

# Model Repair for Markov Decision Processes

Taolue Chen<sup>1</sup>, Ernst Moritz Hahn<sup>1</sup>, Tingting Han<sup>1</sup>, Marta Kwiatkowska<sup>1</sup>, Hongyang Qu<sup>2</sup> and Lijun Zhang<sup>3</sup>

<sup>1</sup> Department of Computer Science, University of Oxford

<sup>2</sup> Department of Automatic Control and Systems Engineering, University of Sheffield

<sup>3</sup> State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences

**Abstract**—Markov decision processes (MDPs) are often used for modelling distributed systems with probabilistic failure or randomisation. We consider the problem of *model repair* for MDPs defined as follows: if the MDP fails to satisfy a property, we aim to find new values for the transition probabilities so that the property is guaranteed to hold, while at the same time the cost of repair is minimised. Because solving the MDP repair problem exactly is infeasible, in this paper we focus on approximate solution methods. We first formulate a region-based approach, which yields an interval in which the minimal repair cost is contained. As an alternative, we also consider sampling-based approaches, which are faster but unable to provide lower bounds on the repair cost. We have integrated both methods into the probabilistic model checker PRISM and demonstrated their usefulness in practice using a computer virus case study.

## I. Introduction

Distributed randomised protocols, for example wireless communication protocols, are naturally modelled using Markov decision processes (MDPs), which combine nondeterminism, needed to express concurrency, with probabilistic behaviour, such as communication failures or random back off. When used in formal verification, their correctness properties are specified using the temporal logic PCTL, which can express properties such as “the minimum probability of reaching an unsafe state is below 0.01”, and “the expected number of steps to completion is not greater than 10”. The PCTL model checking problem for Markov decision processes is to decide whether a given state satisfies such a property [1], which reduces to the computation of the minimum/maximum probability or expectation to reach a certain set of states.

An important question is what to do if the property is not satisfied. Similarly to the classical transition systems setting, *counterexamples* that witness the violation of the property have been proposed. Various algorithms for computing probabilistic counterexamples have been studied [2]. However, counterexamples for probabilistic models are often very involved and computationally expensive, and have thus far found limited application.

The probabilities in MDP models are often not precise, and are determined by characteristics of network connections, or refined later in the design stage. Over the past years, the *parametric* model checking problem has been formulated and studied for Markov chains with probabilities as parameters [3], [4]. Instead of computing the probability value, a rational function over the parameters is obtained. Then, given a property, one can identify parameter valuations under which the property is guaranteed to hold. Recently, based on the parametric model checking approach, the *model repair problem* for Markov chains has been introduced [5], where a repair means a change

in the transition probabilities. The input of the model repair problem is a *controllable* Markov chain, namely, a Markov chain with a set of transition probabilities that can be *repaired* with certain costs, and a PCTL property. Assuming that the property is not satisfied, then the aim is to repair the input model so that the modified model satisfies the desired property, while keeping the repair cost minimal. The possible changes to probabilities can be expressed by adding parameters to the original model, thus giving rise to a parametric Markov chain in the sense of [4]. By exploiting the rational function obtained for the parametric Markov chain, the model repair problem is reduced to a nonlinear optimisation problem. The approach exploits existing efficient solvers for optimisation problems, and turns out to be very attractive as a refinement-based design technique.

In this paper, we consider an extension of the model repair problem to Markov decision processes. As for Markov chains, if the property is violated, we aim to find new values for transition probabilities so that the property holds and the cost of the repair is minimal, that is, the repaired MDP is as close to the input one as possible. Unfortunately, we cannot directly apply the repair technique for Markov chains, since for parametric MDPs we cannot guarantee that a single rational function over the parameters can be obtained. In addition, the exact values of optimal repairs might not be representable using rational numbers, and hence it might be impossible to compute them precisely. Because of this, we consider approximate solutions of this problem.

We consider two complementary approaches. The first variant is a *region-based approach*, which builds on our work on parametric MDPs [6] (see also [7] for a related approach). A region represents a set of parameter valuations. Given a division of the parameter space into regions, we can compute both the lower and upper bounds on optimal parameter values by evaluating the edge points of the regions. If the result is too imprecise, some regions might have to be split. The second variant [8] involves three alternative approaches based on *sampling-based methods*. The main idea here is to define a probability density on the parameter space, and then use stochastic search to find a good sample point, corresponding to a low repair cost. To enforce that a point is chosen in which the model is actually repaired, we use a penalty value which is added to the repair cost if this is not the case.

All methods have been integrated into the probabilistic model checker PRISM [9]. The sampling methods are faster than the region approach, but are unable to compute lower bounds for the minimal repair cost. To show the practicality of our methods, we have successfully applied them to a case study that repairs the behaviour of a virus infecting a network.

## II. Preliminaries

We begin by introducing notation and definitions needed to state the model repair problem and its solution precisely. As a notational convention, we use the vector notation  $\vec{v} = (v_1, \dots, v_n)$  to denote a sequence of variables, constants, etc.

### A. Markov Decision Processes (MDPs)

In this section we define the model used in this paper. Throughout, we assume a fixed set of atomic propositions  $AP$ .

*Definition 1:* A Markov decision process (MDP) is defined as a tuple  $\mathcal{M} = (S, s_0, Act, \mathbf{P}, \mathbf{L}, \mathbf{r})$  where

- $S$  is a finite set of states,
- $s_0 \in S$  is the initial state,
- $Act$  is a finite set of actions,
- $\mathbf{P}: S \times Act \times S \rightarrow [0, 1]$  is the probability matrix,
- $\mathbf{L}: S \rightarrow 2^{AP}$  is a state labelling function, mapping states to a subset of the atomic propositions, and
- $\mathbf{r}: S \times Act \rightarrow \mathbb{R}_{\geq 0}$  is a reward structure.

We require  $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{0, 1\}$  for all states  $s \in S$  and actions  $\alpha \in Act$ . Furthermore, for each  $s \in S$  there is at least one  $\alpha \in Act$  with  $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1$ . We use  $Act(s) = \{\alpha \mid \sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1\}$  to denote the set of enabled actions of the state  $s$ .

The nondeterministic choices are resolved by schedulers. When in state  $s$ , a scheduler can choose any action  $\alpha$  which is enabled. Different types of schedulers exist, for example memoryless, but we do not need to consider them in detail.

Rewards can be interpreted as either costs or bonuses, depending on the model under consideration. If a scheduler chooses action  $\alpha$  in state  $s$ , a reward of  $\mathbf{r}(s, \alpha)$  is obtained.

The behaviour of an MDP is as follows. Starting in the initial state  $s_0$ , the scheduler selects an action  $\alpha_0 \in Act(s_0)$ . A reward of  $\mathbf{r}(s_0, \alpha_0)$  is obtained, and a probabilistic choice of successor states is made, where some successor state, say  $s_1 \in S$ , is selected with probability  $\mathbf{P}(s_0, \alpha_0, s_1)$ . Afterwards, the scheduler repeats the process from  $s_1$ , and so forth.

### B. Parametric MDPs (PMDPs)

To state the model repair problem, we define a parametric variant of MDPs. We fix  $V \stackrel{\text{def}}{=} \{x_1, \dots, x_n\}$  as the set of variables with domain  $\mathbb{R}$ . An evaluation  $v$  is a function  $v: V \rightarrow \mathbb{R}$ . A polynomial  $g$  over  $V$  is a sum of monomials

$$g(\vec{x}) = \sum_{i_1, \dots, i_n} a_{i_1, \dots, i_n} x_1^{i_1} \cdots x_n^{i_n},$$

where each  $i_j \in \mathbb{N}$  and each  $a_{i_1, \dots, i_n} \in \mathbb{R}$ . A rational function  $f$  over a set of variables  $V$  is a fraction  $f(\vec{x}) = \frac{g_1(\vec{x})}{g_2(\vec{x})}$  of two polynomials  $g_1, g_2$  over  $V$ . Let  $\mathcal{F}_V$  denote the set of rational functions from  $V$  to  $\mathbb{R}$ . Given  $f \in \mathcal{F}_V$  and an evaluation  $v$ , we let  $f\langle v \rangle \stackrel{\text{def}}{=} f(v(x_1), \dots, v(x_n))$  denote the rational number obtained by substituting each occurrence of  $x_i$  with  $v(x_i)$ .

*Definition 2:* A parametric Markov decision process (PMDP) is a tuple  $\mathcal{M} = (S, s_0, Act, \mathbf{P}, \mathbf{L}, \mathbf{r})$ , where  $S, s_0, Act$ , and  $\mathbf{L}$  are the same as in Definition 1,

- $\mathbf{P}: S \times Act \times S \rightarrow \mathcal{F}_V$  is the parametric probability matrix, and
- $\mathbf{r}: S \times Act \rightarrow \mathcal{F}_V$  is the parametric reward structure.

For a given evaluation, a PMDP induces a (nonparametric) MDP.

*Definition 3:* Given a PMDP  $\mathcal{M} = (S, s_0, Act, \mathbf{P}, \mathbf{L}, \mathbf{r})$  and an evaluation  $v$ , the MDP induced by  $v$  is defined by  $\mathcal{M}_v \stackrel{\text{def}}{=} (S, s_0, Act, \mathbf{P}_v, \mathbf{L}, \mathbf{r}_v)$  where for  $s, s' \in S, \alpha \in Act$

- $\mathbf{P}_v(s, \alpha, s') \stackrel{\text{def}}{=} \mathbf{P}(s, \alpha, s')\langle v \rangle$ , and
- $\mathbf{r}_v(s, \alpha) \stackrel{\text{def}}{=} \mathbf{r}(s, \alpha)\langle v \rangle$ .

The set of valid evaluations  $Evals(\mathcal{M})$  is defined as the set of all evaluations  $v$  so that

- $\mathcal{M}_v$  is a valid MDP (cf. Definition 1), and
- for all  $s \in S, \alpha \in Act(s)$  and  $s' \in S$ , we have that either  $\mathbf{P}(s, \alpha, s')\langle v \rangle \neq 0$ , or  $\mathbf{P}(s, \alpha, s')\langle v' \rangle = 0$  for all evaluations  $v'$ .

For an evaluation to be valid, we require that all transition probabilities are between zero and one, and that they sum up to one for valid nondeterministic choices. We assume that rewards are nonnegative. In addition, we require that evaluations do not change the structure of the PMDP, that is, an evaluation is valid only if it does not set a probabilistic choice to zero, unless it is zero for all possible evaluations.

### C. Probabilistic CTL

To specify the properties of the models, we utilise the probabilistic temporal logic PCTL [1]. The syntax is given by:

$$\begin{aligned} \Phi &= \top \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathcal{P}_{\bowtie p}(\varphi) \mid \mathcal{R}_{\bowtie m}(\diamond\Phi), \\ \varphi &= \mathcal{X}\Phi \mid \Phi \mathcal{U}\Phi \mid \Phi \mathcal{U}^{\leq n}\Phi, \end{aligned}$$

where  $\bowtie \in \{<, \leq, \geq, >\}$ ,  $n \in \mathbb{N}$ ,  $p \in [0, 1]$ ,  $m \in \mathbb{R}$  and  $a \in AP$ . Here,  $\Phi$  is a formula which has a boolean value in a state, whereas  $\varphi$  is interpreted on paths.

The definition of  $\top$ ,  $a$  and  $\wedge$  being satisfied in a state is standard. For state  $s$ , the formula  $\mathcal{P}_{\bowtie p}(\varphi)$  is fulfilled if for all schedulers the probability of paths which start in  $s$  and fulfil  $\varphi$  meets the bound  $\bowtie p$ . For  $\bowtie \in \{<, \leq\}$  ( $\bowtie \in \{\geq, >\}$ ), this is equivalent to asking whether the maximal (minimal) probability meets the bound  $\bowtie p$ .

Given a path, the next state formula  $\mathcal{X}\Phi$  asks whether on the second state of this path  $\Phi$  holds. The unbounded until formula  $\Phi_1 \mathcal{U}\Phi_2$  requires that some state on the path fulfils  $\Phi_2$ , and, for all states on the path before that point,  $\Phi_1$  must hold. The bounded until formula  $\Phi_1 \mathcal{U}^{\leq n}\Phi_2$  is similar, but additionally requires that  $\Phi_2$  occurs by the  $n$ th step. We write  $\mathcal{M} \models \Phi$  if the initial state of an MDP fulfils the PCTL state formula  $\Phi$ . The reachability reward formula [10], [11]  $\mathcal{R}_{\bowtie m}(\diamond\Phi)$  states that the expected accumulated reward until a state satisfying  $\Phi$  is reached should meet the bound  $\bowtie m$  for all schedulers.

## III. The Model Repair Problem for MDPs

We are now ready to define the model repair problem for MDPs formally. For  $V = \{x_1, \dots, x_n\}$ , we write  $span(V) =$

$\{w_1x_1 + \dots + w_nx_n \mid \vec{w} \in \mathbb{R}^n\} \subset \mathcal{F}_V$  for the set of linear expressions over  $V$ .

## A. Problem Statement

*Definition 4:* A *controllable MDP* is a tuple  $\tilde{\mathcal{M}} = (\mathcal{M}, \mathbf{Z}, \mathbf{z})$  where

- $\mathcal{M} = (S, s_0, Act, \mathbf{P}, \mathbf{L}, \mathbf{r})$  is an MDP,
- $\mathbf{Z}: S \times Act \times S \rightarrow span(V)$  is a *transition repair matrix*, and
- $\mathbf{z}: S \times Act \rightarrow span(V)$  is a *reward repair matrix*.

We require that, for all  $s, s' \in S$  and  $\alpha \in Act$ , we only have  $\mathbf{Z}(s, \alpha, s') \neq 0$  in case  $\alpha \in Act(s)$ . Moreover, for all  $s \in S$  and  $\alpha \in Act(s)$  we require  $\sum_{s' \in S} \mathbf{Z}(s, \alpha, s') = 0$ .

*Definition 5:* Given

- a controllable MDP  $\tilde{\mathcal{M}} = (\mathcal{M}, \mathbf{Z}, \mathbf{z})$  with  $\mathcal{M} = (S, s_0, \mathbf{P}, Act, \mathbf{L}, \mathbf{r})$ ,
- a PCTL formula  $\varphi$  for which  $\mathcal{M} \not\models \varphi$ , and
- a polynomial  $g = w_1x_1^2 + \dots + w_nx_n^2, \vec{w} \in \mathbb{R}_{>0}^n$ ,

we define the PMDP

$$\mathcal{M}' = (S, s_0, Act, \mathbf{P} + \mathbf{Z}, \mathbf{L}, \mathbf{r} + \mathbf{z}).$$

Then the *model repair problem* is to find an evaluation  $v: V \rightarrow \mathbb{R}$  which satisfies the following constraints:

$$v \in \mathbf{arg\,min} \{g(v)\} \quad (1)$$

$$v \in \mathit{Evals}(\mathcal{M}) \quad (2)$$

$$\mathcal{M}'_v \models \varphi \quad (3)$$

Constraint (1) requires that the repaired model has the probability transition matrix  $\mathbf{P} + \mathbf{Z}$  and rewards  $\mathbf{r} + \mathbf{z}$  that are the closest to  $\mathbf{P}$ , in terms of the weighted distance  $g(v)$ , under the given side conditions. Note that the weights  $w_1, \dots, w_n$  can be chosen so that the importance or priority of certain parameters with respect to others can be expressed. The function  $g$  is always positive, continuous, differentiable, and, for  $\vec{w} = \vec{1}_n$ ,  $g$  is the square of the L2-norm  $\|\vec{x}\|_2^2$ . Constraint (2) requires that the MDP is changed by  $v$  in such a way that it remains an MDP, and that does not delete any transitions from the MDP before the repair. Constraint (3) states that the model is repaired, that is, it now fulfills its PCTL specification.

## B. Fast Model Repair Problem

In practice, many computer systems need to adapt dynamically and predictably to rapid changes in system workload, environment and objectives by altering some parameter values, in order to guarantee certain correctness properties, as well as performance [12]. Typical examples are multi-processor systems and modern distributed storage systems, where components (such as memories or processors) may fail and only lower- and upper-bounds on failure probabilities are known. If certain components are replaced with similar ones, the system should be able to adjust the failure rates to maintain the reliability.

We observe that, in practice, there are many scenarios where we do not need to find the optimal probability value, but

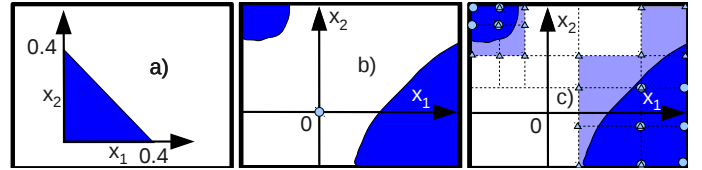


Fig. 1. The procedure of model repair

it suffices to find one such value rapidly. Indeed, finding an optimal repair might not even be possible, as it might correspond to an evaluation which assigns irrational numbers to repair parameters which cannot be represented exactly. Examples where fast repairs are needed include runtime verification of adaptive systems as described in [13].

In this paper we focus on these scenarios. To achieve this, the *fast model repair problem* is to seek an evaluation  $v$  so that the repaired MDP  $\mathcal{M}'_v$  satisfies the constraints (2)-(3) and  $v$  should be obtained as quickly as possible. At the same time, the cost of the repair must be sufficiently low.

## IV. Solving Model Repair Problem for MDPs

In this section, we propose two complementary methods to solve the fast model repair problem. The region-based method focuses on finding a repair with known lower and upper bounds for the repair, whereas the sampling-based methods aim to achieve a faster repair.

### A. Region-based Method

We formulate an approximate solution that involves region refinement through the parameter space, and thus allows one to compute lower and upper bounds for the optimal repair cost. This algorithm builds upon our previous work concerning solving the problem of PCTL model checking for PMDPs [6].

In [6], we assumed that for all parameters we are given a certain *range*, that is, a lower and upper bound for the values of this parameter. Consequently, all parameter valuations respecting these bounds are valid, with finitely many exceptions, and the area of valid evaluations (parameter space) forms a hyper-rectangle. However, for the model repair problem this assumption is no longer valid. For instance, we could have a transition repair matrix which increases two transitions with the initial probability of  $p_1 = p_2 = 0.3$  by  $x_1$  and  $x_2$  respectively, and decreases another transition with the probability of  $p_3 = 0.4$  by  $x_1 + x_2$ . In the resulting PMDP, these transitions thus have the probabilities  $p'_1 = 0.3 + x_1$ ,  $p'_2 = 0.3 + x_2$ , and  $p'_3 = 0.4 - x_1 - x_2$ . Because of this, to ensure that all probabilities are nonnegative, the area of valid evaluations is now a triangle, as depicted in Figure 1 a), and the parameter space is no longer a hyper-rectangle as assumed in [6].

To overcome this problem, we encode the validity of parameter valuations into the formula. Instead of considering the original PCTL formula  $\varphi$ , we consider the formula  $\varphi_{\text{valid}} \stackrel{\text{def}}{=} \varphi \wedge \text{valid}$  where

$$\begin{aligned} \text{valid} \equiv & \bigwedge_{s, s' \in S, \alpha \in Act} 0 \leq \mathbf{P}'(s, \alpha, s') \langle v \rangle \leq 1 \\ & \wedge \bigwedge_{s, s' \in S, \alpha \in Act} \mathbf{P}'(s, \alpha, s') \neq 0 \rightarrow \mathbf{P}'(s, \alpha, s') \langle v \rangle \neq 0. \end{aligned} \quad (4)$$

The requirement that, for all  $s \in S$  and  $\alpha \in Act(s)$ , we have  $\sum_{s' \in S} P'(s, \alpha, s') \langle v \rangle = 1$  already follows from the definition of the original model and that of the repair matrix. We remark that Equation (4) involves statements over parameter valuations  $v$  of the model, which can be handled by our solution algorithm.

We can now apply our previous analysis techniques on the PMDP as in Definition 5, for the modified PCTL formula  $\varphi_{\text{valid}}$ . For this, we first introduce some notation.

*Definition 6:* We define

- the *range* of a parameter  $x$  is an interval  $\text{range}(x) = [L_x, U_x]$ ,
- a *region* is a high-dimensional rectangle  $r = \times_{x \in V} [l_x, u_x]$  so that for all  $x \in V$  we have  $[l_x, u_x] \subseteq \text{range}(x)$ ,
- we define the *volume*  $\mu$  of a region  $r = \times_{x \in V} [l_x, u_x]$  as  $\mu(r) \stackrel{\text{def}}{=} \prod_{x \in V} \frac{u_x - l_x}{U_x - L_x}$ ,
- a set of regions  $K$  is *interior-disjoint* in case the interiors of different regions do not overlap,
- for an interior-disjoint set  $K = \{r_1, \dots, r_n\}$  of regions, we define  $\mu(K) \stackrel{\text{def}}{=} \sum_{i=1}^n \mu(r_i)$ , and
- for evaluation  $v$  we write  $v \in r = \times_{x \in V} [l_x, u_x]$  if for all  $x \in V$  we have  $v(x) \in [l_x, u_x]$ .

Given a PMDP  $\mathcal{M}$  and a PCTL formula  $\varphi$ , a mapping  $m: K \rightarrow \{\perp, ?, \top\}$  over an interior-disjoint set  $K$  with  $\mu(K) = 1$  is an  $\varepsilon$ -*solution mapping* in case

- $K$  is an interior-disjoint set and  $\mu(K) = 1$ ,
- $\mu(\{m(r) = ? \mid r \in K\}) \leq \varepsilon$ ,
- for  $r \in K$  with  $m(r) = \top$  we have  $\mathcal{M}_v \models \varphi$  for all  $v \in r$ ,
- for  $r \in K$  with  $m(r) = \perp$  we have  $\mathcal{M}_v \not\models \varphi$  for all  $v \in r$ ,

Each parameter has a range, which specifies the bounds assumed for this parameter. A region denotes a set of variable valuations. The volume of a region (interior-disjoint set of regions) is the fraction of the whole parameter space a region (set of regions) occupies.

An  $\varepsilon$ -solution mapping divides the parameter space into areas for which a given property is valid or invalid with respect to the induced nonparametric MDPs in the given area. A certain volume of the parameter area is allowed to remain undecided. In [6], we discussed how we can compute such mappings. The basic idea is to prove the truth value ( $\top$  or  $\perp$ ) of an undecided region ( $m(r) = ?$ ) using constraint solvers [14]–[16]. In case the truth value cannot be decided for a region, for instance because there are different truth values for evaluations contained in the same region, a region is split into smaller regions. The algorithm then tries to decide the validity of the smaller regions. This procedure is repeated, until the volume of undecided regions is below  $\varepsilon$ .

For the model repair, we use the PMDP defined in Definition 5, and apply the algorithm on  $\varphi_{\text{valid}}$ .

We illustrate the procedure in Figure 1 b) and c). Part b) shows the parameter space with two parameters  $x_1$  and  $x_2$ . Point  $x_1 = x_2 = 0$  represents the original, non-repaired model. The area in which the modified model satisfies the property is

shown in dark colour. Thus, by setting  $x_i$  to values in those areas, we can obtain a repaired model.

Now, by applying our algorithm, we obtain the mapping depicted in Figure 1 c). Here, the semi-dark regions represent undecided regions, the white regions represent areas in which the property is not fulfilled, and the dark regions the areas in which the property is fulfilled.

To obtain upper bounds on the minimal repair cost, we can evaluate the repair cost function at the vertices of regions  $r$  with  $m(r) = \top$ , and afterwards take the minimum over all these values. In the figure, these vertices are marked by circles. To obtain a lower bound, we do this instead for regions  $r$  with  $m(r) = ?$ . We mark such points by triangles.

## B. Sampling-based Methods

The sampling-based methods involve randomised search through the parameter space, yielding some good parameter values efficiently, rather than finding all such values or finding a closest value. In this paper we apply Monte Carlo sampling techniques to the fast model repair problem. For simplicity, we will focus on the problem of finding a parameter evaluation in a PMDP so that the weighted distance from the original is sufficiently small. Therefore, Constraint (1) and Constraint (3) in the model repair problem are merged into

$$g\langle v \rangle + P(v) \leq b, \quad (5)$$

where  $b$  is a constant and  $P(v)$  is a *penalty function* defined as follows:

$$P(v) = \begin{cases} 0 & \text{if } \mathcal{M}'_v \models \varphi \\ \delta & \text{otherwise.} \end{cases} \quad (6)$$

The penalty function is used to guide the search for a good evaluation. If an evaluation  $v$  does not make  $\mathcal{M}'$  satisfy  $\varphi$ , then a penalty, which is a predefined positive constant value  $\delta$ , e.g., 10000, is generated. This way, the sampling methods get feedback that they are unlikely to find a good evaluation if they continue to follow the current search direction.

Formally, the general rationale of sampling-based methods is to draw samples according to a probability distribution

$$pd(v) = \frac{1}{K} e^{-\beta \mathcal{O}(v)}, \quad (7)$$

where  $\beta$  is some weighting factor,  $K$  is the normalising factor, and  $\mathcal{O}$  is the objective (or *oracle*), which, given  $v$ , checks if the associated MDP  $\mathcal{M}'_v$  satisfies  $\varphi$  and returns  $g\langle v \rangle + P(v)$ . If samples were drawn according to  $pd$ , we would have, for instance, for two points (evaluations)  $v_1$  and  $v_2$  with  $\mathcal{O}(v_1) \ll \mathcal{O}(v_2)$ , that the vicinity of  $v_1$  is more likely to be sampled than that of  $v_2$  in the long run. Note that  $pd$  is *not known a priori*: it is not in closed form, and even computing  $pd(v)$  for a given  $v$  is difficult, since the normalising factor  $K$  is not known. This is one of the main difficulties we have to overcome.

When applying these methods, an *a priori* threshold is given which is the maximal number of the samples being tested. The procedures are terminated when either a good sample point is found which solves the fast repair problem, that is, a sample point in which the property is fulfilled and in which  $g$  is low, or the threshold is reached.

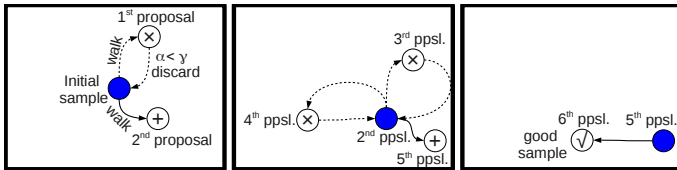


Fig. 2. The evolution of Markov chain Monte Carlo

Below we describe three approaches introduced in [8], namely, the *Markov chain Monte Carlo (MCMC)*, *cross entropy (CE)*, and the *particle swarm optimisation (PSO)*, which turn out to be efficient for our purpose.

**Markov chain Monte Carlo method.** The *Metropolis-Hastings algorithm* (M-H algorithm, [17] [18]) is a variant of the MCMC algorithm. The main idea of the M-H algorithm is to generate a series of samples that are linked in a Markov chain (typically with a continuous state space), where each sample is correlated only with the directly preceding sample. At sufficiently long times (when the equilibrium is reached), the distribution of the generated samples matches the desired probability distribution. Roughly speaking, this algorithm proceeds by randomly attempting to move about the sample space, sometimes accepting the moves and sometimes remaining in place.

In the algorithm, an *acceptance ratio*  $\bar{\alpha}$  is needed to indicate how probable the new proposed sample is with respect to the current sample, according to the distribution  $pd$ . If we attempt to move to a point that is more probable than the existing point (i.e. a point in a higher-density region of  $pd$ ), we will always accept the move. However, if we attempt to move to a less probable point, we will sometimes reject the move, and the higher the relative drop in probability, the more likely we are to reject the new point. Thus, we will tend to stay in (and return large numbers of samples from) the high-density regions of  $pd$ , while only occasionally visiting low-density regions. We refer the reader to [19] for an exposition.

A schematic illustration of the procedure is given in Figure 2. Each iteration of the sampler generates a new *proposal*  $v' \in \text{Evals}(\mathcal{M})$  from the current sample  $v$  using some *proposal scheme*. Note that, here, the support of the distribution is the parameter space  $\text{Evals}(\mathcal{M})$ . The objective  $\mathcal{O}(v')$  is computed for this proposal. We then compute the *acceptance ratio*  $\bar{\alpha} \stackrel{\text{def}}{=} e^{-(\mathcal{O}(v') - \mathcal{O}(v))}$  and accept the proposal randomly, with probability  $\bar{\alpha}$ . Note that, if  $\bar{\alpha} \geq 1$ , then the proposal is definitely accepted. If the proposal is accepted then  $v'$  becomes a new sample; otherwise,  $v$  remains to be the current sample.

Technically, in each iteration, we run a random walk over the parameter space to sample  $\text{Evals}(\mathcal{M})$ . There are many ways to walk randomly but the two ways with the best bounds on the mixing time are the *hit-and-run* and *ball walk*; see [20] for more explanation. Here we give a brief account.

- **Hit-and-run.** (1) Choose a line  $\ell$  through the current point  $v \in \text{Evals}(\mathcal{M})$  uniformly at random. (2) Move to a point  $v'$  chosen uniformly from  $\text{Evals}(\mathcal{M}) \cap \ell$ .

- **Ball walk.** (1) Choose  $v'$  uniformly at random from the ball of radius  $\delta$  centred at the current point  $v$ . (2) If  $v'$  is in the convex set then move to  $v'$ ; if not, try again.

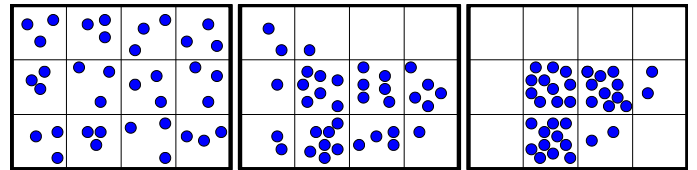


Fig. 3. The evolution of cross entropy

**Cross-Entropy Method.** The cross-entropy method starts from a family of distributions  $\mathcal{U}$  and attempts to find a distribution which is as close to  $pd$  as possible. Note that  $pd$  may not be contained in  $\mathcal{U}$ , but the distributions in  $\mathcal{U}$  usually have a closed form (normal distributions for instance) and are thus easier to sample from. Here closeness of distributions is measured using the standard *Kullback-Liebler divergence* (KL divergence, aka. the cross-entropy) [21].

The general idea of the CE method is that, at each step, it generates samples according to the current *candidate* distribution from the family  $\mathcal{U}$ . Then it uses these samples to *tilt* the current candidate distribution towards a new candidate. We partition the parameter space  $\text{Evals}(\mathcal{M})$  into a set of disjoint measurable cells  $C_1, \dots, C_k$ , where  $C_j$  ( $1 \leq j \leq k$ ) is bounded and has a finite volume. The family of distributions  $\mathcal{U}$  is parameterised by the individual cell sampling probabilities  $\theta: (z_1, \dots, z_k) \in [0, 1]^k$  with  $\sum_{i=1}^k z_i = 1$ . Here  $z_k$  denotes the probability that a point from the cell  $C_i$  is sampled. In order to sample from a given distribution  $p_\theta$  in the family, we choose a cell  $C_i$  with probability  $z_i$  for each  $1 \leq i \leq k$ . As a result, the candidate distribution is expected to get closer to the target distribution. We refer the reader to [22] for details.

A schematic illustration of the method is given in Figure 3, where there are 12 cells and initially the goal is to find 3 samples in each cell (this makes it a uniform distribution with  $\frac{1}{12}$  each). As the procedure goes on, the distribution changes and more samples need to be found in the central cells.

In theory, given the distribution  $pd_{\theta(h)}$  (at the  $h$ -th step) and the samples  $v_1, \dots, v_m$ , the process of tilting is to minimise the empirical KL distance over these samples (at the  $(h+1)$ -st step)  $\theta(h+1)$ . This is standard from the theory of the CE method [22]. In practice, the tilting is usually performed gradually by taking  $\theta(h+1) \stackrel{\text{def}}{=} \beta\theta(h) + (1-\beta)\theta$ , where  $0 < \beta < 1$  is a discount factor.

**Particle swarm optimisation method.** The particle swarm optimisation method [23] [24] is based on swarm intelligence, and its idea is to simulate the movement of a bird flock or fish school. Recall that, given a PMDP  $\tilde{\mathcal{M}}$  with parameters in  $V$ , we associate with  $\mathcal{M}$  constraints  $\text{Evals}(\mathcal{M}) \in \mathbb{R}^m$  as the search space. Moreover, the objective function is given by  $\mathcal{O}(\cdot)$ , which returns the optimal reachability probability for a given valuation  $v \in \text{Evals}(\mathcal{M})$ . The PSO algorithm is based on a population (swarm) of  $n$  particles, each of which is associated with a *velocity*, which indicates where the particle is moving to. By abuse of notation, let the position ( $\vec{v}$ ) and the velocity ( $\vec{r}$ ) of each particle be given as  $m$ -dimensional vectors. For each step  $t \in \mathbb{N}$ , the new position (at  $(t+1)$ -st step) of the  $i$ -th particle ( $1 \leq i \leq n$ ), denoted by  $\vec{v}^i(t+1)$ , is given by

$$\vec{v}^i(t+1) = \vec{v}^i(t) + \vec{r}^i(t+1). \quad (8)$$

The associated velocity vector is updated accordingly by



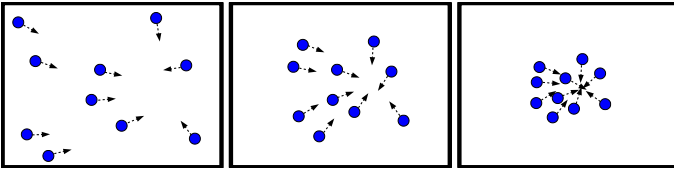


Fig. 4. The evolution of particle swarm

$\vec{r}^i(t+1) = f(\vec{r}^i(t), \vec{v}^i(t))$ , where, intuitively, the function  $f$  is a randomised combination of (1) the direction to the best position of the  $i$ -th particle, and (2) the direction to the best global (among all) particle position.

A schematic illustration of the PSO procedure is given in Figure 4. We remark on the stopping criteria. In our case, clearly, if one particle finds a position  $\vec{v}$  such that  $\mathcal{O}(\vec{v})$  is less than the probability/reward bound in the PCTL formula, then we can stop immediately. Otherwise, to ensure the termination of the procedure, we stop when the norm of the velocity vector  $\vec{v}$  is smaller than some given  $\varepsilon > 0$  for all particles, which is a standard approach for the PSO algorithm. In our experiments, we typically set  $\varepsilon$  to be 0.001, and we apply the L1-norm  $\|\vec{v}\|_1$  for vectors.

## V. PRISM Support

We implemented both model repair methods as an extension of the probabilistic model checker PRISM [9]. In this extension, parametric models can be specified in PRISM’s guarded command language, similarly to nonparametric models. We have decided to allow the specification of parametric models, rather than a nonparametric model and a repair matrix. The reason is that the repair matrix can be easily added to a nonparametric model in the model description. Also, this way of specifying parametric models has the advantage of being general purpose, and is not restricted to model repair problems only. The implementation is currently based on the PRISM “explicit” engine written in JAVA, and hence the state space representation is not represented symbolically. We note that parametric analysis is quite expensive at present, and unlikely to be able to handle large models. Our implementation is written in such a way that it allows for rapid integration of future improvements of, e.g., the representation of regions or functions. The region-based method is a re-implementation of the tool PARAM 2.0 [6], [25]. It will be included in one of the forthcoming releases of PRISM. The implementation for sampling methods was introduced in [8].

The input models used for parametric analyses are specified in the same way as for other types of analyses. Parameters can be expressed as unevaluated constants (e.g. `const double x;`) in the model. Thus, if, for instance, a repairable transition probability is given as 0.5 in the original model description, the PMDP for the repair (cf. Definition 5) can be expressed using the probability of  $0.5+x$ . The user can then specify the ranges of parameters when performing the analysis. These parameters can only be used to specify probabilities or rewards, but cannot occur, e.g., in guards of commands or as lower or upper bounds of model variables. Properties are given as the usual PCTL formulae, where we allow the parameter constants to appear within the formula. This allows one to express, e.g., the constraint from Equation (4). To specify the minimal values

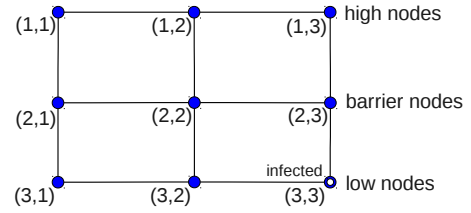


Fig. 5. Network virus case study

from Definition 5, we introduced a new language construct `constfilter`. Similarly to the existing `filter` construct, which filters for the minimum/maximum/average of values over the *states*, `constfilter` allows one to filter for values over *parameter valuations*. Currently, this construct supports computing upper bounds over minima and lower bounds over maxima. For instance, using

```
constfilter (min, x1*x1+x2*x2, <Phi>)
```

one obtains an upper bound for the minima over the function  $g(x_1, x_2) = x_1^2 + x_2^2$  for the values of  $x_1$  and  $x_2$  within their respective ranges, for which the induced model fulfils  $\Phi$ .

In case the `constfilter` construct is used, the output will be the bound to the initial state of the model. Otherwise, an assignment of parameter regions to either rational functions or truth values will be returned. It is also possible to export the result into LATEX code which uses the TIKZ library.

## VI. Case Study

We consider as a case study a parametric model of a computer virus infecting a network [26]<sup>1</sup> derived from [27], [28]. The network is a grid of  $N$  by  $N$  nodes, with each node connected to four neighbours (the nodes that are above, below, to the left and to the right), except for the nodes on the border, for which some of the neighbours are not present.

We model the situation where the virus spawns/multiplies. More concretely, once a node is infected, the virus remains at that node and repeatedly tries to infect uninfected neighbouring nodes.

We suppose that in the network there are “low” and “high” nodes, and that these nodes are divided by “barrier” nodes which scan the traffic between the “high” nodes from the “low” nodes. Initially, only one corner “low” node is infected. A graphical representation of the network for  $N = 3$  is given in Figure 5.

We suppose that both the events of the virus entering a node and infecting a node are *probabilistic*, in that, for each of these steps, there is a chance of success and of failure respectively.

On the other hand, we suppose that the choice as to which node the virus attempts to infect next (out of the neighbouring nodes that are not infected) is *nondeterministic*. This choice depends on the precise topology of the network, that is, the virus can only infect nodes connected to already infected nodes.

We consider a network configuration with  $N = 3$  and suppose that the probability of infection equals 0.5 for all node

<sup>1</sup><http://www.prismmodelchecker.org/casestudies/virus.php>

types. The default probability  $p_{lh}$  of detection by the firewall of the “low” and “high” nodes also equals 0.5 by default, as well as the probability  $p_{ba}$  of the “barrier” nodes. However, both probabilities are subject to *repair* to improve reliability, by increasing the probability by  $p_{lhadd}$  and  $p_{baadd}$  respectively, the *repair cost* of which we assume to be

$$p_{lhadd}^2 + p_{baadd}^2. \quad (9)$$

The corresponding probabilities in the resulting parametric MDP are  $0.5 + p_{lhadd}$  and  $0.5 + p_{baadd}$ . In principle, this would also allow decreasing the probabilities, as opposed to increasing them to repair the model. We do not consider this possibility, because a decrease of the probability of virus detection cannot lead to an improvement of the virus stopping mechanism, so we assume that the minimal values of  $p_{lhadd}$  and  $p_{baadd}$  are 0.

We consider the *minimal expected number of attacks* required by the virus until an infection of the node  $(1, 1)$  starting at  $(N, N)$ . We are interested in ensuring that this value does not become lower than 20, and thus search for the values of  $p_{lhadd}$  and  $p_{baadd}$  which minimise Equation (9), for which the property under consideration is fulfilled.

In Figure 6, we provide a graphical representation of the minimal expected number of attacks depending on the correction by  $p_{lhadd}$  and  $p_{baadd}$ . For the original unmodified model ( $p_{lhadd} = p_{baadd} = 0$ ), this number equals 16, so that the requirement is not satisfied.

### A. Region-based Model Repair Results

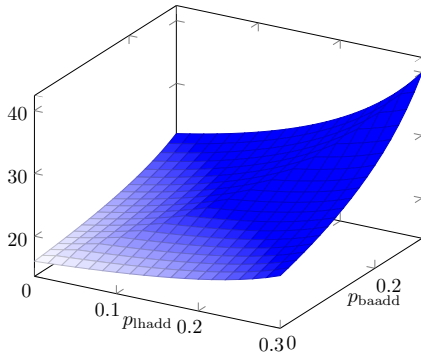


Fig. 6. Minimal expected number of attacks until infection

We apply the region-based method to obtain an upper bound for the minimal repair cost by using the specification

$$\text{constfilter}(\min, \text{detect1} * \text{detect1} + \text{detect2} * \text{detect2}, \text{R}\{\text{"attacks"}\} \geq 20 [\text{F s11}=2]).$$

We used a tolerance (cf. Definition 6) of  $\varepsilon = 0.05$ . In Figure 7, we have divided the parameter space into regions for which  $\text{R}\{\text{"attacks"}\} \geq 20 [\text{F s11}=2]$  is fulfilled (dark), is not fulfilled (white), or this is undecided or different for different parameter valuations within the region (semi-dark).

Undecided regions exist for two different reasons. At first, consider the undecided regions which form the three lines, one of which is a diagonal through the model. These undecided regions exist because, at the lines where they appear, the optimal strategy of the computer virus changes, since it

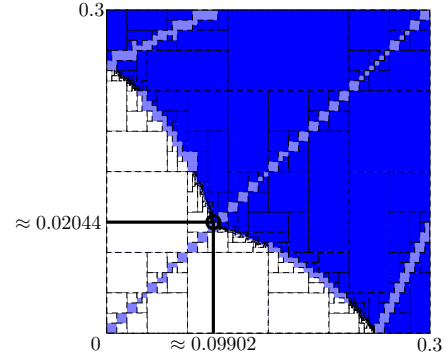


Fig. 7. Checking whether minimal expected number of attacks until infection is higher than 20

depends on the probabilities in the model. Due to the way our parametric analysis works, we cannot decide a rectangle in case it is crossed by such a line, but can only split it into smaller rectangles. Another set of undecided regions appears between the regions in which all parameter valuations are valid (dark), and those in which all of them are not valid (white). Here, the reason for the undecided regions is that these two areas are divided by a curve rather than a straight horizontal or vertical line, such that we cannot cover the whole parameter space by rectangles, but always have to leave a volume for a certain tolerance value undecided.

By evaluating the regions in which the property is fulfilled at their edges, we find a value of  $optval \approx 0.02044$  at  $p_{lhadd} \approx 0.09902$  and  $p_{baadd} \approx 0.10313$  (marked in Figure 7 by two black lines). This means that the minimal cost of repair to ensure that the property holds is no higher than  $optval$ , which is obtained by choosing the repair parameters as described above.

The model of the virus consists of 809 states. The total time to perform the experiments using the region-based approach was two minutes on an Intel(R) Core(TM) i7-3770 CPU with 3.40GHz.

### B. Sampling-based Model Repair Results

To make the sampling-based methods deal with the full model repair problem, i.e., searching for a point at which  $\text{R}\{\text{"attacks"}\} \geq 20 [\text{F s11}=2]$  is satisfied and which minimises Equation (9), we use  $p_{lhadd}^2 + p_{baadd}^2 + P \leq b$  as the objective function, as discussed in Section IV-B.

Table I reports the experimental results for the sampling-based methods. For PSO and MCMC, we restricted the maximum number of trial points to 500. For CE, each parameter domain was partitioned into 5 intervals and refinement was performed 4 times. Due to randomness of these methods, each method was performed for 5 rounds, and we did not compute the average time since the actual time in each round could vary dramatically. We also tested two bounds, i.e.,  $p_{lhadd}^2 + p_{baadd}^2 + P \leq 0.0$  and  $p_{lhadd}^2 + p_{baadd}^2 + P \leq 0.0225$ . The former asked the methods to search for a global minimum point for Equation (9), and the latter told them to stop when a suboptimal point is found. Table I shows that the latter allows the methods to terminate much faster than the former. For each round, we report the running time, the number of samples (#samples), and the result ( $p_{lhadd}^2 + p_{baadd}^2 + P$ ).

TABLE I. EXPERIMENTAL RESULTS FOR THE SAMPLING-BASED METHODS

bound	method	round 1			round 2			round 3			round 4			round 5		
		time (s)	#samples	result	time (s)	#samples	result	time (s)	#samples	result	time (s)	#samples	result	time (s)	#samples	result
0.0	MCMC	18.599	500	0.02701	11.508	500	0.02489	13.362	500	0.02479	11.271	500	0.02227	10.338	500	0.02164
	CE	7.95	663	0.02106	9.358	663	0.02368	12.118	662	0.02193	8.984	665	0.02119	6.515	662	0.02133
	PSO	4.439	327	0.02000	4.406	371	0.02036	6.303	377	0.02001	5.5	319	0.02040	6.322	344	0.02000
0.0225	MCMC	15.449	500	0.02372	14.719	500	0.02359	6.687	355	0.02219	15.304	500	0.02454	1.544	56	0.02170
	CE	8.741	504	0.02228	7.855	485	0.02205	5.143	354	0.02213	16.715	659	0.02288	7.375	307	0.02146
	PSO	3.745	234	0.02056	3.509	192	0.02197	4.408	200	0.02148	6.641	498	0.02374	3.907	209	0.02235

With bound 0.0, MCMC is slower than the other two due to the satisfiability check of constraints. For the larger bound, its performance is not stable: in two rounds, it can find a good point very quickly; but in the other rounds, it fails to obtain a good point, and hence runs slowly. Both CE and PSO are stable regarding finding a good point, but PSO has better performance than CE, since CE has to check all partitions during the search.

### C. Region-based method vs sampling-base methods

We can see from Table I that all sampling-based methods terminated within 20 seconds; in particular, PSO was able to finish within 5 seconds in most rounds. Thus, they were all superior to the region-based method which took 2 minutes. In terms of accuracy of the parameters values, the sampling-based methods can match the region-based method; interestingly, PSO found a smaller value for  $p_{lhadd}^2 + p_{baadd}^2$  than the region-based method. Therefore, sampling-based methods are able to provide a fast solution for model repair. Although they do not guarantee that a good point is found due to random sampling, they provide an efficient solution for real world scenarios if no lower bounds are required.

## VII. Conclusion

We have considered the model repair problem for MDPs, as an extension model repair for Markov chains proposed in [5]. Two approaches to compute approximate solutions to this problems were introduced, and their relative advantages and disadvantages discussed. The methods have been integrated into PRISM, and their practical applicability has been demonstrated on a case study.

**Acknowledgements.** The authors are partly supported by ERC AdG VERIWARE and EPSRC grant EP/F001096.

## References

- [1] A. Bianco and L. D. Alfaro, "Model checking of probabilistic and nondeterministic systems," in *FSTTCS*, ser. LNCS. Springer, 1995, pp. 499–513.
- [2] N. Jansen, E. Ábrahám, B. Zajzon, R. Wimmer, J. Schuster, J.-P. Katoen, and B. Becker, "Symbolic counterexample generation for discrete-time Markov chains," in *FACS*, 2012, pp. 134–151.
- [3] C. Daws, "Symbolic and parametric model checking of discrete-time Markov chains," in *ICTAC*, ser. LNCS. Springer, 2004, pp. 280–294.
- [4] E. M. Hahn, H. Hermanns, and L. Zhang, "Probabilistic reachability for parametric Markov models," in *SPIN*, 2009, pp. 88–106.
- [5] E. Bartocci, R. Grosu, P. Katsaros, C. R. Ramakrishnan, and S. A. Smolka, "Model repair for probabilistic systems," in *TACAS*, ser. LNCS. Springer, 2011.
- [6] E. M. Hahn, T. Han, and L. Zhang, "Synthesis for PCTL in parametric Markov decision processes," in *NFM*, ser. LNCS. Springer, 2011, vol. 6617, pp. 146–161.

- [7] L. Fribourg and É. André, "An inverse method for policy iteration based algorithms," in *INFINITY*, ser. EPTCS. Open Publishing Association, 2009, pp. 44–61.
- [8] T. Chen, T. Han, M. Kwiatkowska, and H. Qu, "Efficient probabilistic parameter synthesis for adaptive systems," Department of Computer Science, University of Oxford, Tech. Rep. CS-RR-13-04, 2013.
- [9] M. Z. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *CAV*, 2011, pp. 585–591.
- [10] B. R. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier, "Model checking performability properties," in *DSN*, 2003, pp. 103–112.
- [11] M. Z. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *SFM*, ser. LNCS. Springer, 2007, pp. 220–270.
- [12] A. Filieri, C. Ghezzi, and G. Tamburrelli, "Run-time efficient probabilistic model checking," in *ICSE*, 2011, pp. 341–350.
- [13] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS management and optimisation in service-based systems," *IEEE TSE*, vol. 37, no. 3, pp. 387–409, 2011.
- [14] M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert, "Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure," *JSAT*, vol. 1, no. 3–4, pp. 209–236, 2007.
- [15] S. Ratschan, "Efficient solving of quantified inequality constraints over the real numbers," *ACM TCL*, vol. 7, no. 4, pp. 723–748, 2006.
- [16] G. O. Passmore and P. B. Jackson, "Combined decision techniques for the existential theory of the reals," in *Calculemus/MKM*, 2009, pp. 122–137.
- [17] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, pp. 1087–1092, 1953.
- [18] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, pp. 97–109, 1970.
- [19] S. Chib and E. Greenberg, "Understanding the Metropolis-Hastings algorithm," *TAS*, vol. 49, no. 4, pp. 327–335, Nov. 1995.
- [20] L. Lovász and R. Kannan, "Faster mixing via average conductance," in *STOC*, 1999, pp. 282–287.
- [21] R. Rubinstein and W. Davidson, "The cross-entropy method for combinatorial and continuous optimization," *Methodology and Computing in Applied Probability*, vol. 1, pp. 129–190, 1999.
- [22] R. Y. Rubinstein and D. P. Kroese, *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Springer, 2004.
- [23] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE IJCNN*, vol. 4. IEEE, 1995, pp. 1942–1948.
- [24] Y. Shi and R. Eberhart, "A modified particle swarm optimization," in *IEEE International Conference on Evolutionary Computation*. IEEE, 1995, pp. 69–73.
- [25] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang, "PARAM: A model checker for parametric Markov models," in *CAV*, ser. LNCS. Springer, 2010, pp. 660–664.
- [26] M. Z. Kwiatkowska, G. Norman, D. Parker, and M. G. Vigiotti, "Probabilistic mobile ambients," *TCS*, vol. 410, no. 12-13, pp. 1272–1303, 2009.
- [27] A. D. Pierro, C. Hankin, and H. Wiklicky, "Continuous-time probabilistic KLAIM," *ENTCS*, vol. 128, no. 5, pp. 27–38, 2005.
- [28] R. De Nicola, J.-P. Katoen, D. Latella, and M. Massink, "Towards a logic for performance and mobility," *ENTCS*, vol. 153, no. 2, pp. 161–175, 2006.