability pfar and brakes with probability $1 - \mathsf{pfar}$.

It is straightforward to connect the above description to a formal definition of a NiMC. The state space $\mathcal{X} = \mathbb{R}^m$. The set of initial states Θ is a hyperrectangle in $\mathcal{X}$ (such that $s_1 > s_2 \ldots > s_m$). For any state, $x \in \mathcal{X}$ the probability transition function is given by the equations in lines 2, 6, 7, 9 and 10. We define the set of unsafe states $\mathcal{U} = \{(s_1, s_2, ..., s_m) \in \mathcal{X} \mid \exists i \in \{2, ..., m\}, \text{ such that } gap_i \leq \delta\} \subseteq \mathcal{X}$ for some constant collision threshold δ.

Given that cars start their motion at any initial state from Θ, the goal is to find the maximum probability of hitting the unsafe set $\mathcal{U}$. For $m = 2$, $\mathsf{vb} = 1$, $\mathsf{vc} = 2$, $\mathsf{pfar} = 0.85$ $\mathsf{pnear} = 0.15$, $\mathsf{near} = 3$, $\delta = 1$, and initial $s_1 \in (20, 23), s_2 \in (0, 2)$, Figure 1 shows estimates of probabilities of hitting the unsafe set from different initial separations between cars. As our intuition suggests, for large enough time horizons the probability of hitting the unsafe set approaches 1 from all initial states, but, for smaller time horizons the maximum probability of unsafety arises when the initial gap is smaller.

3 Statistical Model Checking with Optimistic Optimization

3.1 Hierarchical Optimistic Optimization

Multi-Fidelity Hierarchical Optimistic Optimization (MFHOO) [35] is an black-box optimization algorithm from the *multi-armed bandits* literature [4, 5, 6]. The setup is the following: suppose we want to maximize the function $f : \mathcal{X} \to \mathbb{R}$, which is assumed to have a unique global maximum. Let $f^* = \sup_{x \in \mathcal{X}} f(x)$. MFHOO allows the choice of evaluating f at different fidelities with different costs. This flexibility matters for SMC because it will be beneficial to evaluate the probability of unsafety $p_{k,x_0}(\mathcal{U})$ for certain initial states more precisely, for example, with longer number of simulations, while for other initial states a less precise evaluation may be adequate.

Thus, MFHOO has access to a biased function $f_z(x)$ that depends on fidelity parameter $z \in [0, 1]$. Setting $z = 0$ gives the lowest fidelity (and lowest cost) and $z = 1$ corresponds to full fidelity (and highest cost). At full fidelity, $f_1(x) = f(x)$, and the evaluation is unbiased. More generally, $|f_z(x) - f(x)| \leq \zeta(z)$ and evaluating $f_z(x)$ costs $\lambda(z)$, where the functions $\zeta, \lambda : [0, 1] \to \mathbb{R}_{>0}$ are respectively, non-increasing and non-decreasing, and called the *bias* and the *cost* functions [35].

A bandit algorithm chooses a sequence of sample points (arms) $x_1, x_2, \ldots \in \mathcal{X}$, evaluates them at fidelities $z_1, z_2, \ldots$, and receives the corresponding sequence of noisy observations (rewards) $y_1, y_2, \ldots$. We assume that each y_j is drawn from a unknown distribution M_{z_j,x_j} satisfying $f_{z_j}(x_j) = \int u dM_{z_j,x_j}(u)$. Further the distribution has a sub-Gaussian component, with variance σ^2, which captures uncertainty in the observations. The algorithm *actively chooses* x_{j+1} based on past choices $x_1, \ldots, x_j$ and observations $y_1, \ldots, y_j$. When the budget Λ is exhausted, the algorithm decides the optimal point $\bar{x}_{n(\Lambda)} \in \mathcal{X}$ and the optimal value $f(\bar{x}_{n(\Lambda)})$ with the aim of minimizing *regret*, which is defined as

$$R(\Lambda) = f^* - f(\bar{x}_{n(\Lambda)}).$$

The MFHOO algorithm (Algorithm 1) for selecting x_{j+1} estimates f^* by building a binary tree in which each height in the tree represents a partition of the state space $\mathcal{X}$. The algorithm maintains estimates of an upper-bound on f for each partition subset, and uses the principle of optimism for choosing the next sample x_{j+1}. That is, it chooses the samples in those partitions where the estimated upper-bounds are the highest.

Each node in the constructed tree is labeled by a pair of integers (h, i), where h is the height of the node in the tree, and i satisfying $0 \leq i \leq 2^h$ is its position within height level h. The root is labeled $(0, 1)$, and each node (h, i) can have two children $(h + 1, 2i - 1)$ and $(h + 1, 2i)$. Node (h, i) is associated with subset a of $\mathcal{X}$, denoted by $\mathcal{P}_{h,i}$, where $\mathcal{P}_{h,i} = \mathcal{P}_{h+1,2i-1} \cup \mathcal{P}_{h+1,2i}$, and for each h

these disjoint subsets satisfy $\cup_{i=1}^{2^h} \mathcal{P}_{h,i} = \mathcal{X}$. Therefore, larger values of h represent finer partitions of $\mathcal{X}$.

For each node (h,i) in the tree, the algorithm maintains the following information: (i) $count_{h,i}$: the number of times the node is visited; (ii) $\hat{f}_{h,i}$: the empirical mean of observations over points visited in $\mathcal{P}_{h,i}$; (iii) $U_{h,i}$: an initial estimate of the maximum of f over $\mathcal{P}_{h,i}$; and (iv) $B_{h,i}$: a tighter and optimistic upper bound on the maximum of f over $\mathcal{P}_{h,i}$.

The algorithm proceeds as follows. The *tree* starts out with a single node, the root $(0,1)$, initializes the B-values of its two children $B_{1,1}$ and $B_{1,2}$ to $+\infty$, and initializes the cost C to 0. At a high-level, in each iteration of the **while** loop (line 3), the algorithm adds a new node $(hnew, inew)$ in the *tree* and updates all of the above quantities for several nodes in *tree*. In more detail, first a *path* from the root to a leaf is found by traversing the child with the higher B-value (with ties broken arbitrarily). Let the child with the higher B-value of the traversed leaf be $(hnew, inew)$ (line 4). An arbitrary point x in the partition of this node $\mathcal{P}_{hnew,inew}$ is chosen (line 5). Then, this point is evaluated at fidelity $z_{hnew} = \zeta^{-1}(\nu\rho^{hnew})$ and a reward y is received (line 6). Next, *tree* is extended by inserting $(hnew, inew)$ in the *tree* (line 8) and for all the nodes (h,i) in *path* including $(hnew, inew)$, the $count_{h,i}$ and the empirical mean $\hat{f}_{h,i}$ are updated (line 9). Finally, in line 13, for all nodes (h,i) in *tree*, $U_{h,i}$ and $B_{h,i}$ are updated using the smoothness parameters $\nu_1 > 0$ and $\rho \in (0,1)$ and the parameter σ. Once the sampling budget Λ is exhausted, a leaf with maximum B-value is returned by the Algorithm 1 [35].

Algorithm 1 Multi-Fidelity Hierarchical Optimistic Optimization [35]

```
1: input: Budget: $\Lambda$, parameter: $\sigma$, bias $\zeta(.)$, cost $\lambda(.)$, smoothness params $\nu > 0$ and $\rho \in (0,1)$
2: $tree = \{(0,1)\}$, $B_{1,1} = B_{1,2} = \infty$, $C = 0$
3: while $C \leq \Lambda$ do
4:     $(path, (hnew, inew)) \leftarrow Traverse(tree)$
5:     choose $x \in \mathcal{P}_{hnew,inew}$
6:     query $x$ at fidelity $z_{hnew} = \zeta^{-1}(\nu\rho^{hnew})$ and get observation $y$
7:     $C \leftarrow C + \lambda(z_{hnew})$
8:     $tree.Insert((hnew, inew))$
9:     for all $(h,i) \in path$ do
10:        $count_{h,i} \leftarrow count_{h,i} + 1$
11:        $\hat{f}_{h,i} \leftarrow (1 - 1/count_{h,i})\hat{f}_{h,i} + y/count_{h,i}$
12:    $B_{hnew+1,2inew-1} \leftarrow +\infty$, $B_{hnew+1,2inew} \leftarrow +\infty$
13:    for all $(h,i) \in tree$ do from leaves up to root:
14:        $U_{h,i} \leftarrow \hat{f}_{h,i} + sqrt(2\sigma^2 \ln n/count_{h,i}) + \nu\rho^h + \zeta(z_h)$
15:        $B_{h,i} \leftarrow \min\{U_{h,i}, \max\{B_{h+1,2i+1}, B_{h+1,2i}\}\}$
16: return $\underset{(h,i)\in tree}{\text{argmax}}\ B_{h,i}$
```

We aim to solve the statistical model checking problem of maximizing $p_{k,\mathcal{U}}(x)$ of Equation (2) for a given NiMC $\mathcal{M}$ and a time horizon k, using MFHOO. In order to apply the MFHOO algorithm, one has to make several critical choices regarding the objective function, the budget, the cost, the parameters for fidelity and smoothness, and the multi-fidelity bias function. In this section we discuss the rationale behind our choices.

3.2 Objective function, budget, cost, and fidelity

Fidelity parameter z. Consider a NiMC $\mathcal{M} = (\mathcal{X}, \Theta, P)$ with the unsafe set $\mathcal{U} \subseteq \mathcal{X}$. We have to maximize $p_{k_{max},\mathcal{U}}(x)$ over all initial states $x \in \Theta$, and for a long time horizon k_{max}. Given $x \in \Theta$, the fidelity of evaluating $p_{k_{max},\mathcal{U}}(x)$ will depend on the actual length of the simulations drawn for creating the observation y for the state x. Suppose we fix k_{min} as the shortest simulation to be used. We define the fidelity of an observation (or evaluation) with simulations of length $k \in [k_{min}, k_{max}]$ as $z = (k - k_{min})/(k_{min} - k_{max})$.

Objective function f and observations. A natural choice for the objective function would be to define $f_{z=1}(x) := p_{k_{max},\mathcal{U}}(x)$, for any initial state $x \in \Theta$. Computing this probability, however, is infeasible when the probability transition function $P_{\mathcal{M}}$ is unknown. Even if $P_{\mathcal{M}}$ is known, calculating $p_{k,\mathcal{U}}(x)$ involves summing over many executions (as in (1)). Instead, we take advantage of the fact that MFHOO can work with noisy observations. For any initial $x \in \Theta$, and execution $\alpha \in \text{Execs}_x(k)$ we define the observation:

$$Y = 1 \text{ if } \alpha \in \text{Execs}_x(k, \mathcal{U}), \ \ \textit{and} \ \ = 0 \ \textit{otherwise}. \tag{3}$$

Recall that for a fixed initial states x, $\mathcal{M}$ is a Markov chain and defines a probability distribution over the set of executions $\text{Execs}_x(k)$ as given by Equation (1). Thus, given an initial state x, $Y = 1$ with probability $p_{k,\mathcal{U}}(x)$, and $Y = 0$ with probability $1 - p_{k,\mathcal{U}}(x)$. That is, Y is a Bernoulli random variable with mean $p_{k,\mathcal{U}}(x)$ at fidelity z. In MFHOO, once an initial state $x \in \mathcal{P}_{hnew,inew}$ is chosen (line 5), we simulate $\mathcal{M}$ upto k steps several times starting from x and calculate the empirical mean of Y, which serves as the noisy observation y at fidelity z.

4 HooVer tool and experimental evaluation

We have implemented a prototype tool called HooVer which uses MFHOO for solving the SMC problem of (2). We compare the performance of HooVer with that of Prism [39], Storm [40], and PlasmaLab [41] on several benchmarks we have created.

4.1 Benchmark models

We have created several NiMC models for evaluation of probabilistic and statistical model checking tools. The benchmarks are variants of SLplatoon2. The executable models are available from [42].

SLplatoon3 models a platoon of m cars where in every time step, each car can choose to move with one of *three* speeds: vbrake, vcruise, and vspeedup. The first vehicle always moves at some constant speed. For all the others, these three actions are chosen probabilistically, according to probability distributions that depend on their distance to the preceding vehicle. For example, if the distance $s_{i-1} - s_i$ is less than a threshold constant th_close then the speed is chosen according to a probability distribution pclose.

The state variables of the model are defined as follows: for the i^{th} car, we denote $s_i \in \mathbb{R}$ as the position along the lane. With out loss of generality, we assume $s_1 > s_2 > \ldots > s_m$. We also define an auxiliary variable gap_i for all $i > 1$ as the distance to the preceding car, i.e. $gap_i = s_{i-1} - s_i$. The set of initial states and the unsafe set have the same definition as in SLplatoon2.

The constants in the model are defined as follows: th_far and th_close are some distance thresholds; vbrake, vcruise and vspeedup are some velocities; pclose, pfine and pfar are probability

```
if $i == 1$
    $s_i \leftarrow s_i + \text{vcruise}$;
    return
else
 if $gap_i < \text{th\_close}$
    $p = \text{pclose}$
 else if $gap_i < \text{th\_far}$
    $p = \text{pfine}$
 else
    $p = \text{pfar}$

$s_i \leftarrow s_i + \text{vbrake}$; $w.p.\ p[brake]$
$s_i \leftarrow s_i + \text{vcruise}$; $w.p.\ p[cruise]$
$s_i \leftarrow s_i + \text{vspeedup}$; $w.p.\ p[speedup]$
```

Figure 2: Model of a platoon of cars on a single lane trying not to collide

distributions for different modes. For example, pclose is the probability distribution over three actions, "brake", "cruise" and "speed up", and we denote $\Pr(v = \text{vbrake}) = \text{pclose}[brake]$ as the probability of choosing action "brake" when this probability distribution is used.

With these variables, the behavior of each car at every time step is described in Fig. 2.

MLplatoon models a platoon of m cars on ℓ lanes where in every time step, each car can choose to move with one of *five* actions: moving forward with speed vbrake, vcruise or vspeedup, or chaging to the left or right lane. These actions are chosen probabilistically, according to probability distributions that depend on their distances to the vehicles on its current lane, left lane or the right lane.

The state variables of the model are defined as follows: for the i^{th} car, we denote $s_i \in \mathbb{R}$ as the position along the lane and $y_i \in \{1, \ldots, \ell\}$ as the ID of the current lane. Then, we define some auxiliary variables that can be derived from the state variables. We denote $d\_ahead[i]$ as the distance to the preceding car on the same lane. If the i^{th} car is the leading car on its current lane, then $d\_ahead[i] = \infty$. Then, we denote $d\_left[i]$ as the minimal s-distance (i.e. only considering the difference of the s variables) to the cars on the left lane. If there is no car on the left lane, then $d\_left[i] = \infty$. If the i^{th} car is on the left-most lane, i.e. $y_i = 1$, then $d\_left[i] = -\infty$. Similarly, we define $d\_right[i]$.

The definition of the initial state space of this model is a little bit different due to the y variables. When choosing the initial state, all the y variables are set to 1, i.e. all cars start from the left-most lane. Then, $(s_1, \ldots, s_m)$ is picked from a rectangle in $\mathbb{R}^n$ just as what we have done in SLplatoon2 and SLplatoon3. Finally, we define the unsafe set $\mathcal{U} = \{(s_1, \ldots, s_m) \mid \exists i \in \{1, \ldots, m\}, d\_ahead[i] < \text{unsafe\_rule}\}$.

The constants in the model are defined as follows: th_far, th_close and th_clear are some thresholds; pturn is the probability of chaning to the target lane if allowed to do that; vbrake, vcruise, vspeedup, pclose, pfine and pfar are defined with the same meaning as in SLplatoon3.

With these variables, the behavior of each car at every time step is described in Fig. 3.

4.2 HooVer implementation and metrics

Our implementation of the HooVer tool uses the MFHOO implementation presented in [35] to solve the model checking problem of Equation (2). It works in two stages: First, it uses MFHOO to find the best partition $\mathcal{P}_{h,i}$ and a putative "best" (most unsafe) initial point $x \in \mathcal{P}_{h,i}$ with the maximum estimate for the probability of hitting the unsafe set $\mathcal{U}$. In the second stage, HooVer uses additional simulations to do a Monte Carlo estimation of the probability $p_{k,\mathcal{U}}(x)$. In the experiments reported

```
if $d\_left[i] > \mathsf{th\_clear}$
    $p\_l = \mathsf{pturn}$
else
    $p\_l = 0$

if $d\_right[i] > \mathsf{th\_clear}$
    $p\_r = \mathsf{pturn}$
else
    $p\_r = 0$

$p\_keep = (1 - p\_l - p\_r)$

if $d\_ahead[i] < \mathsf{th\_close}$
    $p\_x = \mathsf{pclose}$
else if $d\_ahead[i] < \mathsf{th\_far}$
    $p\_x = \mathsf{pfine}$
else
    $p\_x = \mathsf{pfar}$

$s_i \leftarrow s_i + \mathsf{vcruise};\ y_i \leftarrow y_i - 1;\ w.p.\ p\_l$
$s_i \leftarrow s_i + \mathsf{vcruise};\ y_i \leftarrow y_i + 1;\ w.p.\ p\_r$
$s_i \leftarrow s_i + \mathsf{vbrake};\ y_i \leftarrow y_i;\ w.p.\ p\_keep * p\_x[brake]$
$s_i \leftarrow s_i + \mathsf{vcruise};\ y_i \leftarrow y_i;\ w.p.\ p\_keep * p\_x[cruise]$
$s_i \leftarrow s_i + \mathsf{vspeedup};\ y_i \leftarrow y_i;\ w.p.\ p\_keep * p\_x[speedup]$
```

Figure 3: Model of a platoon of cars on multiple lanes trying not to collide

below, a constant number of 26K simulations are used in all experiments in the second stage.

To achieve the theoretically optimal performance, MFHOO requires the smoothness parameters ρ and ν which are unknown for our benchmarks. To circumvent this HooVer chooses several parameter configurations (3 sets in our experiments), runs an instance of MFHOO for each, and returns the result with the highest hitting probability. For each instance of MFHOO, we set a time budget which is the maximum time allowed to be consumed by the simulator.

Metrics. We report the regret of HooVer. In order to calculate the regret, first we have to calculate the actual maximum hitting probability for each benchmark. This is computed using Prism [39] which uses numerical and symbolic analysis. The regret is the difference between the exact probability and the estimated probability. Then, we report the running time and memory usage. The memory usage is measured by calculating the total size of the Python object which contains the constructed tree and all other data of MFHOO. We also report the number of *queries* for each method, which is total number of simulations used in both stage 1 and 2. All the experiments are conducted on a Linux workstation with two Xeon Silver 4110 CPUs and 32 GB RAM.

Table 1 shows the running time, the memory usage, the number of queries, and the resulting actual regret for HooVer using MFHOO as well as HooVer(1) using HOO. On every benchmark, HooVer gives low regrets. With the same simulation budget, HooVer devotes longer simulations in the interesting parts Θ, as a consequence, it is usually faster than HooVer(1) as shown in Figure 4.

4.3 Comparison with other model checkers

We compare the performance of HooVer with other model checkers. Prism [39] and Storm [40] are leading probabilistic model checkers for Markov chains and MDPs and compute exact probability of reaching the unsafe states. As Storm has the same functionality as Prism and we found it to be much more efficient than Prism in all our experiments, we only compare HooVer with Storm. PlasmaLab [41] is one of the few *statistical* model checkers that can handle MDPs. For probability estimation problems, PlasmaLab uses smart sampling algorithm [43] to efficiently assign the simulation budgets to each scheduler and then estimates the probability for the putative “best” scheduler.

Model	Method	Time(s)	Memory(MB)	#queries	Regret
SLplatoon2	Storm	68.88	1019	NA	0
$m = 2$	PlasmaLab	37.73	NA	749711	0.0017 ± 0.0020
$p_{k_{max},\mathcal{U}}(x^*) = 0.8800$	HooVer(1)	59.37	9.13	31381	0.0011 ± 0.0025
	HooVer	24.03	6.23	**29415**	$\mathbf{0.0002 \pm 0.0020}$
SLplatoon3	Storm	89.10	1974	NA	0
$m = 2$	PlasmaLab	22.61	NA	749711	0.0022 ± 0.0021
$p_{k_{max},\mathcal{U}}(x^*) = 0.8727$	HooVer(1)	44.52	7.30	30315	0.0038 ± 0.0113
	HooVer	26.46	4.53	**28520**	$\mathbf{0.0010 \pm 0.0018}$
MLplatoon	Storm	NA	OOM	NA	NA
$m = 2,\ \ell = 2$	PlasmaLab	49.14	NA	749711	0.0032 ± 0.0026
$p_{k_{max},\mathcal{U}}(x^*) = 0.5918$	HooVer(1)	110.64	9.37	31555	0.0016 ± 0.0025
	HooVer	76.35	5.27	**28955**	$\mathbf{0.0007 \pm 0.0043}$

Table 1: Comparison of HooVer, Storm and PlasmaLab. HooVer(1) corresponds to using HOO (i.e. the full fidelity algorithm) in stage 1 of HooVer. For regret of HooVer and PlasmaLab, we run the experiments for 10 times and report the "mean ± std". For all the experiments, $|\overline{\Theta}| = 4096$.

Discretizing and scaling the benchmarks. The theory for MFHOO is based on a continuous state spaces, however, most model checking tools, including the ones mentioned above, are designed for discrete state space models and they do not support floats as state variables. Therefore, a direct comparison of the approaches on the same model is infeasible, and we created equivalent, discretized (quantized) versions of the benchmarks.

In HooVer, the algorithm keeps partitioning Θ hierarchically and stops at a depth h, which can be considered as searching over all the 2^h partitions at depth h. In order to make a fair comparison, we make sure that the discrete version of the benchmark has 2^h initial states, i.e. $|\overline{\Theta}| = 2^h$ where $\overline{\Theta}$ is the discretized initial state space.

Before stating the discretizing process, we give two key observations of the benchmarks. First, considering the nature of our benchmarks, it is obvious that if we scale the velocities, distance thresholds and initial distances by a constant factor, the probability of hitting unsafe set doesn't change. Second, taking SLplatoon2 as an example, we set all the constants (velocities and distance thresholds) in the model as integers, which leads to the function $p_{k_{max},\mathcal{U}}(x)$ shown in Figure 1. It's clear that for any state $x \in \mathcal{X}$, there exists an integer state $\overline{x}$ such that $p_{k_{max},\mathcal{U}}(x) = p_{k_{max},\mathcal{U}}(\overline{x})$. If we make sure that for each integer interval in the original continuous space, the discretized space has an value in that interval, then the maximum probability doesn't change. With these observations, we discretize the benchmark as follows. First, we sample 2^h points uniformly in the continuous initial state space. Due to the second observation mentioned above, if 2^h is larger than the number of integer points in the original space, the maximum probability doesn't change. Then, we find an integer scaling factor c such that by multiplying c all the 2^h points become integer points. Other constants in the model are also multiplied by c. According to the first observation, scaling with a constant doesn't change the probability. Thus, we now have a model where all state variables are integers and it has the same maximum probability as the original model. Then we evaluate Storm and PlasmaLab on this new model.

All of the model checkers mentioned above support the Prism [39] language. Thus, we implement each benchmark in Prism language, and then we check the equivalence between the Prism implementation and the Python implementation by calculating and comparing the probability $p_{k_{max},\mathcal{U}}(x)$ at all integer points x.

For Storm, we report the running time and the memory usage. These metrics are directly measured by the software itself. For PlasmaLab, we report the running time. We do not report the memory usage of PlasmaLab because it doesn't provide an interface for that and it is hard to track the actual memory used by the algorithm inside the JAVA virtual machine. We also report the regret for PlasmaLab. We use the term "regret" here just for simplicity, which also refers to the difference between the estimated probability and the exact probability.

The results of HooVer, PlasmaLab and Storm are summarized in Table 1. We show in Figure 4 how the performance of different methods changes as the $|\overline{\Theta}|$ increases. As the size of the initial state space increases, the memory usage of Storm grows quickly, which limits its application on large models. In contrast, HooVer and PlasmaLab scale well. The running time of PlasmaLab is determined by the parameters of the smart sampling [43] algorithm. We use the same parameters regardless of $|\overline{\Theta}|$, and thus the running time of PlasmaLab is almost constant.

As shown in Table 1, Storm fails to check the MLplatoon model due to the memory limitation. Compared with PlasmaLab, HooVer requires more running time. However, we note that PlasmaLab is a considerably more mature tool than HooVer. As shown in Table 1, compared to PlasmaLab, HooVer requires much fewer queries to reach comparable regrets, which attests to the sample efficiency of our proposed method.

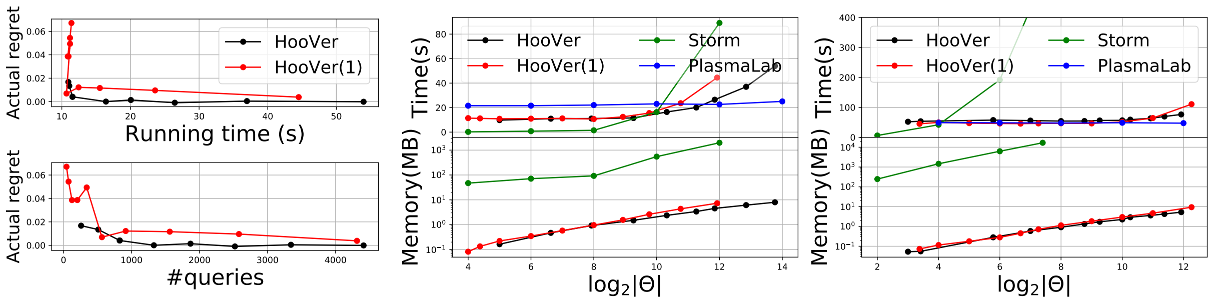

Figure 4: *Left*: Comparison of HooVer and HooVer(1) on SLplatoon3. (Here, we only consider the number of queries in stage 1, because that for stage 2 is just a constant.) *Middle*: Performance of different methods as the $|\overline{\Theta}|$ increases for SLplatoon3. *Right*: Performance of different methods as the $|\overline{\Theta}|$ increases for MLplatoon.

5 Conclusions

In this paper, we formulated the statistical model checking problem for a special type of MDP models as a multi-armed bandit problem and showed how a hierarchical optimistic optimization algorithm from [35] can be used to solve it. In the process, we modified the existing definition of near-optimality dimension to accommodate the non-smoothness of the typical functions we have to optimize and recovered the regret bounds of the algorithm. We created several benchmarks, developed a SMC tool HooVer, and experimentally established the sample efficiency and scalability of the method.

In order to enlarge the area of application of this method, it is necessary to study more general temporal properties for NiMC beyond bounded time safety. In this paper, we focused on a very special type of MDP models with nondeterminism only in the initial set. The results suggest that it will be interesting to study black-box optimization algorithms for model checking more general MDP models.

References

[1] William R Thompson. On the theory of apportionment. *American Journal of Mathematics*, 57(2):450–456, 1935.

[2] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[3] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[4] Rémi Munos. From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning. Technical report, 2014. 130 pages.

[5] Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. X-armed bandits. *J. Mach. Learn. Res.*, 12:1655–1695, July 2011.

[6] Sébastien Bubeck and Nicolò Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.

[7] Håkan L. S. Younes and Reid G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *International Conference on Computer Aided Verification (CAV2002)*, pages 223–235. Springer, 2002.

[8] Daniel Gburek and Christel Baier. Bisimulations, logics, and trace distributions for stochastic systems with rewards. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC 2018, Porto, Portugal, April 11-13, 2018*, pages 31–40, 2018.

[9] Doron Bustan, Sasha Rubin, and Moshe Y. Vardi. Verifying omega-regular properties of markov chains. In *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*, pages 189–201, 2004.

[10] Holger Hermanns, Björn Wachter, and Lijun Zhang. Probabilistic CEGAR. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, pages 162–175, 2008.

[11] David N. Jansen, Joost-Pieter Katoen, Marcel Oldenkamp, Mariëlle Stoelinga, and Ivan S. Zapreev. How fast and fat is your probabilistic model checker? an experimental performance comparison. In *Hardware and Software: Verification and Testing, Third International Haifa Verification Conference, HVC 2007, Haifa, Israel, October 23-25, 2007, Proceedings*, pages 69–85, 2007.

[12] Jan J. M. M. Rutten, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Prakash Panangaden. *Mathematical techniques for analyzing concurrent and probabilistic systems*, volume 23 of *CRM monograph series*. American Mathematical Society, 2004.

[13] Koushik Sen, Mahesh Viswanathan, and Gul Agha. On statistical model checking of stochastic systems. In *Proceedings of the 17th International Conference on Computer Aided Verification*, CAV'05, pages 266–280, Berlin, Heidelberg, 2005. Springer-Verlag.

[14] Håkan L. S. Younes. Probabilistic verification for "black-box" systems. In *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, pages 253–265, 2005.

[15] Ymer. 2015. ymer tool. Retrieved from `http://www.tempastic.org/ymer/`.

[16] Koushik Sen, Mahesh Viswanathan, and Gul A. Agha. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *QEST*, pages 251–252, 2005.

[17] Multivesta. 2015. multivesta tool. Retrieved from `http://sysma.imtlucca.it/tools/multivesta/`.

[18] Benoît Boyer, Kevin Corre, Axel Legay, and Sean Sedwards. PLASMA-lab: A flexible, distributable statistical model checking library. In *Proceedings of the 10th International Conference on Quantitative Evaluation of Systems*, QEST'13, pages 160–164, Berlin, Heidelberg, 2013. Springer-Verlag.

[19] Modes. 2006. modes tool. Retrieved from `http://www.modestchecker.net`.

[20] Uppaal 2015. uppaal. Retrieved from `http://www.uppaal.org`.

[21] Mrmc. 2011. mrmc tool. Retrieved from `http://www.mrmc-tool.org`.

[22] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, Jonas van Vliet, and Zheng Wang. Statistical model checking for networks of priced timed automata. In *Formal Modeling and Analysis of Timed Systems - 9th International Conference, FORMATS 2011, Aalborg, Denmark, September 21-23, 2011. Proceedings*, pages 80–96, 2011.

[23] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Automated performance and dependability evaluation using model checking. In *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures*, pages 261–289, 2002.

[24] Sumit Kumar Jha, Edmund M. Clarke, Christopher James Langmead, Axel Legay, André Platzer, and Paolo Zuliani. A bayesian approach to model checking biological systems. In *Computational Methods in Systems Biology, 7th International Conference, CMSB 2009, Bologna, Italy, August 31-September 1, 2009. Proceedings*, pages 218–234, 2009.

[25] David Kyle, Jeffery P. Hansen, and Sagar Chaki. Statistical model checking of distributed adaptive real-time software. In *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, pages 269–274, 2015.

[26] Sucheendra K. Palaniappan, Benjamin M. Gyori, Bing Liu, David Hsu, and P. S. Thiagarajan. Statistical model checking based calibration and analysis of bio-pathway models. In *Computational Methods in Systems Biology - 11th International Conference, CMSB 2013, Klosterneuburg, Austria, September 22-24, 2013. Proceedings*, pages 120–134, 2013.

[27] Paolo Zuliani, André Platzer, and Edmund M. Clarke. Bayesian statistical model checking with application to stateflow/simulink verification. *Formal Methods in System Design*, 43(2):338–367, 2013.

[28] José Meseguer and Raman Sharykin. Specification and analysis of distributed object-based stochastic hybrid systems. In *Proceedings of the 9th International Conference on Hybrid Systems: Computation and Control*, HSCC'06, pages 460–475, Berlin, Heidelberg, 2006. Springer-Verlag.

[29] João Martins, André Platzer, and João Leite. Statistical model checking for distributed probabilistic-control hybrid automata with smart grid applications. In *Proceedings of the 13th International Conference on Formal Methods and Software Engineering*, ICFEM'11, pages 131–146, Berlin, Heidelberg, 2011. Springer-Verlag.

[30] Richard Lassaigne and Sylvain Peyronnet. Approximate planning and verification for large markov decision processes. *International Journal on Software Tools for Technology Transfer*, 17(4):457–467, 2015.

[31] Arnd Hartmanns and Holger Hermanns. The modest toolset: An integrated environment for quantitative modelling and verification. In *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, pages 593–598, 2014.

[32] David Henriques, Joao G Martins, Paolo Zuliani, André Platzer, and Edmund M Clarke. Statistical model checking for markov decision processes. In *2012 Ninth International Conference on Quantitative Evaluation of Systems*, pages 84–93. IEEE, 2012.

[33] Gul Agha and Karl Palmskog. A survey of statistical model checking. *ACM Trans. Model. Comput. Simul.*, 28(1):6.1–6.39, January 2018.

[34] Axel Legay and Mahesh Viswanathan. Statistical model checking: Challenges and perspectives. *Int. J. Softw. Tools Technol. Transf.*, 17(4):369–376, August 2015.

[35] Rajat Sen, Kirthevasan Kandasamy, and Sanjay Shakkottai. Noisy blackbox optimization using multi-fidelity queries: A tree search approach. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2096–2105, 2019.

[36] Simone Fabris. Method for hazard severity assessment for the case of undemanded deceleration. *TRW Automotive, Berlin*, 2012.

[37] Chuchu Fan, Bolun Qi, and Sayan Mitra. Data-driven formal reasoning and their applications in safety analysis of vehicle autonomy features. *IEEE Design & Test*, 35(3):31–38, 2018.

[38] ISO. ISO 26262:road vehicles – functional safety. Norm ISO 26262, 2011.

[39] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[40] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *International Conference on Computer Aided Verification*, pages 592–600. Springer, 2017.

[41] Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Plasma lab: a modular statistical model checking platform. In *International Symposium on Leveraging Applications of Formal Methods*, pages 77–93. Springer, 2016.

[42] Negin Musavi, Dawei Sun, Sayan Mitra, Geir Dullerud, and Sanjay Shakkottai. Hoover: a statistical model checking tool with optimistic optimization, October 2019. Available from https://github.com/sundw2014/HooVer.

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

[44] Rémi Munos et al. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. *Foundations and Trends® in Machine Learning*, 7(1):1–129, 2014.

Appendix A Derivation of near-optimality dimension from Taylor expansion

Near-optimality dimension is widely used in the proofs of the error bound of optimistic optimization methods [44]. In this section, we give a simple method to calculate the given the Taylor expansion at the optima.

A.1 1-D case

Theorem 1. Considering a semi-metric $\ell(x,y) = ||x-y||_2^\beta$, a function $f : \mathbb{R} \mapsto \mathbb{R}$, the minimizer $x^* = \arg\min_{x\in\mathbb{R}} f(x)$, and the Taylor expansion of f at x^*, $f(x) = f(x^*) + \sum_{i=1}^{\infty} c_i(x-x^*)^i$, if there exists an even number $\alpha \geq 2$ such that $c_\alpha \neq 0$ and $c_i = 0$, $\forall i < \alpha$, then the near-optimality of f is $d = \frac{1}{\beta} - \frac{1}{\alpha}$.

Proof. We have $f(x) - f(x^*) = c_\alpha(x-x^*)^\alpha + o((x-x^*)^\alpha)$. If ϵ is small enough, equation $f(x) - f(x^*) = \epsilon$ has two real roots $x^- \in (-\infty, x^*)$ and $x^+ \in (x^*, \infty)$, and $\mathcal{X}_\epsilon = [x^-, x^+]$. Because $c_\alpha(x^+ - x^*)^\alpha + o((x^+ - x^*)^\alpha) = \epsilon$, we have $\lim_{\epsilon\downarrow 0}\left[c_\alpha\frac{(x^+-x^*)^\alpha}{\epsilon} + \frac{o((x^+-x^*)^\alpha)}{\epsilon}\right] = 1$, and thus we must have $\lim_{\epsilon\downarrow 0}\frac{(x^+-x^*)^\alpha}{\epsilon}$ exists and is not zero. Therefore, we have $\lim_{\epsilon\downarrow 0}\frac{x^+-x^*}{\epsilon^{1/\alpha}} = C_1 \neq 0$ Similarly, we have $\lim_{\epsilon\downarrow 0}\frac{x^*-x^-}{\epsilon^{1/\alpha}} = C_2 \neq 0$. Then the near-optimality dimension is $d = \lim_{\epsilon\downarrow 0}\frac{\ln\frac{(x^+-x^*)+(x^*-x^-)}{\epsilon^{1/\beta}}}{\ln\epsilon^{-1}} = \lim_{\epsilon\downarrow 0}\frac{\ln\frac{(x^+-x^*)+(x^*-x^-)}{\epsilon^{1/\alpha}} + \ln\epsilon^{1/\alpha-1/\beta}}{\ln\epsilon^{-1}} = \frac{1}{\beta} - \frac{1}{\alpha}$. □

A.2 N-D case

Proposition 1. Considering a semi-metric $\ell(x,y) = ||x-y||_2^\beta$, a function $f : \mathbb{R}^D \mapsto \mathbb{R}$, the minimizer $x^* = \arg\min_{x\in\mathbb{R}^D} f(x)$, if the Taylor expansion of f at x^* is dominated by the α-order term, then the near-optimality of f is $d = D(\frac{1}{\beta} - \frac{1}{\alpha})$.

Appendix B Usage of HooVer

The code is available at https://github.com/sundw2014/HooVer.The MFTreeSearchCV code base was developed by Rajat Sen (https://github.com/rajatsen91/MFTreeSearchCV) which in-turn was built on the blackbox optimization code base of Kirthivasan Kandasamy (https://github.com/kirthevasank).

To run the existed benchmarks, run command like this:

```
python example.py --model [Slplatoon2/Slplatoon3/Mlplatoon]\
--budget [time budget for simulator] --numRuns [number of runs] [--useHOO]
```

If ones want to add new benchmarks, they can create their own models following the interface used in models/Slplatoon2.py. For the Python implementation, the things that have to be defined include initial state space, step-forward simulation function, and the unsafe set.