Proceedings of the 2nd International Workshop on Biological Processes & Petri Nets (BioPPN2011)
online: http://ceur-ws.org/Vol-724 pp.53-70

Parameter Estimation of Biological Pathways Using Data Assimilation and Model Checking

Chen Li[1,†], Keisuke Kuroyanagi[2,†], Masao Nagasaki[1,⋆], and Satoru Miyano[1]

[1]Human Genome Center, Institute of Medical Science,
[2]Graduate School of Information Science and Technology, University of Tokyo, 4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan
{chenli,ksk9687,masao,miyano}@hgc.jp
http://www.springer.com/lncs

Abstract. This paper presents a novel method to estimate kinetic parameter of biological pathways by using observed time-series data and other knowledge that cannot be formulated in the form of time-series data. Our method utilizes data assimilation (DA) framework and model checking (MC) technique, with a quantitative modeling and simulation architecture named hybrid functional Petri net with extension (HFPNe). Proposed method is applied to an HFPNe model underlying circadian rhythm in *mouse*. We first translate 23 rules of biological knowledge with temporal logic for the model checking, which are not described in the time-series data. Next, we employ particle filter often applied to DA for our estimation procedure. Each particle checks whether its simulation result satisfies the rules or not, and the result of the checking is used for its resampling step. Our simulation results show that proposed method is faster and more accurate than previous method.

Keywords: Hybrid functional Petri net with extension, parameter estimation, data assimilation, particle filter, model checking, temporal logic

1 Introduction

Modeling and simulating large-scale biological pathways have played an important role in systems biology. Owing to their importance, many formal description methods of biological pathway models have been made so far [1–3]. Petri net and its related concepts are one of the succeeded ways of describing biological models [4–6], which have been used for modeling a wide variety of biological pathways and succeeded in reproducing consistent time-series profiles of biological elements such as the concentrations of mRNAs and proteins by means of computer simulations.

Simulation studies on biological pathways promises a deep understanding of complex cellular mechanisms by investigating the dynamic feature. Simulation-based models are commonly governed by a series of parameters, *e.g.* initial values,

⋆ Corresponding author email: masao@hgc.jp. †These authors made equal contributions.

reaction speeds and threshold values of activities. Before the model can be simulated, all parameters must be assigned in advance. However, most parameters are often unknown or not obvious. In general, such parameters are carefully tuned by experts to fit the simulated elements with observed *in vivo/vitro* experiment results. Due to the nonlinearity of the model, parameter estimation is difficult and requires a lot of trial and errors. Small differences of the parameters make large gap between reality and simulation results. Therefore, conventional hand tuning method severely limits the size and complexity of simulation models built as more output data (*e.g.*, microarray gene expression data) are being measured.

The aim of this paper is to develop a novel method to automatically and efficiently estimate kinetic parameters of a given model or a model starting from scratch by combining data assimilation (DA) and model checking (MC) approaches, coupled with observed experiment data. Observed experiment data includes well-defined time-series data and other knowledge that cannot be formulated in the form of time-series data.

DA was originally established in the field of geophysical simulation science. Nagasaki *et al.* [7] have proposed a so-called *genomic data assimilation* approach. Their DA framework enables users to handle both the model construction and parameter tuning in the context of statistical inferences, and establishes a link between the HFPNe simulation model and observed data, *e.g.*, microarray gene expression data [7] or time series proteomic data [8]. Current DA approach has some issues because it depends on providing successive time points (at least 10-20 time points) of time-series data by biological experiments. That is, for a small time-series data set including for example two or three time points, DA will cost massive computational resource, in some cases, it will be completely impossible to estimate parameters. The response to this difficulty of dealing with sparse and/or not well-defined time-series data is the use of model checking.

MC is a method for automatic verification of system requirements [9], which firstly used in hardware field because of its determined behavior and limited value space. Today, this technique has been applied to more complex biological models. There are several studies that use model checking for parameter estimation, *e.g.* Donaldson and Gilbert developed a computational system named MC2(GA) [10] that couples model checking with genetic algorithm for parameter estimation. Li *et al.* [11] proposed online model checking approach based parameter estimation framework applied to the HFPNe class. In order to check the model, one need to write biological rules of the knowledge with temporal logic. For example, "A biological phenomena *Foo* keeps decreasing until *Bar* rises." can be written as "`d([Foo])`$\leq$`0 U d([Bar])>0`" in a kind of temporal logic. Various knowledge can be described in this way. Nevertheless, in order to improve the estimation efficiency and accuracy, it is expected to find a general methodology to determine parameters by combining DA and MC dealing with both time-series experimental data and biological queries.

The paper is organized as follows. In **Methods**, we briefly explain how to (i) construct biological models with HFPNe, (ii) estimate unknown parameters in a nonlinear state space model, (iii) translate biological rules with a temporal logic for querying system properties, and (iv) combine MC with DA for parameter estimation. In **Results**, we compare our novel method with previous method by applying them to *mouse* circadian rhythm model represented by HFPNe. We show that our method is potentially faster and more accurate than previous one that excludes MC technique.

2 Methods

2.1 Hybrid Functional Petri Net with Extension (HFPNe)

HFPNe is developed as a biosimulation tool for pathway modeling and simulation extended from original Petri net [13]. HFPNe can deal with three types of data: discrete, continuous and generic, whereas the original Petri net deal with only discrete data. HFPNe consists of three types of elements: *entity*, *process* and *connector* (see Figure 1 (a)). Figure 1 (b) shows connection rules in HFPNe. For more definitions and usages of HFPNe, see Nagasaki *et al.* [13].

Figure 2 shows circadian rhythm model of *mouse* represented by HFPNe [14]. This model is composed of of 12 entities, 28 processes and 45 connectors. Due to the space restriction, **Appendix** gives the details of these elements. Initial value of entities $m_i(0)$ $(i{=}1,\cdots,12)$, reaction speed parameters k_i ($i{=}1,2$ and k_i is a common parameter to control speeds of similar biological processes: k_1 for protein binding; k_2 for translation), and threshold parameters s_i $(i{=}1,\cdots,3)$ are unknown parameters.

2.2 Data Assimilation for Parameter Estimation

We here explain how to estimate parameters in a simulation model from time-series data with the use of DA.

Data Assimilation with Nonlinear State Space Model DA is an approach to improve the accuracy of the models by combining with data, which can deal with models formulated by nonlinear state space model (SSM) given by two equations [15]:

$$\boldsymbol{m}_t{=}\boldsymbol{f}(\boldsymbol{m}_{t-1},\boldsymbol{w}_t,\boldsymbol{\theta}_{sys})\,, \tag{1}$$

$$\boldsymbol{y}_t{=}\boldsymbol{H}\boldsymbol{m}_t+\boldsymbol{\epsilon}_t\,. \tag{2}$$

Equation (1) is called "system model" and **Equation** (2) is called "observed model". In the system model, $\boldsymbol{m}_t{\equiv}(m_{1t},\cdots,m_{pt})^T$ is a state vector consisting of p state variables m_{it} $(i{=}1,\cdots,p)$ at discrete time point t. $\boldsymbol{w}_t$ denoting system noise is an l-dimensional white noise at t with a density $q(\boldsymbol{w})$. $\boldsymbol{f}$ is a vector-valued function, $\boldsymbol{f}:\mathbb{R}^{p+l}\mapsto\mathbb{R}^p$, and $\boldsymbol{\theta}_{sys}$ is a vector of model parameters. In the observed model, $\boldsymbol{y}_t{\in}\mathbb{R}^d$ is an observation vector at t, $\boldsymbol{H}{\in}\mathbb{R}^d\times\mathbb{R}^p$ is an observation matrix; H_{ij} takes value one if observed value of jth entity corresponds to the ith element of $\boldsymbol{y}_t$, otherwise zero. $\boldsymbol{\epsilon}_t$ called observational noise is a d-dimensional white noise at t with a density $r(\boldsymbol{\epsilon})$. The likelihood of the parameter is given by

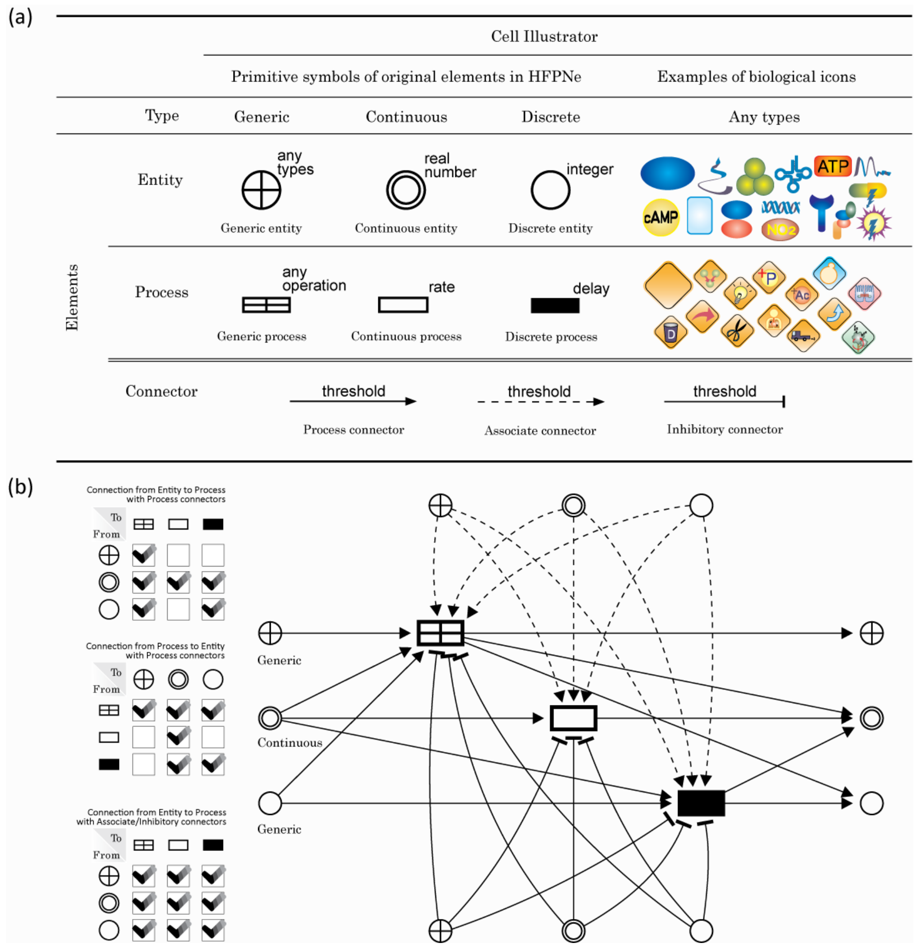

Fig. 1. (a) Basic HFPNe elements and biological icons in Cell Illustrator [12] in which HFPNe was implemented. (b) Connection rules (left side) and corresponding network (right side) in HFPNe. For instance, for the uppermost block labeled with "Connection from Entity to Process with Process connectors", the check-mark denotes the availability connected from corresponding entities to processes, *e.g.*, only the generic process can be selected as the output connected from generic entity with process connector.

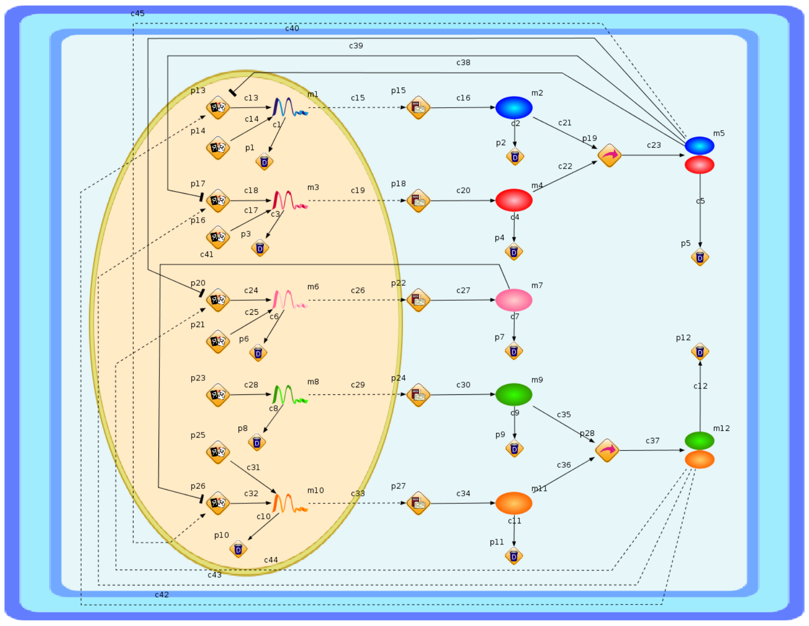
Fig. 2. Circadian rhythm model of *mouse* with HFPNe .

$L(\boldsymbol{\theta}_{sys})=p(\boldsymbol{y}_1,\cdots,\boldsymbol{y}_T|\boldsymbol{\theta}_{sys})=\prod_{t=1}^{T}p(\boldsymbol{y}_t|\boldsymbol{Y}_{t-1},\boldsymbol{\theta}_{sys})=\prod_{t=1}^{T}\int p(\boldsymbol{y}_t|\boldsymbol{m}_t,\boldsymbol{\theta}_{sys})p(\boldsymbol{m}_t|\boldsymbol{Y}_{t-1},\boldsymbol{\theta}_{sys})d\boldsymbol{m}_t$, where $\boldsymbol{Y}_t=\{\boldsymbol{y}_1,\cdots,\boldsymbol{y}_t\}$. The maximum likelihood estimator (MLE) for $\boldsymbol{\theta}_{sys}$ is given by

$$\underset{\boldsymbol{\theta}_{sys}}{\operatorname{argmax}}\log L(\boldsymbol{\theta}_{sys}).$$

We can estimate parameters with MLE, however, this maximum likelihood method has an important issue that the value of $\log L(\boldsymbol{\theta}_{sys})$ computed by Monte Carlo filter includes an approximation error. For accurate estimation, huge computation resources are thus required.

Self-Organizing State Space Model To deal with the difficulty of maximum likelihood method, we use self-organizing state space model (SOSSM) [16–18]. We can estimate parameters based on Bayesian inference with SOSSM by weaving parameters in the state vector as

$$\boldsymbol{z}_t=\begin{bmatrix}\boldsymbol{m}_t\\ \boldsymbol{\theta}_{sys}\end{bmatrix}.$$

The SSM for this vector is given by $\boldsymbol{z}_t=\boldsymbol{F}^*(\boldsymbol{z}_{t-1},\boldsymbol{w}_t)$ and $\boldsymbol{y}_t=\boldsymbol{H}^*\boldsymbol{z}_t+\boldsymbol{\epsilon}_t$, where

$$\boldsymbol{F}^*(\boldsymbol{z}_{t-1},\boldsymbol{w}_t)=\begin{bmatrix}\boldsymbol{f}(\boldsymbol{m}_{t-1},\boldsymbol{w}_t,\boldsymbol{\theta}_{sys})\\ \boldsymbol{\theta}_{sys}\end{bmatrix}$$
$$\boldsymbol{H}^*\boldsymbol{z}_t=\boldsymbol{H}\boldsymbol{m}_t.$$

We can obtain marginal posterior densities without obtaining MLE of $\boldsymbol{\theta}$ as

$$p(\boldsymbol{m}_T \mid \boldsymbol{Y}_T) = \int p(\boldsymbol{z}_T \mid \boldsymbol{Y}_T) d\boldsymbol{\theta}_{sys}$$

$$p(\boldsymbol{\theta}_{sys} \mid \boldsymbol{Y}_T) = \int p(\boldsymbol{z}_T \mid \boldsymbol{Y}_T) d\boldsymbol{m}_T.$$

In this way, we can avoid the issue of maximum likelihood method during the estimation.

Particle Filter As mentioned above, we have to calculate distribution $p(\boldsymbol{z}_T|\boldsymbol{Y}_T)$ for estimating parameters. However, it generally becomes a non-gaussian distribution in SSM. Therefore, it is needed to represent this distribution with some method. We here use a sequential Monte Carlo method called *particle filter* (PF) [19]. Figure 3 shows the overview of particle filter's algorithm. In particle filter, predictive distribution $\boldsymbol{p}(\boldsymbol{z}_t|\boldsymbol{Y}_{t-1})$ and filter distribution $\boldsymbol{p}(\boldsymbol{z}_t|\boldsymbol{Y}_t)$ are approximated by m in which each realization is called *particle* as follows:

$$\boldsymbol{p}_{t|t-1} \equiv \{\boldsymbol{p}_{t|t-1}^{(1)}, \cdots, \boldsymbol{p}_{t|t-1}^{(m)}\} \sim \boldsymbol{p}(\boldsymbol{z}_t \mid \boldsymbol{Y}_{t-1})$$
$$\boldsymbol{p}_{t|t} \equiv \{\boldsymbol{p}_{t|t}^{(1)}, \ldots, \boldsymbol{p}_{t|t}^{(m)}\} \sim \boldsymbol{p}(\boldsymbol{z}_t \mid \boldsymbol{Y}_t),$$

where $\boldsymbol{p}_{t|t-1}^{(j)}$ $(j{=}1, \cdots, m)$ and $\boldsymbol{p}_{t|t}^{(j)}$ $(j{=}1, \cdots, m)$ are $(p+|\boldsymbol{\theta}_{sys}|)$-dimensional numbers. This algorithm is processed with the following steps.

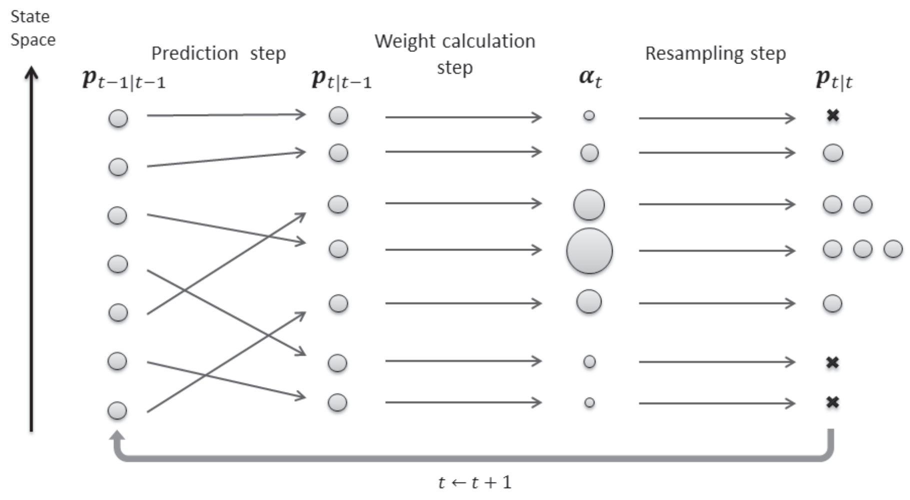

Fig. 3. Overview of particle filter. The left-most column of particles shows state at $t-1$. The second column shows the predicted states. The third column shows the weights of corresponding particles ($\boldsymbol{\alpha}_t \equiv \alpha_t^{(1)}, \cdots, \alpha_t^{(m)}$). The right-most column shows results of resampling step.

Step 1 Generate the $(p+|\boldsymbol{\theta}_{sys}|)$-dimensional random number $\boldsymbol{p}_{0|0}^{(j)}$ for j=$1, \cdots, m$.

Step 2 Repeat the following three steps for the observed time points t=$1, \cdots, T$.

Step 2.1 (Prediction step) Generate m system noises $\boldsymbol{w}_t^{(j)}$ independently and identically from $q(\boldsymbol{w}_t)$, and compute particles by inputting filtered states into simulation model, $\boldsymbol{p}_{t|t-1}^{(j)}$=$\boldsymbol{F}^*(\boldsymbol{p}_{t-1|t-1}^{(j)}, \boldsymbol{w}_t^{(j)})$ for j=$1, \cdots, m$.

Step 2.2 (Weight calculation step) Compute the weights of importance for the particles according to

$$\alpha_t^{(j)}=p(\boldsymbol{y}_t|\boldsymbol{p}_{t|t-1}^{(j)})=r(\boldsymbol{y}_t - \boldsymbol{H}^*\boldsymbol{p}_{t|t-1}^{(j)}) \tag{3}$$

for j=$1, \cdots, m$. These weights are then normalized as $\bar{\alpha}_t^{(j)}$=$\alpha_t^{(j)}\sum_{j=1}^{m}\alpha_t^{(j)}$.

Step 2.3 (Resampling step) Generate filtered state $\boldsymbol{p}_{t|t}^{(j)}$ for j=$1, \cdots, m$ by resampling $\{\boldsymbol{p}_{t|t-1}^{(1)}, \cdots, \boldsymbol{p}_{t|t-1}^{(m)}\}$ with the probabilities $\{\bar{\alpha}_t^{(1)}, \cdots, \bar{\alpha}_t^{(m)}\}$.

2.3 Model Checking

We explain how to represent requirements of biological pathway models for model checking. Model checking is a method to verify whether models satisfy the requirements or not. Temporal logic formulae are often used to describe the system requirements. In this study, we selected PLTL (*Probabilistic Linear-time Temporal Logic*) for querying dynamic models of cellular networks [20, 21] which extends original LTL to a stochastic setting with a probability operator and a filter criterion defining the starting state where the property is satisfied.

Table 1. Syntax of PLTL.

ψ	$::= \mathbf{P}_{\trianglelefteq x}(LTL) \mid \mathbf{P}_{=?}(LTL) \mid LTL$
LTL	$::= \phi\{AP\} \mid \phi$
ϕ	$::= \mathbf{X}\phi \mid \mathbf{G}\phi \mid \mathbf{F}\phi \mid \phi\ \mathbf{U}\ \phi \mid \phi\ \mathbf{R}\ \phi \mid \neg\phi \mid \phi\&\&\phi \mid \phi\Vert\phi \mid \phi \Rightarrow \phi \mid AP$
AP	$::= value\ comp\ value \mid value_{boolean}$
$value$	$::= value\ op\ value \mid [variableName] \mid Function_{numeric} \mid Integer \mid Real$
$value_{boolean}$	$::= true \mid false \mid Function_{boolean}$
$comp$	$::= == \mid != \mid > \mid \geq \mid < \mid \leq$
op	$::= + \mid - \mid * \mid / \mid \hat{}\,$,

with $\trianglelefteq \in \{<, \leq, >, \geq\}$, $x \in [0, 1]$.

[PLTL Syntax] Table 1 shows definition of PLTL syntax, which is used to ask for the probability of user's query via a PLTL formulae ψ. In the LTL expression $\phi\{AP\}$, ϕ will be checked from the state that AP is satisfied rather than from the default initial state, where AP is called *atomic proposition* and takes boolean domain. PLTL allows (i) LTL expression to contain temporal operators, *i.e.*, **X**, **F**, **G**, **U**, **R**. Five temporal operators are used to describe the sequencing of the states along the execution; and (ii) the usage of ψ without probabilistic operators (*i.e.* simply in the form of LTL), which is useful when the model is deterministic.

Table 2. Semantics of temporal operators

Operator	Meaning	Explanation
$\mathbf{X}\phi$	Next time	ϕ must be true at the next time point.
$\mathbf{G}\phi$	Globally	ϕ must always be true.
$\mathbf{F}\phi$	Finally	ϕ must be true at least once.
$\phi_1\mathbf{U}\phi_2$	Until	ϕ_1 must be true until ϕ_2 becomes true; ϕ_2 must become true eventually.
$\phi_1\mathbf{R}\phi_2$	Release	ϕ_2 must be true until and including the time point ϕ_1 becomes true; if ϕ_2 never true, ϕ_1 must always be true.

[**PLTL Semantics**] The semantics of PLTL is defined over the finite sets of finite paths through system's state space, obtained by repeated simulation runs of HFPNe models. The PLTL formula is built upon two components: probabilistic operator and property LTL. For each simulation run, the LTL expression is evaluated to a boolean truth value, and the probability of the LTL statement holding true is calculated based on the whole set of simulation results.

For the probability operator components, there are two distinct operators: (i) $\mathbf{P}_{\trianglelefteq x}(LTL)$ is any inequality comparison of the probability of the property LTL holding true, for example $\mathbf{P}_{\geq 0.5}(LTL)$; and (ii) $\mathbf{P}_{=?}(LTL)$ returns the value of the probability of the property holding true. The semantics of the temporal logic operators are described in Table 2. Concentrations of biochemical species in the model are denoted by [*variableName*]. A special variable, [*time*], stands for simulation time.

Due to the ability of PLTL, it is possible to define functions of two different natures: functions that return a real number and functions that return a boolean value. An example of the real number function is $d([variableName])$ which returns the subtracted value of [*variableName*] between time i and $i{-}1$. Note that, d([$variableName$]) equals zero at time point zero. One example of a boolean function is $similarAbsolute(value\ a, value\ b, value\ \epsilon)$, which returns true if $|a{-}b|{<}\epsilon$ or else it returns false. Table 3 shows the rules written in PLTL for circadian rhythm model of *mouse*.

2.4 Combining Model Checking with DA

As stated in **Introduction**, we have applied either DA or MC to pathway model in the previous researches, but used time scales are different. We here employ common time scale called *Petri net time* for combining DA with MC, which is the virtual time unit of the HFPNe model denoted by [pt]. We define: (i) $simInt{\in}\mathbb{R}$ as simulation interval; (ii) $mcInt{\in}\mathbb{R}$ which is a multiple of $simInt$ as a model checking interval; and (iii) $MapOttoPt : \mathbb{N}{\mapsto}\mathbb{R}$ as a mapping from observed time point to Petri net time for combining. From the MC's viewpoint, $\mathbf{X}\phi$ means that ϕ must be true at the state after $mcInt$ Petri net time. Meanwhile, a simulation from time $MapOttoPt(t{-}1)$ to $MapOttoPt(t)$ is a run in $\boldsymbol{f}$ from DA's viewpoint.

Hürseler and Künsch mentioned that it is difficult to generate a good initial distribution of parameters with SOSSM [22]. This is because parameters in resampled particles are the subset of parameters in the initial particles and a model with randomly generated parameters rarely satisfy all rules. To generate good initial distribution of parameters, two considerations are designed:

Table 3. Biological rules for circadian rhythm model of *mouse*. Rule 1 to Rule 9 describes the range of concentrations; Rule 10 to Rule 17 describes that of peak concentrations, Rule 18 to Rule 21 specifies concentration relationships when they reach peaks, and Rule 22 to Rule 23 specifies normal concentration relationships.

No.	Rule
	LTL translation
Rule 1	Concentration of per_mRNA is between 0.2 and 3.8.
	`G([per_mRNA] < 3.8 && [per_mRNA] > 0.2)`
Rule 2	Concentration of Rev-Erv_mRNA is between 0.2 and 3.4.
	`G([Rev-Erv_mRNA] < 3.4 && [Rev-Erv_mRNA] > 0.2)`
Rule 3	Concentration of Bmal_mRNA is between 0.2 and 4.3.
	`G([Bmal_mRNA] < 4.3 && [Bmal_mRNA] > 0.2)`
Rule 4	Concentration of Bmal_mRNA is between 2.4 and 2.6 after time becomes 20.
	`G(similarAbsolute([Clock_mRNA],2.5,0,1)){[time] > 20}`
Rule 5	Concentration of Cry_mRNA is between 3.8 and 0.2.
	`G([Cry_mRNA] < 3.8 && [Cry_mRNA] > 0.2)`
Rule 6	Concentration of PER is between 2.6 and 0.2.
	`G([PER] < 2.6 && [PER] > 0.2)`
Rule 7	Concentration of CRY is between 2.6 and 0.2.
	`G([CRY] < 2.2 && [CRY] > 0.2)`
Rule 8	Concentration of PER/CRY is between 2.7 and 0.2.
	`G([PER/CRY] < 2.7 && [PER/CRY] > 0.2)`
Rule 9	Concentration of REV_ERB is between 1.5 and 0.2.
	`G([REV_ERB] < 1.5 && [REV_ERB] > 0.2)`
Rule 10	Local maximum concentration of per_mRNA is greater than 2.0.
	`G(d([per_mRNA]) ≥ 0 && X(d([per_mRNA])<0) ⇒ [per_mRNA] > 2.0)`
Rule 11	Local minimum concentration of per_mRNA is less than 1.0.
	`G(d([per_mRNA])≤0 && X(d([per_mRNA])>0)⇒ [per_mRNA] < 1.0)`
Rule 12	Local maximum concentration of Rev-Erv_mRNA is greater than 1.5.
	`G(d([Rev-Erv_mRNA])≥0 && X(d([Rev-Erv_mRNA])<0) ⇒ [Rev-Erv_mRNA] > 1.5)`
Rule 13	Local minimum concentration of Rev-Erv_mRNA is less than 1.0.
	`G(d([Rev-Erv_mRNA])≤0 && X(d([Rev-Erv_mRNA])>0) ⇒ [Rev-Erv_mRNA] < 1.0)`
Rule 14	Local maximum concentration of Bmal_mRNA is greater than 1.5.
	`G(d([Bmal_mRNA]) ≥0 && X(d([Bmal_mRNA])<0) ⇒ [Bmal_mRNA] > 1.5)`
Rule 15	Local minimum concentration of Bmal_mRNA is less than 1.0.
	`G(d([Bmal_mRNA])≤0 && X(d([Bmal_mRNA])>0) ⇒ [Bmal_mRNA] < 1.0)`
Rule 16	Local maximum concentration of Cry_mRNA is greater than 2.0.
	`G(d([Cry_mRNA])≥0 && X(d([Cry_mRNA])<0) ⇒ [Cry_mRNA] > 2.0)`
Rule 17	Local minimum concentration of Cry_mRNA is less than 1.0.
	`G(d([Cry_mRNA])≤0 && X(d([Cry_mRNA])>0) ⇒ [Cry_mRNA] < 1.0)`
Rule 18	When concentration of Bmal_mRNA takes local minimum, concentration of Bmal_mRNA is less than concentration of per_mRNA.
	`G(d([Bmal_mRNA])≤0 && X(d([Bmal_mRNA])>0) ⇒ [Bmal_mRNA] < [per_mRNA])`
Rule 19	When concentration of Bmal_mRNA takes local maximum, concentration of Bmal_mRNA is grater than concentration of per_mRNA
	`G(d([Bmal_mRNA])≥0 && X(d([Bmal_mRNA])<0) ⇒ [Bmal_mRNA] > [per_mRNA])`
Rule 20	When concentration of per_mRNA takes local minimum, concentration of per_mRNA is less than concentration of per_mRNA.
	`G(d([per_mRNA])≤0 && X(d([per_mRNA])>0) ⇒ [per_mRNA] < [Bmal_mRNA])`
Rule 21	When concentration of per_mRNA takes local maximum, concentration of per_mRNA is grater than concentration of Bmal_mRNA
	`G(d([per_mRNA])≥0 && X(d([per_mRNA])<0) ⇒ [per_mRNA] > [Bmal_mRNA])`
Rule 22	Concentration of per_mRNA is greater than concentration of Rev-Erv_mRNA.
	`G([per_mRNA] > [Rev-Erv_mRNA])`
Rule 23	Concentration of CLOCK is greater than concentration of PER.
	`G([CLOCK] > [PER])`

Firstly, we repeat particle filters many times by regenerating $\boldsymbol{p}_{0|0}$ from $\boldsymbol{p}_{T|T}$ like a crossover in genetic algorithm since the length of time courses for biological use is generally shorter than that for other fields; Secondly, a threshold value Th is used to discard particle whose unsatisfied number by the checking is greater than Th. Th is changed for each run of particle filter. In this study, system noise is not taken into account in order to accelerate the estimation. Nevertheless, a generic process can be mapped to Java object supporting HFPNe model in a nondeterministic settings.

Algorithm 1 shows pseudocode of our parameter estimation method. In step 9, *PredictandMC* returns predicated particle and the result whether it is worthless or not. The detail of this function is displayed in **Algorithm 2**. In steps 10–15, if the simulation results unsatisfy more than Th rules, the weight of the particle will become zero, or else the weight is calculated by **Equation** (3) via r. To calculate r, we assume that observed noise is a gaussian white noise and its mean is zero. The variance of observed noise is thus needed to be estimated. We generate multiple candidate values and use the value that has the maximum likelihood as the variance. Steps 17–19 are designed to break the run of particle filter if all the particles unsatisfied more than Th rules; Otherwise Th is incremented or decremented in steps 23–27. $\boldsymbol{p}_{0|0}$ for the next run of particle filter is generated from $\boldsymbol{p}_{T|T}$ in step 28.

Algorithm 1. Pseudecode of our parameter estimation method.

```
1: function $EstimateParameters(\boldsymbol{Y}_T, Rules)$
2: $\boldsymbol{p}_{0|0} \leftarrow GenerateInitialParticles()$
3: $Th \leftarrow initialThreshold$
4: $F \leftarrow false$
5: loop
6:     for $t \leftarrow 1, \ldots, T$ do
7:         $F \leftarrow false$
8:         for $j \leftarrow 1, \ldots, m$ do
9:             $(\boldsymbol{p}^{(j)}_{t|t-1}, SatisfiedRules) \leftarrow PredictandMC(\boldsymbol{p}^{(j)}_{t-1|t-1}, t, Rules, Th)$
10:            if $SatisfiedRules$ then
11:                $\alpha^{(j)}_t \leftarrow CalculateWeight(\boldsymbol{p}^{(j)}_{t|t-1}, \boldsymbol{y}_t)$
12:                $F \leftarrow true$
13:            else
14:                $\alpha^{(j)}_t \leftarrow 0$
15:            end if
16:        end for
17:        if $!F$ then
18:            break
19:        end if
20:        $\bar{\boldsymbol{\alpha}}_t \leftarrow \boldsymbol{\alpha}_t / \sum_{j=1}^{M} \alpha^{(j)}_t$
21:        $\boldsymbol{p}_{t|t} \leftarrow Resampling(\boldsymbol{p}_{t|t-1}, \bar{\boldsymbol{\alpha}}_t)$
22:    end for
23:    if $F$ then
24:        $Th \leftarrow \max(0, Th - 1)$
25:    else
26:        $Th \leftarrow Th + 1$
27:    end if
28:    $\boldsymbol{p}_{0|0} \leftarrow RegenerateParticles(\boldsymbol{p}_{T|T})$
29: end loop
```

In **Algorithm 2**, steps 2–3 convert time scale from observed time to Petri net time. Step 4 separates particle $\boldsymbol{p}$ into $\boldsymbol{m}(start)$ and $\boldsymbol{\theta}$, where $\boldsymbol{m}(t)$ is a vector consisting of the values of p entities at Petri net time t and $\boldsymbol{\theta}$ is a vector of parameters to be estimated. $\boldsymbol{m}(pt{+}simInt)$ is predicted by simulation in step 6. Step 7 returns the number of unsatisfied rules via $ModelChecking()$. In steps 9–11, if simulation result violates more than Th rules, simulation will stop immediately.

Algorithm 2 Prediction step of particle filter and model checking.

```
1:  function $PredictandMC(\boldsymbol{p}, t, Rules, Th)$
2:  $start \leftarrow MapOttoPt(t-1)$
3:  $end \leftarrow MapOttoPt(t) - simInt$
4:  $(\boldsymbol{m}(start), \boldsymbol{\theta}) \leftarrow \boldsymbol{p}$
5:  for $pt \leftarrow start$ to $end$ step $simInt$ do
6:     $\boldsymbol{m}(pt + simInt) \leftarrow Simulation(\boldsymbol{m}(pt), \boldsymbol{\theta})$
7:     if $(pt + simInt)\%mcInt = 0$ then
8:        $N \leftarrow ModelChecking(\boldsymbol{m}, Rules)$
9:        if $Th < N$ then
10:          return $([\boldsymbol{m}(pt + simInt)^T, \boldsymbol{\theta}^T]^T, false)$
11:       end if
12:    end if
13: end for
14: return $([\boldsymbol{m}(MapOttoPt(t))^T, \boldsymbol{\theta}^T]^T, true)$
```

3 Results and Discussions

3.1 Estimation environment and evaluation criteria

We estimate 17 parameters of circadian rhythm model of *mouse* (*i.e.*, 12 initial values of entities, two reaction speeds and three threshold values). We use synthesized data set coupled with simulation and observed noise $\boldsymbol{\epsilon}_t$ ($\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, 0.05^2 I_{12})$) as observed data. It contains 312 data of 26 time points (see Figure 4) for each 12 biological entities – five mRNAs, five proteins and two complex proteins –. One observed time point is mapped to five Petri net times. Table 4 summarizes details of our estimation environment. Parameter search range is set from zero to 15.

Table 4. Default parameters of the estimation.

Parameter name	Value	Meaning
m	50,000	the number of particles
p	12	the number of entities
d	12	the number of entities which have observed data
T	26	the number of observed time points
$simInt$	0.1	simulation interval
$mcInt$	1.0	model checking interval
$initialThreshold$	8	initial value of threshold for model checking

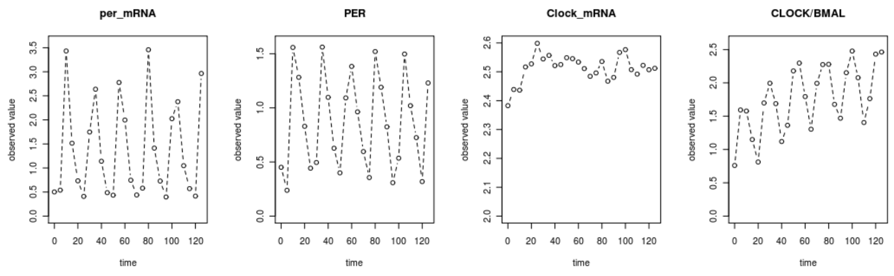

Fig. 4. Examples of observed data.

To evaluate the results of estimation, following *Score*s are defined and calculated before the step *RegenerateParticles* step given in **Algorithm 1**.

$$Score_{mean} = \sum_{t=1}^{T} \sum_{e=1}^{p} \frac{|SimResult(Params_{mean},e,t) - y_{t,p}|^2}{|\boldsymbol{Y}_T|}$$
$$Score_{mode} = \sum_{t=1}^{T} \sum_{e=1}^{p} \frac{|SimResult(Params_{mode},e,t) - y_{t,p}|^2}{|\boldsymbol{Y}_T|}$$
$$Score_{median} = \sum_{t=1}^{T} \sum_{e=1}^{p} \frac{|SimResult(Params_{median},e,t) - y_{t,p}|^2}{|\boldsymbol{Y}_T|}$$
$$Score_{best} = \sum_{t=1}^{T} \sum_{e=1}^{p} \frac{|SimResult(Params_{best},e,t) - y_{t,p}|^2}{|\boldsymbol{Y}_T|}$$
$$Score = \min\{Score_{mean}, Score_{mode}, Score_{median}, Score_{best}, Score_{current}\},$$

where $y_{t,p}$ is an observed value of entity p at observed time point t. $Params_{best}$ denotes parameters of a particle which is made of particles. $Params_{mean}$, $Params_{mode}$ and $Params_{median}$ represent mean, mode and median value of all particles' parameters respectively. $SimResult(Params, e, t)$ returns the value of entity e at time t which is calculated by simulation with $Param$. $Score_{current}$ is ∞ at first time of the calculation, otherwise it is the $Score$ of previous calculation.

3.2 Experimental Result

Comparison between PFMC and PF We compare the performance between our new method Particle Filter with Model Checking (PFMC) and previous method Particle Filter (PF). The estimation experiments are carried out on workstation of Intel Xeon E5450 (3.0GHz) with 32G bytes of memory. We performed estimation 100 times for both methods. Figure 5 shows the result with respect to the distribution of *Score* with elapsed time. Mean of *Score* is also analyzed by Welch's t-test (See Table 5). Nearly all *Score* of PFMC and PF are good on and after 600 seconds. Both medians are also good on and after 600 seconds, but there are many bad cases before 1,000 seconds for PF. That is, roughly speaking, if there are enough amounts of observed data, there is no much difference by using either PMFC or PF for the estimation. However, in almost all cases, we can finish the estimation within a short time incorporating model checking.

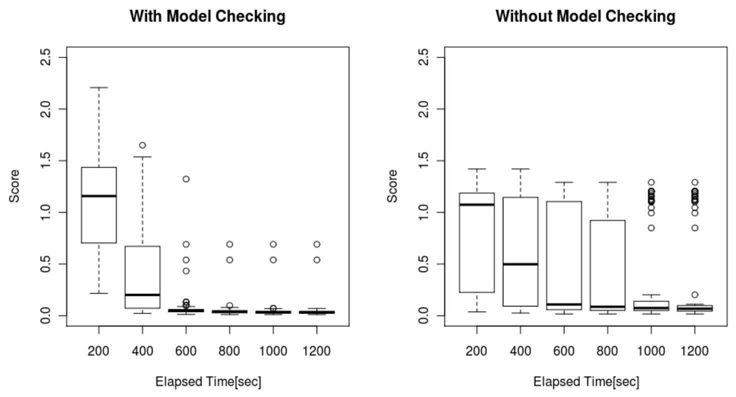

Fig. 5. Distributions of *Score* at corresponding time points. The left diagram shows the result of PFMC, and the right one shows that of PF. X-axis denotes elapsed time (second), while y-axis denotes *Score*.

Table 5. Results of Welch's t-test.

Elapsed time[sec]	P-value	Mean of PFMC *Score*	Mean of PF *Score*
200	1.584×10^{-6}	1.1055600	0.7746972
400	0.02169	0.4656204	0.5243369
600	1.834×10^{-10}	0.07760144	0.44547495
800	6.165×10^{-9}	0.0516071	0.3580496
1000	9.631×10^{-8}	0.04642736	0.31403475
1200	2.974×10^{-7}	0.04446433	0.29820645

Estimation with small amount of observed data To investigate the performance in the case of small amount of observed data, we use only first ten time points of five biological entities' data for the estimation: per_mRNA, Cry_mRNA, Rev_Erv_mRNA, $Clock\_mRNA$ and $Bmal\_mRNA$. More detailedly, we use all default parameters for *Score* calculation and just overwrite p to five and T to ten. The estimation results are exhibited in Figure 6 and Table 6.

The results clearly show that on and after 1,200 seconds, in contrast to the previous experiment, there is difference not only between bad cases, but also between medians of PFMC and PF. This is because it is difficult to estimate parameters which makes certain rhythms with only two cycles of observed data. Therefore, median *Score*s of PF are not good before 3,000 seconds. Moreover, convergence of PFMC is worse than previous experiment.

Effects of the rules We also investigate the effect of rules by checking which rule is unsatisfied which results in the cutting of a particle. We run estimation 100 times with default parameters and all of observed data. All the runs finished after 20 minutes and the results are shown in Figure 7.

Two kinds of effects used in model checking approach can be considered. First, it cuts useless particles,which enables us to estimate with more particles or more runs of particle filters. Second, it cuts bad results which facilitates the

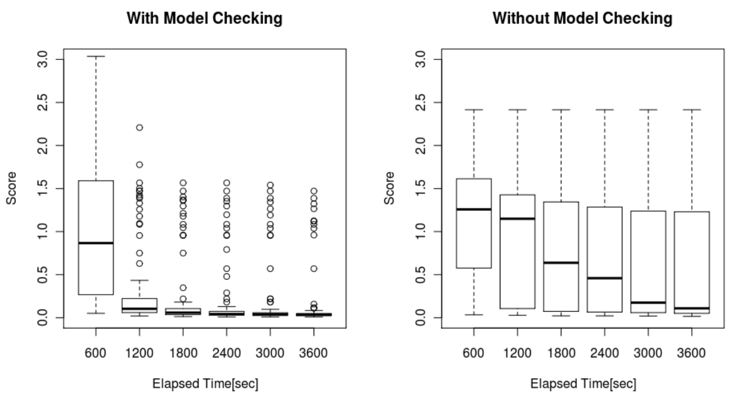

Fig. 6. Distributions of *Score* with small amount of observed data.

Table 6. Results of Welch's t-test.

Elapsed time[sec]	P-value	Mean of PFMC Scores	Mean of PF Scores
600	0.3340	6.401179	1.137218
1200	3.725×10^{-10}	0.3219618	0.8769369
1800	1.393×10^{-10}	0.2144606	0.7418111
2400	1.797×10^{-10}	0.1849873	0.6784018
3000	8.596×10^{-9}	0.1677461	0.6048147
3600	3.071×10^{-8}	0.1469600	0.5597454

estimation only with observed data. From the first effect's viewpoint, the rule that cut more particles is a good rule. This is due to the fact that rule is able to cut many particles from early time because the number of unique particle decreases with the time elapse in our method. Rule 1 to 3, 5 to 9, 22 and 23 are such rules. From the second effect's viewpoints, good rules are different depending on behaviors of target models. For the circadian rhythm model, it is important to reproduce the oscillations within a certain range. Rule 10 to 21 are specified for verifying the behavior of oscillation. Generally, it is not easy to prepare this kind of rules before trial. Nevertheless, unlike observed data, it will be a great help in improving the efficiency and accuracy of conventional parameter estimation process and eventually leading to better understanding of biological pathways.

4 Conclusions

We propose a novel parameter estimation method for biological pathways. By combining model checking with DA framework, our method enables us to use various knowledge in addition to observed time series data. We extract 23 biological rules with temporal logic which cannot be formulated in the form of time-series data. Proposed method and previous method are applied to *mouse* circadian rhythm model of HFPNe by means of performance evaluation. Results

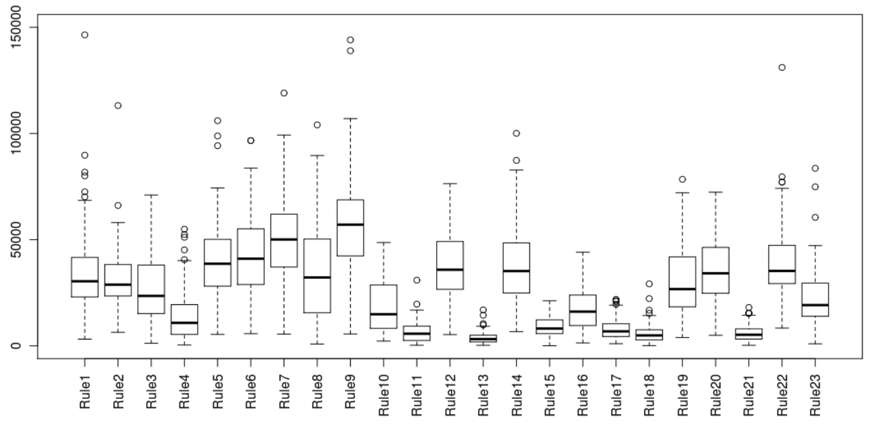

Fig. 7. The number of the particle cutting happened by model checking with respect to each rule. X-axis denotes rule number, and y-axis denotes the number of the cutting.

shows that (i) if estimations execute with enough amounts of observed data, our method can practically give good parameters in a short time; and (ii) if estimations execute with small amount of observed data, new method is much faster than the method without model checking.

References

1. Yan, H., Zhang, B., Li, S., Zhao, Q.: A Formal Model for Analyzing Drug Combination Effects and its Application in TNF-alpha-induced NFkappaB Pathway. BMC Syst Biol., 4(50) (2010)
2. Antoniotti, M., Policriti, A., Ugel, N., Mishra, B.: Model Building and Model Checking for Biochemical Processes. Cell Biochem. Biophys., 38(3), 271–286 (2003)
3. Regev, A., Silverman, W., Shapiro, E.: Representation and Simulation of Biochemical Processes Using the Pi-calculus Process Algebra. Pac. Symp. Biocomput., 459–470 (2001)
4. Chaouiya, C.: Petri Net Modelling of Biological Networks. Brief Bioinform., 8(4), 210–219 (2007)
5. Koch, I., Heiner, M.: Petri nets. In Analysis of Biological Networks. Edited by Junker, B.H., Schreiber, F. A Wiley Interscience Publication, 139–180 (2008).
6. Matsuno, H., Li, C., Miyano, S.:Petri Net Based Descriptions for Systematic Understanding of Biological Pathways. IEICE Trans. Fundamentals, E89-A(11), 3166–3174 (2006)
7. Nagasaki, M., Yamaguchi, R., Yoshida, R., Seiya, I., Doi, A., Tamada, Y., Matsuno, H., Miyano, S.: Genomic Data Assimilation for Estimating Hybrid Functional Petri Net from Time-course Gene Expression Data. Genome Inform., 17(1), 46–61, 2006.

8. Tasaki, S., Nagasaki, M., Kozuka-Hata, H., Semba, K., Gotoh, N., Hattori, S., Inoue, J., Yamamoto, T., Miyano, S., Sugano, S., Oyama, M.: Phosphoproteomics-based Modeling Defines the Regulatory Mechanism Underlying Aberrant EGFR Signaling. PLoS One, 5(11), e13926 (2010)
9. Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P., McKenzie, P.: Systems and Software Verification: Model-Checking Techniques and Tools, Springer (2001)
10. Donaldson, R., Gilbert, D.: A Model Checking Approach to the Parameter Estimation of Biochemical Pathways. In: Proceedings of the 6th International Conference on Computational Methods in Systems Biology (CMSB '08), pp. 269–287. Springer-Verlag, Berlin, Heidelberg (2008).
11. Li, C., Nagasaki, M., Hock Koh, C., Miyano, S.: Online model checking approach based parameter estimation to a neuronal fate decision simulation model in C. elegans with hybrid functional Petri net with extension, Mol. Biosyst., 7(5), 1576–1592 (2011)
12. Nagasaki, M., Saito, A., Jeong, E., Li, C., Kojima, K., Ikeda, E., Miyano, S.: Cell Illustrator 4.0: A Computational Platform for Systems Biology, In Silico Biol., 10, 0002 (2010)
13. Nagasaki, M., Doi, A., Matsuno, H., Miyano, S.: A Versatile Petri Net Based Architecture for Modeling and Simulation of Complex Biological Processes. Genome Inform., 15(1), 180–197 (2004)
14. Circadian rhythms in Mus musculus, `http://www.csml.org/models/csml-models/circadian-rhythms-in-mouse/`
15. Kitagawa, G.: Non-Gaussian State-space Modeling of Nonstationary Time Series. Journal of the American Statistical Association, 82(400), 1032–1063, 1987.
16. Kitagawa, G.: Self-organizing State Space Model. J. of the American Statistical Association, 93, 1203–1215 (1998)
17. Higuchi, T.: Self-organizing Time Series Model. In Sequential Monte Carlo Methods in Practice, Springer-Verlag New York, 429–444 (2001)
18. Kitagawa, G. and Sato, S.: Monte Carlo Smoothing and Self-Organising State-space Model. In Sequential Monte Carlo Methods in Practice, Springer-Verlag New York, 177–195 (2001)
19. Kitagawa, G.: Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models.Journal of Computational and Graphical Statistics, 5(1), 1–25 (1996)
20. Heiner, M., Lehrack, S., Gilbert, D., Marwan, W.: Extended Stochastic Petri Nets for Model-based Design of Wetlab Experiments. Edited by Priami, C., et al., Trans. on Comput. Syst. Biol. XI, LNBI, 5750, 138–163 (2009)
21. Donaldson, R., Gilbert, D.: A Monte Carlo Model Checker for Probabilistic LTL with Numerical Constraints, Technical report, University of Glasgow, Department of Computing Science (2008)
22. Hürzeler and Hans, M., Künsch, R.: Approximating and maximising the likelihood for a general state-space model. In: Sequential Monte Carlo Methods in Practice, Springer-Verlag New York, 159–175 (2001)

Appendix:

Table 7. Biological entities in the model of Figure 2. Variable $m_i(t)$ $(i = 1, \ldots, 12)$ indicates the concentration of corresponding entity at time t. $m_i(0)$ $(i = 1, \ldots, 12)$ is the initial value of corresponding entity.

Entity Name	Variable	Initial Value	Biological Type
per_mRNA	$m_1(t)$	$m_1(0)$	mRNA
PER	$m_2(t)$	$m_2(0)$	protein
Cry_mRNA	$m_3(t)$	$m_3(0)$	mRNA
CRY	$m_4(t)$	$m_4(0)$	protein
PER/CRY	$m_5(t)$	$m_5(0)$	complex protein
Rev-Erv_mRNA	$m_6(t)$	$m_6(0)$	mRNA
REV-ERV	$m_7(t)$	$m_7(0)$	protein
Clock_mRNA	$m_8(t)$	$m_8(0)$	mRNA
CLOCK	$m_9(t)$	$m_9(0)$	protein
Bmal_mRNA	$m_{10}(t)$	$m_{10}(0)$	mRNA
BMAL	$m_{11}(t)$	$m_{11}(0)$	protein
CLOCK/BMAL	$m_{12}(t)$	$m_{12}(0)$	complex protein

Table 8. Processes and their reaction speeds in the model. (k_1 and k_2 are common parameters to control speeds of similar biological processes: k_1 for protein binding. k_2 for translation.

Process Name	Biological Process Type	Speed of corresponding processes
$p1$	degradation	$v_1(t) = m_1(t) \times 0.2$
$p2$	degradation	$v_2(t) = m_2(t)/7$
$p3$	degradation	$v_3(t) = m_3(t) \times 0.2$
$p4$	degradation	$v_4(t) = m_4(t) \times 0.1$
$p5$	degradation	$v_5(t) = m_5(t)/15$
$p6$	degradation	$v_6(t) = m_6(t) \times 0.2$
$p7$	degradation	$v_7(t) = m_7(t) \times 0.1$
$p8$	degradation	$v_8(t) = m_8(t) \times 0.2$
$p9$	degradation	$v_9(t) = m_9(t)/7$
$p10$	degradation	$v_{10}(t) = m_0(t) \times 0.2$
$p11$	degradation	$v_{11}(t) = m_1(t) \times 0.1$
$p12$	degradation	$v_{12}(t) = m_2(t)/15$
$p13$	transcription	$v_{13}(t) = 1$
$p14$	transcription	$v_{14}(t) = 0.05$
$p15$	translation	$v_{15}(t) = m_1(t)/k_2$
$p16$	transcription	$v_{16}(t) = 0.05$
$p17$	transcription	$v_{17}(t) = 1$
$p18$	translation	$v_{18}(t) = m_3(t)/k_2$
$p19$	binding	$v_{19}(t) = m_2(t) \times m_4(t)/k_1$
$p20$	transcription	$v_{20}(t) = 1$
$p21$	transcription	$v_{21}(t) = 0.05$
$p22$	translation	$v_{22}(t) = m_6(t)/(2 \times k_2)$
$p23$	transcription	$v_{23}(t) = 0.5$
$p24$	translation	$v_{24}(t) = m_8(t)/k_2$
$p25$	transcription	$v_{25}(t) = 0.05$
$p26$	transcription	$v_{26}(t) = 1.1$
$p27$	translation	$v_{27}(t) = m_{10}(t)/k_2$
$p28$	binding	$v_{28}(t) = m_9(t) \times m_1(t)/k_1$

Table 9. Threshold parameters s_i $(i = 1, \ldots, 3)$ and the corresponding connectors in the model.

Threshold parameters	Name of regulation	Corresponding connector
s_1	Bmal_mRNA active regulation	$c45$
s_2	Rev-Erv_mRNA inhibitory regulation	$c40$
s_3	Cry_mRNA inhibitory regulation	$c39$