Falsifying Oscillation Properties of Parametric Biological Models

Thao Dang
CNRS-VERIMAG, Grenoble, France
Thao.Dang@imag.fr

Tommaso Dreossi
VERIMAG, University Joseph Fourier, Grenoble, France
Dept. of Mathematics and Computer Science
University of Udine, Udine, Italy
Tommaso.Dreossi@imag.fr

We propose an approach to falsification of oscillation properties of parametric biological models, based on the recently developed techniques for testing continuous and hybrid systems. In this approach, an oscillation property can be specified using a hybrid automaton, which is then used to guide the exploration in the state and input spaces to search for the behaviors that do not satisfy the property. We illustrate the approach with the Laub-Loomis model for spontaneous oscillations during the aggregation stage of Dictyostelium.

1 Introduction

Understanding periodic responses in living organisms is an important problem since such oscillations are a common phenomenon in biology. To reveal possible molecular mechanisms underlying this phenomenon, mathematical models have been developed. These models require validation before they can be used to make predictions. Such validation is often based on a comparison between the model behavior and experimental data obtained by temporal measurements. One major difficulty in biological model validation is that biological models often require many parameters, and most parameter values are neither measurable nor available in literature. Since there are often many sets of parameter values that can match the data, parameter identification is based not only on the error between the model simulation output and the data, but also on model robustness with respect to parameter variation. From a modelling point of view, robust parameters allow the model to fit new data without compromising the fit to the previous data. From a biological point of view, with robust parameters the system is resilient to perturbations.

The focus of this work is twofold. On one hand, we are interested in studying biological oscillating behaviors. On the other hand, we want to study the influence of parameters on the system behavior, that is how much the parameters can be varied without violating a given property. Typical behavioral changes include self-oscillations (that is the developments of periodical orbits from an equilibrium) and the occurence of a bifurcation. To illustrate this, we consider a dynamical system described by the following differential equations:

$$\dot{x} = f(x,k)$$

where $x \in \mathbb{R}^n$ is the state variables, and $k \in \mathbb{R}$ is a real-valued parameter. In a more general context, the dynamics of the system can be hybrid and contain more than one parameter. To characterize the impact of parameter variation, we want to know under which condition the two systems $\dot{x} = f(x,k)$ and $\dot{x} = f(x,k')$ (under two different parameter values k and k') are qualitatively similar, that is there exists an inversible and continuous homeomorphism that maps a trajectory of one system to a trajectory of the other. In this case, an oscillating trajectory and a steady state equilibrium of one system correspond respectively to an oscillating trajectory and a steady state equilibrium of the other. When the parameter changes and reaches a value at which the behaviors are no longer qualitatively similar, such a behavioral change is often called a bifurcation and this value is called a critical value or a value of bifurcation. This can be illustrated with a linear dynamical system when the real parts of its eigenvalues change their sign under a parameter variation.

A set of parameter values is called *robust* if the system does not undergo a bifurcation under any variation of the parameter within that set. In this paper we propose to use a testing approach to analyze the robustness of biological models with respect to preservation of oscillation properties under admissible parameter variations. When applied to a model, this testing approach can be seen as systematic simulation that can check whether the model can replicate some essential behaviors observed during experiments. However, in general it can also be applied to a biological system (viewed as a black-box system). The key steps of the approach we propose are the following:

1. *Specifying the property.* A hybrid automaton $\mathcal{A}$ is used to describe the expected oscillating behaviors. We call $\mathcal{A}$ a *property automaton*. This automaton also encodes the satisfaction/violation of the property and incorporates realistic variations of the parameter values.

2. *Generating test cases for property falsification.* The generation of test cases from the automaton $\mathcal{A}$ is randomized but guided by the property, that is it favors the exploration of the trajectories leading to a violation of the property of interest.

We choose hybrid automata as specification formalism for two reasons. First, numerous phenomena in biology exhibit switching behaviors. Second, hybrid automata can naturally describe transitions between different qualitative behaviors, as we will show later. In the hybrid systems research, formal specification of oscillation properties of biological systems are considered in [3, 6].

Concerning bifurcation detection, the theory of bifurcation in smooth systems is well developed. The existing methods (such as using analysis of the eigenvalues of the Jacobian matrix [10], Routh-Hurwitz stability criteria [8, 9], the Floquet multipliers [14]) are developed for continuous systems and it is not easy to extend them to hybrid systems with discontinuities in the dynamics. Another approach (such as [7]) involves first generating the model outputs by simulation and then finding the parameters by fitting the simulation outputs to the experimental data, based on a grid over the parameter space. Our testing-based approach with a property guided search enables a quick detection without exploring a large number of parameter values. In addition, the approach has the potential to be more scalable than analytical and grid-based methods.

The rest of the paper is organized as follows. We first describe how to use hybrid automata to specify oscillation properties. This specification formalism can be applied to a large class of temporal properties due to the expressiveness of hybrid automata. We then show how to generate test cases from a property automaton for falsification purposes. The approach is applied to analyze the robustness of the Laub-Loomis model under parameter variation. This model has been proposed for describing the dynamical behavior of the molecular network underlying adenosine 3'5'-cyclic monophosphate (cAMP) [11].

2 Using Hybrid Automata to Specify Oscillation Properties

We first present a commonly used definition of hybrid automata and then show how they can be used to for oscillating property specifications.

2.1 Hybrid Automata

In the development of formal models for designing engineering systems, hybrid automata [1] emerged as an extension of timed automata [2] with more general dynamics. Unlike in a timed automaton where a clock c is a continuous variable with time derivative equal to 1, that is $\dot{c} = 1$, in a hybrid automaton its continuous variables x can evolve according to some differential equations, for example $\dot{x} = f(x)$. This allows hybrid automata to capture the evolution of a wide range of physical entities.

Definition 1 (Hybrid automaton)**.** *A* hybrid automaton *is a tuple* $\mathcal{A} = (\mathcal{X}, Q, E, F, \mathcal{I}, \mathcal{G}, (q_0, x_0))$ *where*

- $\mathcal{X} \subseteq \mathbb{R}^n$ *is the continuous state space;*
- Q *is a finite set of locations (or discrete states);*
- $E \subseteq Q \times Q$ *is a set of discrete transitions;*
- $F = \{F_q \mid q \in Q\}$ *specifies for each location a* continuous vector field. *In each location* $q \in Q$ *the evolution of the continuous variables x are governed by a differential equation* $\dot{x}(t) = f_q(x(t), u(t))$ *where* $u(\cdot) \in \mathcal{U}_q$ *is an input function of the form* $u : \mathbb{R}^+ \to U_q \subset \mathbb{R}^m$*. The set* $\mathcal{U}_q$ *is the set admissible inputs and consists of piecewise continuous functions. We assume that all* f_q *are Lipschitz continuous;*
- $\mathcal{I} = \{\mathcal{I}_q \subseteq \mathcal{X} \mid q \in Q\}$ *is a set of invariants. The* invariant *of a location q is defined as a subset* $\mathcal{I}_q$ *of* $\mathcal{X}$*. The system can evolve inside q if* $x \in \mathcal{I}_q$*;*
- $\mathcal{G} = \{\mathcal{G}_e \mid e \in E\}$ *is a set of* guards *specifying the conditions for switching between locations. For each discrete transition* $e = (q, q') \in E$, $\mathcal{G}_e \subseteq \mathcal{I}_q$*;*
- $\mathcal{R} = \{\mathcal{R}_e \mid e \in E\}$ *is a set of* reset *maps. Each transition* $e = (q, q') \in E$ *is associated with a reset map* $\mathcal{R}_e : \mathcal{G}_e \to 2^{\mathcal{I}_{q'}}$ *that defines how x may change when the automaton* $\mathcal{A}$ *switches from q to* q'*;*
- *The initial state of the automaton is denoted by* (q_0, x_0).

A state (q, x) of $\mathcal{A}$ can change in the following two ways:

1. by a *continuous evolution*, where the continuous state x evolves according to the dynamics f_q while the location q remains constant;
2. by a *discrete evolution*, where x satisfies the guard of an outgoing transition and the system changes location by taking this transition and updating the values of x accordingly to the associated reset map.

It is important to note that hybrid automata allow modelling non-determinism in both continuous and discrete evolutions. This non-determinism is useful for describing disturbance from the environment or under-specified control, as well as for taking into account imprecision in modeling.

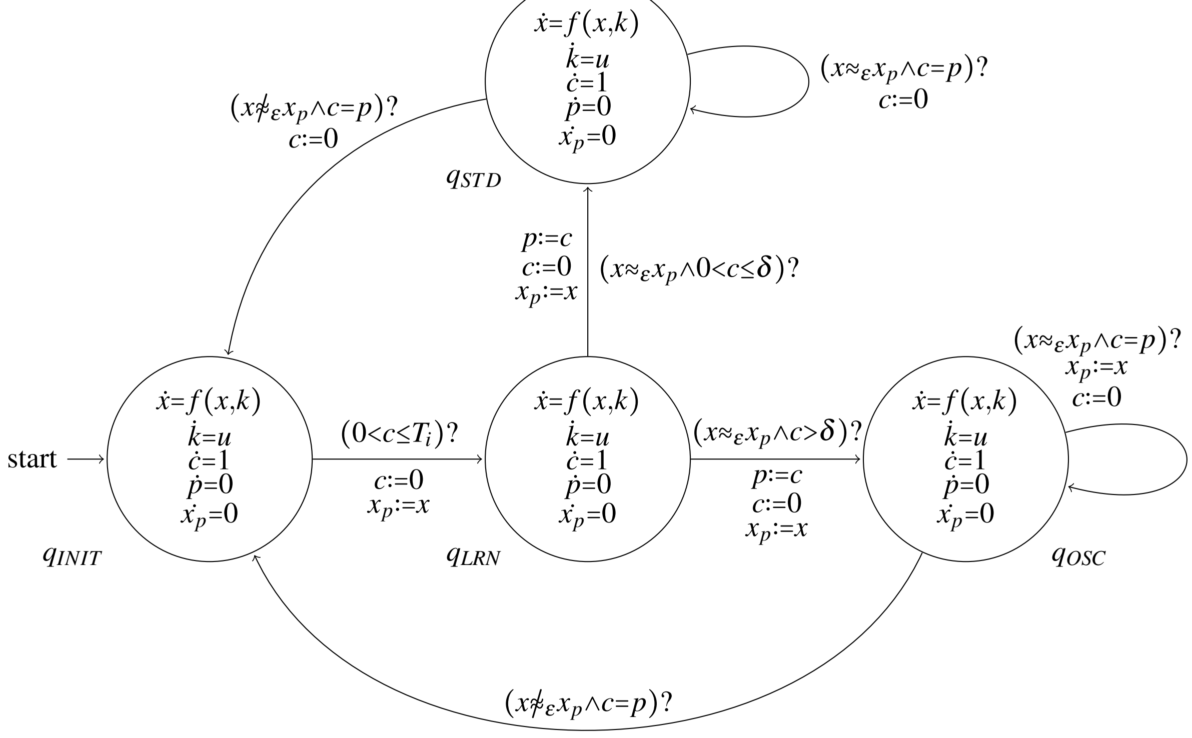

$\dot{x}=f(x,k)$ $\dot{k}=u$ $\dot{c}=1$ $\dot{p}=0$ $\dot{x_p}=0$
q_{STD}
$(x\approx_\varepsilon x_p \wedge c=p)?$ $c:=0$
$(x\not\approx_\varepsilon x_p \wedge c=p)?$ $c:=0$
$p:=c$ $c:=0$ $x_p:=x$ $(x\approx_\varepsilon x_p \wedge 0<c\leq\delta)?$
start
q_{INIT}
$(0<c\leq T_i)?$ $c:=0$ $x_p:=x$
q_{LRN}
$(x\approx_\varepsilon x_p \wedge c>\delta)?$ $p:=c$ $c:=0$ $x_p:=x$
q_{OSC}
$(x\approx_\varepsilon x_p \wedge c=p)?$ $x_p:=x$ $c:=0$
$(x\not\approx_\varepsilon x_p \wedge c=p)?$

Figure 1: An oscillation property automaton.

2.2 Property Automata

We now show how to formalize some common temporal properties of particular interest for biological systems using hybrid automata. We will call them *property automata*.

A dynamical system starting from a given initial state can evolve to a steady state or to an irregular behavior. The steady state may be stationary (that is, the system remains in the same state as time passes), which is also called an equilibrium. The system can also evolve to a periodic state (or a limit cycle). Stationary states and periodic states can be stable (that is, attracting neighboring trajectories), unstable (that is, repelling neighboring trajectories), or non-stable (saddle). The stationary and periodical states are important since they help determine the long-term behavior of the system. It is often of great interest, in particular for biological systems, to know how the stationary and periodic states change when the parameters of the system change.

Suppose that we are interested in checking whether a given dynamical system exhibits the following behavioral pattern: the system from a given initial state evolves to a limit cycle and then under a some admissible parameter perturbation it evolves only from one limit cycle to another one. In other words, this parameter perturbation does not make the system undergo a structural behavior change (or a bifurcation). Another question is to know under which parameter changes the system moves from an oscillating behavior to a steady state.

The hybrid automaton depicted in Figure 1 can be used to specify the above-described oscillating behaviors. In this property automaton, the parameters $k \in \mathbb{R}^m$ form part of the continuous state of the automaton. In the remainder of this paper, we assume that the evolution of x can be described by:

$$\dot{x} = f(x,k)$$
$$\dot{k} = u$$

where u is the input. This can be thought of as an abstract view of the dynamics of x, which can be described using complex concrete models, such as a hybrid automaton. Additionally, as we will show later, for test generation purposes, the continuous dynamics in the property automaton is abstracted away, which results in a discrete abstraction. This abstraction retains only important information about the expected temporal behavioral patterns of the variables under study.

In this example, we restrict the derivatives of the parameters to be constant and they can take values in some set U. Therefore, this allows capturing piecewise linear evolution of the parameters. It is also worth noting that one can use other classes of functions to describe the parameter change.

In addition, to describe the desired temporal behavioral pattern, we augment the continuous state of the property automaton with three special variables: c, p and x_p, where c is a clock used to measure time lapses, p is used to store the oscillation period, and x_p is used to memorize a point to which the system should return after a period.

The discrete structure of the property automaton consists of four locations q_{INIT}, q_{LRN}, q_{OSC} and q_{STD}. The location q_{INIT} corresponds to transient behaviors (between different qualitative behaviors) that can have a maximal duration T_i. After this amount T_i of transient time, the automaton jumps to q_{LRN} and while doing so, as specified by the associated reset map, it stores the current value of x in the variable x_p which is used as an expected periodic point.

The role of the location q_{LRN} is to "learn" the period of a limit cycle that the system is expected to enter. At location q_{LRN}, if after a strictly positive time δ the system returns to the point x_p, then the automaton resets the clock c after storing its value in the variable p. We use a strictly positive amount of time lapse here to exclude Zeno behaviors. Therefore, if the system has entered a limit cycle, the value of the variable p is exactly the period of that limit cycle (see Figure 2). In case the system reaches x_p after exactly δ time, the automaton switches to the location q_{STD} which is used to model a steady state (see Figure 3). Note that the test generation algorithm interacts with the system under test in discrete time, and the value of δ represents the smallest clock period that the test generation algorithm can handle.

After the learning phase at the location q_{LRN} the variable p contains the value of the expected period. When the automaton switches from the location q_{LRN} to the location q_{OSC}, the variable x_p is updated with the current value of x. At the location q_{OSC}, the automaton checks after every p time whether x returns to the periodic point x_p. There are two cases:

- If x is in the ε-neighborhood of the periodic point x_p, the system is considered oscillating and the clock is reset and the self-loop transition is traversed in order to check the next oscillation cycle.
- Otherwise, the automaton jumps back to the initial location q_{INIT}. This models the scenario where the system leaves the current limit cycle and may then evolve to another limit cycle.

To allow measurement imprecision, in the guard conditions x is not required to return exactly to x_p but to some ε-neighborhood of x_p. This is denoted by $x \approx_\varepsilon x_p$.

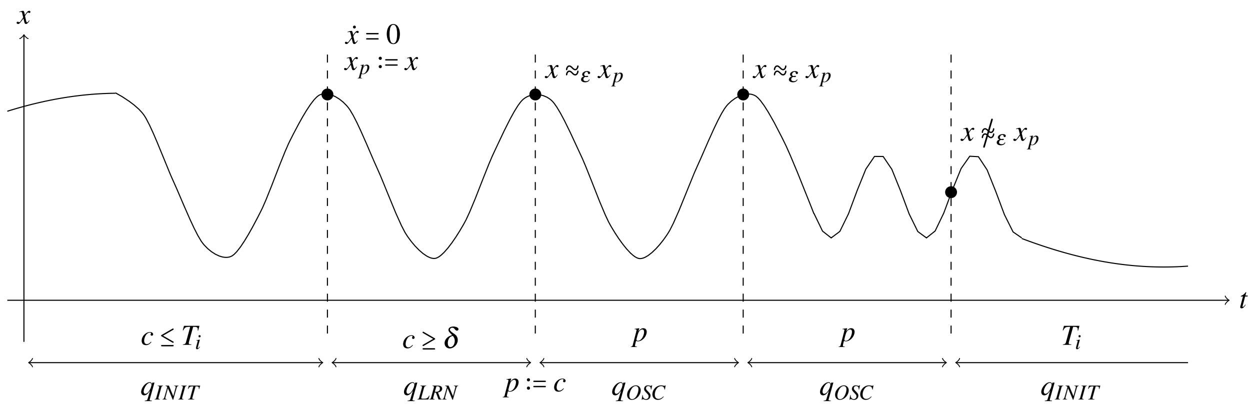


Figure 2: Detection of an oscillation using a periodic point x_p.

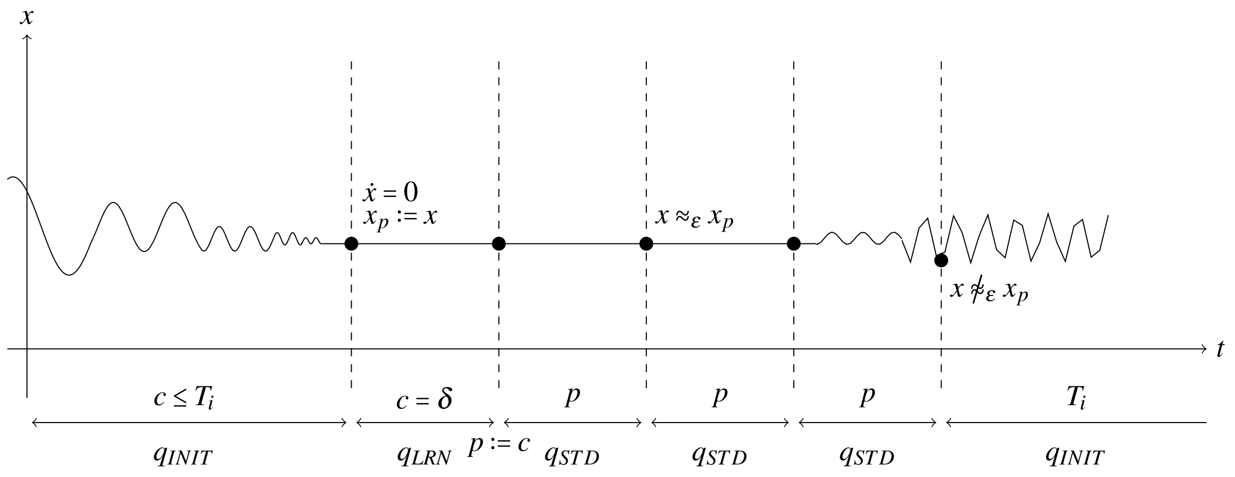


Figure 3: Detection of a steady state.

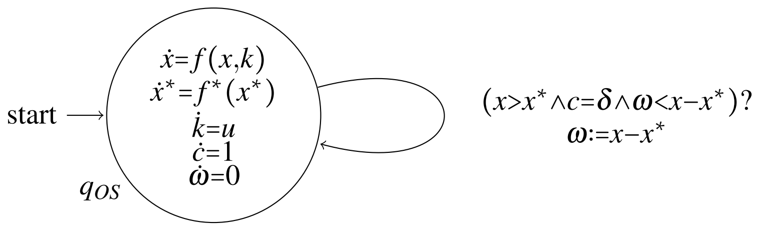


Figure 4: Overshoot detection.

Another property that can be easily expressed using hybrid automata is the maximal *overshoot*, that is the maximum peak value of x measured from a desired response of the system. The automaton has only one location q_{OS} and is equipped with an auxiliary variable ω which stores the maximum distance from the desired response x^* (see Figure 4).

3 Guided Exploration to Falsify a Property

Given a property automaton, our problem now is to explore the parameter space to detect behaviors that do not satisfy the property expressed by this automaton. To do so, we make use of the test generation algorithm gRRT [5]. This algorithm is based on the star discrepancy coverage notion and allows achieving good coverage of the reachable state space. When the objective is not to cover the whole reachable space but to quickly detect some specific behavioral patterns, we can use on top of the gRRT algorithm a property-based guiding tool. The goal of this tool is to specify some critical regions to visit and then the algorithm gRRT can be used to cover those regions. Before continuing, let us briefly recall the algorithm gRRT.

Given a hybrid automaton $\mathcal{A}$, the algorithm gRRT generates a *test case* represented by a tree where each node is associated with a state of $\mathcal{A}$ and each arc is associated with a control input action, which is either a continuous input value or a discrete control action (that is, the action of traversing a transition). Note that in the context of this work, both continuous inputs (described by u in the definition of hybrid automata) and transitions are controllable by the tester. To execute such a test case, the tester applies a control input sequence to the system, measures the variables of interest and decides whether the system under test satisfy the property. The algorithm thus can be thought as a procedure to find input signals that correspond to the beahviors we want to observe. The main steps of the coverage-guided test generation algorithm gRRT [5] are the following:

- Step 1: a goal state (q_{goal}, x_{goal}) is sampled from the state space;
- Step 2: a neighbor state (q_{near}, x_{near}) of the goal state is determined;
- Step 3: from the neighbor state, an appropriate continuous input u is applied for a time step h, or a transition is taken, in order to steer the system towards the goal state.

Step 2 can be done using a notion of distance between two hybrid states that capture the effects of discrete transitions. The choice of continuous input u in Step 3 can be done by a random selection from a discretization of the input set U. Indeed, more sophisticated methods based on trajectory sensitivity to input variation can be used but they cost more computation effort. It is important to note that a good selection of goal states is key to a good coverage result, because the success of randomized algorithms depends on finding good starting states. For a more thorough description of the algorithm gRTT and its properties, the reader is referred to [5]. For the example in Section 4 a uniform randomized selection of u is used, which already allows an efficient exploration.

To bias the goal state sampling while taking into account the property to falsify, we first construct a discrete abstraction of the property automaton $\mathcal{A}$ that reflects the expected behavioral patterns. This abstraction is then used to biased the goal state samplings, so that it favors the exploration of the behavioral patterns of interest. As an example, to falsify the oscillation property presented in the previous section, that is to show that after a given initial transient time there exists a parameter change that leads the system out of an expected limit cycle, the trajectories that lead to the transition from q_{OSC} to q_{INIT} are favored. In other words, the exploration is biased in a way to increase the probability of sampling the goal states in the guard set of this transition.

To define a discrete abstraction, we need some additional definitions. A n-dimensional predicate is defined as $\pi(x) := g(x) \sim 0$ where $g : \mathbb{R}^n \to \mathbb{R}$ is a function of n variables, and $\sim \in \{\geq, >\}$. Let λ be a function that specifies for each location $q \in Q$ a vector of m_q predicates, that is $\lambda(q) = (\pi_1, \dots, \pi_{m_q})$.

We define for each location q an abstraction function $\alpha_q : \mathcal{X} \to \mathbb{B}^{m_q}$ such that

$$\alpha_q(x) = (\pi_1(x), \dots, \pi_{m_q}(x)).$$

We say that the Boolean abstraction vector of x with respect to α_q is the Boolean vector $(\pi_1(x), \dots, \pi_{m_q}(x))$. The abstraction function α_q associated with a location $q \in Q$ partitions the set of continuous states at location q into at most 2^{m_q} subsets of continuous states such that all the continuous states in each subset have the same Boolean abstraction vector with respect to the abstraction function α_q.

In the other direction, for each location q we define the concretization function $\gamma_q : \mathbb{B}^{m_q} \to 2^{\mathcal{X}}$ such that for a given Boolean vector $b \in \mathbb{B}^{m_q}$, $\gamma_q(b) = \{x \in \mathcal{X} \mid \alpha_q(x) = b\}$.

The discrete abstraction of $\mathcal{A}$ with respect to λ is a transition system $\mathcal{D} = \{S, \rightsquigarrow, s_0\}$.

- Each location q of the hybrid automaton corresponds to a set S_q of abstract states, each of which corresponds to a pair (q,b) where $b \in \mathbb{B}^{m_q}$ is a value of the Boolean abstraction vector. For convenience, we call them q-abstract states. Two q-abstract states $s = (q,b)$ and $s' = (q,b')$ are adjacent if their corresponding sets of concrete states, that is $\gamma_q(b)$ and $\gamma_q(b')$, have non-empty intersection and they intersect only their boundaries. The whole abstract state space S is the union

$$S = \bigcup_{q \in Q} S_q.$$

- The transition relation $\rightsquigarrow \subset S \times S$ between the abstract states is defined as the union of the following two relations $\rightsquigarrow_d$ and $\rightsquigarrow_c$. Let $s = (q,b)$ and $s' = (q',b')$ be two abstract states; the transition relation between them is defined as follows:
  - $s \rightsquigarrow_c s'$ if $q = q'$ and s_1 and s_2 are adjacent.
  - $s \rightsquigarrow_d s'$ if $q \neq q'$ and $\gamma_q(s) \cap \mathcal{G}_{qq'} \neq \varnothing$ and $\mathcal{R}(\gamma_q(s) \cap \mathcal{G}_{qq'}) \subseteq \mathcal{I}_{q'}$.

  The relation $\rightsquigarrow_d$ represents the transitions in the abstract state space due to discrete switches in the original hybrid automaton $\mathcal{A}$, the relation $\rightsquigarrow_c$ represents the continuous evolution in $\mathcal{A}$.
- The initial abstract state $s_0 = (q_0, b_0)$ where $b_0 = \alpha_{q_0}(x_0)$.

The abstraction $\mathcal{D}$ can be thought of as an over-approximation of $\mathcal{A}$ since it is easy to see that any execution of $\mathcal{A}$ corresponds to an execution of $\mathcal{D}$. Moreover, it can be refined based on the exploration results in order to distinguish different qualitative behaviors that are important with respect to the property to validate.

In order for such a discrete abstraction to reflect the behavioral patterns we want to explore, we should choose for each location a set of predicates that can capture the discrete transitions of $\mathcal{A}$ and separate critical regions from the rest; therefore the set should include the predicates defining the guard and invariant conditions. This will be illustrated by the example in Section 4.

To biased the search, we use the Metropolis-Hastings method to perform a random walk [12] on $\mathcal{D}$ starting at the abstract state s_0. We first specify a target probability distribution over the abstract states

$$\pi = \{\pi_s \mid s \in S\}.$$

We then construct the following transition matrix $P(\mathcal{D})$. Between two abstract states s and s', we assign a probability to the transition from s to s':

$$\begin{cases} p_{ss'} = \dfrac{1}{deg(s)} min\{\dfrac{deg(s)\pi_{s'}}{deg(s')\pi_s}, 1\} & \text{if } s \rightsquigarrow s' \\ p_{ss'} = 1 - \sum\limits_{w \neq s} p_{sw} & \text{if } s = s' \\ p_{ss'} = 0 & \text{otherwise} \end{cases}$$

The above transition matrix $P(\mathcal{D})$ guarantees that the stationary distribution of the resulting random walk on the abstraction $\mathcal{D}$ is the target distribution π [13]. Therefore the abstract states corresponding to the region we want to visit are assigned with high target probabilities.

The Metropolis-Hastings method was proved to have good hitting times, which allows quickly reaching a desired abstract state, indeed the hitting time from s to s' of this random walk is of $\mathcal{O}(rN_v^2)$ where N_v is the number of abstract states and $r = \max\{\dfrac{\pi_s}{\pi_{s'}} \mid s, s' \in \rightsquigarrow\}$.

4 Application

4.1 Laub-Loomis Model

In this section we apply on the Laub-Loomis model [11] the techniques previously exposed. The model consists of a parametrized ODE system extracted from a molecular network that describes the aggregation stage of Dictyostelium. Our main intent is to show that for some parameter variation with bounded derivatives, the spontaneous oscillations of the system do not occur any more. Roughly speaking, we want to falsify the oscillation robustness of the system.

To this end, we derive a discrete abstraction from the property automaton in Figure 1 and guide the simulation of the ODE system towards the areas in the state space of the property automaton where the oscillation disappears. The derivatives u of the parameter variables are the inputs that we use to guide the exploration.

A revisited model that slightly differs from the original one presented by Laub and Loomis [11] is the following [8]:

$$\dot{x} = f(x,k) = \begin{bmatrix} k_1x_7 - k_2x_1x_2 \\ k_3x_5 - k_4x_2 \\ k_5x_7 - k_6x_2x_3 \\ k_7 - k_8x_3x_4 \\ k_9x_1 - k_{10}x_4x_5 \\ k_{11}x_1 - k_{12}x_6 \\ k_{13}x_6 - k_{14}x_7 \end{bmatrix}$$

Par.	Val.	Par.	Val.
k_1	2.0 min^{-1}	k_8	1.3 min^{-1}
k_2	0.9 min^{-1}	k_9	0.3 min^{-1}
k_3	2.5 min^{-1}	k_{10}	0.8 $min^{-1}\mu M^{-1}$
k_4	1.5 min^{-1}	k_{11}	0.7 min^{-1}
k_5	0.6 min^{-1}	k_{12}	4.9 min^{-1}
k_6	0.8 $min^{-1}\mu M^{-1}$	k_{13}	23.0 min^{-1}
k_7	1.0 $min^{-1}\mu M^{-1}$	k_{14}	4.5 $min^{-1}\mu M^{-1}$

Table 1: Oscillations parameter values.

The variables x correspond to seven protein concentrations: $x_1 = [ACA]$, $x_2 = [PKA]$, $x_3 = [ERK2]$, $x_4 = [REGA]$, $x_5 = [Internal\ cAMP]$, $x_6 = [External\ cAMP]$ and $x_7 = [CAR1]$. The coefficient vector $k = [k_1,\dots,k_{14}]$ contains the system parameters. Table 1 shows the parameter values for which spontaneous oscillations occur [11].

4.2 Constructing a Discrete Abstraction

In the property automaton in Figure 1, the transition from q_{OSC} to the location q_{INIT} is critical since it takes the system from an oscillation phase to a non-oscillation phase. We thus want to control the system's behavior in order to satisfy the condition $(x \not\approx_\varepsilon x_p)$?.

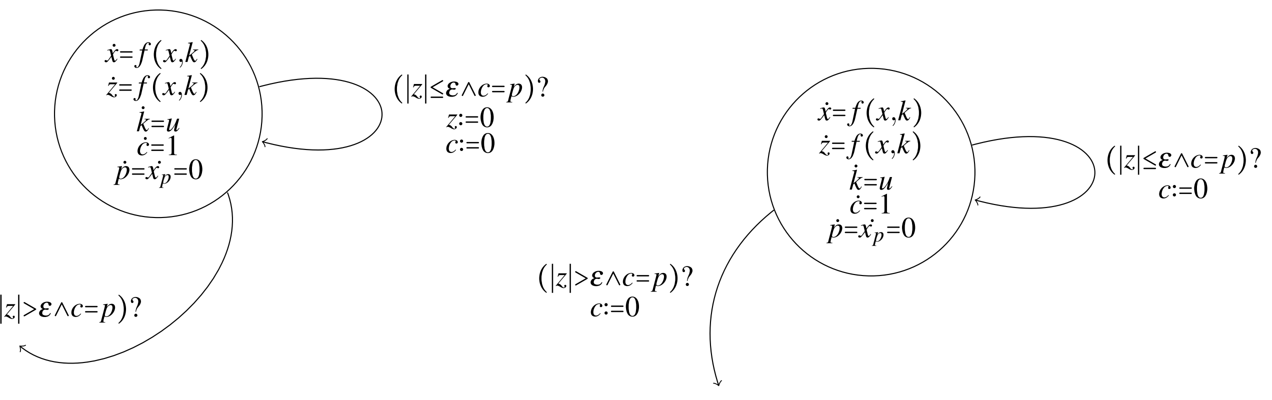
In addition, we modify the property automaton so that it results in an abstraction with predicates involving only one state variable, which is more suitable for the algorithm gRRT. Indeed the star discrepancy is defined for states inside some rectangular sets; for more general sets, box approximations are required. To do so, we modify the condition $(x \approx_\varepsilon x_p)$? by introducing a new variable $z = x - x_p$, the derivative of which is $\dot{z} = \dot{x} - \dot{x_p} = \dot{x}$ (recall that by definition $\dot{x_p} = 0$). The guard on the self-loop transition over q_{OSC} becomes $(x \approx_\varepsilon x_p)? \equiv (|x-(x-z)| < \varepsilon)? \equiv (|z| \leq \varepsilon)?$, while the reset is rewritten as $(x_p := x) \equiv (x - z := x) \equiv (z := 0)$. Similarly the guard condition of the transition from q_{OSC} to q_{INIT} becomes $(|z| > \varepsilon)?$ and $z := 0$, respectively (see Figure 5a). The guard conditions and resets concerning the clock c remain unchanged. The same reasoning can be easily applied to the location q_{STD} (see Figure 5b).

We now proceed with the definition of the function λ which is the basis of the abstraction. Let $\lambda : Q \to 2^\Pi$ be defined as follows:

$$\lambda(q) = \begin{cases} (z \geq \varepsilon, \varepsilon > z, z > -\varepsilon, -\varepsilon \geq z) & \text{if } q = q_{OSC}, \\ (\top) & \text{otherwise.} \end{cases}$$

Note that abstraction function partitions the space of z in q_{OSC} into the sets $(+\infty;+\varepsilon]$, $(+\varepsilon,-\varepsilon)$ and $[-\varepsilon,-\infty)$ with the respective Boolean abstraction vectors $(1,0,0,0)$, $(0,1,1,0)$ and $(0,0,0,1)$. From λ we obtain the transition system $\mathcal{D} = \{S, \rightsquigarrow, s_0\}$, with abstract states

$$S = \{s_i = (q_i,\top) \mid q_i \in Q . q_i \neq q_{OSC}\} \cup \{s'_{OSC} = (q_{OSC},(1,0,0,0)); s''_{OSC} = (q_{OSC},(0,1,1,0)); s'''_{OSC} = (q_{OSC},(0,0,0,1))\},$$



(a) q_{OSC} modified. (b) q_{STD} modified.

Figure 5: Modified property automaton.

transition relation

$$\begin{aligned}\rightsquigarrow = \quad & \rightsquigarrow_C \cup \rightsquigarrow_D = \\ & \{(s'_{OSC}, s''_{OSC}); (s''_{OSC}, s'_{OSC}); (s''_{OSC}, s'''_{OSC}); (s'''_{OSC}, s''_{OSC})\} \cup \\ & \{((q,b),(q',b')) \mid (q,q') \in E. q, q' \neq q_{OSC}\} \cup \\ & \{(s'_{OSC}, s_{INIT}); (s''_{OSC}, s''_{OSC}); (s'''_{OSC}, s_{INIT})\},\end{aligned}$$

and the initial abstract state $s_0 = s_{INIT} = (q_{INIT}, \top)$.

Before specifying the target probabilities over the abstract states, it is necessary to make another modification to the abstract transition system, in order to be able to distinguish the self-loop transitions originated from the abstraction process from those introduced by the transition probability definition. This modification consists in duplicating the locations that with self-loop transitions and replacing these self-loop transitions with the transitions connecting the original location to its copy, and vice versa. Hence, for this example we add two locations s_{STD^L} and s''_{OSC^L} to S and we replace in $\rightsquigarrow$ the transitions (s_{STD}, s_{STD}) and (s''_{OSC}, s''_{OSC}) with (s_{STD}, s_{STD^L}), (s_{STD^L}, s_{STD}), (s''_{OSC}, s''_{OSC^L}) and (s''_{OSC^L}, s''_{OSC}). Figure 6 shows the resulting actraction without such self-loop transitions.

We now define the target probabilities over the abstract states. Since we are interested in detecting that the system stops oscillating, it makes sense to attribute higher probabilities to those abstract states which bring the system from an oscillation phase to a non-oscillation one, i.e., the states s'_{OSC} and s'''_{OSC}. Thus, defining the target probabilities as $\pi_{s_{INIT}} = \pi_{s_{LRN}} = \pi_{s_{STD}} = \pi_{s_{STD^L}} = \pi_{s''_{OSC}} = \pi_{s''_{OSC^L}} = 0.1$ and $\pi_{s'_{OSC}} = \pi_{s'''_{OSC}} = 0.25$, we obtain the following probability transition matrix:

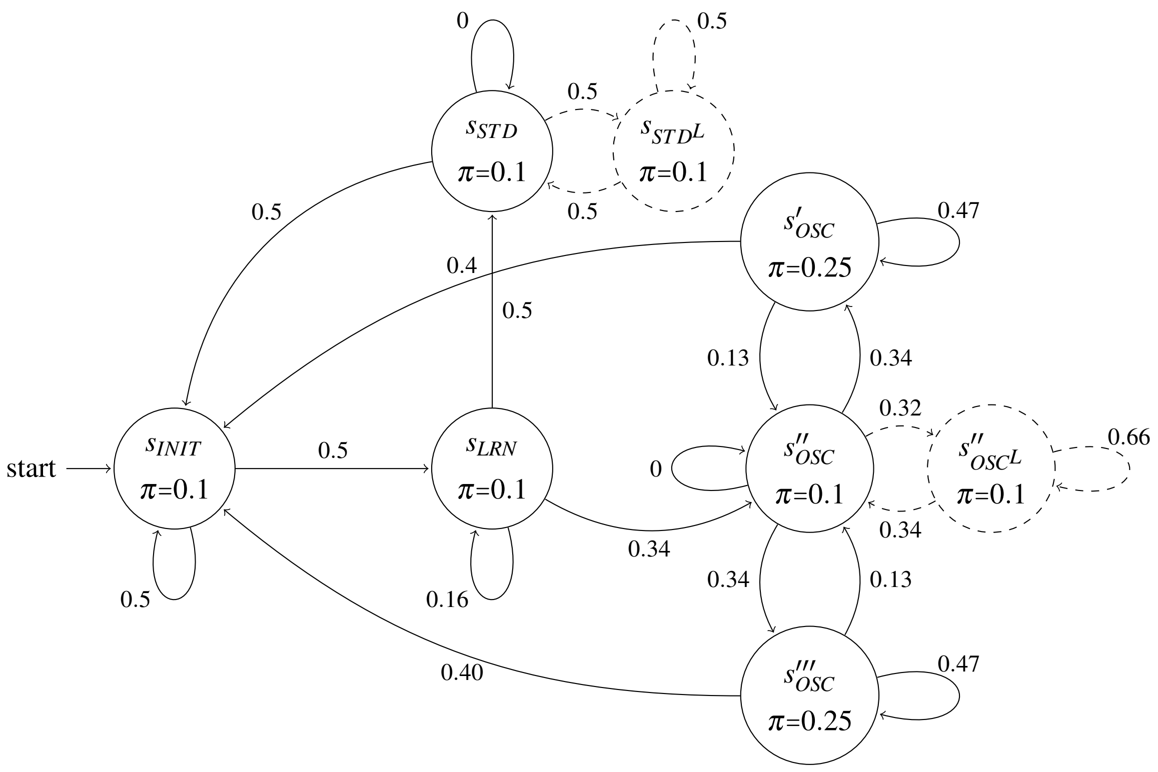


Figure 6: Abstract transition system of the property automaton. Dashed states and transitions are introduced to eliminate self-loop transitions in the property automaton.

	s_{INIT}	s_{LRN}	s_{STD}	s_{STD^L}	s'_{OSC}	s''_{OSC}	s''_{OSC^L}	s'''_{OSC}
s_{INIT}	0.50	0.50	0	0	0	0	0	0
s_{LRN}	0	0.16	0.50	0	0	0.34	0	0
s_{STD}	0.50	0	0	0.50	0	0	0	0
s_{STD^L}	0	0	0.50	0.50	0	0	0	0
s'_{OSC}	0.40	0	0	0	0.47	0.13	0	0
s''_{OSC}	0	0	0	0	0.34	0	0.32	0.34
s''_{OSC^L}	0	0	0	0	0	0.34	0.66	0
s'''_{OSC}	0.40	0	0	0	0	0.13	0	0.47

that leads to the system shown in Figure 6.

4.3 Experimental Results

We have implemented the above described method and incorporated it in the HTG tool [4], which is our previous C++ implementation of the gRRT algorithm. In particular, we extended HTG with the following new functions: defining a discrete abstraction over the considered hybrid automaton, specifying the target probabilities for each abstract state and performing a random walk on the abstract transition system in order to identify the areas that need to be explored.

In our experiments, we focused on the parameter k_1 and on its derivative modelled by the input variable u_1. Moreover, we monitor the two variables x_1 and z_1 since k_1 is involved in both of their dynamics. The values of the other parameters of the automaton in Figure 1 are fixed as follows: $T_i = 7.3781$, $\delta = 0.05$ and $\varepsilon = 0.2$. As an initial value of k_1 we choose its oscillating nominal value 2.0 (see Table 1).

We performed three experiments with different ranges of the input u_1. In the first case u_1 can be sampled within the interval $[-0.01, 0.01]$ (see Figure 7), in the second within $[-0.1, 0.1]$ (see Figure 8), while in the third within [-1.0,1.0]. In all the experiments the state space of k_1 is $[1.8, 2.2]$. In the first case, even if at the end of each period the value of z_1 is not exactly equal to zero, it is always included in the interval defined by $\varepsilon = 0.2$ and thus, for all the simulation runs, the system is considered oscillating. Differently, in the case where k_1 can evolve faster, the variable z_1 ends an oscillation phase at a value smaller than $-\varepsilon$. This means that already for values of $k_1 \in [1.8, 2.2]$ and $\dot{k_1} = u_1 \in [-0.1, 0.1]$ the system leaves the current limit cycle. We can interpret such a behavioral change under a small variation of the nominal parameter values as weak robustness of the Laub-Loomis model. Finally, for values of $u_1 \in [-1.0, 1.0]$ we found that, not very surprisingly, the variable z_1 drifts very far away from zero, showing that the system has stopped oscillating.

All the experiments were performed on a MacBook 3,1 having 2GB RAM. Each experiment involved the computation of 30000 points, with integration time step equal to 0.05. In the first experiment, the tool required $6.23s$, in the second $5.94s$ and in the third $6.53s$. To give an idea of the scalability of the technique, a simulation with 10000 points requires $1.22s$, with 25000 points $4.93s$, while with 50000 points $17.42s$.

5 Conclusion

In this paper, we described a framework for falsifying oscillation properties and study the robustness of biological models. The experimental results are encouraging and we intend to pursue this work in two

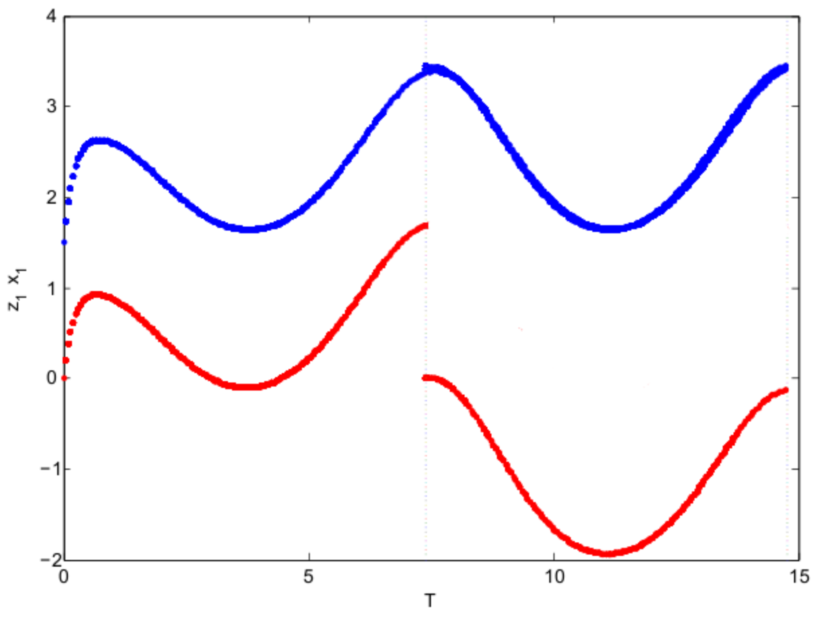


Figure 7: Evolutions of x_1 (blue) and z_1 (red) for $u_1 \in [-0.01, 0.01]$

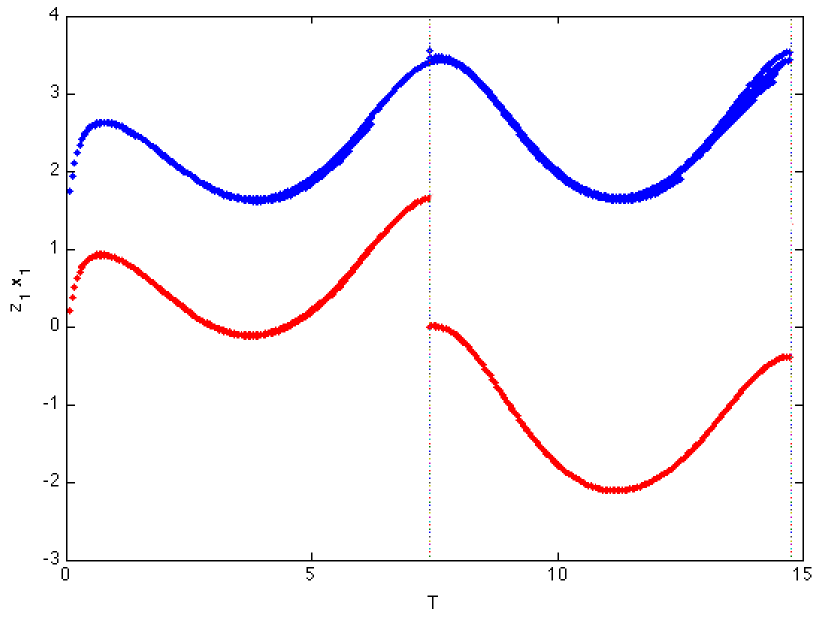


Figure 8: Evolutions of x_1 (blue) and z_1 (red) for $u_1 \in [-0.1, 0.1]$

directions. One is a more efficient parameter sampling, which can be guided by local analysis using Floquet theory. The other direction concerns the application of this approach to analyze other types of bifurcation in biological systems.

References

[1] R. Alur, C. Courcoubetis, T. A. Henzinger & P. H. Ho (1992): *Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems*. In R. L. Grossman, A. Nerode, A. P. Ravn & H. Richel, editors: *Hybrid Systems*, LNCS, Springer, pp. 209–229, doi:10.1007/3-540-57318-6_30.

[2] R. Alur & D. L. Dill (1994): *A Theory of Timed Automata*. *Theoret. Comput. Sci.* 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[3] E. Bartocci, F. Corradini, E. Merelli & L. Tesei (2009): *Model Checking Biological Oscillators*. *Electronic Notes in Theoretical Computer Science* 299(1), pp. 41–58, doi:10.1016/j.entcs.2009.02.004.

[4] T. Dang (2010): *Model-based testing of hybrid systems*. Model-Based Testing for Embedded Systems, CRC Press, doi:10.1201/b11321-15.

[5] T. Dang & T. Nahhal (2009): *Coverage-guided test generation for continuous and hybrid systems*. *Form. Methods Syst. Des.* 34(2), pp. 183–213, doi:10.1007/s10703-009-0066-0.

[6] P. Dluhos, L. Brim & D. Safrnek (2012): *On Expressing and Monitoring Oscillatory Dynamics*. In: *HSB 2012*, doi:10.4204/EPTCS.92.6.

[7] M. M. Donahue, G. Buzzard & A. E. Rundell (2009): *Robust parameter identification with adaptive sparse grid-based optimization for nonlinear systems biology models*. In: *ACC Conference*, doi:10.1109/ACC.2009.5160512.

[8] R. Ghaemi & D. Del Vecchio (2007): *Evaluating the robustness of a biochemical network model*. In: *Decision and Control, 2007 46th IEEE Conference on*, pp. 615–620, doi:10.1109/CDC.2007.4434348.

[9] R. Ghaemi, J. Sun, P. A. Iglesias & D. Del Vecchio (2009): *A Method for determining the robustness of bio-molecular oscillator models*. *BMC Systems Biology* 3(95), doi:10.1186/1752-0509-3-95.

[10] J. Kim, D. G. Bates, I. Postlethwaite, L. Ma & P. A. Iglesias (2006): *Robustness analysis of biochemical network models*. *IEE Proc. Systems Biology* 153(2), pp. 96–104, doi:10.1049/ip-syb:20050024.

[11] M.T. Laub & W.F. Loomis (1998): *A molecular network that produces spontaneous oscillations in excitable cells of Dictyostelium*. *Molecular biology of the cell* 9(12), pp. 3521–3532, doi:10.1091/mbc.9.12.3521.

[12] R. Motwani & P. Raghavan (1995): *Randomized algorithms*. Cambridge University Press, New York, NY, USA, doi:10.1017/CBO9780511814075.

[13] Y. Nonaka, H. Ono, K. Sadakane & M. Yamashita (2010): *The hitting and cover times of Metropolis walks*. *Theoret. Comput. Sci.* 411(1618), pp. 1889 – 1894, doi:10.1016/j.tcs.2010.01.032.

[14] Kuznetsov Y. (2004): *Elements of Applied Bifurcation Theory* . Springer.