Bayesian Statistical Parameter Synthesis for Linear Temporal Properties of Stochastic Models

Luca Bortolussi[1] and Simone Silvetti[2,3](✉)

[1] University of Trieste, Trieste, Italy
lbortolussi@units.it
[2] University of Udine, Udine, Italy
simone.silvetti@gmail.com
[3] Esteco S.p.A., Trieste, Italy

Abstract. Parameterized verification of temporal properties is an active research area, being extremely relevant for model-based design of complex systems. In this paper, we focus on parameter synthesis for stochastic models, looking for regions of the parameter space where the model satisfies a linear time specification with probability greater (or less) than a given threshold. We propose a statistical approach relying on simulation and leveraging a machine learning method based on Gaussian Processes for statistical parametric verification, namely Smoothed Model Checking. By injecting active learning ideas, we obtain an efficient synthesis routine which is able to identify the target regions with statistical guarantees. Our approach, which is implemented in Python, scales better than existing ones with respect to state space of the model and number of parameters. It is applicable to linear time specifications with time constraints and to more complex stochastic models than Markov Chains.

Keywords: Parameter synthesis · Parametric verification
Smoothed model checking · Gaussian Processes

1 Introduction

Overview. Stochastic models are commonly used in many areas to describe and reason about complex systems, from molecular and systems biology to performance evaluation of computer networks. In all these cases, the system dynamics is usually described by high-level languages as Chemical Reaction Networks [1], population models [2] or Stochastic Petri Nets [3], which generate an underlying *Continuous Time Markov Chain* (CTMC). Formal reasoning about these models often amounts to the computation of reachability probabilities. This is the basic tool behind successful *Stochastic Model Checking* tools like PRISM [4] or the more recent STORM [5]. These tools implement numerical algorithms that compute probabilities up to a given precision, suffering though from state space

D. Beyer and M. Huisman (Eds.): TACAS 2018, LNCS 10806, pp. 396–413, 2018.
https://doi.org/10.1007/978-3-319-89963-3_23

explosion, as well as simulation engines that allow statistical estimation when models are too large.

All classic quantitative verification tools assume that a model is fully specified, which is typically a strong assumption, particularly in application domains like system biology, where many model parameters are estimated from data or are only known to belong to a given range. An alternative approach is that of parameterised verification, which tries to verify properties for a whole set of models, indexed by some parameters. In case of stochastic models, this typically requires us to compute how reachability probabilities change as a function of model parameters, which is a much harder task [6]. A related problem is that of synthesis [7], where one looks for a subset of the parameter space where a given property (or multiple properties [8]) is guaranteed to be satisfied. Alternatively, one can try to design a system by finding a value that maximises the probability of satisfying a specification.

Problem Statement. In this paper, we focus on parameter synthesis for CTMC models described by chemical reaction networks, benchmarking against the approach of [7].

More specifically, we consider the following problem. We have a collection of CTMCs, indexed by a parameter vector $\boldsymbol{\theta} \in \Theta$, taking values in a bounded and compact hyperrectangle $\Theta \subset \mathbb{R}^k$. We assume that the CTMCs depends on $\boldsymbol{\theta}$ through their rates, and that this dependency is smooth. We consider a linear time specifications ϕ described by Metric Interval Temporal Logic [9], with bounded time operators. For each ϕ and $\boldsymbol{\theta}$, we can in principle compute the probability that a random trajectory, generated by that specific CTMC, satisfies it, i.e. $P_\phi(\boldsymbol{\theta})$.

Our goal is to find a partition of the parameter space Θ composed by three classes. The positive class $\mathcal{P}_\alpha$ which is composed by parameters where the probability of satisfying ϕ is higher than a threshold value α, the negative class $\mathcal{N}_\alpha$ composed by parameters where this probability is lower than α and the undefined class $\mathcal{U}_\alpha$ which collects all the other parameters. Following [7], we will look for a partition where the volume of the undefined class is lower a fraction of the volume of Θ. This is the *threshold synthesis problem.*

Our approach will be statistic: we assume that models are too complex to numerically compute bounds on the reachability probability, and we only rely on the possibility of simulating the model. As a consequence, our solution to the parameter synthesis problem will have only statistical guarantees of being correct. For example, if a parameter belongs to $\mathcal{P}_\alpha$, the confidence of this point satisfying $P_\phi(\boldsymbol{\theta}) \geq \alpha$ will be larger than a prescribed probability (typically 95% or 99%), though for most points this probability will be essentially one, and similarly for $\mathcal{N}_\alpha$. The challenge of such an approach is that estimating the satisfaction probability at many different points in the parameter space by simulation is very expensive and inefficient, unless we are able to share the information carried by simulation runs at neighbouring points in the parameter space.

Contributions. We propose a Bayesian statistical approach for parameter synthesis, which leverages a statistical parameterised verification method known as Smoothed Model Checking [6] and the nice theoretical approximation properties of Gaussian Process [10]. Being based on a Bayesian inference engine, this naturally gives statistical error bounds for the estimated probabilities. Our algorithm uses active learning strategies to steer the exploration of the parameter space only where the satisfaction probability is close to the threshold. We also provide a prototype implementation of the approach in Python.

Despite being implemented in Python, our approach turns to be remarkably efficient, being slightly faster than [7] for small models, and outperforming it for more complex and large models or when the number of parameters is increased, at the price of a weaker form of correctness. Compared to [7], we also have an additional advantage: the method treats the simulation engine and the routine to verify of the linear time specification on individual trajectories as black boxes. This means that we can not only treat arbitrary MTL properties (while in [7] they is an essential restriction to non-nested CSL properties, i.e. reachability), but also other more complex linear time specifications (e.g. using hybrid automata, provided that the satisfaction probability is a smooth function of model parameters), and we can also apply the same approach to more complex stochastic models for which efficient simulation routines exist, like stochastic differential equations.

Related Work. Parameter synthesis of CTMC is an active field of research. In [7,11] the authors use Continuous Stochastic Logic (CSL) and uniformization methods for computing exact probability bounds for parameteric models of CTMCs obtained from chemical reaction networks. In [12] the same authors extend their algorithm to GPU architecture to improve the scalability. Authors in these two papers solve two problems: one is the threshold synthesis, the other is the identification of a parameter configuration maximising the satisfaction probability. In this paper we focus on the former, as we already presented a statistical approach to deal the latter problem in [13] for the single objective case and in [8] for the multi-objective case. An alternative statistical approach for multi-objective optimisation is that of [14], where authors use ANOVA test to estimate the dominance relation. Another approach to parameter synthesis for CTMC is [15], where the authors rely on a combination of discretisation of parameters with a refinement technique.

In this work we use a statistical approach to approximate the satisfaction probability function, building on Smoothed Model Checking [6]. This approach is applicable to CTMC with rate functions that are smooth with respect to parameters, and leverages statistical tools based on Gaussian Process regression [10] to learn an approximation of the satisfaction function from few observations. Moreover, this approach allows us to deal with a richer class of linear time properties than reachability, like those described by Metric Temporal Logic [9,16], for which numerical verification routines are heavily suffering from state space explosion [17]. Another statistical approach is that of [18], which combines sensitivity analysis, statistical model checking and uniform continuity to

approximate the satisfaction probability function, but it is restricted to cases when the satisfaction probability is monotonic in the parameters. In contrast, Gaussian Process-based methods have no restriction (as Gaussian Processes are universal approximators), and have also the advantage of requiring much less simulations than pointwise statistical model checking, as information is shared between neighbouring points (see [6] for a discussion in this sense). Parametric verification and synthesis approaches are more consolidated for Discrete Time Markov Chains [19], where mature tools like PROPhESY exist [20], which rely on an symbolic representation of the reachability probability, which does not generalise to the continuous time setting.

Paper Structure. The paper is organized as follows. In Sect. 2 we discuss background material, including Parametric CTMCs, MITL, and Smoothed Model Checking and Gaussian Processes. In Sect. 3 we present our method in detail. In Sect. 4 we discuss experimental results, comparing with [7]. Conclusions and future work are discussed in Sect. 5.

2 Background

In this section we introduce the relevant background material: a formalism to describe the systems of interest, i.e. Parametric Chemical Reaction Networks, and one to describe linear time properties, i.e. Signal Temporal Logic. We then present smoothed model checking [21] and Gaussian Processes [10], which form the underlying statistical backbone of the parameter synthesis.

2.1 Parametric Chemical Reaction Networks

Chemical Reaction Networks [1] are a standard model of population processes, known in literature also as Population Continuous Time Markov Chains [2] or Markov Population Models [22]. We consider a variant with an explicit representation of kinetic parameters.

Definition 1. *A Parametric Chemical Reaction Network (PCRN) $\mathcal{M}$ is a tuple $(S, \mathbf{X}, D, \mathbf{x_0}, \mathcal{R}, \Theta)$ where*

- *$S = \{s_1, \ldots, s_n\}$ is the set of species;*
- *$\mathbf{X} = (X_1, \ldots, X_n)$ is the vector of variables counting the amount of each species, with values $\mathbf{X} \in D$, with $D \subseteq \mathbb{N}^n$ the state space;*
- *$\mathbf{x_0} \in D$ is the initial state;*
- *$\mathcal{R} = \{r_1, \ldots, r_m\}$ is the set of chemical reactions, each of the form $r_j = (\mathbf{v_j}, \alpha_j)$, with $\mathbf{v_j}$ the stoichiometry or update vector and $\alpha_j = \alpha_j(\boldsymbol{X}, \boldsymbol{\theta})$ the propensity or rate function. Each reaction can be represented as*

$$r_j : u_{j,1}s_1 + \ldots + u_{j,n}s_n \xrightarrow{\alpha_j} w_{j,1}s_1 + \ldots + w_{j,n}s_n,$$

where $u_{j,i}$ ($w_{j,i}$) is the amount of elements of species s_i consumed (produced) by reaction r_j. With $\boldsymbol{u_j} = (u_{j,1}, \ldots, u_{j,n})$ (and similarly $\boldsymbol{w_j}$), $\boldsymbol{v_j} = \boldsymbol{w_j} - \boldsymbol{u_j}$.

– $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$ *is the vector of (kinetic) parameters, taking values in a compact hyperrectangle* $\Theta \subset \mathbb{R}^k$.

To stress the dependency of $\mathcal{M}$ on the parameters $\boldsymbol{\theta} \in \Theta$, we will often write $\mathcal{M}_{\boldsymbol{\theta}}$. A PCRN $\mathcal{M}_{\boldsymbol{\theta}}$ defines a Continuous Time Markov Chain [2,23] on D, with infinitesimal generator Q, where $Q_{\boldsymbol{x},\boldsymbol{y}} = \sum_{r_j \in \mathcal{R}} \{\alpha_j(\boldsymbol{x}, \boldsymbol{\theta}) \mid \boldsymbol{y} = \boldsymbol{x} + \boldsymbol{v_j}\}$, $\boldsymbol{x} \neq \boldsymbol{y}$. We denote by $P_{\boldsymbol{\theta}}$ the probability over the paths $Path^{\mathcal{M}_{\boldsymbol{\theta}}}$ of $\mathcal{M}_{\boldsymbol{\theta}}$ of such a CTMC.

2.2 Metric Interval Temporal Logic

Metric Interval Temporal Logic (MITL [16]) is a discrete linear time temporal logic used to reason about the future evolution of a path in continuous time. Generally this formalism is used to qualitatively describe the behaviors of trajectories of differential equations or stochastic models. The temporal operators we consider are all time-bounded, like in Signal Temporal Logic [9], a signal-based version of MITL. This implies that time-bounded trajectories are sufficient to verify every formula. The atomic predicates of MITL are inequalities on a set of real-valued variables, i.e. of the form $\mu(\boldsymbol{X}) := [g(\boldsymbol{X}) \geq 0]$, where $g : \mathbb{R}^n \to \mathbb{R}$ is a continuous function and consequently $\mu : \mathbb{R}^n \to \{\top, \bot\}$.

Definition 2. *A formula* $\phi \in \mathcal{F}$ *of MITL is defined by the following syntax:*

$$\phi := \bot \mid \top \mid \mu \mid \neg\phi \mid \phi \vee \phi \mid \phi \mathbf{U}_{[T_1,T_2]} \phi, \tag{1}$$

where μ *are atomic predicates as defined above, and* $T_1 < T_2 < +\infty$.

Eventually and globally modal operators are defined as customary as $\mathbf{F}_{[T_1,T_2]}\phi \equiv \top \mathbf{U}_{[T_1,T_2]}\phi$ and $\mathbf{G}_{[T_1,T_2]}\phi \equiv \neg \mathbf{F}_{[T_1,T_2]} \neg\phi$. MITL formulae are interpreted over the paths $\boldsymbol{x}(t)$ of a PCRN $\mathcal{M}_{\boldsymbol{\theta}}$. We will consider here the Boolean semantics of [9], which given a trajectory $\boldsymbol{x}(t)$, returns either true or false, referring the reader to [9] for its definition and for a description of monitoring algorithms. Combining this with the probability distribution $P_{\boldsymbol{\theta}}$ over trajectories induced by a PCRN model $\mathcal{M}_{\boldsymbol{\theta}}$, we obtain the satisfaction probability of a formula ϕ as

$$P_\phi(\boldsymbol{\theta}) \equiv P(\phi \mid \mathcal{M}_{\boldsymbol{\theta}}) := P_{\boldsymbol{\theta}}(\{\boldsymbol{x}(t) \in Path^{\mathcal{M}_{\boldsymbol{\theta}}} \mid (\boldsymbol{x}, 0) \models \phi\})$$

2.3 Parametric Verification and Smoothed Model Checking

Given an MITL formula ϕ and a CTMC $\mathcal{M}_{\boldsymbol{\theta}}$, we consider two verification tasks:

– (Classic) Verification: compute or estimate the satisfaction probability $P_\phi(\boldsymbol{\theta})$ for a fixed $\boldsymbol{\theta}$.
– Parametric verification: compute or estimate the satisfaction probability $P_\phi(\boldsymbol{\theta})$ as a function of $\boldsymbol{\theta} \in \Theta$.

The classic verification task can be solved with specialised numerical algorithms [17,24]. These methods calculate $P_\phi(\boldsymbol{\theta})$ by a clever numerical integration of the Kolmogorov equations of the CTMC. This approach, however, suffers from the curse of state space explosion, becoming inefficient for big or complex models. A viable alternative is rooted in statistics. The key idea is to estimate the satisfaction probability by combining simulation and monitoring of MITL formulas. In practice, for each trajectory $\boldsymbol{x}$ generated by a simulation of the CTMC $\mathcal{M}_{\boldsymbol{\theta}}$, we verify if $\boldsymbol{x} \models \phi$. This produces observations of a Bernoulli random variable Z_ϕ, which is equal to 1 if and only if the trajectory satisfies the property, and 0 otherwise. By definition, the probability of observing 1 is exactly $P_\phi(\boldsymbol{\theta})$, which can thus be estimated by frequentist or Bayesian statistical inference [25,26].

Parametric verification brings additional challenges. For PCRN, the numerical approach of [27] provides upper and lower bounds on the satisfaction function. By decomposing the parameter space in small regions, one can provide a tight approximation of the satisfaction function, at the price of a polynomial cost in the dimension of the state space and of an exponential cost in the dimension of the parameter space [27].

The statistical counterpart for parametric verification is known as Smoothed Model Checking [6]. This method combines simulations in few points of the parameter space with state-of-the-art generalised regression methods from statistics and machine learning to infer an analytic approximation of the satisfaction function, mapping each $\boldsymbol{\theta}$ to the corresponding value of $P_\phi(\boldsymbol{\theta})$. The basic idea is to cast the estimation of the satisfaction function as a learning problem: from the observation of few simulation runs at some points of the parameter space, we wish to learn an approximation of the satisfaction function, with statistical error guarantees. Smoothed Model Checking solves this problem relying on Gaussian Process (generalised) regression, a Bayesian non-parametric method that returns in each point an estimate of the value of the satisfaction function together with confidence bounds, defining the region containing the true value of the function with a prescribed probability. The only substantial requirement for Smoothed Model Checking is that the satisfaction probability is smooth with respect to the parameters. This holds for MITL properties interpreted over PCTMCs [6]. Smoothed Model Checking will be the key tool for our synthesis problem, hence we will introduce it in more detail, after a brief introduction of its underlying inference engine, i.e. Gaussian Processes.

Gaussian Processes. Gaussian Processes (GPs) are a family of distributions over function spaces, used mostly for Bayesian non-parametric classification or regression. More specifically, a GP is a collection of random variables $f(\boldsymbol{x}) \in \mathbb{R}$ ($\boldsymbol{x} \in E$, a compact subset of $\mathbb{R}^h$) of which any finite subset defines a multivariate normal distribution. A GP is uniquely determined by its mean and covariance functions (called also kernels) denoted respectively with $m : E \to \mathbb{R}$ and $k : E \times E \to \mathbb{R}$ such that for every finite set of points $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$:

$$f \sim \mathcal{GP}(m, k) \iff (f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \ldots, f(\boldsymbol{x}_n)) \sim \mathcal{N}(\mathbf{m}, K) \tag{2}$$

where $\mathbf{m} = (m(t_1), m(t_2), \ldots, m(t_n))$ is the vector mean and $K \in \mathbb{R}^{n \times n}$ is the covariance matrix, such that $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. From a functional point of view, GP is a probability distribution on the set of functions $g : E \to \mathbb{R}$. The choice of the covariance function is important from a modeling perspective because it determines which functions will be sampled with higher probability from a GP, see [10].

GP are popular as they provide a Bayesian non-parametric framework for regression and classification. Starting from a training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1,\ldots,n}$ of input $\boldsymbol{x}_i$ and output y_i pairs, and a prior GP, typically with zero mean and a given covariance function, GP regression computes a posterior distribution given the observations, which is another GP, whose mean and covariance depend on the prior kernel and the observation points. In particular, for real valued y_i and Gaussian observation noise, the posterior mean at a point $\boldsymbol{x}^*$ is a linear combination of the prior kernel $k(\boldsymbol{x}^*, \boldsymbol{x}_i)$ evaluated at $\boldsymbol{x}^*$ and observation points $\boldsymbol{x}_i$ with coefficients depending on the observations y_i. The prior kernel thus plays a central role, and it sometimes depends on hyperparameters, that can be set automatically by optimising the marginal likelihood, as traditionally happens in Bayesian methods [10].

In this work we use the Gaussian Radial Basis Function (GRBF) kernel [10], as samples from a GP defined by it can approximate arbitrarily well any continuous function on a compact set E. The kernel is defined as

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp(-\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2 / l^2),$$

where l is the lengthscale hyperparameter, which roughly governs how far away observations are contributing to predictions in a point (as if $\boldsymbol{x}^*$ and $\boldsymbol{x}_i$ are much more distant than l, then $k(\boldsymbol{x}^*, \boldsymbol{x}_i)$ is approximately zero). Moreover, l determines the Lipschitz constant of the GRBF kernel, which is $\frac{\sqrt{2/e}}{l}$, and *a fortiori* of the prediction itself (being a linear combination of kernel functions).

Smoothed Model Checking. Smoothed Model Checking is a statistical method to estimate the function $P_\phi(\boldsymbol{\theta})$, casting it into a learning problem taking as input the truth value of ϕ for several simulations at different parameter values $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$, with few simulation runs ($M \ll +\infty$) per parameter point. The method tries to reconstruct a real-valued latent function $f(\boldsymbol{\theta})$, which is squeezed to $[0, 1]$ via the Probit transform[1] Ψ to give the satisfaction probability at $\boldsymbol{\theta}$: $P_\phi(\boldsymbol{\theta}) = \Psi(f(\boldsymbol{\theta}))$. Let us denote with $\mathcal{O} = [\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_n]$ the matrix whose rows $\mathbf{o}_i$ are the Boolean m-vectors of the evaluations in $\boldsymbol{\theta}_j$. Hence, we have that each observation $\mathbf{o}_i$ is an independent draw from a $Binomial(M, P_\phi(\boldsymbol{\theta}_j)))$.

Smoothed Model Checking plugs these observations into a Bayesian inference scheme, assuming a prior $p(f)$ for the latent variable f. As f is a random function, one can take as a prior a GP, specifying its mean and kernel function,

[1] The Probit $\Psi(x) = p(Z \leq x)$ is the cumulative distribution function of a standard normal distribution $Z \sim \mathcal{N}(0, 1)$, evaluated at the point x.

and then invoke Bayes theorem to compute the joint posterior distribution of f at a prediction point $\boldsymbol{\theta}^*$ and at the observation points $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$ as

$$p(f(\boldsymbol{\theta}^*), f(\boldsymbol{\theta}_1), \ldots, f(\boldsymbol{\theta}_n) \mid \mathbf{o}) = \frac{1}{Z} p(f(\boldsymbol{\theta}^*), f(\boldsymbol{\theta}_1), \ldots, f(\boldsymbol{\theta}_n)) \prod_{j=1}^{n} p(\mathbf{o}_j \mid f(\boldsymbol{\theta}_j)).$$

In the previous expression, on the right hand side, Z is a normalisation constant, while $p(f(\boldsymbol{\theta}^*), f(\boldsymbol{\theta}_1), \ldots, f(\boldsymbol{\theta}_n))$ is the prior, which is Gaussian distribution with mean and covariance matrix computed according to the GP. $p(\mathbf{o}_j \mid f(\boldsymbol{\theta}_j))$, instead, is the noise model, which in our case is given by a Binomial density. By integrating out the value of the latent function at observations points in the previous expression, one gets the predictive distribution

$$p(f(\boldsymbol{\theta}^*) \mid \mathcal{O}) = \int \prod_{j=1}^{n} d(f(\boldsymbol{\theta}_j)) p(f(\boldsymbol{\theta}^*), f(\boldsymbol{\theta}_1), \ldots, f(\boldsymbol{\theta}_n) \mid \mathcal{O}).$$

The presence of a Binomial observation model makes this integral analytically intractable, and forces us to resort to an efficient variational approximation known as Expectation Propagation [6,10]. The result is a Gaussian form for the predictive distribution for $p(f(\boldsymbol{\theta}^*) \mid \mathcal{O})$, whose mean and δ-confidence region are then Probit transformed into $[0, 1]$.

It is important to stress that the prediction of Smoothed Model Checking, being a Bayesian method, depends on the choice of the prior. In case of Gaussian Processes, choosing the prior means fixing a covariance function, which makes assumptions on the smoothness and density of the functions that can be sampled by the GP. The Gaussian Radial Basis Function is dense in the space of continuous functions over a compact set [28], hence it can approximate arbitrarily well the satisfaction probability function. By setting its lengthscale via marginal likelihood optimization, we are picking the best prior for the observed data.

3 Methodology

3.1 Problem Definition

We start by rephrasing the parameter synthesis problem defined in [7] in the context of Bayesian statistics, where truths are quantified probabilistically. The basic idea is that we will exhibit a set of parameters that satisfy the specification with high confidence, which in the Bayesian world means with high posterior probability. To recall and fix the notation, let $\mathcal{M}_\theta$ be a PCRN defined over a parameter space Θ, ϕ a MITL formula and $\tilde{P}_\phi(\boldsymbol{\theta})$ be a statistical approximate model of the satisfaction probability of ϕ at each point $\boldsymbol{\theta}$. In the Bayesian setting, $\tilde{P}_\phi(\boldsymbol{\theta})$ is in fact a posterior probability distribution over $[0, 1]$, hence we can compute for each measurable set $B \subseteq [0, 1]$ the probability $p(\tilde{P}_\phi(\boldsymbol{\theta}) \in B)$.

Problem (Bayesian Threshold Synthesis): Let $\mathcal{M}_\theta$, Θ, ϕ, and $\tilde{P}_\phi(\boldsymbol{\theta})$ as before. Fix a threshold α and consider the threshold inequality $P_\phi(\boldsymbol{\theta}) > \alpha$, for the true

satisfaction probability $P_\phi(\boldsymbol{\theta})$. Fix $\epsilon > 0$ a volume tolerance, and $\delta \in (0.5, 1]$ a confidence threshold. The *Bayesian threshold synthesis problem* consists in partitioning the parameter space Θ in three classes $\mathcal{P}_\alpha$ (positive), $\mathcal{N}_\alpha$ (negative) and $\mathcal{U}_\alpha$ (undefined) as follows:

- for each $\theta \in \mathcal{P}_\alpha$, $p(\tilde{P}_\phi(\boldsymbol{\theta}) > \alpha) > \delta$
- for each $\theta \in \mathcal{N}_\alpha$, $p(\tilde{P}_\phi(\boldsymbol{\theta}) < \alpha) > \delta$
- $\mathcal{U}_\alpha = \Theta \setminus (\mathcal{P}_\alpha \cup \mathcal{N}_\alpha)$, and $\frac{vol(U)}{vol(\Theta)} < \epsilon$, where vol is the volume of the set.

Note that the set $\mathcal{P}_\alpha$ solves the threshold synthesis problem defined above, while $\mathcal{N}_\alpha$ solves the threshold synthesis problem $P_\phi(\boldsymbol{\theta}) < \alpha$.

3.2 Bayesian Parameter Synthesis: The Algorithm

Our Bayesian synthesis algorithm essentially combines smoothed Model Checking (smMC) with an active learning step to adaptively refine the sets $\mathcal{P}_\alpha, \mathcal{N}_\alpha, \mathcal{U}_\alpha$, trying to keep the number of simulations of the PCRN $\mathcal{M}_\theta$ to a minimum. smMC is used to compute a Bayesian estimate of the satisfaction probability, given the samples of the truth of ϕ accumulated up to a certain point. More specifically, we use the posterior distribution $p(\tilde{P}_\phi(\boldsymbol{\theta}))$ of the satisfaction probability at each $\boldsymbol{\theta}$ returned by smMC to compute the following two functions of $\boldsymbol{\theta}$:

- $\lambda^+(\boldsymbol{\theta}, \delta)$ is such that $p\left(\tilde{P}_\phi(\boldsymbol{\theta}) < \lambda^+(\boldsymbol{\theta}, \delta)\right) > \delta$
- $\lambda^-(\boldsymbol{\theta}, \delta)$ is such that $p\left(\tilde{P}_\phi(\boldsymbol{\theta}) > \lambda^-(\boldsymbol{\theta}, \delta)\right) > \delta$

Essentially, at each point $\boldsymbol{\theta}$, $\lambda^+(\boldsymbol{\theta}, \delta)$ is the upper bound for the estimate $\tilde{P}_\phi(\boldsymbol{\theta})$ at confidence δ (i.e. with probability at least δ, the true value $P_\phi(\boldsymbol{\theta})$ is less than λ^+), while $\lambda^-(\boldsymbol{\theta}, \delta)$ is the lower bound. These two values will be used to split the parameter space into the three regions $\mathcal{P}_\alpha, \mathcal{N}_\alpha, \mathcal{U}_\alpha$ as follows:

- $\boldsymbol{\theta} \in \mathcal{P}_\alpha$ iff $\lambda^-(\boldsymbol{\theta}, \delta) > \alpha$
- $\boldsymbol{\theta} \in \mathcal{N}_\alpha$ iff $\lambda^+(\boldsymbol{\theta}, \delta) < \alpha$
- $\mathcal{U}_\alpha = \Theta \setminus (\mathcal{P}_\alpha \cup \mathcal{N}_\alpha)$, $\frac{vol(U)}{vol(\Theta)} < \epsilon$

To dig into how λ^+ and λ^- are computed, recall that smMC computes a real-valued Gaussian process $f_\phi(\boldsymbol{\theta})$, with mean function μ and covariance function k, from which the pointwise standard deviation can be obtained as $\sigma(\boldsymbol{\theta}) = \sqrt{k(\boldsymbol{\theta}, \boldsymbol{\theta})}$. At each $\boldsymbol{\theta}$, the function $f_\phi(\boldsymbol{\theta})$ is Gaussian distributed, hence we can compute the upper and lower confidence bounds for the Gaussian, and then squeeze them into $[0, 1]$ by the Probit transform Ψ. Letting $\beta_\delta = \Psi^-1(\frac{\delta+1}{2})$, as customary while working with Normal distributions, we get:

- $\lambda^+(\boldsymbol{\theta}, \delta) = \Psi(\mu(\tilde{f}_\phi(\boldsymbol{\theta})) + \beta_\delta \sigma(\tilde{f}_\phi(\boldsymbol{\theta})))$
- $\lambda^-(\boldsymbol{\theta}, \delta) = \Psi(\mu(\tilde{f}_\phi(\boldsymbol{\theta})) - \beta_\delta \sigma(\tilde{f}_\phi(\boldsymbol{\theta})))$

Algorithm 1. Bayesian Parameter Synthesis.

Input: Θ parameter space, $\mathcal{M}$ PCRN, ϕ MTL formula, α threshold, ϵ volume precision, δ confidence

$\mathcal{S} \leftarrow$ `initial_samples`$(\Theta, \mathcal{M}, \phi)$
$\mathcal{P}_\alpha \leftarrow \emptyset, \mathcal{N}_\alpha \leftarrow \emptyset, \mathcal{U}_\alpha \leftarrow \Theta$
while true **do**
  $\lambda^+, \lambda^- \leftarrow$ `smoothed_MC`$(\Theta, \mathcal{S})$
  $\mathcal{P}_\alpha, \mathcal{N}_\alpha, \mathcal{U}_\alpha \leftarrow$ `update_regions(` $\lambda^+, \lambda^-, \mathcal{P}_\alpha, \mathcal{N}_\alpha, \mathcal{U}_\alpha$`)`
  if $vol(\mathcal{U}_\alpha)/vol(\Theta) < \epsilon$ **then**
    return $\mathcal{P}_\alpha, \mathcal{N}_\alpha, \mathcal{U}_\alpha$
  else
    $\mathcal{S} \leftarrow$ `refine_samples(` $\mathcal{S}, \mathcal{U}_\alpha$`)`
  end if
end while

The Bayesian synthesis procedure is described in Algorithm 1, which after initialisation enters the main loop, in which the computation of the positive, negative, and uncertain sets are carried out adaptively until convergence. Before proceeding further, we introduce some notation to describe regular grids, as they are used in the current implementation of the method. Let us consider the hyper-rectangular parameter space $\Theta = \bigtimes_{i=1}^{n} [w_i^-, w_i^+] \subset \mathbb{R}^n$, where w_i^- and w_i^+ are respectively the lower and the upper bound of the domain of the parameter θ_i. An $\boldsymbol{h}$-grid of Θ is the set $\boldsymbol{h}\text{-grid} = \cup_{\boldsymbol{m} \in M} \{\boldsymbol{w}^- + \boldsymbol{m} * \boldsymbol{h}\}$ where $\boldsymbol{h} = \{h_1, \ldots, h_n\}$, $M = \bigtimes_{i=1}^{n} \{0, \ldots, \frac{w_i^+ - w_i^-}{h_i}\}$, $\boldsymbol{w}^- = (w_1^-, \ldots, w_n^-)$ and $*$ is the elementwise multiplication. Given a grid, we define as *basic cell* a small hyperrectangle of size $\boldsymbol{h}$ whose vertices are points of the grid.

Initialisation. The initialisation phase consists in running some simulations of the PCRN at some points of the parameter space, to have a first reconstruction of the satisfaction function. As we do not need to be very precise in every part of the parameter space, but only for points $\boldsymbol{\theta}$ whose satisfaction probability $P_\phi(\boldsymbol{\theta})$ is close to the threshold α, we start by simulating the model on all parameters of a coarse grid $\boldsymbol{h_0}$-grid, with $\boldsymbol{h_0}$ chosen such that the total number of parameters $\boldsymbol{\theta}$ explored is reasonably small for smMC to be fast. The actual choice will depend on the number of dimensions of the parameter space, as grids depend exponentially on it. Once the grid $\boldsymbol{h_0}$-grid is fixed, we simulate N runs of the model per each point and pass them to a monitoring algorithm for MITL, obtaining N observations of the truth value of the property ϕ at each point of $\boldsymbol{h_0}$-grid, collected in the set $\mathcal{S}$. We also initialise the sets $\mathcal{P}_\alpha$, $\mathcal{N}_\alpha$, and $\mathcal{U}_\alpha$.

Computation of $\boldsymbol{\mathcal{P}_\alpha}$, $\boldsymbol{\mathcal{N}_\alpha}$, and $\boldsymbol{\mathcal{U}_\alpha}$ Regions. The algorithm then enters the main loop, first running smMC with the current set of sample points $\mathcal{S}$ to compute the two functions λ^+ and λ^-. These are then used to update the regions $\mathcal{P}_\alpha$, $\mathcal{N}_\alpha$, and $\mathcal{U}_\alpha$. Here we discuss several possible approaches.

Approach 1: Fixed Grid. The simplest approach is to partition the parameter space in small cells, i.e. using a $\boldsymbol{h}$-grid with $\boldsymbol{h}$ small, and then assign each cell to one of the sets. The assignment will be discussed later, but it involves evaluating the functions λ^+ and λ^- in each point of the grid. The method is accurate if each basic cell contains only a fraction of the volume much smaller than ϵ. However, this requires to work with fine grids, whose size blows up quickly with the number of parameters. Practically, this approach is feasible up to dimension 3 or 4 of the parameter space.

Approach 2: Adaptive Grid. To scale better with the dimension of the parameter space, we can start evaluating the $\lambda^{+/-}$ functions on a coarse grid, and refine the grid iteratively only for cells that are assigned to the uncertain set, until a minimum grid size is reached.

Central in both approaches is how to guarantee that all points of a basic cell are all belonging to one set, inspecting only a finite number of them. In particular, we will limit the evaluation of the $\lambda^{+/-}$ functions to the vertices of each cell c, i.e. to the points in the grid $\boldsymbol{h}$-grid. Intuitively, this will work if the cell has a small edge size compared to the rate of growth of the satisfaction function, and the values of the satisfaction function in its vertices are all (sufficiently) above or below the threshold. However, we need to precisely quantify this "sufficient". We sketch here two exact methods and an heuristic one, which performs well in practice. We discuss here how to check that a cell belongs to the positive set, the negative one being symmetric.

Method 1: Global Lipschitz bound. This approach relies on computing the Lipschitz constant L of the satisfaction function. This can be obtained by estimating its derivatives (e.g. by finite difference or better by learning it using methods discussed in [10]), and performing a global optimization of the modulus of the gradient after each call to smMC. Let $d(\boldsymbol{h})$ be the length of the largest diagonal of a basic cell c in a $\boldsymbol{h}$-grid. Consider the smallest value of the satisfaction function in one of the vertices of c, and call it $\hat{p}$. Then the value of the satisfaction function in the cell is surely greater than $\hat{p} - Ld(\boldsymbol{h})/2$ (after decreasing for half the diagonal, we need to increase again to reach the value of another vertex). The test then is $\hat{p} - Ld(\boldsymbol{h})/2 \geq \alpha$.

Method 2: Local Lipschitz bound. The previous method will suffer if the slope of the satisfaction function is large in some small region, as this will result in a large Lipschitz constant everywhere. To improve it, we can split the parameter space is subregions (for instance, by using a coarse grid), and compute the Lipschitz constant in each subregion. An alternative we are investigating is to compute in each cell of the grid a lower bound of the function $f(\theta)$ learned from the GP from its analytic expression.

Heuristic Method. In order to speed up computation and avoid computing Lipschitz constants, we can make the function λ^- more strict. Specifically, we

can use a larger β_δ than the one required by our confidence level δ. For instance for a 95% confidence, $\beta_\delta = 1.96$, while we can use instead $\beta_\delta = 3$, corresponding roughly to a confidence of 99%. Coupling this with a choice of the grid step $\boldsymbol{h}$ at least one order of magnitude smaller than the lenghtscale of the kernel learned from the data (which is proportional to the Lipschitz constant of the kernel and of the satisfaction function), which guarantees that the satisfaction function will vary very little in each cell, we can be confident that if the strict λ^- is above the threshold in all vertices of the cell, then the same will hold for all points inside c for the less strict λ^-.

Refinement Step. After having build the sets $\mathcal{P}_\alpha$, $\mathcal{N}_\alpha$, and $\mathcal{U}_\alpha$, we check if the volume of $\mathcal{U}_\alpha$ is below the tolerance threshold. If so, we stop and return these sets. Otherwise, we need to increase the precision of the satisfaction function near the uncertain region. This means essentially reducing the variance inside $\mathcal{U}_\alpha$, which can be obtained by increasing the number of observations in this region. Hence, the refinement step samples points from the undefined regions $\mathcal{U}$, simulates the model few times in each of these points, computes the truth of ϕ for each trace, and add these points to the training set $\mathcal{S}$ of the smoothed model checking process. This refinement will reduce the uncertainty bound in the undefined regions which leads some part of this region to be classified as Positive $\mathcal{P}$ or Negative $\mathcal{N}$. We iterate this process until the exit condition $\frac{vol(U)}{vol(\Theta)} < \epsilon$ is satisfied. The convergence of the algorithm is rooted in the properties of smoothed Model Checking, which is guaranteed to converge to the true function with vanishing variance as the number of observation points goes to infinity. In practice, the method converges quite fast, unless the problem is very hard (the true satisfaction function is close to the threshold for a large fraction of the parameter space).

4 Results

Implementation. We have implemented our algorithm in Python 3.6. The code is available at http://simonesilvetti.com/pycheck/. To improve the scalability of our algorithm, we profiled it to identify the most computationally expensive steps, among simulating the PCRN, checking the MITL formulae at each step, running smMC and partitioning the state space. The most expensive part in our test turned out to be the simulation step, which we performed using Gillespie SSA algorithm [1]. To speed up simulations, we ran them in parallel leveraging the Numba [29] package of Python which is optimal to execute array-oriented and math-heavy Python code. The smoothed model checking step, instead, is substantially independent with respect the number of repetitions. Its execution time depends on the cardinality of the training points. This is why, compared with [6], we increased the number of simulations per parameter point and reduced their number. We ran all the experiments on a Dell XPS, Intel Core i7-7700HQ 2.8 GHz, 8 GB 1600 MHz memory, equipped with Windows 10 Pro.

SIR Epidemic Model. We consider the popular SIR epidemic model [30], which is widely used to simulate the spreading of a disease among a population. The population of N individuals is divided in three classes:

- *susceptible* S individuals that are healthy and vulnerable to the infection;
- *infected* I individuals that are actively spreading the disease;
- *recovered* R individuals, which gained immunity to the disease.

The version of SIR model we consider is defined by the following two chemical reactions:

$$r_1 : S + I \xrightarrow{\alpha_1} 2I \qquad \alpha_1 = k_i \cdot \frac{X_s \cdot X_i}{N}$$
$$r_2 : I \xrightarrow{\alpha_2} R \qquad \alpha_2 = k_r \cdot X_i$$

Here, r_1 describes the possibility that an healthy individual gets the disease and becomes infected and the reaction r_2 models the recovery of an infected agent. We described the model as a PCRN where $k_i \in [0.005, 0.3]$, $k_r \in [0.005, 0.2]$ and initial population $(S, I, R) = (95, 5, 0)$ and we consider the following MITL formula:

$$\phi = (I > 0)\, \mathcal{U}_{[100,120]}\, (I = 0) \tag{3}$$

This formula expresses that the disease becomes extinct (i.e.; $I = 0$) between 100 and 120 time units. Note that for this model extinction will eventually happen with probability one, but the time of extinction depends on he parameters $\boldsymbol{\theta} = (k_i, k_r)$. In the following, we report experiments to synthetise the parameter region such that $P_\phi(\boldsymbol{\theta}) > \alpha$, with $\alpha = 0.1$, volume tolerance $\epsilon = 0.1$, and confidence $\delta = 95\%$. We consider all possible combinations of free parameters to explore (i.e. k_i alone, k_r alone, and k_i and k_r). The initial train set of the smoothed model checking approach has been obtained by sampling the truth value on the parameters disposed in a grid as described in Sect. 3, of size 40 points for 1D case and 400 points for the 2D case. The satisfaction probability of each parameter vector which compose the training set, as well as, the parameter vectors sampled by the refinement process have been obtained by simulating the PCRN and evaluating the MITL formula 3 with 1000 repetitions per parameter point.

Efficiency, Accuracy, and Scalability. The execution times of the experiments are reported in Table 1 (left). The results shows a good performance of our statistical algorithms, despite being implemented in Python rather then in a more efficient language like C. The execution time (in percentage) with respect to the results of the exact method reported in [7] are 42%, 18% and 7% for Case 1, Case 2 and Case 3. Our results are reported using the heuristic method to compute the sets and a fixed grid of small stepsize $\boldsymbol{h}$.

In Case 1, we also compare the three methods to classify the regions, computing the derivative of the satisfaction probability function by finite differences and (i) optimising it globally to obtain the Lipschitz constant (equal to 4.31), (ii) optimising it in every cell of the fine prediction grid to compute a local Lipschitz constant (in each cell). As for the heuristic method, we use $\beta'_\delta = 3$ instead of $\beta_\delta = 1.96$, and a grid step of order 10^{-4}, three orders of magnitude less than the lengthscale of the kernel, set by marginal likelihood optimization equal to 0.1. All three methods gave the same results for the grid size we used. More specifically, the maximum displacement of the approximated satisfaction probability inside the cell is estimated to be 0.003

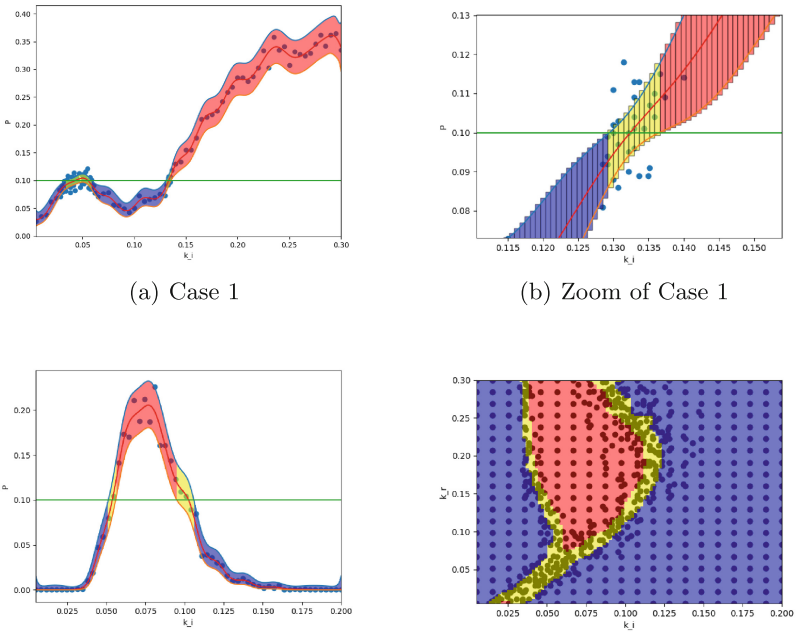
As a statistical accuracy test, we computed the "true" value of the satisfaction probability (by deep statistical model checking, using 10000 runs) for points in the positive and negative set close to the undefined set, and counted how many times these points were misclassified. More specifically, in Case 1 we consider 300 equally-spaced points between 0.1 and 0.07 (consider that a portion of the undefined region is located in a neighborhood of 0.05, see Fig. 1). All points turned to be classified correctly, pointing out to the accuracy of the smMC prediction.

We performed also a scalability test with respect to the size of the state space of the PCRN model, increasing the initial population size N of the SIR model (case 1). The results are reported in Table 1 (right). We increase the initial population size maintaining the original proportion $\frac{I}{S} = \frac{1}{19}$. Moreover we consider different thresholds α and volume tolerance ϵ in order to force the algorithm to execute at least one refinement step, as the shape of the satisfaction function changes with N. The execution time increase moderately, following a linear trend.

Table 1. (LEFT) Results for the Statistical Parameter Synthesis for the SIR model with $N = 100$ individuals and the formula $\phi = (I > 0)\,\mathcal{U}_{[100,120]}\,(I = 0)$. We report the mean and standard deviation of the execution time of the algorithm. The volume tolerance is set to 10% and the threshold α is set to 0.1. The h-grid column shows the size $\boldsymbol{h}$ of the grid used to compute the positive, negative, and uncertain sets. (RIGHT) Scalability of the method w.r.t. the size of the state space of the SIR model, increasing initial population N. α and δ are the threshold and volume tolerance used in the experiments.

Case	$k_i \times k_r$	$\boldsymbol{h}$-grid	Time (sec)
1	$[0.005, 0.3] \times 0.05$	0.0007	17.92 ± 2.61
2	$0.12 \times [0.005, 0.2]$	0.0005	4.87 ± 0.01
3	$[0.005, 0.3] \times [0.005, 0.2]$	(0.003,0.002)	116.4 ± 4.06

Pop. Size	α	δ	Time (sec)
200	0.13	10%	$13,05 \pm 3,22$
400	0.08	10%	$13,86 \pm 5,99$
800	0.2	4%	$15,02 \pm 0,05$
1000	0.23	4%	$17,44 \pm 0,23$
2000	0.3	4%	$28,81 \pm 0,07$



(a) Case 1 (b) Zoom of Case 1

(c) Case 2 (d) Case 3

Fig. 1. (a),(c) and (d) show the partition of the parameter space for Cases 1, 2, and 3 respectively. The positive area $\mathcal{P}_\alpha$ is depicted in red, the negative area $\mathcal{N}_\alpha$ is in blue and the undefined region $\mathcal{U}_\alpha$ is in yellow. (a) and (c) are one dimensional case: in the x-axis we report the parameter explored (respectively k_i and k_r), on the y-axis we show the value of the satisfaction function and the confidence bounds (for $\beta_\delta = 3$). The green horizontal line is the threshold $\alpha = 0.1$ (d) shows a two dimensional parameter space, hence no confidence bound has been represented. The circle dot represent the training set. In (b) we have zoomed a portion of the parameter space of (a) to visualize the cells with base length equals to h and height equal to the span of the confidence bounds. (Color figure online)

5 Conclusions

We presented an efficient statistical algorithm for parameter synthesis, to identify parameters satisfying MITL specifications with a probability greater than a certain threshold. The algorithm is based on Bayesian statistics and leverages the powerful parametric verification framework of Smoothed Model Checking, integrating it into an active learning refinement loop which drives the computational effort of simulations near the critical region concentrated around the threshold α. The developed approach shows good performance in terms of execution time and outperforms the exact algorithm developed in [7], retaining good accuracy at the price of having only statistical guarantees.

Note that we compared with the performance of [7] and not of their GPU implementation [12], as our method uses only CPU computing power at the

moment. However, it can be implemented on a GPU, leveraging e.g. [31]. We expect a substantial increase of the performance. Fully distributing on CPU the computations of the algorithm, beyond only stochastic simulation, is also feasible, the hard part being to parallelise GP inference [32].

Other directions for future work include the implementation of the adaptive grid strategy to construct the $\mathcal{P}_\alpha$, $\mathcal{N}_\alpha$, and $\mathcal{U}_\alpha$ regions, given the output of the smMC, and a divide and conquer strategy to split the parameter space (and the uncertain set $\mathcal{U}_\alpha$) in subregions, to reduce the complexity of the smMC. These two extensions are mandatory to scale the method in higher dimensions, up to 6–8 parameters. To scale even further, we plan to integrate techniques to speed up GP reconstruction: more classical sparsity approximation techniques [10] and more recent methods for GPs tailored to work on grids [33,34]. This techniques have a computational cost of $O(n)$ instead of standard implementation which costs $O(n^3)$. Finally, we aim to combine our approach with the exact algorithm developed in [7]. The idea is to use our approach for a rough exploration of the parameter space to cut out the region with higher statistical confidence to be higher or lower than the considered threshold, applying the exact approach in the remain area, when feasible.

References

1. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**(25), 2340–2361 (1977)
2. Bortolussi, L., Hillston, J., Latella, D., Massink, M.: Continuous approximation of collective systems behaviour: a tutorial. Perform. Eval. **70**(5), 317–349 (2013)
3. Haas, P.J.: Stochastic Petri nets for modelling and simulation. In: Winter Simulation Conference, vol. 1 (2004)
4. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
5. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A storm is coming: a modern probabilistic model checker. arXiv preprint (2017). arXiv:1702.04311
6. Bortolussi, L., Milios, D., Sanguinetti, G.: Smoothed model checking for uncertain continuous-time markov chains. Inf. Comput. **247**, 235–253 (2016)
7. Češka, M., Dannenberg, F., Paoletti, N., Kwiatkowska, M., Brim, L.: Precise parameter synthesis for stochastic biochemical systems. Acta Informatica **54**(6), 589–623 (2017)
8. Bortolussi, L., Policriti, A., Silvetti, S.: Logic-based multi-objective design of chemical reaction networks. In: Cinquemani, E., Donzé, A. (eds.) HSB 2016. LNCS, vol. 9957, pp. 164–178. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47151-8_11
9. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12
10. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press, Cambridge (2006)

11. Češka, M., Dannenberg, F., Kwiatkowska, M., Paoletti, N.: Precise parameter synthesis for stochastic biochemical systems. In: Mendes, P., Dada, J.O., Smallbone, K. (eds.) CMSB 2014. LNCS, vol. 8859, pp. 86–98. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12982-2_7
12. Češka, M., Pilař, P., Paoletti, N., Brim, L., Kwiatkowska, M.: PRISM-PSY: precise GPU-accelerated parameter synthesis for stochastic systems. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 367–384. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_21
13. Bartocci, E., Bortolussi, L., Nenzi, L., Sanguinetti, G.: System design of stochastic models using robustness of temporal properties. Theoret. Comput. Sci. **587**, 3–25 (2015)
14. David, A., Du, D., Guldstrand Larsen, K., Legay, A., Mikučionis, M.: Optimizing control strategy using statistical model checking. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 352–367. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38088-4_24
15. Han, T., Katoen, J.P., Mereacre, A.: Approximate parameter synthesis for probabilistic time-bounded reachability. In: Proceedings of Real-Time Systems Symposium, pp. 173–182 (2008)
16. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM **43**(1), 116–146 (1996)
17. Barbot, B., Chen, T., Han, T., Katoen, J.-P., Mereacre, A.: Efficient CTMC model checking of linear real-time objectives. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 128–142. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_12
18. Jha, S.K., Langmead, C.J.: Synthesis and infeasibility analysis for stochastic models of biochemical systems using statistical model checking and abstraction refinement. Theoret. Comput. Sci. **412**(21), 2162–2187 (2011)
19. Katoen, J.P.: The probabilistic model checking landscape. In: LICS, pp. 31–45 (2016)
20. Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Bruintjes, H., Katoen, J.-P., Ábrahám, E.: PROPhESY: a probabilistic parameter synthesis tool. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 214–231. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_13
21. Bortolussi, L., Sanguinetti, G.: Smoothed model checking for uncertain continuous time Markov chains. arXiv preprint. arXiv:1402.1450 (2014)
22. Henzinger, T.A., Jobstmann, B., Wolf, V.: Formalisms for specifying markovian population models. In: Bournez, O., Potapov, I. (eds.) RP 2009. LNCS, vol. 5797, pp. 3–23. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04420-5_2
23. Norris, J.R.: Markov Chains. Number 2008. Cambridge University Press, Cambridge (1998)
24. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans. Softw. Eng. **29**(6), 524–541 (2003)
25. Younes, H.L., Simmons, R.G.: Statistical probabilistic model checking with a focus on time-bounded properties. Inf. Comput. **204**(9), 1368–1409 (2006)
26. Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A bayesian approach to model checking biological systems. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 218–234. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03845-7_15

27. Brim, L., Češka, M., Dražan, S., Šafránek, D.: Exploring parameter space of stochastic biochemical systems using quantitative model checking. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 107–123. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_7
28. Steinwart, I.: On the influence of the kernel on the consistency of support vector machines. J. Mach. Learn. Res. **2**, 67–93 (2002)
29. Lam, S.K., Pitrou, A., Seibert, S.: Numba: a LLVM-based Python JIT compiler. In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, p. 7. ACM (2015)
30. Gardner, T.S., Cantor, C.R., Collins, J.J.: Construction of a genetic toggle switch in escherichia coli. Nature **403**(6767), 339–342 (2000)
31. Dai, Z., Damianou, A., Hensman, J., Lawrence, N.: Gaussian process models with parallelization and GPU acceleration. arXiv:1410.4984 (2014). [cs, stat]
32. Zhang, M.M., Williamson, S.A.: Embarrassingly parallel inference for Gaussian processes. arXiv:1702.08420 (2017). [stat]
33. Wilson, A., Nickisch, H.: Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In: International Conference on Machine Learning, pp. 1775–1784 (2015)
34. Wilson, A.G., Hu, Z., Salakhutdinov, R., Xing, E.P.: Deep kernel learning. In: Artificial Intelligence and Statistics, pp. 370–378 (2016)