Department of Computer Science

EFFICIENT PROBABILISTIC PARAMETER SYNTHESIS FOR ADAPTIVE SYSTEMS

Taolue Chen
Tingting Han
Marta Kwiatkowska
Hongyang Qu

RR-13-04

Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford OX1 3QD

Efficient Probabilistic Parameter Synthesis for Adaptive Systems

Taolue Chen Tingting Han Marta Kwiatkowska Hongyang Qu
Department of Computer Science, University of Oxford, United Kingdom
{firstname.lastname}@cs.ox.ac.uk

***Abstract*—Probabilistic modelling has proved useful to analyse performance, reliability and energy usage of distributed or networked systems. We consider parametric probabilistic models, in which probabilities are specified as expressions over a set of parameters, rather than concrete values. We address the parameter synthesis problem for parametric Markov decision processes and parametric Markov reward models, which asks for a valuation for the parameters such that the resulting (concrete) probabilistic model satisfies a given property. To solve parametric probabilistic models for quantitative reachability properties, we propose efficient, robust methods, either based on sampling, for which we provide two algorithms, Markov chain Monte Carlo and the cross entropy algorithm, or on swarm intelligence, for which we adapt the particle swarm algorithm, a nonlinear optimisation method from evolutionary computation. We implement the methods in PRISM and demonstrate the effectiveness of our approach on several case studies, including adaptive systems and online model repair.**

I. Introduction

Almost all aspects of our everyday life are enabled or managed by computer systems: from software controllers found in cars and aeroplanes, through healthcare and manufacturing, to entertainment and business supported by cloud computing. The prevalence of such systems, combined with their increasing complexity, means that effective methods to ensure their reliability and performance are essential. Model-based analysis techniques provide an automated way to achieve this.

Probabilistic modelling is a valuable tool for quantifying aspects such as failure, reliability and dependability of systems. Messages transmitted across wireless networks, for example, may be susceptible to losses and delays, or system components may be prone to failure, which can be expressed probabilistically. Another source of probabilistic behaviour is the use of randomisation, for example, to achieve decentralisation when coordinating distributed systems, to manage workflows, or to distribute workload. Thus, system models are often probabilistic in nature, typically represented as variants of Markov chains, constructed from a formal description in some high-level modelling language. Then quantitative, probabilistic verification [26] techniques can be applied to automatically verify whether a given quantitative property hold for the model. While conventional formal verification techniques can establish functional correctness of system models, quantitative verification can be used to analyse non-functional properties such as performance or reliability.

Recently, adaptiveness has become increasingly important in modern software systems, in order to better cope with rapid changes in their running environment so that certain correctness properties, as well as performance, can be guaranteed. In the literature, several frameworks, e.g., [6], [15], have been proposed for dynamic management and optimisation of non-functional requirements of adaptive systems, which can be achieved through quantitative runtime verification [5]. Although these frameworks are naturally appealing for the deployment of real-world systems, obtaining "good" values for the parameters with respect to non-functional requirements can be time consuming, and impairs system performance.

The problem of finding a good valuation can be formalised as *parameter synthesis*. We consider *parametric probabilistic models*, where transition probabilities are not fixed, but depend on a set of parameters, specified as expressions over the parameters. For example, failure probability may be dependent on factors such as the number of attempts to send. Then, given a parametric system model and a quantitative property, the parameter synthesis problem is to find a good parameter valuation such that the property is guaranteed to hold in the model. In this paper, we focus on parametric Markov decision processes (PMDPs) and parametric Markovian reward models (PMRMs), which subsume parametric discrete-time Markov chains (PDTMCs). Note that the parameter synthesis problem is orthogonal to the verification problem, although they are closely related to each other. The main usage of parameter synthesis is in automatically finding parameters contained in a given range so that the (reachability/reward) property is guaranteed to be satisfied. The dimensionality of the method is dependent on the number of parameters to be synthesised, although the size of the system also plays a part, as it takes more time for a larger model to calculate the probability.

Traditionally, there are at least two obvious, "exact" approaches to solve parameter synthesis: (1) reduce to a mathematical programming problem, and (2) encode the problem in first-order theory of real closed fields, which admits quantifier elimination. However, although theoretically appealing, the scalability of these approaches is rather poor. (They usually only work for up to 10 states, while the problem we are handling is often of magnitude of at least 100,000 [17].) Hence, we instead concentrate on developing "inexact" solutions that involve randomised search through the parameter space, yielding some good parameter values efficiently, rather than finding all such values.

In this paper, we propose three efficient approaches to solve the parameter synthesis problem with respect to quantitative

reachability properties in parametric probabilistic models. The key novelty is that we incorporate techniques from areas such as Monte Carlo sampling and evolutionary computation to the setting of quantitative verification, which offer inexact, randomised, algorithms that perform well in practice. In particular, our contribution is to show how to adapt two sampling-based approaches, i.e., *Markov chain Monte Carlo* (MCMC) and the *cross entropy* (CE) method, and a swarm-intelligence method, i.e., the *particle swarm optimisation* (PSO), to the synthesis problem of parametric probabilistic models. Technically, this adaptation is challenging, since we have to decide: (1) which optimisation methods are suitable in our setting, and (2) how to instantiate the general framework (MCMC, CE, PSO) in our particular case to yield highly efficient solutions. For instance, regarding (1), many optimisation techniques (although appealing at a first glance), e.g., McCormick convex relaxation [31], do not perform well, since the optimisation problem in parameter synthesis is neither linear nor convex. Regarding (2), tailoring the definition of Equation (2) in Section III-A and the random walk to sample the parameter space was non-trivial for the models considered here.

We implemented all the algorithms based on the PRISM model checker [27]. The applicability and effectiveness of the techniques developed here is demonstrated through several case studies, drawn from dynamic QoS management of adaptive systems (cf [6], which could be integrated with our methods) and *model repair*. While model checking checks whether a model $\mathcal{M}$ satisfies a property, model repair aims to provide a new model $\mathcal{M}'$, based on $\mathcal{M}$, which is guaranteed to satisfy the property when $\mathcal{M}$ does not. This idea has been demonstrated in [3] for probabilistic systems, where $\mathcal{M}'$ is obtained by adjusting parameters in $\mathcal{M}$. *Online model repair* computes $\mathcal{M}'$ when $\mathcal{M}$ is executing in order to allow adaptation to changes in the environment. In this setting, time efficiency of the model repair procedure is crucial, because frequent changes in the environment may have invalidated $\mathcal{M}'$ before it is generated. We propose the use of parameter synthesis to perform model repair, generalising the approach of [3] to a larger class of Markovian models and significantly improving performance.

Related work. The parameter synthesis problem for parametric probabilistic models has received considerable attention recently. In [13], Daws studied parametric discrete-time Markov chains for the first time, where a language-theoretic approach to solve the reachability problem is proposed. In [19], Hahn *et al* improved on the approaches in [13], and implemented them in the tool PARAM [18], which also supports bounded reachability and reachability rewards. Parametric continuous-time Markov chains were addressed in [20], where rate expressions over variables indicate the average speed of state changes and are expressed using polynomials over the reals. Furthermore, in [17], Hahn *et al* extended the approach of [19] to solve the PCTL synthesis problem for parametric MDPs, where PCTL formulae with the reachability reward properties were also considered. The basic technique is to divide the possible parameter space into hyper-rectangles and to check whether the PCTL property holds on each of the Markov processes represented by the hyper-rectangle. As it is normally impossible to cover the whole parameter space by hyper-rectangles, some areas remain undecided. All of these works provide exact, deterministic methods, and differ from our approach that is based on randomisation and guided search from AI (evolutionary computation).

Our work is also related to quantitative runtime verification [5]. [14] [16] have used parametric DTMCs to improve efficiency of runtime verification and software product lines. [6] employed parametric DTMCs, selecting optimal parameters using the QoSMOS framework. Regarding the use of Monte Carlo sampling, our work can be seen as akin to statistical model checking, which performs approximate verification based on simulation and techniques from statistics. In particular, [38] [24] applied the cross-entropy method to optimise importance sampling parameters. Our work is also related to [32] [35], where sampling is applied to the verification of hybrid systems which inspired our work.

II. PRELIMINARIES

Given a finite set S, we write $\mathfrak{D}(S)$ for the set of *probability distributions* over S.

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

- S is a finite set of states and $s_0 \in S$ is the initial state;
- Act is a finite set of actions; and
- $\mathbf{P} : S \times Act \times S \in [0,1]$ is a transition probability matrix where $\sum_{s' \in S} \mathbf{P}(s,a,s') \in \{0,1\}$ for any $s \in S$ and $a \in Act$; and
- $\rho : S \times Act \to \mathbb{R}_{\geqslant 0}$ is a reward function.

For a state s and action $a \in Act$ in $\mathcal{M}$, we say a is enabled at s if $\sum_{s' \in S} \mathbf{P}(s,a,s') = 1$. We write $\delta(s) \subseteq Act$ for the set of actions which are enabled at s.

A path in an MDP is of the form $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \cdots$ where $a_i \in \delta(s_i)$ and $\mathbf{P}(s_i, a_i, s_{i+1}) > 0$ for each $i \geq 0$. We use $\pi[i]$ to denote the i-th state on π. Let $Paths_s^{\mathcal{M}}$ denote the set of paths in $\mathcal{M}$ starting from s. A finite path is a prefix of an infinite path ending in a state. Let $Paths_{\mathcal{M}}^{\star}$ be the set of finite paths. The superscript or subscript $\mathcal{M}$ may be omitted if it is clear from the context. For path π, let $rew(\pi, j) = \sum_{i=0}^{j} \rho(s_i, a_i)$ be the accumulated reward along π. Note that, in the rest of the paper we mainly deal with MDPs without the reward function. Our techniques can be easily generalised to those with rewards, and results for reward-based properties will be shown in one of the case studies.

In order to formally reason about the probabilistic behaviour of an MDP $\mathcal{M}$, we require the notion of *scheduler* (aka. *strategy*, *policy* or *adversary*), which is one possible resolution of the nondeterministic choices in $\mathcal{M}$. Formally, a scheduler is a function $\sigma : Paths^{\star} \to Act$ such that, in each state s, σ selects an action in $\delta(s)$ based on the finite path ending in s (the history of choices made so far). A scheduler σ restricts the behaviour of the MDP to a set of paths $Paths_s^{\sigma} \subseteq Paths_s$, which induces a probability distribution $\Pr_s^{\sigma}$ over the paths

$Paths^{\sigma}_{s}$ in a standard way (cf. [2, Chapter 10]). We write $\Sigma_{\mathcal{M}}$ for the set of schedulers in $\mathcal{M}$.

Properties to be verified against MDPs are usually expressed in temporal logics, such as PCTL [21] [4] and LTL [11]. Performing verification reduces to the computation of a few key properties of MDPs, among which the main ones are the minimum or maximum *reachability probabilities*, i.e. the minimum or maximum probability that a path through the MDP eventually reaches a state in some target set $G \subseteq S$, quantified over all possible schedulers:

$$p^{\min}_{\mathcal{M},s}(G) = \inf_{\sigma \in \Sigma_{\mathcal{M}}} \Pr^{\sigma}_{s}(\Diamond G) \text{ and } p^{\max}_{\mathcal{M},s}(G) = \sup_{\sigma \in \Sigma_{\mathcal{M}}} \Pr^{\sigma}_{s}(\Diamond G) \quad (1)$$

where $\Diamond G$ is the event $\{\pi \mid \exists n.\pi[n] \in G\}$.

It is well known [4] that to achieve the maximum (or minimum) reachability probabilities *memoryless* schedulers, i.e., the schedulers which select the action only depending on the current state, suffice. Moreover, these probabilities can be computed by *value iteration*, an iterative numerical method which can approximate the values up to some desired accuracy. In practice, this method is widely used since it scales well to large MDPs.

A. Parametric MDPs

Let $X = \{x_1, \ldots, x_m\}$ to be the set of variables (parameters), each of which ranges over $\mathbb{R}$. A *valuation* v over X is a function $v : X \to \mathbb{R}$. In practice, usually each variable x is associated with a closed interval $\mathrm{range}(x) = [L_x, U_x]$. For instance, if we have a parameter denoting the velocity of a train, then the range should be $[0, 400]$. An *affine* function over X is of the form $f(\mathbf{x}) = \mathbf{a} \cdot \mathbf{x} + \mathbf{b}$, for $\mathbf{a}, \mathbf{b} \in \mathbb{R}^m$, where $\cdot$ denotes the scalar product. Let FV denote the set of affine functions from X to $\mathbb{R}$. Given $f \in FV$ and a valuation v, we let $f\langle v \rangle := f(v(x_1), \ldots, v(x_m))$ denote the value obtained by replacing each occurrence of x_i with $v(x_i)$. Let $\overline{Dist}(S)$ be the set of all parametric discrete probability distributions over S, i.e., $\overline{\mu} : S \to FV$.

Formally, a *parametric Markov decision process (PMDP)* is a tuple $\mathcal{M} = (S, s_0, Act, \mathbb{P}, \rho, X)$ where S, and s_0 are the same as for MDPs, and X is a set of variables. The transition matrix $\mathbb{P}$ is of the same form as in MDPs, but defined on parametric distributions.

For any state s, a parametric distribution $\overline{\mu} \in \mathbb{P}(s)$ is a vector of affine functions over X, and we usually write $\overline{\mu} = F_s \cdot \mathbf{x} + f_s$, where $F_s \in \mathbb{R}^{n \times m}$, and $f_s \in \mathbb{R}^m$.

Given a PMDP $\mathcal{M}$ with parameters in X, we associate with $\mathcal{M}$ a system of constraints $\mathcal{K}(\mathcal{M})$ specifying which values of X are valid. In particular, $\mathcal{K}(\mathcal{M})$ consists of:

- Basic constraints. Namely, for each state s and action $a \in \delta(s)$, we have that

  $$\mathbf{0} \leq F_{s,a} \cdot \mathbf{x} + f_{s,a} \leq \mathbf{1} \quad \text{and} \quad \mathbf{1} \cdot (F_{s,a} \cdot \mathbf{x} + f_{s,a}) = 1$$

  Moreover, as discussed above, we have $L_x \leq x \leq U_x$ for each $x \in X$.
- Additional constraints. This is provided by the user to specify the correlation among the parameters in X.

Each point in $\mathcal{K}(\mathcal{M})$ corresponds to a valuation for X. An MDP is *induced* from a PMDP and a valuation v by replacing each $x \in X$ by its value in v.

Remark 1: It turns out that our approaches are very general, in the sense that one does not need to restrict to linear functions for the transition probabilities. However, for simplicity and also for implementation, we only present our results for linear (affine) functions.

Problem statement. Assume a given PMDP $\mathcal{M}$. For a *parameter space* $\Xi = \mathcal{K}(\mathcal{M}) \subseteq \mathbb{R}^m$, and for each $\mathbf{x} \in \Xi$, we obtain an MDP $\mathcal{M}\langle \mathbf{x} \rangle$, and one can compute the *optimal* (i.e., maximum or minimum) reachability probability using standard (efficient) techniques, typically value iteration. Throughout the paper we write $\mathcal{O}$ for the *oracle*, which, given $\mathbf{x}$, returns the maximum or minimum reachability probability (cf. (1)) of the associated MDP.

We are interested in the *parameter synthesis* problem, which asks whether there exists some valuation v for the parameters in $\mathcal{M}$ such that the maximum/minimum reachability probability of $\mathcal{M}\langle v \rangle$ satisfies $\bowtie \lambda$, where $\bowtie \in \{\leq, <, >, \geq\}$ and $\lambda \in [0, 1]$ is some threshold. Note that we formulate this in an existential way, i.e., we ask whether there exists some valuation. Alternatively, one could formulate the problem in a universal way, i.e., ask whether, for all valuations v for the parameters in $\mathcal{M}$, the maximum/minimum reachability probability of $\mathcal{M}\langle v \rangle$ satisfies $\bowtie \lambda$. It is evident that these two formulations are equivalent.

To solve the parameter synthesis problem, a simple observation is that one only needs to find the optimal valuation among the parameter space $\mathcal{K}(\mathcal{M})$. For example, there exists some valuation v such that the maximum reachability probability to goal states G of $\mathcal{M}\langle v \rangle$ is no greater than λ iff the *minimum* over Ξ of the maximum reachability probability of the associated MDP is *no greater than* λ. The general strategy is to search $\mathbf{x} \in \mathcal{K}(\mathcal{M})$ *minimizing* $p^{\max}_{\mathcal{M}\langle \mathbf{x} \rangle, s}(G)$ where one could immediately stop if some valuation v is found (not necessarily the minimum) fulfilling the requirement. Accordingly, if one is interested in *no less than* λ, one should search $\mathbf{x}$ *maximizing* $p^{\max}_{\mathcal{M}\langle \mathbf{x} \rangle, s}(G)$ instead. NB., in Section III-A and Section III-B we assume that the question is to find some valuation such that the optimal reachability probability is no greater than λ, so we should find the *minimum* over Ξ. The dual case (i.e., to find some valuation such that the optimal reachability probability is no less than λ) can be handled in a similar way. Both cases have been implemented, but we only present the former for brevity.

B. Parametric MRMs

A *discrete-time Markov chain* (DTMC) $\mathcal{D} = (S, s_0, \mathbf{P}, \rho)$ can be regarded as an MDP with $|Act| = 1$. A *Markovian reward model* (MRM, i.e., a *continuous-time Markov chain* with rewards) [9] is $\mathcal{C} = (S, s_0, \mathbf{P}, E, \rho)$ where $(S, s_0, \mathbf{P}, \rho)$ is a DTMC, $E : S \to \mathbb{R}_{\geqslant 0}$ assigns an *exit rate* to each state. In MRM, for a timed path $\pi = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \cdots$ with $t_i \in \mathbb{R}_{\geq 0}$ and $t = \sum_{i=0}^{j-1} t_i + t'$ with $t' \leq t_k$, let

$r(\pi,t) = \sum_{i=0}^{j-1} t_i \cdot \rho(s_i) + t' \cdot \rho(s_j)$ be the accumulated reward along π up to time t. Similarly to PMDPs, MRMs can be made parametric, resulting in *parametric MRMs* (PMRMs) which are defined as $\mathcal{C} = (S, s_0, \mathbf{P}', E, \rho, X)$, where $\mathbf{P}' : S \times S \to FV$. Accordingly, notions for PMDPs in Section II-A can be adapted for PMRMs without any difficulties. We refer the reader to [20] for further details.

To verify a (P)MRM, we could apply the uniformisation technique [1] and reduce to the problem for DTMCs with rewards [9]. As a result, all the techniques we develop for PMDPs (of which (P)DTMCs are a special case) can be applied to PMRMs. In the rest of the paper, weonly present the results for PMDPs and show a case study which is modelled by PMRMs in Section IV.

III. Parameter Synthesis

A. Sampling Based Approach

In this section, we apply Monte Carlo sampling techniques to the parameter synthesis problem of PMDPs. The general rationale of sampling-based methods is to draw samples according to a probability distribution. In a nutshell, let $p(x)$ be the *probability density function* (pdf) over the support X. We have to provide a sampling scheme producing a sequence of samples $x_1, \cdots, x_N \in X$ such that, for any measurable subset $Y \subseteq X$, we have that

$$\lim_{N\to\infty} \frac{\sum_{i=1}^{N} \mathbf{1}_Y(x_i)}{N} = \int_Y p(x)dx,$$

where $\mathbf{1}_Y$ is the *characteristic function* for Y (i.e, $\mathbf{1}_Y(x) = 1$ if $x \in Y$; and 0 otherwise.)

To apply this general idea to our problem, as the first step we should provide a probability distribution, sampling according to which one has a higher chance of getting a sample with as small value as possible. Note that, here, the support of the distribution is the parameter space Ξ, which is a polytope. We define the probability distribution (in terms of pdf) as

$$p(\mathbf{x}) = \frac{1}{K} e^{-\beta \mathcal{O}(\mathbf{x})}, \tag{2}$$

where $\mathcal{O}$ is the oracle, β is some weighting factor and K is the normalizing factor. It is not hard to see that the probability of having a smaller $\mathcal{O}(\mathbf{x})$ is exponentially larger than that of having a larger $\mathcal{O}(\mathbf{x})$. (The precise ratio is controlled by β though.) Hence, if samples are drawn according to the distribution p, we will have, for instance, for two points $\mathbf{a}$ and $\mathbf{b}$ with $\mathcal{O}(\mathbf{a}) \ll \mathcal{O}(\mathbf{b})$, that $\mathbf{a}$ is more likely to be sampled than $\mathbf{b}$ in the long run. However, p is *not* known *a priori*: it is not in a closed form, and even computing $p(\mathbf{x})$ for a given $\mathbf{x}$ is difficult as the normalizing factor K is not known. This is the main difficulty we have to overcome. Below we propose two approaches, namely, the *Markov chain Monte Carlo* (MCMC) and *cross entropy* (CE), which turn out to be efficient for our purpose.

Remark 2: Equation (2) is a technical formula for the MCMC and CE methods to guide the search of the parameter space; it is *not* the hypothesised distribution of the parameters. Users only specify the range of each parameter (without a priori knowledge of Equation (2)) and apply one of our approaches.

MCMC. In this section, we apply the *Metropolis-Hastings algorithm* (M-H algorithm, [30] [22]). The general 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. Note that the *acceptance ratio* α indicates how probable the new proposed sample is with respect to the current sample, according to the distribution p. 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 p), 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) high-density regions of p, while only occasionally visiting low-density regions.

Algorithm 1: M-H Algorithm

Input: Oracle $\mathcal{O}$, para. space Ξ, prob. threshold λ
Output: Find a good sample or not

```
Choose some initial input x ∈ Ξ;
FIND := false; sum := 0;
repeat
    x' := Generate(x); sum := sum + 1;
    r := U[0,1]; /*uniform random number in [0,1]*/
    α := e^{-β(O(x')-O(x))};        /*acceptance ratio*/
    if α ≥ r; then
        x := x';
        if O(x) ≤ λ then
            FIND := true;
        end
    end
until FIND = true or sum ≥ MaxNumSamples;
return FIND
```

The pseudocode of our procedure is presented in Algorithm 1, which is actually a description of the Metropolis algorithm, a special case of the M-H algorithm. Each iteration of the sampler generates a new *proposal* $\mathbf{x}' \in \Xi$ from the current sample $\mathbf{x}$ using some *proposal scheme*. The objective $\mathcal{O}(\mathbf{x}')$ is computed for this proposal. We then compute the *acceptance ratio* $\alpha := e^{-\beta \cdot (\mathcal{O}(\mathbf{x}') - \mathcal{O}(\mathbf{x}))}$ and accept the proposal randomly, with probability α. Note that if $\alpha \geq 1$ then the proposal is accepted definitely. If the proposal is accepted then $\mathbf{x}'$ becomes a new sample; otherwise, $\mathbf{x}$ remains to be

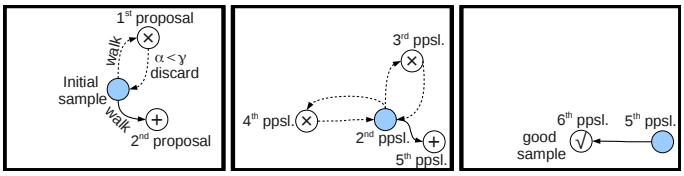

Fig. 1. The evolution of Markov chain Monte Carlo

the current sample. In this algorithm, we also set a upper limit $MaxNumSamples$ on the number of samples being tested to guarantee the termination of the algorithm. A schematic illustration of the procedure is given in Figure 1. Below we discuss some peculiarities of the procedure.

(1) Proposal scheme. To apply the M-H algorithm, we first should give a probability density $Q(x' \mid x)$ (the proposed density or jumping distribution), which suggests a new sample value x' given a sample value x. We require the proposed density to be be symmetric, i.e., $Q(x' \mid x) = Q(x \mid x')$.

In our setting, the sample space is the parameter space Ξ, which is a polytope. The standard proposal scheme which samples a normal distribution centered at $\mathbf{x}$ with a suitably adjusted standard deviation (by some covariance matrix) does not work properly here, simply because the constraint $\mathbf{x}' \in \Xi$ for a newly proposed $\mathbf{x}'$ is very hard to guarantee. Here, inspired by the work on random walk over a convex body [29], we use random walk as a mechanism to generate the proposal, taking advantage that Ξ is convex (actually a polytope), and thus powerful techniques from convex (linear) programming can be leveraged.

Technically, in each iteration, we run a random walk to sample the polytope Ξ. 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 [28] for extensive exposition. Here we give a brief account.

- Hit-and-run. (1) Choose a line ℓ through the current point $\mathbf{x} \in \Xi$ uniformly at random. (2) Move to a point $\mathbf{y}$ chosen uniformly from $\Xi \cap \ell$.
- Ball walk. (1) Choose $\mathbf{y}$ uniformly at random from the ball of radius δ centered at the current point $\mathbf{x}$. (2) If $\mathbf{y}$ is in the convex set then move to $\mathbf{y}$; if not, try again.

We have implemented both these methods. The experience shows that there are no essential differences in terms of efficiency. However, the hit-and-run scheme is easier to implement as one only needs to solve a linear programming problem, while the issue of ball walking is that the radius δ is difficult to select in practice. (In the experiment in Section IV, we only present results using hit-and-run.)

(2) Weighting factor. Recall that, in Equation (2), we introduce the parameter β, which deserves some explanation. Essentially, one intends to direct the search of the sample space Ξ in such way that points with lower values of $\mathcal{O}$ are sampled with an exponentially higher probability compared to points with a higher value of the function $\mathcal{O}$. In our experiments, inspired by [32], we regularly adjust the values of β to ensure that the ratio of accepted samples vs. rejected samples remains close to a fixed value (1 in our experiments). In detail, we record the acceptance ratio during the sampling process based on which we adapt β in the following way: when the acceptance ratio is getting higher, we decrease β and vice versa.

Cross-Entropy Method. Recall that, for the sampling based method, we draw samples according to the probability distribution p given in Equation (2). In this section, we take a different approach than the MCMC in Section III-A, namely, the cross-entropy method (CE), which was introduced by Rubinstein [33]. This method starts from a family of distributions $\mathcal{U}$ and attempts to find a distribution which is as close to p as possible. Note that p may not be contained in $\mathcal{U}$, but, on the other hand, the distributions in $\mathcal{U}$ usually have a closed form (normal distributions for instance) and thus are easier to sample from. Here closeness of distribution is measured using the standard *Kullback-Liebler divergence* (KL divergence), which is defined as follows.

Definition 2: [Kullback-Liebler divergence] Given two distributions η_1 and η_2 over X,

$$d(\eta_1, \eta_2) = \int_X \log\left(\frac{\eta_1(x)}{\eta_2(x)}\right) \cdot \eta_1(x) dx.$$

We note that KL divergence is not a distance, as it is not symmetrical. However, d is always nonnegative and $d(\eta_1, \eta_2) = 0$ iff $\eta_1 = \eta_2$. Therefore, d can be useful in assessing how close two densities are. The KL divergence is also known as the cross-entropy, hence the name of the method.

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. As a result, the candidate distribution is expected to get closer to the target distribution. We refer the reader to [34] for details.

To apply the CE method, the first step is to fix a family of distributions $\mathcal{U}$. For the PMDP synthesis, we use a piecewise-uniform family. More specifically, we partition the parameter space Ξ into a set of disjoint measurable cells $C_1, \ldots, C_k$, where C_j $(1 \leq j \leq k)$ is bounded and has a finite volume. In our experiments, each C_i is a polytope as well. This partition is fixed. The family of distributions $\mathcal{U}$ is parameterized by the individual cell sampling probabilities $\theta : (z_1, \cdots, z_k) \in [0,1]^k$ 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_θ in the family, we choose a cell C_i with probability z_i for each $1 \leq i \leq k$.

Formally, we define Θ to be a finite set of distributions of the form $(z_1, \cdots, z_k)$ and $\mathcal{U} = \{p_\theta \mid \theta \in \Theta\}$. The CE method attempts to choose a distribution p_θ from $\mathcal{U}$ which minimizes the KL divergence $d(p, p_\theta)$. One immediate difficulty is that p is not given in a closed form, so in general it is infeasible to compute $d(p, p_\theta)$. To overcome this difficulty, one typical way is to achieve the minimization only over the current set of samples which gives an empirical estimation. Namely, the CE method proceeds by *approximating* $d(p, p_\theta)$ empirically from samples and adaptively choosing values of θ. Another issue arising in practice is that it is usually not clear how to

decide Θ before the whole procedure. In our experiments, we actually generate Θ in the running time, and we only need to start with Θ which merely contains a (uniform) distribution $\theta = (\frac{1}{k}, \cdots, \frac{1}{k})$. This will be made clear later.

Algorithm 2: CE Algorithm

Input: Oracle $\mathcal{O}$, para. space Ξ, prob. threshold λ
Output: Find a good sample or not

1 Choose $\theta(0)$ from $\mathcal{U}$;
2 FIND:= false; $h := 0$; iter := 0;
3 **repeat**
4 Draw a fixed number of N samples $\mathfrak{S} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ according to $p_{\theta(h)}$; iter := iter + 1;
5 Sort $\mathfrak{S}$ in ascending order according to $\mathcal{O}(\mathbf{x}_1), \cdots, \mathcal{O}(\mathbf{x}_N)$;
6 **if** $\mathcal{O}(\mathbf{x}_1) \leq \lambda$ **then**
7 FIND:=true
8 **end**
9 Let $\mathbf{x}_0, \cdots, \mathbf{x}_m$ be the top m samples ($m \ll N$);
10 $z_j := \frac{\sum_{i=1}^{m} \mathbf{1}_{\mathbf{x}_i \in C_j} \cdot \gamma_i}{\sum_{i=1}^{m} \gamma_i}$ for each $1 \leq j \leq k$,
11 where $\gamma_i = \frac{e^{-\beta \mathcal{O}(\mathbf{x}_i)}}{p_{\theta(h)}(\mathbf{x}_i)}$; /* tilting*/
12 $\theta := (z_1, \cdots, z_k)$;
13 $\theta(h+1) := \beta\theta(h) + (1-\beta)\theta$;
14 **until** FIND **or** iter $\geq$ MaxNumIters;
15 **return** FIND

The pseudocode of our procedure is presented in Algorithm 2, and a schematic illustration of the procedure is given in Figure 2. There are 12 cells and initially we need to find 3 samples in each cell (this makes it a uniform distribution with $1/12$ each). As the procedure goes on, the distribution changes and more samples need to be found in the center cells. Note that we limit the number of iterations by $MaxNumIters$ as a termination condition in case no good sample has been found. Some explanations are in order.

Tilting. Lines 10–13 are for tilting. In theory, given the distribution $p_{\theta(h)}$ and the samples $\mathbf{x}_1, \cdots, \mathbf{x}_m$, we seek to minimize the empirical KL distance over these samples, i.e.,

$$\theta(h+1) := \text{argmin}_\theta \left(-\frac{1}{m} \sum_{i=1}^{m} \left(\frac{\log(p_\theta(\mathbf{x}_i))p(\mathbf{x}_i)}{p_{\theta(h)}(\mathbf{x}_i)} \right) \right).$$

This is standard from the theory of the CE method [34].

To compute this, we write $\gamma_i = \frac{e^{-\beta \mathcal{O}(\mathbf{x}_i)}}{p_{\theta(h)}(\mathbf{x}_i)}$ which can be computed easily. Note that, from our definition (cf. (2)), $p(\mathbf{x}) = \frac{1}{K} e^{-\beta \mathcal{O}(\mathbf{x})}$, and hence it follows that

$$\theta(h+1) = \text{argmax}_\theta \left(\sum_{i=1}^{m} \gamma_i \log(p_\theta(\mathbf{x}_i)) \right).$$

Based on this equation, one can find the optimal θ which maximises $\sum_{i=1}^{m} \gamma_i \log(p_\theta(\mathbf{x}_i))$ disregarding the constraint that $\theta \in \Theta$. Standard method from mathematical analysis

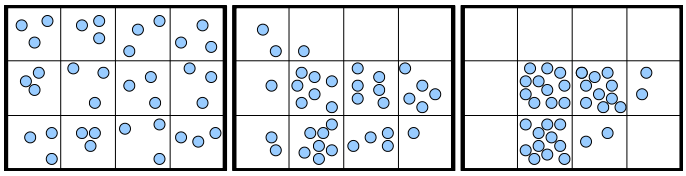

Fig. 2. The evolution of cross entropy

yields that $\theta = (\theta_1, \cdots, \theta_k)$ where

$$\theta_j = \frac{\sum_{i=1}^{m} \mathbf{1}_{\mathbf{x}_i \in C_j} \cdot \gamma_i}{\sum_{i=1}^{m} \gamma_i}$$

for each $1 \leq j \leq k$, and $\mathbf{1}$ is the characteristic function for $\mathbf{x}_i \in C_j$. We set Θ to be $\Theta \cup \{\theta\}$. In practice, the tilting is usually performed gradually by taking $\theta(h+1) := \beta\theta(h) + (1-\beta)\theta$ (as we see at line 13), where $0 < \beta < 1$ is a discount factor.

B. SI-Optimisation

In this section, we apply the particle swarm optimisation (PSO, [25] [36]) to PMDP synthesis problems. The basic idea of PSO is to simulate the movement of a bird flock or fish school. Recall that, given a PMDP $\mathcal{M}$ with parameters in X, we associate with $\mathcal{M}$ constraints $\Xi = \mathcal{K}(\mathcal{M}) \in \mathbb{R}^m$ as the search space. Moreover, the oracle function is given by $\mathcal{O}(\cdot)$, which returns the optimal reachability probability for a given valuation $\mathbf{x} \in \Xi$. The PSO algorithm is based on a population (swarm) of s particles, each of which is associated with a *velocity* which indicates where the particle is moving to. The position ($\mathbf{x}$) and the velocity ($\mathbf{v}$) of each particle are 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 ($1 \leq i \leq s$), denoted by $\mathbf{x}^i(t+1)$, is given as

$$\mathbf{x}^i(t+1) = \mathbf{x}^i(t) + \mathbf{v}^i(t+1). \tag{3}$$

The associated velocity vector is updated accordingly by

$$\mathbf{v}^i(t+1) = \iota(t)\mathbf{v}^i(t) + \mu\omega_1(t)(\mathbf{y}^i(t) - \mathbf{x}^i(t)) + \nu\omega_2(t)(\hat{y}(t) - \mathbf{x}^i(t)),$$

where

- $\iota(t)$ is a weighting factor, called *inertial*;
- $\mu > 0$ and $\nu > 0$ are two parameters called *cognition* and *social* respectively;
- $\omega_1(t)$ and $\omega_2(t)$ are two random vectors with each entry randomly drawn from $(0, 1)$;
- $\mathbf{y}^i(t)$ is the position of the i-th particle with the best objective function value so far calculated; and
- $\hat{\mathbf{y}}(t)$ is the particle position with the best objective function value found so far among all particles. Formally

$$\hat{\mathbf{y}}(t) = \text{argmin}_{\mathbf{z} \in \{\mathbf{y}^1(t), \cdots, \mathbf{y}^s(t)\}} \mathcal{O}(\mathbf{z})$$

Note that the minimisation is taken componentwise. There might be several $\mathbf{z}$ achieving the minimum. In this situation, we simply select the first one.

Intuitively, we add to the previous velocity vector 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.

In the PMDP case, since the search (parameter) space is a polytope which is bounded, we also need to make sure that the constraints are enforced. Here, we define the operation ξ as follows and, when applying (3), we define

$$\xi(\mathbf{x}^i(t), \mathbf{v}^i(t+1)) = \mathbf{x}_i(t) + k^\star \cdot \mathbf{v}_i(t+1)$$

where $k^\star = \min\{\text{argmax}\{k \mid \mathbf{x}_i(t) + k \cdot \mathbf{v}_i(t+1) \in \Xi\}, 1\}$. Note that as Ξ is a polytope, $k^\star$ (and hence ξ) can be easily computed by a reduction to a linear programming problem.

Algorithm 3: Particle Swarm Algorithm

Input: Oracle $\mathcal{O}$, para. space Ξ, prob. threshold λ
Output: Find a good sample or not

Choose initial swarm positions $\mathbf{x}^1(0), \cdots, \mathbf{x}^s(0)$ and the initial swarm velocities $\mathbf{v}^1(0), \cdots, \mathbf{v}^s(0)$.
FIND:= false; $t := 0$;
$\mathbf{y}^i(0) := \mathbf{x}^i(0)$ for $1 \le i \le s$;
$\hat{\mathbf{y}}(1) := \hat{\mathbf{y}}(0) := \text{argmin}_{\mathbf{z} \in \{\mathbf{y}_i(0) | 1 \le i \le s\}} \mathcal{O}(\mathbf{z})$;
repeat
  $t := t + 1$;
  if $\mathcal{O}(\hat{\mathbf{y}}(t)) \le \lambda$ **then**
    FIND := true;
  end
  for $i :=$ *from 1 to* s **do**
    $\mathbf{x}^i(t) := \xi(\mathbf{x}^i(t-1), \mathbf{v}^i(t-1))$; /* compute the current position */
  end
  for $i :=$ *from 1 to* s **do**
    if $\mathcal{O}(\mathbf{x}^i(t)) < \mathcal{O}(\mathbf{y}^i(t))$ **then**
      $\mathbf{y}^i(t+1) := \mathbf{x}^i(t)$;
      if $\mathcal{O}(\mathbf{y}^i(t+1)) < \mathcal{O}(\hat{\mathbf{y}}(t+1))$ **then**
        $\hat{\mathbf{y}}(t+1) := \mathbf{y}^i(t+1)$;
      end
    end
    else
      $\mathbf{y}^i(t+1) := \mathbf{y}^i(t)$;
    end
  end
  $\mathbf{v}^i(t+1) := \iota(t)\mathbf{v}^i(t) + \mu\omega_1(t)(\mathbf{y}^i(t) - \mathbf{x}^i(t)) + \nu\omega_2(t)(\hat{\mathbf{y}}^i(t) - \mathbf{x}^i(t))$;
until FIND *or* $||\mathbf{v}^i(t)|| < \varepsilon$ *for all* $1 \le i \le s$;
return FIND

The pseudocode of our PSO procedure is presented in Algorithm 3, and a schematic illustration of the procedure is given in Figure 3. We remark on the stopping criteria of the procedure. In our case, clearly, if one particle finds a position $\mathbf{x}$ such that $\mathcal{O}(\mathbf{x}) \le \lambda$, then we can stop immediately. Otherwise, to ensure the termination of the procedure, we stop when the norm of the velocity vector $\mathbf{v}$ 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 ε to be 0.001, and we apply the 1-norm for vectors.

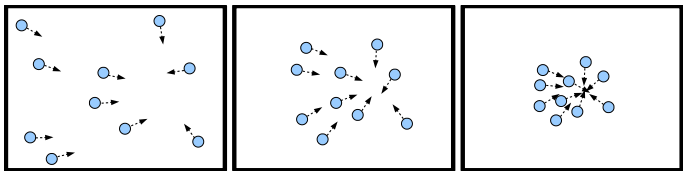

Fig. 3. The evolution of particle swarm

IV. Application to Adaptive Systems

In this section, we apply the techniques developed in Section III in the context of dynamic management of (non-functional) requirements for adaptive systems. We focus on the framework QoSMOS (QoS Management and Optimisation of Service-based systems) proposed by Calinescu *et al* [6], which is based on probabilistic model checking using PRISM [27]. QoSMOS adopts probabilistic temporal logic (PCTL and CSL) to formulate a variety of QoS requirements, and can accommodate parametric Markovian models (PDTMC, PMDP, PMRM, etc) as modelling formalisms to determine the reliability and performance metrics of service-based systems.

QoSMOS augments the standard service-based system architecture with a component called *autonomic manager*. This component performs monitoring, analysing, planning and executing (so called MAPE loop) to achieve continuous adaptation to meet specified QoS requirements as follows: (1) *Monitoring*. This step involves monitoring performance/reliability of the system and the workload/allocated resource of each system component. The information is used to build and/or to update a Markovian model. (2) *Analysis*. In this step, the model is employed to analyse the QoS requirements by executing a probabilistic verification task. The model is *parameterised* by the configurable parameters, and this step aims to identify configurations that satisfy the QoS requirements for the system. (3) *Planning*. In this step, the analysis results are used to derive a plan for adapting the configuration, typically by changing the workflow of the system or by modifying the resources allocated to individual system components. (4) *Execution*. In this step, the adaption plan is implemented.

We remark that the performance of QoSMOS is dominated by the need to enumerate the candidate parameter sets and execute a verification task for each. We propose to significantly improve efficiency by casting the problem as the *parameter synthesis* problem for the parameterised Markovian models adopted in QoSMOS, and solving this using the methods described in Section III. We have implemented the three algorithms described in previous sections in PRISM [27]. In particular, the PSO algorithm was adopted from [37]. The experiments were carried out on a 64-bit PC with an Intel Xeon CPU X5660 2.80GHz and 32GB RAM. The models, properties and implementation can be found at [39].

TeleAsistance. To demonstrate the usefulness of our techniques for adaptive systems, we first consider an example on which the QoSMOS framework was applied [6]. A remote medical assistance system *TeleAssistance* is composed of *Alarm Service*, *Medical Analysis Service* and *Drug Service*. When the system is executing, certain reliability and

TABLE I
TELEASISTANCE RESULTS – TIME & #SAMPLES

TeleAsistance ($\|\mathcal{M}_D\| = 16$, $\|\mathcal{M}_C\| = 102$)										
method	round 1		round 2		round 3		round 4		round 5	
	time (s)	#samples	time (s)	#samples	time (s)	#samples	time (s)	#samples	time (s)	#samples
MCMC	0.164	23	0.053	28	0.03	16	0.078	57	0.016	14
CE	0.116	5	0.054	12	0.112	51	0.013	9	0.013	9
PSO	0.018	12	0.006	11	0.056	94	0.02	39	0.001	1

performance requirements are continuously monitored, and a global utility function is formulated based on the quantitative measurement of these requirements. If the value of the utility function becomes negative, then a system admin alarm is triggered and a set of new parameter values is computed to adapt the system behaviour to meet these requirements. In this paper, the utility function is based on the following three requirements:

1) During the life time of the system, an alarm failure cannot occur with probability higher than 0.13.
2) During the life time of the system, a service failure cannot occur with probability higher than 0.14.
3) The probability that the number of pending changeDrug requests exceeds 75% of the request queue capacity in the long run must be less than 0.2.

The first two requirements are checked on a discrete-time Markov chain model of the system, while the third is checked on a continuous-time Markov chain model, and hence we use PMRMs. The DTMC model has three parameters, a, b and c, modelling probabilities of various system failures, and the CTMC model has one parameter, cpu, representing the fraction of CPU resources on the server allocated to *Drug Service*. Interested readers are referred to [6] for more details. The (simplified) utility function is defined as follows:

$$utility = (\sum_{1}^{3} R_i) - 10 \cdot cpu, \tag{4}$$

where $R_i = 1$ if requirement i is satisfied; 0 otherwise.

The QoSMOS framework [6] precomputes all the possible configurations (parameter valuations) *a priori*, and, when a new configuration is needed, it looks up the table of configurations to select a good one. This strategy works well when the number of configurations is limited. However, it may result in unacceptably slow response times if the number of candidate valuations is high. Worse still, the approach becomes impractical if parameters are continuous variables, which can take any value in a given range.

To improve efficiency, we apply our approaches to search for a good configuration "online". We run each method five times, and do not take the average as the difference between different runs may be quite large. It is possible that a good sample is found in some run in a short time, but no good samples are found in another run, due to the random nature of our algorithms. As a result, we list all the results in Table I to give a fair overview. The size of the model $\mathcal{M}$, denoted $|\mathcal{M}|$, is the number of states. "time" and #samples are the time (in seconds) spent and the number of samples checked before a good sample is found, respectively. For MCMC and PSO, we set $MaxNumSamples = 2000$, namely, the algorithm terminates automatically after 2000 samples are explored; while for the CE method we pick the smallest integer such that $N \times MaxNumIters \geq 2000$ and N is the fixed number of samples drawn in each iteration. Here, the algorithm terminates when roughly 2000 samples are explored.

The results are reported in Table I. It is clear that each method can find a good sample efficiently in this case study. As each parameter is defined as a bounded real number, $a, b, c, cpu \in (0, 1)$, the QoSMOS framework cannot handle this situation due to the infinite number of possible parameter valuations.

Weather service. The second example is a *Weather Service*, which is modelled by PMRMs. This example is drawn from experiment scenarios of the EU project CONNECT [10], where a weather service server provides weather information, and a number of clients query this information via a connector that performs "protocol mapping". The Weather Service is continuously monitored, recording the *average latency* for the clients. The requirement of the system is that the latency is no greater than a given threshold, when all clients successfully obtain weather information from the server without a server failure. If the latency is above the threshold, a system admin alarm is triggered and a set of new parameter values is computed to adjust the system behaviour to meet the requirement.

To enhance the reliability of the server, we use *error checking* that incorporates redundant data for discovery of and recovery from errors caused by hardware or software faults [8]. Error checking inevitably degrades the system's performance because it requires extra processing time, and this is typically performed upon every request. However, *probabilistic error checking* can be applied to reduce the overhead. In particular, each access operation will be followed by error checking with probability $r \in [0, 1)$, instead of certainly (i.e., $r=1$). The server is *1-correctable*, i.e., the system can recover from a single error, but fails if two or more errors occur. We suppose that all requests, as well as the error checking, are atomic and all delays involved (e.g., arrivals, checks) are exponentially distributed. Requests are *accepted* with (exponential distribution) rate λ and *responded* to with rate μ; the hardware/software will *fail* with rate γ, while the *error checking* takes place with rate σ. When taking the error checking into consideration, with rate $r \cdot \mu$ the server is being error checked, and with rate $(1 - r) \cdot \mu$ the server responds. We also assume the clients make a request with rate λ'.

In the case study, we fixed λ' and μ, as λ' is controlled by

TABLE II
WEATHER SERVICE RESULTS – TIME & #SAMPLES VS. REWARDS THRESHOLD

Weather service (#paras= 4)													
$\#clients$	$\|\mathcal{M}\|$	l	method	round 1		round 2		round 3		round 4		round 5	
				time (s)	#samples	time (s)	#samples	time (s)	#samples	time (s)	#samples	time (s)	#samples
1	23	$R\leq 25.8$	MCMC	**0.518**	**2000**	0.287	572	**0.495**	**2000**	0.335	745	**0.543**	**2000**
			CE	**0.503**	**1836**	**0.54**	**1829**	**0.487**	**1825**	**0.461**	**1837**	**0.467**	**1824**
			PS	0.049	172	0.059	160	0.05	152	0.05	147	0.054	203
		$R\leq 26.8$	MCMC	0.17	207	0.1	23	0.222	346	0.187	263	0.171	203
			CE	0.528	1625	0.319	582	0.307	528	0.107	70	0.32	530
			PS	0.059	165	0.06	169	0.053	129	0.061	111	0.02	11
2	529	$R\leq 29.5$	MCMC	**1.194**	**2000**	**1.129**	**2000**	**1.136**	**2000**	**1.152**	**2000**	1.124	1975
			CE	**1.277**	**1833**	**1.288**	**1829**	**1.268**	**1832**	**1.275**	**1830**	**1.276**	**1830**
			PS	0.154	286	0.128	230	0.129	234	0.093	139	0.215	441
		$R\leq 30.5$	MCMC	0.106	44	0.203	169	0.282	290	0.234	218	0.569	779
			CE	0.575	581	0.612	639	0.611	615	0.969	1097	0.591	598
			PS	0.101	169	0.091	146	0.105	179	0.093	153	0.093	137
3	12167	$R\leq 31.5$	MCMC	**29.339**	**2000**	**28.913**	**2000**	**29.224**	**2000**	**29.467**	**2000**	**29.488**	**2000**
			CE	**34.782**	**1832**	**34.96**	**1828**	**34.921**	**1832**	**34.894**	**1833**	**34.936**	**1836**
			PS	2.113	141	3.569	240	4.086	278	4.002	272	3.399	230
		$R\leq 32.5$	MCMC	23.311	1588	6.486	426	2.974	176	9.93	661	1.796	109
			CE	**34.895**	**1828**	13.249	587	13.113	580	**34.988**	**1830**	**35.128**	**1832**
			PS	3.41	231	2.778	185	2.117	141	2.265	151	3.448	233
4	279841	$R\leq 33$	MCMC	**1028.822**	**2000**	**1030.77**	**2000**	**1046.088**	**2000**	**1024.239**	**2000**	**1027.954**	**2000**
			CE	**1249.162**	**1835**	**1241.261**	**1836**	**1240.472**	**1832**	**1244.047**	**1829**	**1256.268**	**1833**
			PS	115.91	223	60.884	118	116.22	227	226.518	433	170.718	328
		$R\leq 34$	MCMC	375.58	716	579.279	1117	220.751	424	341.109	657	81.194	154
			CE	484.189	624	459.773	578	1025.209	1398	**1263.933**	**1836**	463.601	585
			PS	69.43	134	71.578	139	100.411	187	128.868	244	153.478	296

the clients and μ is restricted by the bandwidth of communication channels. Parameters λ, r, γ and σ can be adjusted by the system so the system adaptation can be planned and perfored in the planning and execution phases. We also assume that there are n clients sending requests simultaneously.

We apply the three methods to search for a good sample, i.e., a parameter configuration. The experimental results are shown in Table II. In this case study, we investigate how the reward threshold (column l) affects the average running time, as well as the total number of samples explored. We also vary the model size by modifying the number of clients, and pick two reward thresholds for each model. The numbers in bold indicate that no good samples are found when the algorithm terminates.

We then compare the three methods. Overall, the PS method performed the best and was the most stable one, although occasionally MCMC might beat PS. The MCMC method was quite dependent on the models and thresholds. If the "good region" in the sample space was small, then it was likely that it failed to find one, or took longer time. This is because MCMC always started in the centre of the sample space and "walked" towards the good region. The next sample was found based on the current sample. This is different, however, in the PS method, where the next sample was determined by the current swarm. This difference justified the better performance of PS. As it explored the sample space evenly (in each cell), the CE method had to spend quite some time in each cell, even if the cell was not good at all. This increased the overhead and made the CE method not so efficient. However, we comment that, if the good region was scattered over the sample space (which might not be the case in this case study), then the CE method might have better performance.

Cloud infrastructure. The third example is a three-tier software service deployed on a *cloud* infrastructure [7]. There are three components of the service, i.e., Web, Application, and Database, and several instances of each of the three components are running on different virtual machines, which are located on three physical servers: Server A, Server B and Server C. In particular, Server A and Server B are running two virtual machines respectively, and each virtual machine supports either Web or Application. Server C is a database server which has two virtual machines, each of which supports one database.

We assume that the system is subject to three different failures: (1) disk failure, which happens with probability p_d; (2) RAID failure, which happens with probability p_r; and (3) memory failure, which happens with probability p_m. In addition, each virtual machine i ($1 \leq i \leq 6$) is suffering from failures as well, denoted by p_v^i. The model is a PMDP derived as a parallel composition of all components of the system.

The requirement of the system is that the probability that eventually the system fails is below some threshold. The system failures are continuously monitored. If the rate during a period is above the threshold, a system admin alarm is triggered and a set of new parameter values is computed to allow the system to adapt. In this example, the parameters are p_d, p_r, p_m and p_v^i ($1 \leq i \leq 6$).

In this case study, we fixed a probability threshold and study how the number of parameters affected the average running time as well as the total number of samples explored. We picked 3 parameters (p_d, p_r and p_m) and all 9 parameters because they were representative of two very different behaviours of the system. The results are shown in Table III.

In case of 3 parameters, the MCMC method performed the worst; the CE method performed in a rather unpredictable way, and the PS method was stable and performed well in general.

TABLE III
CLOUD INFRASTRUCTURE RESULTS – TIME & #SAMPLES VS. #PARAS

Cloud ($\|\mathcal{M}\| = 4270168$, $p \leq 0.59$)											
#paras	method	round 1		round 2		round 3		round 4		round 5	
		time (s)	#samples	time (s)	#samples	time (s)	#samples	time (s)	#samples	time (s)	#samples
3	MCMC	**5249.3**	**2000**	**5422.5**	**2000**	**5311.3**	**2000**	**5390.6**	**2000**	**5359.7**	**2000**
	CE	**8320.2**	**2048**	4183.8	1025	81.6	2	4132.5	1025	**8133.5**	**2048**
	PS	644.0	212	534.2	175	673.4	230	531.7	174	527.7	171
9	MCMC	1252.4	336	619.2	159	2921.6	854	1611.1	455	530.9	134
	CE	77.7	1	78.5	1	75.1	1	64.3	1	74.3	1
	PS	116.4	12	155.1	24	107.4	10	157.2	25	489.3	119

The case of 9 parameters surprisingly behaved better than the case of 3 parameters. This is because with 9 parameters there was more likelihood (or combination) to have a good sample. Comparing the three methods, this time the CE was far superior to the others. This is because the starting cell lay in the good region, hence the good performance. The PS method also performed well. The MCMC method started from the "centre" of the sample space and had to move all the way to the "corner", and hence it needed more time.

V. APPLICATION TO ONLINE MODEL REPAIR

In this section, we apply our parameter synthesis to the *online model repair* problem, a variant of model repair in [3]. When a probabilistic model $\mathcal{M}$ with a set of n controllable parameters does not satisfy a property ϕ, model repair generates a new model $\mathcal{M}'$ from $\mathcal{M}$ such that the property $\mathcal{M}'$ holds on $\mathcal{M}'$ and $\mathcal{M}'$ is the closest model to $\mathcal{M}$. The distance between $\mathcal{M}$ and $\mathcal{M}'$ is measured by the function $g = w_1(x'_1 - x_1)^2 + \cdots + w_n(x'_n - x_n)^2$, where $w_i \in \mathbb{R}_+$ and x_i and x'_i are the value of the i-th parameter in $\mathcal{M}$ and $\mathcal{M}'$ respectively. In other words, the distance can be seen as the cost of repair, which should be minimised.

For simplicity, we focus on the problem of finding a parameter valuation in a PMDP such that the weighted distance is sufficiently small. Therefore, the model repair problem can be formalised as finding a valuation of parameters satisfying

$$g\langle v\rangle + P(v) \leq \mathtt{b}, \tag{5}$$

where $\mathtt{b} \in \mathbb{R}_+$ is a bound and $P(v)$ is a *penalty function* defined as follows: $P(v) = 0$ if $\mathcal{M}'\langle v\rangle \models \phi$; and δ otherwise. The penalty function is used to guide the search of good valuations by the sampling methods. If a valuation v does not make $\mathcal{M}'$ satisfy ϕ, then a penalty, which is a predefined positive constant value δ (e.g., 10000) is generated. This way, the sampling methods know it is unlikely to find a good valuation if they continue to follow the current search direction. Thus, the oracle $\mathcal{O}$ checks if the associated MDP $\mathcal{M}'\langle v\rangle$ satisfies ϕ and returns $g\langle v\rangle + P(v)$ for a given v.

We test this method on the Kaminsky case study first presented in [3], which is a PMRM model. We use the same values for the fixed parameters (popularity=3, guess=150,other_legitimate_requests=150) as in [3], and vary the parameter times_to_request_url to generate different model sizes. The only controllable parameter is port_id, which is a 16-bit integer, i.e., $1 \leq$ port_id ≤ 65535. To our surprise, all three methods found a good sample on the first trials due to the large region of good samples in the sample space. For instance, it took about 1.6 seconds to find a good sample in two trials using particle swarm when times_to_request_url=10 (14992 states). By contrast, the time for finding a good sample in [3] was 528 minutes. These figures suggest that, for this cast study, our technique can provide a faster solution than [3] does, although we could not perform an accurate comparison because the implementation of the technique in [3] was not available.

BRP. To analyse the performance of our online model repair methods in depth, we study *Bounded Retransmission Protocol* (BRP, [23]), a variant of the alternating bit protocol for file transfer service. This protocol is used in one of Philips' products. The BRP protocol sends a file in a number of chunks (i.e., parts of a file), but allows only a bounded number of retransmissions of each chunk. So, eventual delivery is not guaranteed and the protocol may abort the file transfer. In probabilistic verification, we are interested in quantifying the probability of such abortion. The protocol consists of a sender S and a receiver R which exchange data via two unreliable (lossy) channels, K and L. The details of the protocol can be found in [12].

In our experiment, the protocol is modelled as a PMDP, where two parameters are introduced for the lossy channels K and L to denote the probability of successful transmission of the message, respectively. We assume both probabilities are 0.7. For the property, we are interested in that "eventually the sender reports a successful transmission with probability at least p", which can be encoded as a reachability property. When this property is violated, both probabilities are subject to repair by increasing the probability of successful transmission by p_{Kadd} and p_{Ladd} respectively, and the repair cost is

$$\big((0.7+p_{\mathrm{Kadd}})-0.7\big)^2+\big((0.7+p_{\mathrm{Ladd}})-0.7\big)^2 = p^2_{\mathrm{Kadd}}+p^2_{\mathrm{Ladd}}. \tag{6}$$

To make the sampling-based methods deal with the full model repair problem, i.e., searching for a point at which the property is fulfilled and which minimises Equation (6), we use $p^2_{\mathrm{Kadd}} + p^2_{\mathrm{Ladd}} + P(p_{\mathrm{Kadd}}, p_{\mathrm{Ladd}}) \leq \mathtt{b}$ as the objective function, where P and b are the same as in Equation (5).

Table IV reports the experimental results for the sampling-based methods. We tested various bounds b and threshold p to investigate their impact on effectiveness of our techniques. Bound 0.0 asked the methods to search for a global minimum

point for Equation (6), and the non-zero bounds told them to stop when a suboptimal point is found. Table IV shows that the latter allows the methods to terminate much faster than the former. For each round, we report the running time, #samples, and the result ($p^2_{\mathrm{Kadd}} + p^2_{\mathrm{Ladd}} + P(p_{\mathrm{Kadd}}, p_{\mathrm{Ladd}})$). The model size is fixed by letting $Max = 4$, which represents the number of retransmissions of a packet.

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 improved, and getting close to that of CE because it tried fewer samples that CE did. In some runs, MCMC even found a good sample very quickly. Both CE and PSO are stable regarding finding a good point, but PSO has better performance than CE, as CE has to check all partitions during the search.

VI. Conclusion

We have considered the parameter synthesis problem for parametric probabilistic models, which asks for a valuation for the parameters such that the resulting (concrete) probabilistic model satisfies quantitative reachability properties. We proposed efficient, robust methods to solve this problem, based on Markov chain Monte Carlo, on the cross entropy method, or on the particle swarm optimisation. We demonstrated the applicability and effectiveness of these methods by case studies of adaptive systems and online model repair. with very encouraging results: for instance, we managed to find a good valuation for a PMDP with 4 million states and 9 parameters in less than 100 seconds. The experimental results showed that our methods improved the efficiency of previous implementations, e.g., [3], by orders of magnitude in some cases. The time can be further shortened using parallel computation. Indeed, our techniques have little correlation to the size of systems or the number of possible parameter valuations.

The experimental results suggest the following guidelines for applicability of the three methods. The PS method has stable performance and always managed to find a good sample. Since the initial locations of particles are random, it is not sensitive to the location of the good region. Although, in most of cases, CE could find a good sample by narrowing down to the good cell(s), it usually performed well when the initial cell was near the good region. The MCMC method was quite dependent on the models and thresholds. If the "good region" in the sample space was small, then it was likely that it failed to find one, or took longer time. This is because the MCMC method always started from the centre of the sample space. It could quickly find a good sample if the good region was near the centre. In practice, although no single method performs well in all classes of systems, we could select a method based on the system characteristic. As a general guideline, we choose MCMC if we know a good region can be found near the centre of the sample space; choose CE if a good region is close to the cells that are dealt with first; or choose PS in other cases.

Our techniques can be easily integrated into existing frameworks for adaptive systems, e.g., QosMOS. Essentially, the second step, i.e., the analysis, can be replaced by our parameter synthesis procedures, as demonstrated in Section IV.

Future work includes improvements of the current methods/implementation by incorporating parallelisation and incremental verification, extension to probabilistic timed automata, as well as more case studies to identify relative strengths and weaknesses.

Acknowledgments

This work is supported by the ERC Advanced Grant VERIWARE, the FP7 EU project CONNECT-IP, and the EPSRC project LSCITS.

References

[1] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003.

[2] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[3] E. Bartocci, R. Grosu, P. Katsaros, C. R. Ramakrishnan, and S. A. Smolka. Model repair for probabilistic systems. In *TACAS*, volume 6605 of *Lecture Notes in Computer Science*, pages 326–340. Springer, 2011.

[4] A. Bianco and L. de Alfaro. Model checking of probabalistic and nondeterministic systems. In *FSTTCS*, pages 499–513, 1995.

[5] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, 55(9):69–77, 2012.

[6] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS management and optimisation in service-based systems. *IEEE Transactions on Software Engineering*, 37(3):387–409, 2011.

[7] R. Calinescu, K. Johnson, and S. Kikuchi. Compositional reverification of probabilistic safety properties for large-scale complex it systems. In *Large-Scale Complex IT Systems – Development, Operation and Management*, pages 303–329, 2012.

[8] I.-R. Chen and I.-L. Yen. Analysis of probabilistic error checking procedures on storage systems. *Comput. J.*, 38(5):348–354, 1995.

[9] L. Cloth, J.-P. Katoen, M. Khattri, and R. Pulungan. Model checking markov reward models with impulse rewards. In *DSN*, pages 722–731, 2005.

[10] CONNECT Consortium. Deliverable 6.3 – Experiment scenarios, prototypes and report - Iteration 2, 2012. http://hal.inria.fr/docs/00/69/56/39/PDF/CONNECT_deliverable_D6_3.pdf.

[11] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.

[12] P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *PAPM-PROBMIV*, pages 39–56, 2001.

[13] C. Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *ICTAC*, pages 280–294, 2004.

[14] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In *ICSE*, pages 341–350, 2011.

[15] A. Filieri, C. Ghezzi, and G. Tamburrelli. A formal approach to adaptive software: continuous assurance of non-functional requirements. *Formal Aspect of Computing*, 24(2):163–186, 2012.

[16] C. Ghezzi and A. Sharifloo. Verifying non-functional properties of software product lines: Towards an efficient approach using parametric model checking. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 170 –174, aug. 2011.

[17] E. M. Hahn, T. Han, and L. Zhang. Synthesis for PCTL in parametric Markov decision processes. In *NASA Formal Methods*, pages 146–161, 2011.

[18] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. PARAM: A model checker for parametric Markov models. In *CAV*, pages 660–664, 2010.

[19] E. M. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric Markov models. *STTT*, 13(1):3–19, 2011.

[20] T. Han, J.-P. Katoen, and A. Mereacre. Approximate parameter synthesis for probabilistic time-bounded reachability. In *IEEE Real-Time Systems Symposium*, pages 173–182, 2008.

TABLE IV
BRP RESULTS – TIME & #SAMPLES VS. PROB./REPAIR THRESHOLD

BRP																			
Max, $\lvert\mathcal{M}\rvert$	Bounds		method	round 1			round 2			round 3			round 4			round 5		
	b	p		time (s)	#samples	result	time (s)	#samples	result	time (s)	#samples	result	time (s)	#samples	result	time (s)	#samples	result
3, 110206	0.0	0.7	MCMC	195.5	2000	0.1050	197.9	2000	0.1053	191.1	2000	0.1048	195.1	2000	0.1047	195.4	2000	0.1058
			CE	157.4	2330	0.1048	136.1	2326	0.1046	140.6	2331	0.1046	132.5	2330	0.1050	140.6	2337	0.1046
			PSO	130.1	2002	0.1047	45.8	2001	0.1046	27.9	1220	0.1046	19.7	864	0.1058	20.7	908	0.1047
		0.8	MCMC	202.5	2000	0.1131	198.3	2000	0.1127	197.4	2000	0.1126	191.8	2000	0.1128	192.1	2000	0.1123
			CE	164.8	2328	0.1123	142.7	2323	0.1123	142.5	2334	0.1123	143.7	2328	0.1126	144.0	2326	0.1123
			PSO	56.6	1855	0.1122	35.3	1487	0.1122	36.8	1555	0.1122	32.1	1353	0.1124	47.4	2001	0.1122
		0.9	MCMC	199.1	2000	0.1231	202.0	2000	0.1231	202.4	2000	0.1237	198.0	2000	0.1231	194.3	2000	0.1231
			CE	149.7	2319	0.1238	131.1	2318	0.1236	140.1	2322	0.1231	138.8	2327	0.1232	138.1	2326	0.1230
			PSO	143.6	1465	0.1232	80.6	955	0.1233	58.8	2001	0.1243	42.9	1910	0.1232	20.6	908	0.1230
	0.105	0.7	MCMC	82.2	788	0.1047	84.2	851	0.1047	203.9	2000	0.1053	202.0	2000	0.1051	197.1	2000	0.1052
			CE	67.6	603	0.1047	123.4	1886	0.1047	72.4	867	0.1043	76.6	1009	0.1047	93.3	1287	0.1048
			PSO	20.1	249	0.1048	102.1	1440	0.1049	119.1	1374	0.1051	25.2	365	0.1048	55.0	734	0.1048
	0.115	0.8	MCMC	28.1	249	0.1124	2.0	18	0.1138	4.7	50	0.1139	13.1	132	0.1128	9.2	105	0.1131
			CE	65.8	578	0.1127	28.3	327	0.1148	23.5	278	0.1134	29.6	328	0.1146	29.1	328	0.1145
			PSO	12.3	104	0.1133	10.0	135	0.1143	8.8	102	0.1127	9.6	145	0.1143	6.7	88	0.1143
	0.125	0.9	MCMC	9.6	77	0.1239	21.4	193	0.1245	27.1	287	0.1247	11.7	116	0.1245	20.7	210	0.1240
			CE	63.2	560	0.1246	47.5	566	0.1237	69.5	875	0.1235	54.6	665	0.1244	28.8	326	0.1245
			PSO	61.6	918	0.1233	3.0	128	0.1240	3.8	163	0.1239	4.1	178	0.1244	6.9	285	0.1247
4, 136244	0.0	0.7	MCMC	295.2	2000	0.0746	294.8	2000	0.0742	294.3	2000	0.0745	288.8	2000	0.0739	286.5	2000	0.0746
			CE	242.1	2336	0.0737	214.5	2337	0.0738	216.5	2333	0.0739	220.3	2331	0.0738	230.9	2332	0.0737
			PSO	112.4	788	0.0739	81.1	747	0.0737	315.7	2001	0.0737	198.5	1283	0.0738	126.5	782	0.0737
		0.8	MCMC	301.0	2000	0.0822	290.2	2000	0.0820	301.6	2000	0.0824	287.1	2000	0.0822	300.5	2000	0.0823
			CE	229.8	2322	0.0817	215.8	2329	0.0818	213.2	2332	0.0818	209.9	2323	0.0817	206.5	2329	0.0817
			PSO	213.6	1301	0.0817	77.9	788	0.0822	78.9	674	0.0817	114.0	1005	0.0817	211.4	1124	0.0817
		0.9	MCMC	304.3	2000	0.0942	295.3	2000	0.0936	310.2	2000	0.0942	291.9	2000	0.0940	286.0	2000	0.0938
			CE	229.7	2331	0.0939	209.8	2323	0.0941	212.2	2333	0.0940	220.5	2326	0.0939	209.9	2324	0.0940
			PSO	255.1	1684	0.0937	108.9	906	0.0937	187.2	1109	0.0938	212.8	1533	0.0954	227.2	1426	0.0936
	0.075	0.7	MCMC	28.2	169	0.0743	217.1	1490	0.0739	65.6	468	0.0742	54.3	363	0.0741	86.0	604	0.0744
			CE	62.4	275	0.0747	75.9	597	0.0737	38.1	275	0.0738	68.5	529	0.0739	110.3	964	0.0702
			PSO	22.2	138	0.0739	19.4	170	0.0739	20.6	191	0.0748	13.3	115	0.0738	19.0	166	0.0745
	0.085	0.8	MCMC	92.2	573	0.0831	61.5	410	0.0840	20.4	135	0.0834	2.7	23	0.0840	5.1	40	0.0845
			CE	86.4	509	0.0845	32.1	232	0.0846	77.2	563	0.0820	35.8	267	0.0828	98.3	865	0.0836
			PSO	15.6	88	0.0847	10.9	77	0.0843	11.6	124	0.0842	12.9	81	0.0835	9.4	66	0.0841
	0.095	0.9	MCMC	28.1	156	0.0944	116.7	770	0.0948	100.7	657	0.0945	141.9	922	0.0941	18.6	148	0.0944
			CE	94.8	574	0.0947	113.2	954	0.0948	109.6	861	0.0947	102.8	899	0.0943	45.4	322	0.0943
			PSO	14.2	47	0.0947	21.3	236	0.0947	7.3	231	0.0947	4.2	129	0.0940	42.8	1376	0.0947

[21] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Asp. Comput.*, 6(5):512–535, 1994.

[22] W. K. Hastings. Monte Carlo samping methods using Markov chains and their applications. *Biometrika*, pages 97–109, 1970.

[23] L. Helmink, M. P. A. Sellink, and F. W. Vaandrager. Proof-checking a data link protocol. In *TYPES*, pages 127–165, 1993.

[24] C. Jégourel, A. Legay, and S. Sedwards. Cross-entropy optimisation of importance sampling parameters for statistical model checking. In *CAV*, pages 327–342, 2012.

[25] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[26] M. Kwiatkowska. Quantitative verification: Models, techniques and tools. In *ESEC/FSE'07*, pages 449–458. ACM Press, 2007.

[27] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV'11*, 2011.

[28] L. Lovász and R. Kannan. Faster mixing via average conductance. In *STOC*, pages 282–287, 1999.

[29] L. Lovász and S. Vempala. Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. *J. Comput. Syst. Sci.*, 72(2):392–417, 2006.

[30] 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*, 21:1087–1092, 1953.

[31] A. Mitsos, B. Chachuat, and P. I. Barton. Mccormick-based relaxations of algorithms. *SIAM Journal on Optimization*, 20(2):573–601, 2009.

[32] T. Nghiem, S. Sankaranarayanan, G. E. Fainekos, F. Ivancic, A. Gupta, and G. J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *HSCC*, pages 211–220, 2010.

[33] R. Rubinstein and W. Davidson. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1:129–190, 1999.

[34] 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.

[35] S. Sankaranarayanan and G. E. Fainekos. Falsification of temporal properties of hybrid systems using the cross-entropy method. In *HSCC*, pages 125–134, 2012.

[36] Y. Shi and R. Eberhart. A modified particle swarm optimization. In *IEEE International Conference on Evolutionary Computation*, pages 69–73. IEEE, 1995.

[37] A. I. F. Vaz and L. N. Vicente. A particle swarm pattern search method for bound constrained global optimization. *J. Global Optimization*, 39(2):197–219, 2007.

[38] P. Zuliani, C. Baier, and E. M. Clarke. Rare-event verification for stochastic hybrid systems. In *HSCC*, pages 217–226, 2012.

[39] http://www.prismmodelchecker.org/subm/ase13synthesis/.