Machine Learning Methods in Statistical Model Checking and System Design – Tutorial

Machine Learning Methods in Statistical Model Checking and System Design – A Tutorial Introduction *

Luca Bortolussi[1 2 3], Dimitrios Milios[4], and Guido Sanguinetti[4 5]

[1] Modelling and Simulation Group, University of Saarland, Germany
[2] Department of Mathematics and Geosciences, University of Trieste
[3] CNR/ISTI, Pisa, Italy
[4] School of Informatics, University of Edinburgh
[5] SynthSys, Centre for Synthetic and Systems Biology, University of Edinburgh

Abstract. Recent research has seen an increasingly fertile convergence of ideas from machine learning and formal modelling. Here we review some recently introduced methodologies for model checking and system design/ parameter synthesis for logical properties against stochastic dynamical models. The crucial insight is a regularity result which states that the satisfaction probability of a logical formula is a smooth function of the parameters of a CTMC. This enables us to select an appropriate class of *functional priors* for Bayesian model checking and system design. We give a tutorial introduction to the statistical concepts, as well as an illustrative case study which demonstrates the usage of a newly-released software tool, U-check, which implements these methodologies

1 Introduction

Verification of temporal logic formulae via model checking is one of the major success stories of theoretical computer science [1]. An important development has been the introduction of probabilistic model checking [2,3,4], which aims to verify logical formulae on trajectories of stochastic processes such as Continuous Time Markov Chains (CTMCs): here, due to the intrinsic stochasticity of the system and the fact that formulae may evaluate differently on different trajectories of the same system, the purpose of probabilistic model checking is therefore to quantify the probability of a formula being true. From the theoretical point of view, probabilistic model checking has stimulated a remarkable cross-fertilisation between applied mathematics and computer science, resulting in a renaissance in algorithms for computing transient probabilities in Markovian processes [5]. From the practical point of view, the impact of probabilistic model

* D.M. and G.S. acknowledge support from the ERC under grant MLCS 306999. L.B. acknowledges partial support from EU-FET project QUANTICOL (nr. 600708), by FRA-UniTS, and by the German Research Council (DFG) as part of the Cluster of Excellence on Multimodal Computing and Interaction at Saarland University.

checking has arguably been even greater, since ideas from verification and formal modelling could now be applied to a wide array of models from the physical sciences and engineering disciplines. Partly as a result of these developments, formal modelling is now a major player in applications as diverse as systems biology, ecology, performance modelling and smart cities.

While probabilistic model checking is undoubtedly a major player in all applications of formal modelling, it is not without its challenges. While many elegant algorithms for computing transient probabilities have been developed, exact probabilistic model checking is still computationally too demanding to be deployed on many realistic problems. *Statistical* model checking (SMC) algorithms are often employed in these circumstances [6,7,8] if the underlying system can be simulated efficiently, one can simply draw several independent trajectories from the system, evaluate the formula of interest on each trajectory and obtain in this way a Monte Carlo estimate of the desired probability. However, in many applications, a more fundamental difficulty is encountered, as models are often incompletely specified, relying on a parametrisation which can be highly uncertain. For example, in a systems biology application a parameter may represent a kinetic reaction rate which can only be measured with considerable approximation; in a smart-city application, a parameter may model how a group of transport users may behave in a hypothetical scenario, which cannot be accurately measured before the scenario is actually implemented. In many cases, therefore, it is of primary importance not only to quantify the probability that a trajectory of the system will satisfy a certain property, but also how this probability may depend on uncertain parameter, and how to select parameter values which will (robustly) yield consistent behaviour. While direct parameter exploration is in some cases possible [9,10], it is always computationally intensive.

Recently, we proposed a novel family of algorithms for statistical model checking and parameter synthesis on CTMC models with parametric uncertainty, which can achieve considerable computational savings over parameter exploration methods [11,12,13,14,15]. Our methods are theoretically grounded on a novel characterisation of the functional dependence of the satisfaction probability of a formula on the parameters. This regularity result enables us to use powerful methodologies from Bayesian machine learning, obtaining efficient algorithms with theoretical guarantees. Recently, some of these methods have been implemented in the open source U-check software suite [16], a flexible tool which can interface with some of the most widely used modelling languages.

In this paper, we provide a tutorial introduction to these novel algorithms and their use. We focus on providing an accessible introduction to the relevant statistical machine learning concepts, as well as demonstrating the use of the U-check tool on a non-trivial case study. The rest of the paper is organised as follows: we start by reviewing briefly basic concepts of temporal logics and (Bayesian) statistical model checking. We then introduce the notion of satisfaction function for models with parametric uncertainty, and introduce Gaussian processes, a suitable class of functional priors. GPs are at the heart of Smoothed model checking, a Bayesian statistical algorithm that directly performs model check-

ing of the whole satisfaction function. We then discuss the parameter synthesis problem and introduce the GP-UCB algorithm, a provably globally convergent optimisation algorithm which is at the basis of the parameter synthesis routines in U-check. Finally, we illustrate the use of the U-check tool on an example, with the aim of facilitating the use of these novel tools in practical applications.

2 CTMCs, temporal logics and statistical model checking

2.1 Continuous-Time Markov Chains

In this paper we will be mostly concerned with Continuous-Time Markov Chains (CTMC), which are memoryless stochastic processes on a countable state space evolving in continuous time [17]. CTMC are the most widespread class of stochastic models in many areas, including performance and systems biology. In these domains, they usually take the form of Population CTMC (PCTMC), describing how a population of agents evolves in time [18]. Typically, the state of a PCTMC $\mathcal{M}$ is represented as a vector of integer-valued variables $\boldsymbol{X} = (X_1, \ldots, X_n)$, while the dynamical evolution is represented by a set of transition classes. These specify events changing the state of some agents, and are easily represented in the biochemical reaction style as follows:

$$r_1 X_1 + \ldots + r_n X_n \xrightarrow{\alpha(\boldsymbol{X},\theta)} s_1 X_1 + \ldots + s_n X_n.$$

Each such a rule specifies that r_1 agents in state X_1, r_2 in state X_2, and so on, interact and are replaced by s_1 copies of X_1 agents, s_2 of X_2 agents, and so on. Hence, the net change of agents is given by the update vector $\boldsymbol{v}$ defined by $v_i = s_i - r_i$. Each reaction happens with a rate or frequency given by the function $\alpha(\boldsymbol{X}, \theta)$, depending on the current state of of the system and on a vector of parameters θ.

Example. In order to illustrate the description of a PCTMC, we consider here a simple example from system biology, describing the (uncontrolled) transcription and translation of a gene into a protein. We need two variables, X_m and X_p, counting the amount of messenger RNA and of protein in the system. We further need four transition classes: transcription of DNA into mRNA ($\emptyset \xrightarrow{k_p} X_m$) , degradation of mRNA ($X_m \xrightarrow{k_{dm} X_m} \emptyset$), translation of mRNA into the protein $X_m \xrightarrow{k_t X_m} X_p$, and degradation of the protein $X_p \xrightarrow{k_{dp} X_p} \emptyset$. In all cases, we assume a mass action rate, proportional to the amount of reactants involved. Note that the state space of this model is the countably infinite set $\mathbb{N}^2$.

Uncertain PCTMC. The dynamic behaviour of a PCTMC $\mathcal{M}$ can heavily depend on the parameters θ, a fact we make explicit in the notation $\mathcal{M}_\theta$. As discussed in the introduction, the values of θ are seldom known exactly, hence often we can only assume that they belong to a certain compact, connected subset $\mathcal{D}$ of

$\mathbb{R}^d$, where d is the dimension of the parameter space. By varying $\theta \in \mathcal{D}$, we have a family of PCTMC models, which we will refer as an *uncertain CTMC*.

We conclude this section noting that we can consider different classes of stochastic models, including Stochastic Differential Equations and Stochastic Hybrid Systems. We refer the reader to [12,15,14] for a more detailed discussion in this sense.

2.2 Metric Temporal Logic

In order to describe behavioural properties of biological and complex systems we will consider formal languages satisfying two main constraints:

- the language should capture properties of single executions of the system, as this is the only way we can experimentally observe such a system;
- properties should be time-bounded, as we can observe a real system only for a finite amount of time.

In this paper, we stick to the linear-time temporal-logic based formalism of Metric Interval Temporal Logic [19,20], which is defined by the following syntax:

$$\varphi := \mathtt{tt} \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathbf{U}_{[a,b]} \varphi_2.$$

Atomic predicates $\mu = \mu(\boldsymbol{x}(t))$ are evaluated pointwise on time bounded trajectories $\boldsymbol{x} : [0,T] \to \mathbb{R}^n$, and are usually boolean inequalities of the form $f(\boldsymbol{x}(t)) \geq 0$. Boolean operators work as usual, while the time-bounded until holds in a trajectory at time t_0 whenever the formula φ_1 is satisfied from t_0 to a time point $t \in [t_0 + a, t_0 + b]$, at which φ_2 must hold. The time-bounded eventually and always operators are definable as usual: $\mathbf{F}_{[a,b]}\varphi := \mathtt{tt}\mathbf{U}_{[a,b]}\varphi$ and $\mathbf{G}_{[a,b]}\varphi := \neg\mathbf{F}_{[a,b]}\neg\varphi$. MITL can be given a boolean semantics in a standard way, see[19,20] for details. Furthermore, it can be assigned a quantitative satisfaction score along the lines of [21,22]. This semantics is obtained by essentially using maxima (minima) and suprema (infima) in place of conjunction (disjuction) and universal (existential) quantification, respectively, and by replacing atomic predicates $\mu(\boldsymbol{x}(t)) := f(\boldsymbol{x}(t)) \geq 0$ with the real value $f(\boldsymbol{x}(t))$. It associates with a formula φ and a trajectory $\boldsymbol{x}$ a real number $\rho(\varphi, \boldsymbol{x})$ whose sign is associated with satisfiability of φ (true if and only if $\rho(\varphi, \boldsymbol{x})$ is positive), and whose absolute value measures how robustly the formula is satisfied by the trajectory. Both boolean and quantitative semantics can be efficiently checked on sample trajectories by monitor algorithms, see [20,22].

The boolean semantics of MITL can be extended to stochastic models by considering the probability $p(\varphi) = p(\varphi = \mathtt{tt})$ of the set of trajectories which satisfy a formula φ. For the quantitative semantics, instead, this extension produces a probability distribution over real numbers, see [15,13] for further details.

We remark here that the use of MITL is not mandatory: any linear-time, time bounded formalism will do, provided it is equipped with a monitoring routine.

2.3 Statistical Model Checking

The goal of probabilistic model checking is to compute the satisfaction probability of a MITL formula (in accordance with the boolean semantics) for a given stochastic (PCTMC) model. Numerical algorithms for this problem [23] are prohibitively costly, hence the standard approach is to rely on statistical methods. Basically, one combines a simulation algorithm for the stochastic process (e.g., Gillespie's algorithm [24] for PCTMC) with a monitor routine for the MITL formula φ, thus generating samples from a Bernoulli random variable with probability $p(\varphi)$. Then, standard stastistical tools can be used to obtain an estimate $\hat{p}$ of $p(\varphi)$, with a given error and confidence, or to test if $p(\varphi)$ is greater or smaller than a threshold q, see [6,8,7]. Of particular interest in the context of this paper are Bayesian methods [8], which assume a prior distribution over $p(\varphi)$, and compute the posterior distribution, given the observed Bernoulli samples. Typically, this prior is the Beta distribution, which is conjugate [25] with the Bernoulli distribution, meaning that the posterior is still a Beta and hence analytically computable [8].

When the quantitative semantics is concerned, pipelining a simulation algorithm for PCTMC with a monitor will produce samples of a real valued random variables, which can again be analysed with statistical means. In particular, in [15,13], the authors focus on the average quantitative score as a measure of robustness of the property in the stochastic model, which can be estimated by standard statistics.

3 Smoothness and functional priors

3.1 Satisfaction functions

We now switch our attention to examine the behaviour of the truth probability of a formula as we change the model within a parametric family of models. The scenario we consider is the following: let φ being a proposition in a suitable temporal logic (e.g. MITL) which we wish to verify over the trajectories of a stochastic process. Let $\mathcal{M}_\theta$ be an *uncertain CTMC* whose transition rates depend on a set of parameters $\theta \in \mathcal{D}$ where $\mathcal{D}$ is a connected, compact domain in $\mathbb{R}^d$, as in Section 2.1. For a fixed value of the parameters θ, model checking the formula φ would return the probability $p_\theta(\varphi) = p(\varphi = \mathtt{tt} \mid \mathcal{M}_\theta)$ that the formula will be evaluated as true on a randomly sampled trajectory of the system. This procedure therefore defines a function from the parameters domain $\mathcal{D}$ to the interval $[0, 1]$. We can formalise this in the following definition.

Definition 1. *Let $\mathcal{M}_\theta$ be an uncertain CTMC indexed by the variable $\theta \in \mathcal{D}$, and let φ be a temporal logic formula. The* satisfaction function $f_\varphi \colon \mathcal{D} \to [0, 1]$ *associated with φ and $\mathcal{M}_\theta$ is*

$$f_\varphi(\theta) = p(\varphi = \mathtt{tt} \mid \mathcal{M}_\theta)$$

i.e., with each value θ in the space of parameters $\mathcal{D}$ it associates the satisfaction probability of φ for the model with that parameter value.

The following theorem characterises the dependence of the satisfaction function on its parameters, and was proved in [12].

Theorem 1. *Let φ be a formula in a suitable temporal logic and let $\mathcal{M}_\theta$ be an uncertain CTMC indexed by the variable $\theta \in \mathcal{D}$. Denote as $\boldsymbol{\alpha}(\boldsymbol{X}, \theta)$ the transition rates of the CTMC and assume that these depend smoothly on the parameters θ and polynomially on the state vector of the system $\boldsymbol{X}$. Then, the satisfaction function of φ is a smooth function of the parameters, $f_\varphi \in \mathcal{C}^\infty(\mathcal{D})$.* □

3.2 Prior distributions over smooth functions – Gaussian Processes

Our discussion in Section 3.1 highlights the fact that, for uncertain CTMCs, the concept of satisfaction probability must be replaced with a functional analogue, which takes into account the influence that model parameters may have on the satisfaction probability of the formula. From the statistical model checking perspective, this suggests that Monte Carlo estimation should be replaced by function approximation. We will retain a Bayesian perspective in this paper, and construct a statistical model checking method based on Bayesian functional approximation: this requires the definition of a suitable class of probability distribution over functions. Our theoretical analysis in Theorem 1 enabled us to conclude that the satisfaction function is a smooth function of its arguments, the model parameters: a natural choice of prior distribution over smooth functions is a *Gaussian Process* (GP [26]). Formally, the definition of a GP is as follows:

Definition 2. *A GP is a collection of random variables indexed by an input variable x such that every finite dimensional marginal distribution is a multivariate normal distribution.*

In practice, a sample from a GP is a random function; the random vector obtained by evaluating a sample function at a finite set of points $x_1, \ldots, x_N$ is a multivariate Gaussian random variable. A GP is uniquely defined by its *mean* and *covariance* functions, denoted by $\mu(x)$ and $k(x, x')$; the mean vector (covariance matrix) of the finite dimensional marginals are given by evaluating the mean (covariance) function on every (pair of) point in the finite sample. Naturally, by subtracting the mean function to any sample function, we can always reduce ourselves to the case of *zero mean* GPs; in the following, we will adopt this convention and ignore the mean function.

A popular choice for the covariance function, which we will also use, is the *squared exponential* covariance function

$$k(x, x') = \sigma^2 \exp\left[-\frac{(x - x')^2}{\lambda^2}\right] \tag{1}$$

with two hyper-parameters: the amplitude σ^2 and the characteristic length scale λ^2 [26].

The covariance function endows the space of samples from a GP with a metric: this is an example of a Reproducing Kernel Hilbert Space (RKHS). A

complete characterisation of such spaces is non-trivial; for our purposes, however, it is sufficient to show that their expressivity is sufficient to approximate a satisfaction function by a sample from a GP. The following result is a simple corollary of the results in [27]:

Theorem 2. *Let f be a continuous function over a compact domain $D \in \mathbb{R}^p$. For every $\epsilon > 0$, there exists a sample ψ from a GP with squared exponential covariance such that*

$$\|f - \psi\|_2 \leq \epsilon$$

where $\|\cdot\|_2$ denotes the L_2 norm.

The results of Theorems 1 and 2 jointly imply that the satisfaction function of a formula can be approximated arbitrarily well by a sample from a GP, justifying the use of GPs as priors for the satisfaction function.

4 Smoothed model checking

To see how this fact enables a Bayesian statistical model checking approach *directly at the level of the satisfaction function*, we need to explain the basics of posterior computation in GP models. Let x denote the input value and let $\hat{\mathbf{f}} = \{\hat{f}_1, \ldots, \hat{f}_N\}$ denote observations of the values of the unknown function f at input points $x_1, \ldots, x_N$. We are interested in computing the distribution over f at a *new* input point x^* *given* the observed values $\hat{\mathbf{f}}$, $p(f(x^*)|\hat{\mathbf{f}})$. A priori, we know that the true function values at any finite collection of input points is Gaussian distributed, hence $p\left(f(x^*), f(x_1), \ldots, f(x_N)\right) = \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, with $\boldsymbol{\mu}$ and Σ obtained from the mean and covariance function as explained before. This prior distribution can be combined with likelihood models for the observations, $p(\hat{f}|f)$ in a Bayesian fashion to yield a joint posterior

$$p\left(f(x^*), f(x_1), \ldots, f(x_N)|\hat{\mathbf{f}}\right) = \frac{1}{Z} p\left(f(x^*), f(x_1), \ldots, f(x_N)\right) \prod_i p(\hat{f}_i|f(x_i))$$

where Z is a normalisation constant. The desired posterior predictive distribution can then be obtained by integrating out (*marginalising*) the true function values $f(x_1), \ldots, f(x_N)$

$$p(f(x^*)|\hat{\mathbf{f}}) = \int \prod_{i=1}^{N} df(x_i) p\left(f(x^*), f(x_1), \ldots, f(x_N)|\hat{\mathbf{f}}\right). \tag{2}$$

Equation (2) plays a central role in non-parametric function estimation; the inference procedure outlined above goes under the name of *GP regression*. It is important to note that, in the case of Gaussian observation noise, the integral in equation (2) can be computed in closed form. For further details see e.g. [26].

Important remark: GP regression provides an analytical expression for the predicted mean and variance of the unknown function at all input points.

In our case, observations of the satisfaction function are obtained through boolean evaluations of a formula over individual trajectories at isolated parameter values. The satisfaction of a formula φ over a trajectory generated from a specific parameter value θ is a Bernoulli random variable with success probability $f_\varphi(\theta)$. In order to map this probability to the real numbers, we introduce the *inverse probit* transformation

$$\psi(f) = g \Leftrightarrow f = \int_\infty^g \mathcal{N}(0,1) \quad \forall f \in [0,1], g \in \mathbb{R}$$

where $\mathcal{N}(0,1)$ is the standard Gaussian distribution with mean zero and variance 1. The function $g_\varphi(\theta) = \psi(f_\varphi(\theta))$ is by construction a smooth, real valued function of the model parameters, and can therefore be modelled as a draw from a GP.

We can summarise the algorithm as follows: we draw m binary evaluations of satisfaction at each of N parameter values $\theta_1, \ldots, \theta_N$; these are collected in a (binary) data matrix $\mathcal{D} = [\mathbf{d}_1, \ldots, \mathbf{d}_N]$ whose rows $\mathbf{d}_i$ are the boolean m-vectors of evaluations at θ_i. By construction, at each θ_i value the observations are independent draws from a $Binomial(m, f_\varphi(\theta))$. The inverse probit transform of the satisfaction function $g_\varphi(\theta) = \psi(f_\varphi(\theta))$ is a smooth function of the parameters and is assigned a GP prior. Denote as $(g^*, \mathbf{g})$ the vector containing the values of g_φ at a new parameter value θ^* and at the training parameter values $\theta_1, \ldots, \theta_N$. Using Bayes theorem and the marginalisation property (2) of GPs, the posterior estimate of $g_\varphi(\theta^*)$ at a new parameter value is given by

$$p\left(g_\varphi(\theta^*)|\mathcal{D}\right) \propto \int d\mathbf{g} \mathcal{N}\left((g^*, \mathbf{g})|\mathbf{0}, \Sigma\right) \prod_{i=1}^{N} \left(f_\varphi(\theta_i)\right)^{\sum \mathbf{d}_i} \left(1 - f_\varphi(\theta_i)\right)^{m - \sum \mathbf{d}_i}. \quad (3)$$

Computing this posterior distribution can be done accurately and efficiently using the Expectation-Propagation algorithm described in [12].

5 Learning and designing systems from logical constraints

Smoothed model checking provides an effective algorithm for approximating the satisfaction function of a formula, in other words, to examine the sensitivity of the formula's truth probability to the parameters of the uncertain CTMC. A related set of problems, which can also benefit from a machine learning approach, is concerned with reducing the uncertainty in the model, either by incorporating observations of the system, or by enforcing requirements in a system design scenario. Parameter synthesis from observations of the state of a stochastic process is a well studied problem in computational statistics and machine learning (see, e.g. [28,29]). Here we focus instead on the less studied problem where observations are truth values of a (set of) formula over individual realisations of the process (trajectories). The two questions we aim to address are the following:

1. **Learning problem** Given truth evaluations of a set of formulae over independent individual realisations of a stochastic process, how can we find the parameter set which optimises their probability (maximum likelihood)?

2. **Probabilistic design problem** Given a desired target (joint) probability distribution for the truth of a (set of) formula, how can we find the parameter set for a system which will optimally meet these requirements?
3. **Robust design problem** Given a behavioural requirement expressed as a MITL formula, how can we find the parameter set for a system which will satisfy this requirement as robustly as possible?

In this section, we review a GP-based approach to address these problems, which was first presented in [11,14] for qualitative semantics (problems 1 and 2) and in [13,15] for quantitative semantics (robustness of a formula, problem 3).

5.1 Observations, constraints and objective functions

All of these problems can be effectively addressed as optimisation problems; the first step is therefore to define a suitable objective function. We focus our description on the Learning problem with qualitative semantics as it is the most direct in terms of exposition; similar general considerations apply in the other scenarios, although additional technical difficulties are encountered for quantitative semantics [13,15].

The first step in setting up an optimisation procedure is to define an objective function. Let $\varphi_1, \ldots, \varphi_M$ be the formulae being monitored over system trajectories, and arrange in the $M \times N$ *design matrix* D the observed truth values of the M formulae over N independent trajectories. Assuming one could access the true joint satisfaction function of the formulae $p(\varphi_1, \ldots, \varphi_M|\theta)$ (an 2^M vector valued function of the parameters), a natural objective function would be the *log-likelihood* function

$$\mathcal{L}(D, \theta) = \sum_{j=1}^{N} \log[p(\varphi_1(T_j), \ldots, \varphi_M(T_j)|\theta)] \tag{4}$$

where $p(\varphi_1(T_j), \ldots, \varphi_M(T_j)|\theta)$ denotes the entry of the function $p(\varphi_1, \ldots, \varphi_M|\theta)$ corresponding to the actual truth values observed on trajectory j. If the joint satisfaction function is known analytically, one might be able to apply a variety of optimisation methods to identify the maximum likelihood parameter set. Unfortunately, the log-likelihood (4) is never analytically available, except in the simplest of cases.

In [11,14] we proposed an alternative, statistical approach to optimise the uncomputable log-likelihood (4). This is based on obtaining noisy estimates of the function from a limited number of SMC runs, and then adopting a reinforcement learning approach to obtain a provably optimal solution. In the next subsection, we briefly detail the algorithm we use.

Th same optimisation approach can be used for problems 2 and 3. Probabilistic design can be rephrased as the minimisation of a suitable distance function (the Jansen-Shannon divergence) between the target joint probability and the joint distribution $p(\varphi_1, \ldots, \varphi_M|\theta)$ for a fixed θ, see [11,14] for details. Robust design, instead, has been modelled in [13,15] as the maximisation of the expected quantitative score for the formula φ expressing the desired requirement.

5.2 Optimising un-computable functions – the GP-UCB algorithm

We have seen in Section 4 that Smoothed Model Checking can provide an accurate reconstruction of the satisfaction function of a formula from a limited set of truth evaluations. A naive idea would be to plug the Smoothed Model Checking approximation in (4) and then directly optimise the resulting function. This however would be a suboptimal procedure: as we are often interested in joint probabilities, the computation of the training set for Smoothed Model Checking might become computationally intensive (intuitively, we have to estimate 2^M satisfaction functions). It is therefore advantageous to use an adaptive strategy to select the least possible number of parameter values where to evaluate the log-likelihood (4).

The key insight we adopt comes from reinforcement learning: there, one is tasked with devising an optimal strategy for an agent acting in an incompletely known environment. A central object of study in reinforcement learning is the trade-off between exploitation and exploration: the agent may settle for the best known policy so far (exploitation), or it may choose better policies potentially still unknown (exploration). Bayesian optimisation algorithms export this paradigm to the world of optimisation by constructing a statistical model of the unknown function which not only can predict the unknown function values, but also quantify the uncertainty in the unknown function values.

More specifically, suppose one has already acquired function evaluations $\hat{\mathbf{f}}$ at a number of initial training points $x_1, \ldots, x_N$. In order to chose a subsequent point, we first construct a GP model of the unknown function f by using GP regression. To achieve a trade-off between exploration and exploitation, one then optimises a *quantile* of the process (rather than the mean): in this way, rather than choosing a point where the expectation is maximal, one chooses a point where the function could be even greater. Formally, we introduce the concept of *acquisition function*, an auxiliary function constructed from the statistics of the GP model which is optimised to obtain the next evaluation point. Let the unknown function $f \sim \mathcal{GP}(\mu, k)$ and let μ_N and β_N be the posterior mean and variance after N observations. The acquisition function we will use is defined as

$$\alpha_N(x) = \mu_N(x) + \lambda_N \beta_N(x) \tag{5}$$

where λ_N is a constant factor (which depends on the number of points acquired only).

The algorithm we use, called GP Upper Confidence Bound (GP-UCB), iteratively selects novel points for approximate function evaluation by optimising the auxiliary function (5), which is known analytically due to the properties of GP regression. Importantly, Srinivas et al [30] showed that the GP-UCB algorithm is globally convergent with high probability for a particular choice of the constants λ_N in (5) (which depends on the probability of globally converging). Our approach to learning from logical constraints therefore relies on applying the GP-UCB algorithm to equation (4) (or the analogous objective functions for system design/ robustness optimisation, see [14,15]).

5.3 Related work: learning formulae

Our previous description has focussed on the scenario where a fixed formula was evaluated over runs of an uncertain model. In reality, formulae may themselves come in parametric families, and there may be uncertainty over the parametrisation/ structure of the formula. This is often the case for temporal logic formulae, when the temporal bounds of formulae that best characterise a behaviour may be subject to uncertainty (e.g. an oscillator of imprecisely known period in the case of Signal Temporal Logic). In the most general case, one may have uncertainty over both model *and* formulae parameters.

This general problem can also be addressed using ideas from machine learning and in particular GPs. In [31] a general strategy was proposed where, given observations of the state of a real system, one would learn a statistical model of the system, and then optimise formulae parameters using GP-UCB to obtain formulae that could optimally characterise a system (in the sense that they would be satisfied by the system with high probability). This approach was applied to the problem of discriminating cardiac arrhythmias from electro-cardiogram data: the authors fitted hidden Markov models (HMM) to annotated sequences to learn models of the different type of arrhythmias, and then applied the GP-UCB procedure to determine temporal logic formulae which could optimally discriminate different conditions.

6 The U-check software suite

All the algorithms discussed above have been implemented in the open source tool U-check [16], available online[6]. U-check has been implemented in Java, it runs cross platform, and can be used as a Java library or as a standalone software, with a command line interface.

The simple interface of the tool takes as input an option file, which specifies the algorithm to run. The choice at the moment is between smoothed model checking (Section 4), parameter estimation from qualitative data and system design using the MITL quantitative semantics (Section 5). The option file has additional fields specifying additional properties of the algorithm, see [16] and the online documentation for further details. Furthermore, one has to specify the link to a model file and to the properties file.

U-check supports models specified in several modelling languages, some of them of common use, such as PRISM guarded commands [2] and Bio-PEPA [32]. It also supports models in the SimHyA modelling language [33]. Properties, instead, can be specified only in MITL for the moment, though a spatio-temporal extension of MITL [34] will be supported soon.

In the following section, we will illustrate the methods discussed and the use of U-check through a simple example of a virus infection.

[6] `http://homepages.inf.ed.ac.uk/dmilios/ucheck`

7 Case study: a CTMC model of viral infection

In this section, we show the statistical verification methods at work on a viral infection model appeared in [35]. This model, in particular, is stiff, which makes stochastic simulation very expensive. Therefore, any statistical method to explore the parameter space should minimise the number of simulation runs to keep the analysis efficient. In this scenario, the use of smoothed model checking and of active learning strategy for optimisation is of highest value.

The model has three counting variables keeping track of three species: the viral template T, the viral genome G, and the viral structural protein S. Its dynamics is given by the reactions of Table 1. It is assumed that nucleotides and amino acids are available at constant concentrations, and their contributions to the rate functions are encoded in the model parameters. In the initial state, we assume 1 molecule for T and zero for the rest of the species. In the experiments that follow, we vary c_n and c_a, which are coefficients that control the concentrations of nucleotides and amino acids correspondingly.

Table 1. Rate functions and default parameter values for the viral model.

Reaction	Rate Function	Kinetic Constant
$\text{nucleotides} + T \xrightarrow{k_1} G + T$	$k_1 X_T c_n$	$k_1 = 1,\ c_n = 1$
$\text{nucleotides} + G \xrightarrow{k_2} T$	$k_2 X_G c_n$	$k_2 = 0.025$
$\text{nucleotides} + \text{amino acids} + T \xrightarrow{k_3} S + T$	$k_3 X_T c_n c_a$	$k_3 = 100,\ c_a = 1$
$T \xrightarrow{k_4} \emptyset$	$k_4 X_T$	$k_4 = 0.25$
$S \xrightarrow{k_5} \emptyset$	$k_5 X_S$	$k_5 = 0.2$
$G + S \xrightarrow{k_6} V$	$k_6 X_G X_S$	$k_6 = 7.5 \times 10^{-6}$

The trajectories of the model in question are characterised by irregular fluctuations around a fixed level. We formalise this concept by the following formula that captures fluctuations of a certain magnitude:

$$\varphi = \mathbf{F}_{[100,150]}(G > T_{hi} \ \wedge \ \mathbf{F}_{[1,20]}(G < T_{lo} \ \wedge \ \mathbf{F}_{[1,20]} G > T_{hi})) \tag{6}$$

The property will be satisfied if at lest one fluctuation occurs for the genome population in the area specified by the threshold parameters T_{lo} and T_{hi}.

The PRISM specification of the viral model, the MITL properties considered, and the inputs for the experiments that follow, are distributed along with the source code of U-check.

7.1 Using the Command-Line version of U-check

The U-check executable has to be provided with a configuration file that specifies the properties of a certain experiment. The experiment options are in the form of assignments as follows:

```
OPTION = VALUE
```

where `VALUE` can be a number, a truth value, or a string, depending on the option. A comprehensive summary of the options available is given in [16], while a exhaustive description can be found in the user manual associated with the code release. We highlight the most important options required to execute the algorithms supported.

- `modelFile`: A file that contains the model specification.
- `propertiesFile`: A file that contains one or more MiTL properties.
- `observationsFile`: A file that contains qualitative observations; it is required for parameter inference from qualitative data only.
- `mode`: It can be either `inference`, `robust` or `smoothedmc`.

The parameters to be explored have to be defined by a declaration of the form:

```
parameter NAME = [A, B]
```

which implies that `NAME` is assigned with the interval between `A` and `B`.

7.2 Smoothed Model Checking

We demonstrate the configuration required to perform smoothed model checking for the viral expression model. The code that follows dictates that the c_n parameter (`k_nucleotides`) is explored in the interval $[0.8, 2]$ and c_a (`k_amino_acids`) in $[0.5, 1]$. In the `viral.mtl` we have specified the fluctuation formula in (6) for $T_{lo} = 280$ and $T_{hi} = 320$.

```
modelFile = viral.sm
propertiesFile = viral.mtl
mode = smoothedmc
parameter k_nucleotides = [0.8, 2]
parameter k_amino_acids = [0.5, 1]
endTime = 200
runs = 10
initialObservtions = 100
numberOfTestPoints = 1600
```

According to the `initialObservtions` option, the satisfaction probability will be evaluated on a grid of 100 regularly distributed parameters values. The `runs` and `endTime` options imply that for each parameter value 10 trajectories will be sampled up to time 200. The `numberOfTestPoints` option means that satisfaction function will be eventually estimated on a grid of 1600 points.

Eventually, U-check produces a csv file that contains the estimated satisfaction probabilities for the specified grid of points, as well as a matlab/octave script file that allows easy manipulation of the results. Automatic plotting for up to two dimensions is also possible via the gnuplot program. Figure 1 depicts a screenshot of the current smoothed model checking result.

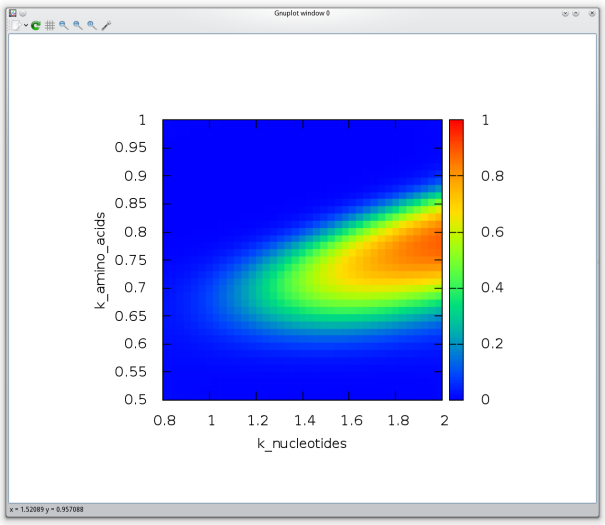

Fig. 1. U-check screenshot: Smoothed model checking result presented via gnuplot

7.3 Robust Parameter Synthesis

The configuration that follows is an example of robustness optimisation.

```
modelFile = viral.sm
propertiesFile = viral.mtl
mode = robust
parameter k_nucleotides = [0.001, 2]
parameter k_amino_acids = [0.001, 2]
endTime = 200
runs = 10
initialObservations = 40
numberOfTestPoints = 100
```

The `initialObservtions` option specifies the number of points that are required for the initialisation of the GP-UCB algorithm; these will serve as the initial training set for the GP that emulates formula robustness. The option `numberOfTestPoints` controls the size of the GP test set used in each iteration of the algorithm. Increasing this value will increase the possibility of discovering a new potential maximum, assuming there is one.

The program output that contains the solution obtained and its robustness value, as well as additional information regarding the progress of the optimisation process; a screenshot can be seen in in Figure 2. The most robust values found for c_n and c_a have been 1.929 and 0.766 correspondingly, with robustness value 20.4. This implies that the system robustly fluctuates around in the area specified by $T_{lo} = 280$ and $T_{hi} = 320$.

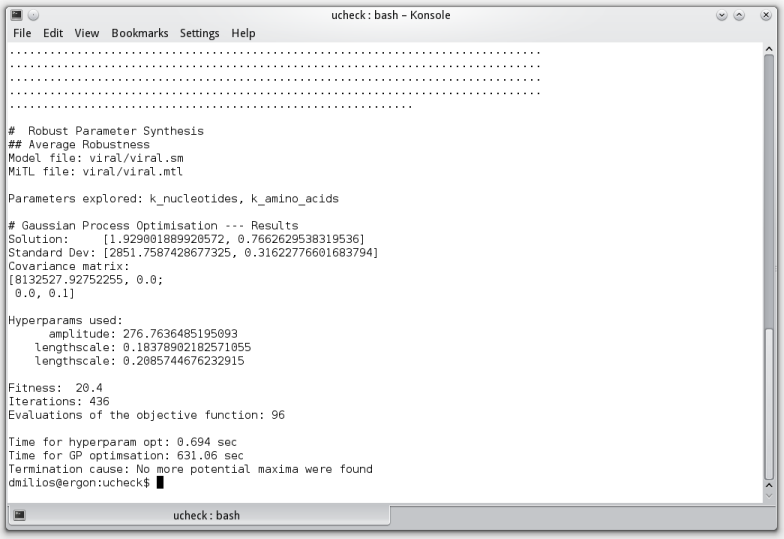

Fig. 2. U-check screenshot: Robust parameter synthesis terminal output

7.4 Learning from Qualitative Observations

We shall demonstrate U-check capability of learning the model parameters from qualitative observations on some artificial data. We make use of three variations of (6), which capture fluctuations of different magnitude. The values used for (T_{lo}, T_{hi}) have been: $(290, 310)$, $(280, 320)$ and $(270, 370)$. Artificial observations have been generated by considering $c_n = 1.5$ and $c_a = 0.75$. We have sampled 100 independent trajectories from this fixed model and performed model checking, resulting in a $n \times m$ matrix of boolean observations, where $n = 100$ and $m = 3$; these are stored in the observations file `viral.dat`. Given that the formulae are specified in a file named `viral_inference.mtl`, the following configuration will perform parameter inference:

```
modelFile = viral.sm
propertiesFile = viral_inference.mtl
observationsFile = viral.dat
mode = inference
parameter k_nucleotides = [0.001, 2]
parameter k_amino_acids = [0.001, 2]
endTime = 200
runs = 10
```

The program output is depicted in figure 3. The optimal values for c_n and c_a have been 1.445 and 0.762 correspondingly. The solution obtained is a good approximation of the actual parameters that produced the data.

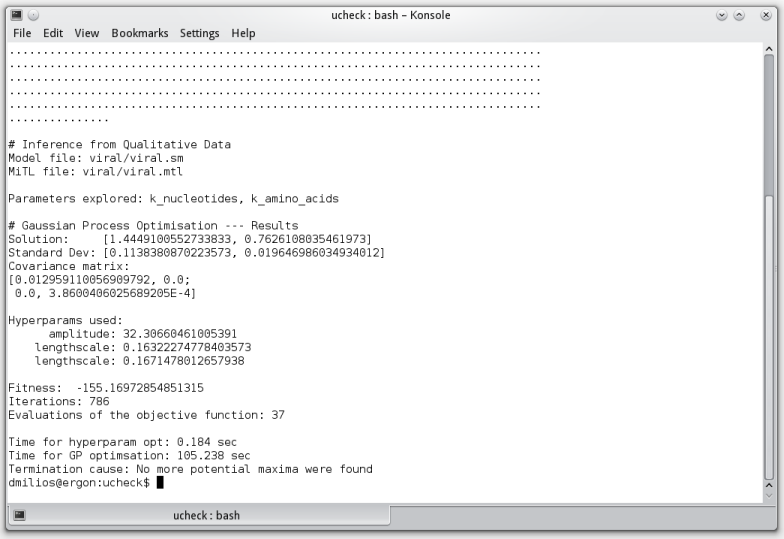

Fig. 3. U-check screenshot: Terminal output for inference from qualitative observations

8 Discussion

In this paper we gave a simple introduction to recent development at the border between machine learning and formal methods, leading to statistically grounded methods for the analysis of uncertainty in stochastic models. We used a simple but representative case study of viral infection taken from system biology, to show how these analyses can be performed with the tool U-check. The reader interested in a more exhaustive discussion about performance and accuracy of these these methods is referred to [12,14,15]. Applications of these techniques are naturally found in domains as diverse as systems and synthetic biology, cyber-physical systems, smart cities and collective adaptive systems. We believe that the methods presented here, and related approaches, can provide effective tools to investigate and design the behaviour of such systems.

Related Work. The cross fertilisation between ideas from statistical machine learning and formal methods/ verification is a novel area of research which is gaining momentum. The bulk of works of the authors discussed in this paper is part of this growing trend.

Parameter estimation using temporal logic and qualitative data has been discussed in [36], using heuristic methods based on evolutionary algorithms to explore the state space. This is in contrast with our approach [11,14], in which we rely on more advanced optimisation techniques from active learning, providing guarantees on finding the maximum likelihood estimate.

The same optimisation approach has then been used in [13,15] with the purpose of system design of stochastic models, using temporal logic specifications and leveraging its quantitative semantics.

Another problem which received considerable attention is the synthesis and parameter exploration problem, where one is interested in the satisfaction probability as a function of some parameters. Besides our statistical approach [12], there are some numerical methods based on exaustive exploration of the state space combined with error bounds [9,10]. These methods, however, are much more affected by the curse of dimensionality and by scalability issues.

A complementary problem which has been tackled with similar methods is that of learning temporal logic formulae that best characterise model properties or that explain observed data. In this respect we recall works of some of the authors [31,37], exploiting active learning, possibly combine with heuristic searches in the space of formulae. Other works dealing with learning of temporal logic specifications are, for instance, [38,39].

Another area of formal methods in which machine learning methods have a large potential is that of abstraction, recasted in a statistical sense. At the moment, these ideas have been used to speed up simulation of systems with multiple time scales [40] and systems where only a small portion is simulated explicitly, while the rest of the system is abstracted by a Gaussian Process [41]. They have also been used for modular decomposition of systems in parameter estimation tasks [42]

Finally, the integration of machine learning and formal methods is happening also at the level of modelling languages. In [43], a novel process algebra is defined, with a semantics in terms of uncertain CTMC, and equipped with inference routines to reduce parametric uncertainty in presence of observations.

References

1. C. Baier, J.-P. Katoen, Principles of Model Checking, MIT Press, 2008.
2. M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: Verification of probabilistic real-time systems, in: Proc. of CAV, 2011, pp. 585–591.
3. C. Baier, B. Haverkort, H. Hermanns, J. Katoen, Model checking continuous-time Markov chains by transient analysis, in: Proc. of CAV, 2000, pp. 358–372.
4. J.-P. Katoen, M. Khattri, I. S. Zapreevt, A markov reward model checker, in: Proc. of QEST, 2005, pp. 243–244.
5. M. Mateescu, V. Wolf, F. Didier, T. Henzinger, Fast adaptive uniformisation of the chemical master equation, IET Systems Biology 4 (6) (2010) 441–452.
6. A. Legay, B. Delahaye, S. Bensalem, Statistical model checking: An overview, in: Proceeding of RV, 2010, pp. 122–135.
7. H. L. Younes, R. G. Simmons, Statistical probabilistic model checking with a focus on time-bounded properties, Information and Computation 204 (9) (2006) 1368–1409.
8. P. Zuliani, A. Platzer, E. M. Clarke, Bayesian statistical model checking with application to simulink/stateflow verification, in: Proc. of HSCC, 2010, pp. 243–252.

9. L. Brim, M. Ceska, S. Drazan, D. Šafránek, Exploring parameter space of stochastic biochemical systems using quantitative model checking, in: Proc. of CAV, 2013, pp. 107–123.
10. M. Češka, F. Dannenberg, M. Kwiatkowska, N. Paoletti, Precise parameter synthesis for stochastic biochemical systems, in: Proc. of CMSB, 2014, pp. 86–98.
11. L. Bortolussi, G. Sanguinetti, Learning and designing stochastic processes from logical constraints, in: Proc. of QEST, 2013, pp. 89–105.
12. L. Bortolussi, D. Milios, G. Sanguinetti, Smoothed model checking for uncertain continuous time Markov chains, CoRR ArXiv 1402.1450.
13. E. Bartocci, L. Bortolussi, L. Nenzi, G. Sanguinetti, On the robustness of temporal properties for stochastic models, in: Proc. of HSB, Vol. 125 of EPTCS, 2013, pp. 3–19.
14. L. Bortolussi, G. Sanguinetti, Learning and designing stochastic processes from logical constraints, Logical Methods in Computer Science.
15. E. Bartocci, L. Bortolussi, L. Nenzi, G. Sanguinetti, System design of stochastic models using robustness of temporal properties, Theoretical Computer Science 587 (2015) Pages 3–25.
16. L. Bortolussi, D. Milios, G. Sanguinetti, U-check: Model checking and parameter synthesis under uncertainty (2015).
17. R. Durrett, Essentials of Stochastic Processes, Springer, 2012.
18. L. Bortolussi, J. Hillston, D. Latella, M. Massink, Continuous approximation of collective systems behaviour: a tutorial, Perform. Eval. 70 (5) (2013) 317–349.
19. R. Alur, T. Feder, T. A. Henzinger, The benefits of relaxing punctuality, J. ACM 43 (1) (1996) 116–146.
20. O. Maler, D. Nickovic, Monitoring temporal properties of continuous signals, in: Proc. of FORMATS, 2004, pp. 152–166.
21. A. Donzé, O. Maler, Robust satisfaction of temporal logic over real-valued signals, in: Proc. of FORMATS, 2010, pp. 92–106.
22. A. Donzé, T. Ferrere, O. Maler, Efficient robust monitoring for STL, in: Proc. of CAV, Springer, 2013, pp. 264–279.
23. T. Chen, M. Diciolla, M. Kwiatkowska, A. Mereacre, Time-bounded verification of ctmcs against real-time specifications, in: Proc. of FORMATS, 2011, pp. 26–42.
24. D. T. Gillespie, Exact stochastic simulation of coupled chemical reactions, J. Phys. Chem. 81 (25) (1977) 2340–2361.
25. C. M. Bishop, Pattern recognition and machine learning, Springer, 2006.
26. C. E. Rasmussen, C. K. I. Williams, Gaussian processes for machine learning, MIT Press, 2006.
27. I. Steinwart, On the influence of the kernel on the consistency of support vector machines, The Journal of Machine Learning Research 2 (2002) 67–93.
28. A. Andreychenko, L. Mikeev, D. Spieler, V. Wolf, Approximate maximum likelihood estimation for stochastic chemical kinetics, EURASIP Journal on Bioinformatics and Systems Biology (1) (2012) 1–14.
29. M. Opper, G. Sanguinetti, Variational inference for markov jump processes, in: Proc. of NIPS, 2007, pp. 1105–1112.
30. N. Srinivas, A. Krause, S. Kakade, M. Seeger, Information-theoretic regret bounds for Gaussian process optimisation in the bandit setting, IEEE Trans. Inf. Th. 58 (5) (2012) 3250–3265.
31. E. Bartocci, L. Bortolussi, G. Sanguinetti, Data-driven statistical learning of temporal logic properties, in: Proc. of FORMATS, 2014, pp. 23–37.
32. F. Ciocchetta, J. Hillston, Bio-PEPA: A framework for the modelling and analysis of biological systems, Theoretical Computer Science 410 (33-34) (2009) 3065–3084.

33. L. Bortolussi, V. Galpin, J. Hillston, Hybrid performance modelling of opportunistic networks, no. 85 in EPTCS, 2012, pp. 106–121.
34. L. Bortolussi, L. Nenzi, Specifying and monitoring properties of stochastic spatio-temporal systems in signal temporal logic, in: Proc. of VALUETOOLS, 2014.
35. E. L. Haseltine, J. B. Rawlings, Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics, The Journal of Chemical Physics 117 (15) (2002) 6959.
36. R. Donaldson, D. Gilbert, A model checking approach to the parameter estimation of biochemical pathways, in: Proc. of CMSB, 2008, p. 269–287.
37. S. Bufo, E. Bartocci, G. Sanguinetti, M. Borelli, U. Lucangelo, L. Bortolussi, Temporal logic based monitoring of assisted ventilation in intensive care patients, in: Proc. of ISoLA 2014, 2014, pp. 391–403.
38. E. Bartocci, R. Grosu, P. Katsaros, C. Ramakrishnan, S. A. Smolka, Model repair for probabilistic systems, in: Proc. of TACAS, 2011, pp. 326–340.
39. Z. Kong, A. Jones, A. M. Ayala, E. A. Gol, C. Belta, Temporal logic inference for classification and prediction from data, Proc. of HSCC 2014 (2014) 273–282.
40. L. Bortolussi, D. Milios, G. Sanguinetti, Efficient stochastic simulation of systems with multiple time scales via statistical abstraction (2015).
41. A. Legay, S. Sedwards, Statistical abstraction boosts design and test efficiency of evolving critical systems, in: Proc. of ISoLA, 2014, pp. 4–25.
42. A. Georgoulas, A. Clark, A. Ocone, S. Gilmore, G. Sanguinetti, A subsystems approach for parameter estimation of ode models of hybrid systems, in: Proc. of HSB, Vol. 92 of EPTCS, 2012.
43. A. Georgoulas, J. Hillston, D. Milios, G. Sanguinetti, Probabilistic programming process algebra, in: Proc. of Quantitative Evaluation of Systems, 2014, pp. 249–264.