ASSA-PBN: A Toolbox for Probabilistic Boolean Networks

Andrzej Mizera, Jun Pang, Cui Su, and Qixia Yuan

Abstract—As a well-established computational framework, probabilistic Boolean networks (PBNs) are widely used for modelling, simulation, and analysis of biological systems. To analyse the steady-state dynamics of PBNs is of crucial importance to explore the characteristics of biological systems. However, the analysis of large PBNs, which often arise in systems biology, is prone to the infamous state-space explosion problem. Therefore, the employment of statistical methods often remains the only feasible solution. We present ASSA-PBN, a software toolbox for modelling, simulation, and analysis of PBNs. ASSA-PBN provides efficient statistical methods with three parallel techniques to speed up the computation of steady-state probabilities. Moreover, particle swarm optimisation (PSO) and differential evolution (DE) are implemented for the estimation of PBN parameters. Additionally, we implement in-depth analyses of PBNs, including long-run influence analysis, long-run sensitivity analysis, computation of one-parameter profile likelihoods, and the visualisation of one-parameter profile likelihoods. A PBN model of apoptosis is used as a case study to illustrate the main functionalities of ASSA-PBN and to demonstrate the capabilities of ASSA-PBN to effectively analyse biological systems modelled as PBNs.

Index Terms—Probabilistic Boolean networks, modelling, simulation and analysis of biological networks, discrete-time Markov chains, steady-state analysis, parameter estimation, long-run analysis.

1 Introduction

Systems biology aims to model and analyse biological systems from a holistic perspective in order to provide a comprehensive, system-level interpretation of cellular behaviour. Computational modelling of a biological system plays a key role in realising this purpose. It provides means to build a computational/mathematical model that complies with available biological knowledge and to identify missing biological information using formal reasoning and tools. However, it faces significant challenges when modelling real-life biological systems due to the huge size of their state space.

Among all the existing modelling frameworks, probabilistic Boolean networks (PBNs) [1], [2] are widely used for modelling large-scale biological systems. As an extension of Boolean networks, PBNs inherit appealing features of Boolean networks such as being simple yet effective at modelling biological systems such as gene regulatory networks (GRNs). In addition, PBNs are capable of handling uncertainties both on the data and model selection levels. The framework enables the analysis of the global long-run network dynamics with the tools and methods of the rich mathematical theory of discrete-time Markov chains (DTMCs). For instance, it provides means to quantify the long-run relative influences of network elements in their interactions with each other and to quantify long-run sensitivities of the system under study to various perturbations. All these characteristics are expressed in terms of specific steady-state probabilities. Therefore, efficient computation of steady-state probabilities is of crucial importance to the analysis of PBNs.

It is well studied how to compute the steady-state distributions of small PBNs (i.e. PBNs with less than 20 nodes) with numerical methods [2]. However, due to their characteristics, utilisation of these methods in the analysis of large systems is hampered by the infamous state-space explosion problem. A few statistical methods, such as Monte Carlo methods [3], have been proposed in the literature to deal with large PBNs. Nevertheless, the applicability of the existing methods/tools to PBNs is still limited by the network size, e.g. to PBNs with less than circa 100 nodes [4]. To make PBNs a generally accepted effective mathematical modelling framework for biological systems, there is a demand for a user-friendly tool which can handle large PBNs efficiently both in terms of computational time and memory usage requirements.

In this paper, we present ASSA-PBN, a software toolbox designed for modelling, simulation, and analysis of PBNs. For modelling, ASSA-PBN supports loading and saving PBNs in both high-level ASSA-PBN format and the BN/PBN MATLAB® toolbox [5] format. In addition, users can generate random PBNs according to their requirements. In terms of simulation, ASSA-PBN provides an efficient simulator, which can overcome the network size limitation. The analyser module of ASSA-PBN provides steady-state probability computation, parameter estimation, long-run influence analysis, long-run sensitivity analysis, computation of one-parameter profile likelihoods, and the visualisation of one-parameter profile likelihoods to explore the characteristics of PBNs. Computation of steady-state probabilities plays a crucial role among all the analysis methods as it forms the basis for all of them. In particular, ASSA-PBN implements numerical methods for exact analysis of small PBNs and statistical methods for approximate analysis of large PBNs. The current version supports three different statistical methods: the perfect simulation

• *A. Mizera is with the Allergology - Immunology - Inflammation Research Unit, Department of Infection and Immunity, Luxembourg Institute of Health. J. Pang and Q. Yuan are with the Computer Science and Communications Research Unit, University of Luxembourg. C. Su is with the Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg. Postal address: 6, avenue de la Fonte, L-4364 Esch-sur-Alzette, Luxembourg.*
E-mail: andrzej.mizera@lih.lu, jun.pang@uni.lu, cui.su@uni.lu, qixia.yuan@uni.lu

Manuscript received XX; revised XX.

algorithm [6], the two-state Markov chain approach [7], [8], and the Skart method [9]. To speed up the computations, ASSA-PBN additionally provides three parallel techniques: structure-based parallelisation, CPU-based parallelisation, and GPU-based parallelisation. These techniques facilitate steady-state computations that require generation of long trajectories, i.e. ones consisting of billions of states. Experimental results show that ASSA-PBN is capable of handling PBNs with thousands of nodes.

Compared to the previous version [8], [10], we add some new functionalities to ASSA-PBN. Firstly, six asynchronous update modes are implemented and the new version of ASSA-PBN supports the analysis of both synchronous and asynchronous PBNs. Secondly, in addition to the already existing particle swarm optimisation (PSO) method, the differential evolution (DE) method is implemented for parameter estimation of PBNs. Moreover, in its new version ASSA-PBN displays a fitness heat map which is a graphical representation of the quality of the fit of the model to the experimental data: colours of the individual entries in the heat map matrix reflect the relative fitness of the corresponding observable of the model to the data. Thirdly, ASSA-PBN supports the long-run sensitivity analysis with respect to a 1-bit function perturbation. Users can interactively flip the value of a selected predictor function in a single entry in the function's truth table and compute the long-run sensitivity with respect to this perturbation. Last but not least, we implement the computation and visualisation of one-parameter profile likelihoods for PBN models, which constitutes a first step towards providing a functionality for parameter identifiability analysis.

The paper is structured as follows. We present preliminaries on DTMCs and PBNs in Section 2 and explain the tool architecture in Section 3. The three modules of ASSA-PBN, i.e. the modeller, the simulator, and the analyser, are introduced respectively in Sections 4, 5, and 6, with a focus on the new features and outstanding methods. Since the analyser module contains more functionalities and new features than the other modules, we explain this module in more details. We then demonstrate the functionalities and the performance of our toolbox on a large real-life biological model in Section 7. Finally, we conclude with some directions for future work in Section 8.

2 Preliminaries

This section recalls the preliminary notions on DTMCs and PBNs needed in the course of the paper.

2.1 Finite discrete-time Markov chains (DTMCs)

A discrete-time Markov chain (DTMC) is defined as (S, s_0, P), where S is a finite set of states, $s_0 \in S$ represents the initial state and $P : S \times S \to [0,1]$ is a state transition matrix. For any two states $s, s' \in S$, $P(s,s')$ is the transition probability to transfer from state s to state s'. $P(s,s')$ satisfies $P(s,s') \geq 0$ and $\sum_{s' \in S} P(s,s') = 1$. A DTMC has an important property that the next state is independent of the past states given the present state. Formally, $\mathbb{P}(X_{t+1} = s_{t+1} \,|\, X_t = s_t, X_{t-1} = s_{t-1}, \ldots, X_0 = s_0) = \mathbb{P}(X_{t+1} = s_{t+1} \,|\, X_t = s_t)$ for all $s_0, \ldots, s_t, s_{t+1} \in S$. Here, we consider *time-homogenous* Markov chains, i.e. chains where $\mathbb{P}(X_{t+1} = s' \,|\, X_t = s)$, denoted $P(s,s')$, is independent of t for any states $s, s' \in S$. If the number of states of a DTMC is n, $\boldsymbol{\pi}_t = [\pi_t^1, \pi_t^2, \ldots, \pi_t^n]$ denotes the state probability distribution of the DTMC at some time point t. All the entries in $\boldsymbol{\pi}_t$ are non-negative and sum to one. The ith component of $\boldsymbol{\pi}_t$ represents the probability that the chain is in state i at time t. The probability distribution at time point $t+1$ can be computed by multiplying $\boldsymbol{\pi}_t$ by P, formally $\boldsymbol{\pi}_{t+1} = \boldsymbol{\pi}_t P$. We call $\boldsymbol{\pi}_t$ the *stationary distribution* of the DTMC, if it satisfies $\boldsymbol{\pi}_t P = \boldsymbol{\pi}_t$.

A path of length m is a sequence $s_1 \to s_2 \to \cdots \to s_m$ such that $P(s_i, s_{i+1}) > 0$ and $s_i \in S$ for $i \in \{1, 2, \ldots, m\}$. State $s' \in S$ is *reachable* from state $s \in S$ if there exists a path such that $s \to \cdots \to s'$. A *strongly connected component (SCC)* is a maximal subset T of S, such that for each pair of states in T, one state is reachable from the other. A DTMC is *irreducible* if all the states form a single SCC. The *period* of a state $s \in S$ is $d(s) = gcd(\{n \in \mathbb{N}_+ : \mathbb{P}(X_n = s \mid X_0 = s) > 0\})$, and we use gcd, i.e. the greatest common divisor of two or more integers, to denote the function for computing the greatest common divisor of a set of integers. Thus, starting from state s, the chain can return to s only at multiples of the period d, and d is the largest one of such integers. If all the states of the DTMC are of period one, then the DTMC is called *aperiodic*. A finite DTMC can be viewed as *ergodic* if it is both irreducible and aperiodic.

According to the famous ergodic theorem for DTMCs [11], an ergodic chain possesses a unique stationary distribution, referred to as the *steady-state distribution*, and it is equal to the *limiting distribution* given by $\lim_{n\to\infty} \boldsymbol{\pi}_0 P^n$, where $\boldsymbol{\pi}_0$ is any initial probability distribution on S. In consequence, the limiting distribution for an ergodic chain is independent of the choice of $\boldsymbol{\pi}_0$. It can be estimated by starting from any initial probability distribution on S and iteratively multiplying it by P.

For a more systematic and detailed explanation of the concepts related to DTMCs, we refer to [11].

2.2 Probabilistic Boolean networks (PBNs)

A probabilistic Boolean network $G(V, \mathscr{F})$ consists of a set of binary nodes (commonly referred to as *genes*) $V = \{x_1, x_2, \ldots, x_n\}$ and a list of sets $\mathscr{F} = (F_1, F_2, \ldots, F_n)$, where n is the number of nodes. Each node $x_i \in V$, $i = 1, 2, \ldots, n$, has associated a set $F_i \in \mathscr{F}$ of Boolean functions, referred to as *predictor functions*: $F_i = \{f_1^{(i)}, f_2^{(i)}, \ldots, f_{\ell(i)}^{(i)}\}$, where $\ell(i)$ is the number of predictor functions of node x_i. Each $f_j^{(i)} \in F_i$ is a Boolean function defined with respect to a subset of V referred to as *parent nodes* for $f_j^{(i)}$ and denoted $Pa(f_j^{(i)})$. For each node $x_i \in V$ there is a probability distribution on F_i: each predictor function $f_j^{(i)} \in F_i$ has an associated *selection probability* denoted $c_j^{(i)}$; it holds that $\sum_{j=1}^{\ell(i)} c_j^{(i)} = 1$. We denote by $x_i(t)$ the value of node x_i at time point $t \in \mathbb{N}$. The state space of the PBN is $S = \{0,1\}^n$ and its size is 2^n. The state of the network at time point t is determined by $\boldsymbol{x}(t) = (x_1(t), x_2(t), \ldots, x_n(t))$. We consider in this paper *independent* PBNs where Boolean functions for different nodes are chosen independently of each other. To update the state of a PBN, there are in general two update modes: *synchronous mode* and *asynchronous mode*, of which we give detailed descriptions as follows.

Synchronous PBNs. In the synchronous mode, the states of all the nodes are updated at the same time. At time point t, the transition from $\boldsymbol{x}(t)$ to $\boldsymbol{x}(t+1)$ is conducted by randomly selecting a predictor function for each node x_i from F_i and by synchronously updating the node values in accordance with the selected functions. With the assumption of independence among the choice of Boolean functions for individual nodes, there exist $N = \prod_{i=1}^{n} \ell(i)$ different ways in which the predictor functions can be selected for all n nodes. These combinations are called *realisations* of the PBN and are represented as n-dimensional function vectors $\boldsymbol{f}_k = (f^{(1)}_{k_1}, f^{(2)}_{k_2}, \ldots, f^{(n)}_{k_n}) \in F_1 \times F_2 \times \ldots \times F_n$, where $k \in \{1,2,\ldots,N\}$ and $k_i \in \{1,2,\ldots,\ell(i)\}$. A realisation selected at time t is referred to as $\boldsymbol{F}_t$. Due to independence, $\mathbb{P}(\boldsymbol{f}_k) = \mathbb{P}(\boldsymbol{F}_t = \boldsymbol{f}_k) = \prod_{i=1}^{n} c^{(i)}_{k_i}$. The state of the PBN at time point $t+1$ is determined by: $\boldsymbol{x}(t+1) = \boldsymbol{F}_t(\boldsymbol{x}(t))$.

Asynchronous PBNs. The asynchronous mode [12] consists of six different submodes, presented as follows:

- For the ROG (a random selection of one gene) mode, at each time point t ($t \in \mathbb{N}$), one gene is randomly selected for state update and the other genes remain unchanged. In this mode, the same gene can be updated in consecutive time steps.
- For the RMG (a random selection of m genes with a fixed m) mode, at each time point t ($t \in \mathbb{N}$), m distinct genes are randomly selected for simultaneous state update and the other genes remain unchanged. The value of m is chosen once prior to the system run from the range 1 to n and kept fixed for all time steps. Within one time step a particular gene cannot be updated more than once. If m is 1, RMG is the same as the ROG mode.
- For the RMG-RM (a random selection of m genes with a random m) mode, at each time point t ($t \in \mathbb{N}$), m distinct genes are randomly selected for simultaneous state update and the other genes remain unchanged. The only difference of this mode with respect to RMG is that the value of m is not fixed but randomly selected from 1 to n at each time step.
- For the RMG-RO (a random selection of m genes in a random order with a fixed m) mode, at each time point t ($t \in \mathbb{N}$), m distinct genes are randomly selected for sequential state update and the other genes remain unchanged. The value of m is chosen once prior to the system run from the range 1 to n and kept fixed for all time steps. The states of the m chosen genes are updated one-by-one in a random order without repetition. In result, the update of the next gene may be influenced by the new states of the previously updated genes in this time step.
- For the RMG-RO-RM (a random selection of m genes in a random order with a random m) mode, at each time point t ($t \in \mathbb{N}$), m distinct genes are randomly selected for sequential state update and the other genes remain unchanged. The only difference of this mode with respect to RMG-RO is that the value of m is not fixed but randomly selected from 1 to n at each time step.
- For the AG-RO (all genes in a random order) mode, at each time point t ($t \in \mathbb{N}$), the states of all genes are updated sequentially in a random order without repetition. If m is equal to n, the RMG-RO mode is equivalent with AG-RO.

PBNs with perturbations. The concept of *perturbations* is introduced to PBNs as a probability parameter p, where $0 < p < 1$, and it is characterised by a random perturbation vector $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_n)$, where $\gamma_i \in \{0,1\}$ and $P(\gamma_i = 1) = p$ for all t and $i \in \{1,2\ldots n\}$. Perturbations provide an alternative way to regulate the dynamics of a PBN: the next state is determined as $\boldsymbol{x}(t+1) = \boldsymbol{F}_t(\boldsymbol{x}(t))$ if $\boldsymbol{\gamma}(t) = 0$ and as $\boldsymbol{x}(t+1) = \boldsymbol{x}(t) \oplus \boldsymbol{\gamma}(t)$ otherwise, where $\oplus$ is the exclusive or operator for vectors. The probability that no node is perturbed is $(1-p)^n$, and the probability that at least one node is perturbed is $1-(1-p)^n$. A PBN with perturbations ensures that the chain simulated from this PBN will never get stuck in any state, and thus the corresponding DTMC is irreducible and aperiodic. Therefore, the dynamics of a PBN with perturbations can be viewed as an ergodic DTMC [13]. This allows us to use the theory of ergodic Markov chains to study the dynamics of a PBN with perturbations. In a PBN with perturbations, the transition probability from state s to state s' can be expressed as

$$\begin{aligned} P(s,s') \quad = \quad & (1-p)^n \textstyle\sum_{k=1}^{N} \mathbb{1}_{[\boldsymbol{f}_k(s)=s']} \mathbb{P}(\boldsymbol{f}_k) + \\ & (1-\mathbb{1}_{[s=s']}) p^{\eta(s,s')} (1-p)^{n-\eta(s,s')}, \end{aligned} \tag{1}$$

where $\mathbb{1}$ is the indicator function and $\eta(s,s')$ is the Hamming distance between states $s, s' \in S$.

According to the ergodic theory, adding perturbations to any PBNs ensures the long-run dynamics of the resulting PBN is governed by a unique limiting distribution, convergence to which is independent of the choice of the initial state. However, the perturbation probability value should be chosen carefully, not to dilute the behaviour of the original PBN. In this way, although the 'mathematical trick' introduces some noise to the original system, it allows to significantly simplify the analysis of the steady-state behaviour.

As an example, let us consider the PBN given in Figure 1. This PBN consists of 3 nodes $V = \{x_1, x_2, x_3\}$ and $\mathscr{F} = (F_1, F_2, F_3)$, where $F_1 = \{f^{(1)}_1\}$, $F_2 = \{f^{(2)}_1, f^{(2)}_2\}$, and $F_3 = \{f^{(3)}_1, f^{(3)}_2\}$. The Boolean functions and the selection probabilities are shown in Figure 1a. The corresponding truth table is shown in Figure 1b. Figure 1c shows the state transition graph of the PBN without perturbation in synchronous mode.

Node influence. The concept of influence in a PBN quantifies the impact of parent nodes on a target node [13], [14]. It is based on the notion of the partial derivative of a Boolean predictor function f with respect to variable x_j ($1 \le j \le n$) defined as

$$\frac{\partial f(x)}{\partial x_j} = f(x^{(j,0)}) \oplus f(x^{(j,1)}), \tag{2}$$

where $\oplus$ is addition modulo 2 (exclusive OR) and for $l \in \{0,1\}$

$$x^{(j,l)} = (x_1, x_2, \ldots, x_{j-1}, l, x_{j+1}, \ldots, x_n). \tag{3}$$

Then, the *influence of node x_j on function f* is the expected value of the partial derivative with respect to some probability distribution $D(x)$:

$$I_j(f) = \mathbb{E}_D\Big[\frac{\partial f(x)}{\partial x_j}\Big] = \mathbb{P}\Big\{\frac{\partial f(x)}{\partial x_j} = 1\Big\} = \mathbb{P}\{f(x^{(j,0)}) \neq f(x^{(j,1)})\}. \tag{4}$$

Let $c^{(i)}_k$ for $k = 1,2,\ldots,\ell(i)$ be the corresponding selection probability of Boolean function $f^{(i)}_k$ and let $I_j(f^{(i)}_k)$ represent the influence of node x_j on the Boolean function $f^{(i)}_k$. Then, the *influence of node x_j on node x_i* is defined as:

$$I_j(x_i) = \sum_{k=1}^{l(i)} I_j(f^{(i)}_k) \cdot c^{(i)}_k. \tag{5}$$

If the distribution $D(x)$ is the steady-state distribution of the PBN, the influence given by Equation (5) is the *long-run influence of node x_j on node x_i*.

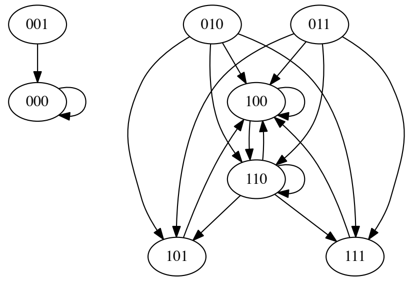
Long-run sensitivities. Long-run sensitivities quantify the extent of changes in the long-run behaviour of a system when certain modifications are introduced into it. We consider two standard

node name	function	probability
x_1	$f_1^{(1)} = x_1 \vee x_2$	1
x_2	$f_1^{(2)} = \neg x_3 \wedge (x_2 \vee x_1)$	0.3
x_2	$f_2^{(2)} = \neg x_1 \wedge x_2 \wedge x_3$	0.7
x_3	$f_1^{(3)} = \neg x_1 \wedge x_2$	0.4
x_3	$f_2^{(3)} = x_1 \wedge x_2 \wedge \neg x_3$	0.6

(a) The Boolean functions and their selection probabilities of the 3-node PBN.

$x_1x_2x_3$	$f_1^{(1)}$	$f_1^{(2)}$	$f_2^{(2)}$	$f_1^{(3)}$	$f_2^{(3)}$
000	0	0	0	0	0
001	0	0	0	0	0
010	1	1	0	1	0
011	1	0	1	1	0
100	1	1	0	0	0
101	1	0	0	0	0
110	1	1	0	0	1
111	1	0	0	0	0
$c_j^{(i)}$	1	0.3	0.7	0.4	0.6

(b) The truth table corresponding to the Boolean functions of the PBN.



(c) The state transition graph of the PBN in synchronous mode.

Fig. 1: The Boolean functions, the truth table, and the state transition graph of a PBN without perturbations in synchronous mode.

long-run sensitivities: of a predictor function and of a node (gene) [14]. We also introduce two other variants of long-run sensitivities by slightly modifying the original definitions in [15]: with respect to a 1-bit function perturbation and with respect to a selection probability perturbation.

Definition 2.1. The *sensitivity of a predictor function at state x* is defined as

$$s_x(f) = \sum_{j=1}^{n} f(x^{(j,0)}) \oplus f(x^{(j,1)}). \tag{6}$$

Then the *average sensitivity of a predictor function* is defined with respect to a probability distribution $D(x)$ as

$$s(f) = \mathbb{E}_D[s_x(f)] = \sum_{j=1}^{n} \mathbb{E}_D[f(x^{(j,0)}) \oplus f(x^{(j,1)})] = \sum_{j=1}^{n} I_j(f). \tag{7}$$

Thus, the average sensitivity of a predictor function is the sum of the influences of each node on this predictor function.

The sensitivity of a node is defined in terms of the influences of one node on another.

Definition 2.2. The *average sensitivity of a node x_i* is defined as

$$s(x_i) = \sum_{j=1}^{n} I_j(x_i). \tag{8}$$

If the distribution $D(x)$ is the steady-state distribution of a PBN, then the average sensitivity of a predictor function/node is referred to as the *long-run average sensitivity of a predictor function/node*.

Definition 2.3. The *long-run sensitivity with respect to a 1-bit function perturbation* is defined for any predictor function $f_k^{(i)}$ ($i \in \{1,2,\ldots,n\}$, $k \in \{1,2,\ldots,\ell(i)\}$) and any state of its parent nodes $\mathbf{x}_{Pa(f_k^{(i)})}$ as

$$s_f[f_k^{(i)}, \mathbf{x}_{Pa(f_k^{(i)})}] = \|\tilde{\pi}[f_k^{(i)}, \mathbf{x}_{Pa(f_k^{(i)})}] - \pi\|_l, \tag{9}$$

where $\tilde{\pi}[f_k^{(i)}, \mathbf{x}_{Pa(f_k^{(i)})}]$ is the steady-state distribution of the perturbed PBN which has a single flipped value in the truth table of $f_k^{(i)}$ at $\mathbf{x}_{Pa(f_k^{(i)})}$, $\|\cdot\|_l$ denotes the l-norm, and π is the steady-state distribution of the original PBN.

Definition 2.4. The *long-run sensitivity with respect to a selection probability perturbation* is defined as

$$s_p[c_j^{(i)} = \rho] = \|\tilde{\pi}[c_j^{(i)} = \rho] - \pi\|_l, \tag{10}$$

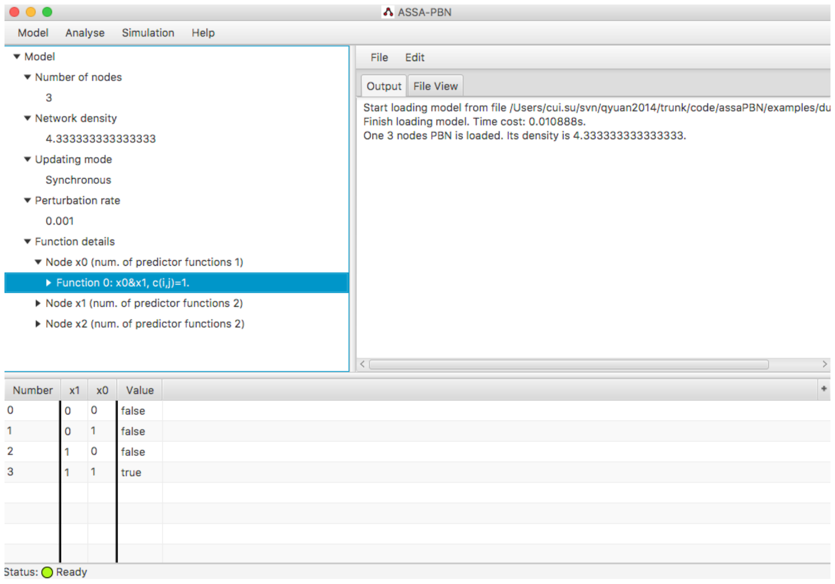
where $\rho \in [0,1]$ is the new value we set for $c_j^{(i)}$. The selection probability of the jth predictor function for node x_i is replaced with $\tilde{c}_j^{(i)} = \rho$ and all $c_k^{(i)}$ selection probabilities for $k \in I_{-j} = \{1,2,\ldots,j-1,j+1,\ldots,\ell(i)\}$ are replaced with

$$\tilde{c}_k^{(i)} = c_k^{(i)} + (c_j^{(i)} - p) \cdot \frac{c_k^{(i)}}{\sum_{l \in I_{-j}} c_l^{(i)}}. \tag{11}$$

The selection probabilities for other nodes remain unchanged. $\|\cdot\|_l$ denotes the l-norm, π and $\tilde{\pi}[c_j^{(i)} = \rho]$ are the steady-state distribution of the original PBN and the perturbed PBN, respectively.

Notice that Definition 2.3 and Definition 2.4 are similar to the definitions in [15] of the *long-run sensitivity with respect to a 1-bit function perturbation* and the *long-run sensitivity with respect to any probabilistic parameter*, respectively. There are however some subtle differences. In the above definition of the long-run sensitivity with respect to a 1-bit function perturbation, the flip of the function value is done for a single entry in the function's truth table. In consequence, the change in general is done not for a single state of a PBN as in [15], but for all states of the network in which the parent nodes of the considered predictor function have values that correspond to the flipped entry. The long-run sensitivity with respect to any probabilistic parameter of [15] is defined as an l-norm of the partial derivative of the steady-state distribution with respect to the considered parameter. Since explicit computation of the derivative is not feasible for large networks, for which ASSA-PBN is designed, we define the respective long-run sensitivity as the l-norm of the difference of the perturbed and unperturbed network steady-state distributions. In this way, our definition is not limited to infinitesimal perturbations in the parameter value, but allows for arbitrarily large changes.

Density of PBNs. The *density* of a PBN is measured with the number of its parent nodes and predictor functions. The density of a PBN G is defined as $\mathscr{D}(G) = \frac{1}{n}\sum_{i=1}^{N_F} \omega(i)$, where n is the number



- Model
  - Number of nodes
    - 3
  - Network density
    - 4.333333333333333
  - Updating mode
    - Synchronous
  - Perturbation rate
    - 0.001
  - Function details
    - Node x0 (num. of predictor functions 1)
      - Function 0: x0&x1, c(i,j)=1.
    - Node x1 (num. of predictor functions 2)
    - Node x2 (num. of predictor functions 2)

```
Start loading model from file /Users/cui.su/svn/qyuan2014/trunk/code/assaPBN/examples/du
Finish loading model. Time cost: 0.010888s.
One 3 nodes PBN is loaded. Its density is 4.333333333333333.
```

Number	x1	x0	Value
0	0	0	false
1	0	1	false
2	1	0	false
3	1	1	true

Fig. 2: Interface after loading a PBN into ASSA-PBN.

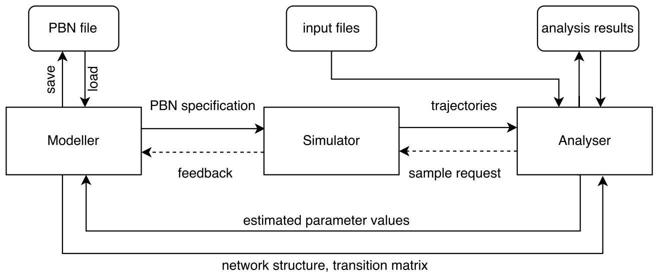


Fig. 3: The architecture of ASSA-PBN. The input files can be one or more of the following files: PBN definition files, property specification files, parameter specification files, and experimental data files.

of nodes, $\omega(i)$ is the number of parent nodes for the ith predictor function and N_F is the total number of predictor functions.

3 Toolbox Architecture

The usability of existing methods/tools for PBNs, such as optPBN [4] and the BN/PBN toolbox for MATLAB® developed by Lähdesmäki and Shmulevich [5], is restricted by the network size. For instance, as discussed in [4], optPBN can only partially analyse a 96-node PBN due to its computational efficiency issues, leaving some hypotheses regarding the network characteristics unverified. The BN/PBN toolbox applies numerical methods for computing steady-state probabilities for PBNs (see Section 7 for a more detailed discussion), which are not scalable and are impractical for the analysis of large biological networks. Therefore, we present ASSA-PBN [8], [10], a toolbox for basic and in-depth analysis of PBNs, which in particular provides several efficient methods for analysing large PBNs.

ASSA-PBN provides both a graphical user interface (GUI) and a command line interface. As shown in Figure 2, the interface is divided into three parts: the menu bar, three panels and the status bar. The panels are used to display PBN specification and the results of simulation and analysis.

The architecture of ASSA-PBN consists of three main modules: a modeller, a simulator, and an analyser, as shown in Figure 3. The three modules allow users to construct, simulate and analyse a PBN model, respectively.

The main function of the modeller is to load a PBN model from a given input file and to create its internal representation in memory or to save a PBN model in an output file. In addition, the modeller facilitates the generation of a random PBN in accordance with a user's requirements. ASSA-PBN supports the loading and saving of PBN models in either the ASSA-PBN format or the BN/PBN toolbox format.

The simulator produces trajectories (also called samples) of the loaded/generated PBN. Since this process is not based on the transition matrix of the loaded PBN, it does not suffer from the state-space explosion problem even for large PBNs. The produced trajectories are presented to the modeller and/or serve as input for further analysis.

The analyser provides several functionalities for the analysis of PBNs and different functionalities require different input files. The core function is the computation of the steady-state probability for a subset of states which is specified in a property file. The computation can be performed in either a numerical manner, suitable for small PBNs, or in a statistical manner, appropriate for large PBNs. The numerical methods are based on the state transition matrix supplied by the modeller; while the statistical methods take as input trajectories produced by the simulator. The statistical methods operate in an iterative way and extensions of the trajectories are requested from the simulator in each iteration until the sample size is large enough to obtain results satisfying the specified precision requirements.

Steady-state probabilities can be utilised by the analyser to estimate selection probability parameters of a PBN model to make it fit experimental steady-state measurements. Further, optimised parameter values are returned to the modeller. Moreover, the analyser facilitates the evaluation of long-run influences and sensitivities of the PBN. The analysis results can be used to verify and optimise the original model. For the parameter estimation and the one-parameter profile likelihood analysis, the input files include the PBN definition file, the parameter specification file, and the experimental data file. The parameter specification file specifies the indices of nodes which function selection probabilities are to be fitted and the experimental data file provides lab measurements. Details of the modules are described in the next three sections.

4 Modeller

The modeller of ASSA-PBN provides two ways for model construction: loading a PBN from a model definition file or generating a random PBN (e.g. for benchmarking and testing purposes) complying with users' requirements [10].

Users can load a PBN from a file either in ASSA-PBN model definition format or BN/PBN toolbox format. The ASSA-PBN model definition file provides various information on a PBN (see Section 2.2), including the update mode, the number of nodes, the Boolean functions for each node, the selection probability for each predictor function and the perturbation rate. ASSA-PBN supports the synchronous update mode and six types of asynchronous update modes. The details on update modes can be found in Section 2.2. A predictor function can be specified in two ways: either in the form of a truth table or with a high-level PBN definition format, where the predictor function is given as its semantic logical formula. The latter makes the node update semantics more explicit and evident. The GUI of ASSA-PBN also provides means to explore and inspect the information on predictor functions, which allows users to check the details of the model structure and semantics.

Figure 2 shows the interface after a PBN has been loaded into ASSA-PBN. The top-left panel displays general information on the loaded PBN, including its number of nodes, network density, updating mode, perturbation rate, and details on its predictor functions. The function details are shown as a tree structure in the panel. After selecting a predictor function, its truth table is shown in the bottom panel. The top-right panel additionally contains information on the the PBN model loading time.

When generating a random model, users provide the node number and they may specify some optional parameters including the maximum (minimum) number of predictor functions for a node, the maximum (minimum) number of parent nodes for a predictor function. The users may modify the default value for the perturbation rate.

Additionally, ASSA-PBN allows the user to disable perturbations for specified nodes. This feature is needed for the modelling of cellular systems where environmental conditions are kept constant, i.e. model input nodes should have fixed values, or modelling of mutants where certain nodes are inactivated or overactivated. This feature should however be used with care as it may cause the PBN's underlying DTMC to become non-ergodic.

ASSA-PBN stores the PBN model in memory with use of dedicated data structures which facilitate efficient simulation.

5 Simulator

At present, statistical approaches are practically the only viable option for the analysis of long-run dynamics of large PBNs due to the infamous state-space explosion problem. Such methods however necessitate generation of long trajectories. Therefore, the simulator module is designed to efficiently produce trajectories with the given initial states (either provided by the user or randomly set by ASSA-PBN).

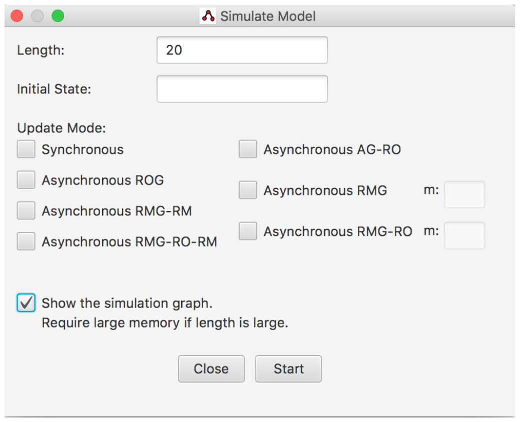
The simulation can be performed with a number of different update modes supported by ASSA-PBN (see Section 2.2), including synchronous, asynchronous ROG, asynchronous RMG and other asynchronous modes. When simulating the next state of a PBN, the simulator first checks whether perturbation should be applied. If yes, the simulator updates the current state according to the perturbation. Otherwise, the simulator updates the state of certain nodes following the update mode. For synchronous update mode, every node in the PBN is updated: a predictor function of each node is chosen according to its selection probability and the state of the node is updated with the chosen predictor function. For asynchronous update mode, depending on which submode is chosen, the states of randomly selected nodes are updated. Notice that the state transition matrix is not needed in the simulation process, which makes ASSA-PBN capable of managing large PBNs. The visualisation of the simulation result is supported in ASSA-PBN. Time-course evolution of the values of selected nodes can be displayed.

Figure 4 shows the simulator interface. Users can set trajectory length and the initial state. For example, for a three-node PBN the initial state $(x_2 = 0, x_1 = 1, x_0 = 1)$ is set by typing the space-delimited sequence 0 1 1 in the 'Initial State' field. If the checkboxes for update modes are left unchecked, the update mode defined in the definition file is used. By checking the 'Show the simulation graph' box, a graph view of the simulation results is shown in a separate window once a trajectory has been generated.

As mentioned above, the analysis of the long-run dynamics of large PBNs often requires generation of long trajectories. Therefore, efficiency of the simulation is crucial to enable the analysis of large biological networks in a reasonable computational time. To achieve this goal, ASSA-PBN offers several ways to speed up the simulation, including the alias method [16], the structure-based parallelisation technique [17], CPU-based parallelisation technique [18], and GPU-based parallelisation technique [19]. Note that the structure-based parallelisation and the GPU-based parallelisation techniques work for synchronous PBNs only.

The consecutive state is obtained by applying properly selected predictor functions to each of the nodes in a PBN. For efficiency reasons, the selection is performed with the alias method. The simulator of ASSA-PBN provides two modes: the *global alias mode* and the *local alias mode*. In the global mode, a single alias table for the joint probability distribution on the all combinations of predictor functions for all PBN nodes is built. In the local mode, individual alias tables for the distributions on predictor functions for each node are constructed. In both cases it is implicitly assumed that the predictor functions for individual nodes are selected independently of each other. In the global mode, two random numbers are needed to perform predictor functions selection for all the nodes at once, while in the local mode the number of random numbers needed is twice the number of nodes. Compared to the local mode, the simulation with the global mode is faster, but more expensive in terms of memory usage for storage of the large alias table. As a consequence, in general the local mode is recommended for large networks.

The structure-based parallel technique [17] can simplify the PBN model with synchronous update mode by reducing the network size and divide nodes into groups for parallel simulation. This is designed for multi-core CPU/multiple CPU architectures and only suits synchronous update mode. The trajectory-level parallel technique is realised both on the CPU and GPU [18], [19]. In terms of the CPU-based parallel technique [18], we


Simulate Model

Length: 20

Initial State:

Update Mode:
- [ ] Synchronous
- [ ] Asynchronous AG-RO
- [ ] Asynchronous ROG
- [ ] Asynchronous RMG m:
- [ ] Asynchronous RMG-RM
- [ ] Asynchronous RMG-RO-RM
- [ ] Asynchronous RMG-RO m:

[✓] Show the simulation graph.
Require large memory if length is large.

Fig. 4: Interface of the simulator window in ASSA-PBN.

combine the Gelman & Rubin method with the two-state Markov chain approach or the Skart method, respectively. The Gelman & Rubin method is used to monitor whether all the trajectories have approximately converged to the steady state, while the two-state Markov chain approach and the Skart method are used to determine the sample size required for steady-state probabilities computation. In terms of the GPU-based parallel technique [19], we parallelise the simulation process using multiple cores of the GPU. In order to improve the efficiency of this technique, we also develop a dynamical data arrangement mechanism to cope with PBNs of different sizes and we introduce a specific way of storing Boolean functions and states of a PBN in the GPU memory. The details of the three parallel techniques can be found in the next section.

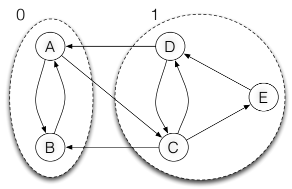
Currently, these three parallel techniques are only available for the analyser module of ASSA-PBN. Since the computation of steady-state probabilities usually requires long trajectories, the main purpose of the three parallel techniques is to speed up the simulation process greatly. The simulator module is mainly used to generate short trajectories for users to visualise the simulation result and check the correctness of the PBN.

6 Analyser

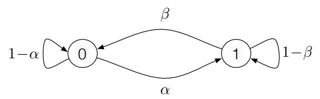
The analyser of ASSA-PBN provides four main functionalities: computation of steady-state probabilities for specified subsets of states, computation of long-run influences and various types of long-run sensitivities, parameter estimation, computation of one-parameter profile likelihoods, and the visualisation of one-parameter profile likelihoods. Computation of steady-state probabilities forms the basis for the other three tasks. The computed steady-state probabilities and the long-run influences/sensitivities provide insight into the characteristics of a given PBN model, which in turn helps to gain a better understanding of the biological system under study. Parameter estimation optimises the values of estimated parameters of the constructed PBN model to fit steady-state experimental measurements. Finally, one-parameter profile likelihoods provide insight into the structural and practical identifiability of considered model parameters.

6.1 Computation of steady-state probabilities

In the following, we first describe a few methods that ASSA-PBN implements for computing steady-state probabilities of PBNs.



(a) Original DTMC



(b) Two-state DTMC

Fig. 5: Conceptual illustration of the idea of the two-state Markov chain construction.

Then, we briefly present three parallel techniques that were recently developed to improve the efficiency of such computations.

Implemented methods. The analyser of ASSA-PBN provides two iterative numerical methods for exact computation of the steady-state distributions of PBNs, up to a pre-specified convergence criterion and numerical precision. In particular, these methods are the Jacobi method and the Gauss-Seidel method. Moreover, the analyser provides three statistical methods for computation of steady-state probabilities: the perfect simulation algorithm [20], the Skart method [9], and the two-state Markov chain approach [7]. Among all the methods, the two-state Markov chain approach is the preferred method for the computation of steady-state probabilities of large-scale PBNs.

Starting from a random initial distribution on the state space of a PBN, iterative numerical methods compute the steady-state distribution by iteratively performing matrix-vector multiplication with use of the state transition matrix. Once the required accuracy threshold is reached, the iterative process terminates and the final steady-state probability distribution is returned. Since iterative numerical methods are based on the state transition matrix, they are expensive both in term of memory and computational time consumption, hence applicable only to *small-size PBNs* (often with less than 20 nodes).

The perfect simulation algorithm [20] draws independent samples which are distributed exactly in accordance with the steady-state distribution of a DTMC. In consequence, it avoids problems related to the convergence to the steady-state distribution or non-zero correlation between consecutive states in a trajectory. The current implementation is in-line with the 'Functional backward-coupling simulation with aliasing' algorithm provided in [6]. This algorithm shortens the average coupling time significantly when only a subset of states is of interest. Nevertheless, due to the nature of this method, each state of the state space needs to be considered at each step of the coupling scheme. Therefore, this approach only suits *medium-size PBNs*, and *large PBNs* are out of its scope. Unfortunately, since PBNs with perturbations are non-monotone systems, the very efficient monotone version of perfect simulation [21], in which only a small subset of the whole state space needs to be considered, is of no use in this context.

The Skart method is a non-overlapping batch means method that can be used to estimate the steady-state probabilities of a DTMC from a simulated trajectory of the DTMC. The Skart method partitions a long simulation trajectory into batches and constructs an interval estimate with the batch means. Then the interval estimate is used to decide whether a steady-state distribution is reached or not. If not, the simulator will be called again to extend the trajectories. For more details on this method we refer to the original publication [9].

The two-state Markov chain approach [7] is a method for estimating the steady-state probability of a subset of states of a DTMC. This approach splits the state space of an arbitrary DTMC into two disjoint parts, referred to as two meta states. One of the meta states consists of the states of interest, numbered 1, the other meta states, numbered 0, is the complement of meta state 1. In this way, an arbitrary DTMC is abstracted into a binary (0-1) stochastic process, which can be approximated with a first-order, two-state DTMC that is composed of two states with transition probabilities α and β between them. Figure 5 illustrates the transformation process of a five-state Markov chain into a two-state Markov chain. The state space of the original discrete-time Markov chain is split into two meta states: states A and B form meta state 0, while states D, C, and E form meta state 1. The split of the state space into meta states is marked with dashed ellipses. To estimate the steady-state probability of meta state 1, simulation of the original DTMC is performed. By applying the two-state Markov chain approach of [7], the steady-state probability estimation is obtained. The accuracy of the estimation is governed by three precision parameters: a steady-state distribution convergence parameter ε, estimate precision parameter r, and confidence level parameter s. The two-state Markov chain approach starts with generating a trajectory of initial 'burn-in' length M_0 and initial sample size N_0. In each iteration of the algorithm, the actual 'burn-in' steps (M) and the actual sample size (N) are re-calculated based on the generated trajectory and the pre-specified values of the three precision parameters. The iteration continues until the re-estimated sample size ($M+N$) is not bigger than the current trajectory length. For more details on this approach, we refer to [7], [8]. Note that in [22], a potential initialisation problem was identified in the original approach of [7] related to the possibly unfortunate choice of the size of the initial sample and three heuristics for avoiding such initialisations have been proposed. ASSA-PBN implements the simple heuristics of [22].

The Skart method and the two-state Markov chain approach both act according to the following scheme. First, the two algorithms call the simulator to generate a trajectory of initial length. Then, the algorithms check whether the trajectory is long enough to provide an estimate of the steady-state probability that satisfies a pre-specified precision and confidence level. If the precision and confidence level requirements are not satisfied, the simulator will be called to extend the trajectory. This process is repeated until the requirements are met. For comprehensive descriptions of these methods we refer to [7], [8], [9]. A detailed comparison of the two methods can be found in [22].

Parallel computation. To produce trajectories of large synchronous PBNs, the simulator of ASSA-PBN needs to check perturbations, select Boolean functions, and perform state update for n nodes in each step. The simulation time cost can be prohibitive in the case of large PBNs and huge trajectory (sample) size. Therefore, three different techniques to speed up the generation of very long trajectories were proposed in previous works and implemented in ASSA-PBN. We briefly describe them in the following paragraphs.

Firstly, a structure-based parallelisation technique to speed up the simulation process of synchronous PBNs was proposed in [17]. This technique performs network reduction by removing unnecessary leaf nodes. Then, nodes are divided into groups and checking perturbations, making selections, and updating nodes is performed in parallel in each group. The key idea of this technique is to speed up the simulation process with the use of more memory.

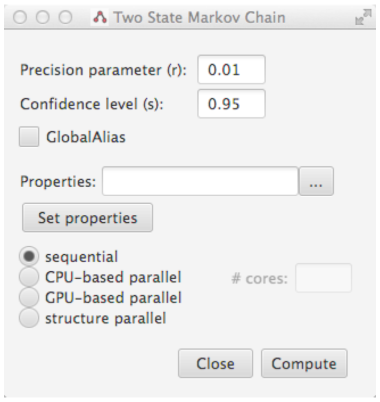
Secondly, parallelising the sample generation process is implemented to speed up the simulation process of both synchronous and asynchronous PBNs. An approach to parallelise steady-state probability computation with multiple CPU cores by combining the Gelman & Rubin method with the two statistical methods, i.e. the two-state Markov chain approach and the Skart method was proposed in [18]. The Gelman & Rubin method makes sure that all the simulated chains have approximately converged to the steady-state distribution while the two-state Markov chain approach and the Skart method [23] are used to determine the sample size required for the steady-state probabilities computation with specified precision. For a given precision, the lengths of trajectories used to estimate steady-state probabilities in the parallel approach are virtually the same as in the original statistical methods. However, since the samples are generated with multiple cores in parallel, the processing time is significantly reduced. Details on the trajectory-level parallelisation can be found in [18].

Thirdly, the trajectory-level parallelisation is further adopted to accelerate the computation of steady-state probabilities in large, currently only synchronous, PBNs with the use of GPUs. The architecture of a GPU is very different from that of a CPU and the performance of a GPU-based program is highly related to how the synchronisation between cores is realised and how memory access is managed. The framework reduces the time-consuming synchronisation cost by allowing each core to simulate one trajectory. With regard to memory management, a PBN's specification, Boolean functions and states are stored in an optimised way in the various types of GPU memory to reduce memory consumption and to improve access speed. Moreover, the framework introduces a dynamical data arrangement mechanism for handling different size PBNs with a GPU to maximise the computation efficiency on a GPU for relatively small-size PBNs. Details on the GPU-based trajectory-level parallelisation can be found in [19].

Figure 6 shows the interface for computing steady-state probabilities with the two-state Markov chain approach in ASSA-PBN. The precision and confidence level are two required parameters of the two-state Markov chain approach. The steady-state convergence parameter ε is in the current implementation fixed to 10^{10}. If "Global Alias" is checked, the simulation will be performed with the global alias mode as described in Section 5. Checking this box could potentially increase the speed of the two-state Markov chain approach at the cost of higher memory consumption. The "Properties" field allows the user to provide a file with the specification of a subset of states for which the steady-state probability is to be computed. The four radio selections are used to specify how the simulation should be performed: either in a sequential way or with the use of one of the three different parallel techniques mentioned above. If "CPU-based parallel" is selected, the gray text field "# cores" will become available and can be filled with the number of cores on the computer to be used.

6.2 Parameter estimation

A common task for building a model for a real-life biological system is to optimise the parameters of the model to make it fit experimental data. The analyser provides the parameter estimation functionality to support this task for PBN models. A few algorithms have been proposed in the literature for parameter estimation of biological systems [24]. ASSA-PBN implements the *particle swarm optimisation (PSO)* and *differential evolution (DE)* algorithms to optimise the specified parameters of PBN models.



Precision parameter (r): 0.01
Confidence level (s): 0.95
☐ GlobalAlias
Properties: ...
● sequential
○ CPU-based parallel
○ GPU-based parallel
○ structure parallel
# cores:

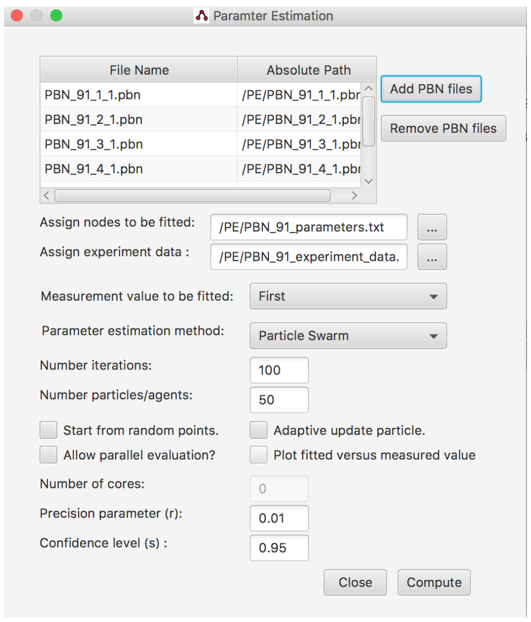
Fig. 6: Interface of computing steady-state probabilities with the two-state Markov chain approach in ASSA-PBN.

PSO [25] is an iterative method to optimise parameters of a model. The set of parameters to be optimised is called a particle. PSO solves the optimisation problem by moving a population of candidate particles around in the search space and by updating the position and speed of the particles according to the considered fitness function. In ASSA-PBN, we use the *mean square error (MSE)*, i.e. $MSE(\theta) = \frac{1}{m \cdot d} \cdot (\sum_{k=1}^{m} \sum_{l=1}^{d} (y_{k,l} - \hat{y}_{k,l}(\theta))^2)$ as the fitness function, where $y_{k,l}$ denotes m steady-state measurements for various mutants of the model for each observable l, i.e. specific subset of states, and $\hat{y}_{k,l}$ is the l-th observable's steady-state probability predicted by the mutant model k with parameters θ. In each iteration, all positions and speed of the particles are updated and verified according to the fitness function. The particle that has the minimum fitness function value is regarded as the optimal particle. Particle values are updated based on the current values and the current best optimal particle values so that each particle is moving towards the direction of the current best optimal particle.

DE is a population-based method introduced by Storn and Price in 1996 [26], [27]. It is developed to optimise real parameters by maintaining a population of candidate solutions that undergo iterations of mutation, recombination and selection. The mutations and recombinations expand the search space by creating new candidate solutions based on the weighted difference between two randomly selected population members added to a third population member. The selection process then keeps the solutions that result in better fitnesses. In conjunction with the selection, the mutation and recombination self-organise the sampling of the problem space, bounding the search space to known areas of interest.

Note that both PSO and DE are commonly known as meta-heuristics and are capable of exploring a large search space. However, meta-heuristics such as PSO and DE do not guarantee that an optimal solution is ever found.

The parameter estimation interface is shown in Figure 7. The parameter estimation method drop-down list provides two available parameter estimation methods: PSO and DE. If the option "Start from random points" is selected, the parameter estimation will start from randomly generated parameter values. Otherwise, it will use the parameters specified in the first PBN model file (usually all the mutant PBN models should have the same parameter values for the same nodes). The option "Adaptive update particle" is specific to the PSO method. If it is checked, PSO will use the adaptive update method to calculate the next position. We refer to [28] for more details on the adaptive update



File Name	Absolute Path
PBN_91_1_1.pbn	/PE/PBN_91_1_1.pbr
PBN_91_2_1.pbn	/PE/PBN_91_2_1.pbr
PBN_91_3_1.pbn	/PE/PBN_91_3_1.pbr
PBN_91_4_1.pbn	/PE/PBN_91_4_1.pbr

Assign nodes to be fitted: /PE/PBN_91_parameters.txt ...
Assign experiment data : /PE/PBN_91_experiment_data. ...
Measurement value to be fitted: First
Parameter estimation method: Particle Swarm
Number iterations: 100
Number particles/agents: 50
☐ Start from random points. ☐ Adaptive update particle.
☐ Allow parallel evaluation? ☐ Plot fitted versus measured value
Number of cores: 0
Precision parameter (r): 0.01
Confidence level (s) : 0.95

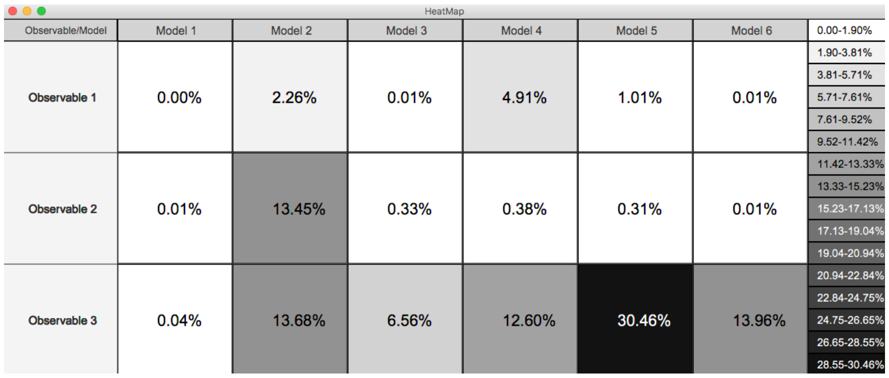
Fig. 7: Interface of parameter estimation in ASSA-PBN.

method. If the option "Allow parallel evaluation" is checked, the parameter estimation method will be run in parallel, which means that in each iteration x particles will be evaluated in parallel, where x is the number of cores. If the option "Plot fitted versus measured value" is checked, the parameter estimation result plot will be presented in a new window to the user at the end of the estimation. Once the parameter estimation procedure is finished, a fitness heat map is shown as illustrated in Figure 8. The heat map is a graphical representation of the information on the obtained model fitness: the individual values in the matrix indicate the percentage contribution of the corresponding observable of the model to the obtained overall fit score value, i.e. the MSE described above. The columns represent PBN models under different experimental conditions or model mutants and the rows represent observables, namely different subsets of states for which the steady-state probabilities are computed and compared with experimental measurements. The vertical colour bar on the right provides a mapping between a colour and corresponding range of percentage values.

6.3 Long-run influence and sensitivity

In a GRN, it is often important to distinguish which parent gene plays a major role in regulating a target gene. To explore the long-run characteristics of the GRNs, the analyser of ASSA-PBN facilitates the computation of long-run influences and sensitivities which have been introduced in Section 2.2. The long-run influences include the long-run influence of a gene on a specified predictor function and the long-run influence of a gene on another gene. The long-run sensitivities include the average long-run sensitivity of a node, the average long-run sensitivity of a predictor function, the long-run sensitivity of a gene with respect to 1-bit function perturbation, and the long-run sensitivity of a gene with respect to selection probability perturbation.

The computations of long-run influences and sensitivities are based on the computations of several steady-state probabilities. Note that ASSA-PBN does not store the generated trajectory for the sake of memory saving. Instead, ASSA-PBN verifies whether


HeatMap

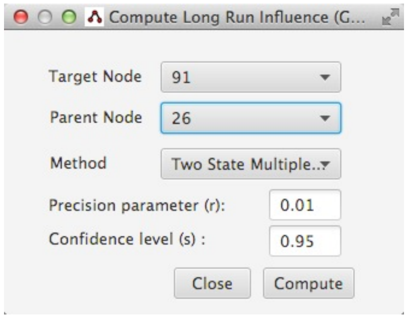
Observable/Model	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6
Observable 1	0.00%	2.26%	0.01%	4.91%	1.01%	0.01%
Observable 2	0.01%	13.45%	0.33%	0.38%	0.31%	0.01%
Observable 3	0.04%	13.68%	6.56%	12.60%	30.46%	13.96%

0.00-1.90%
1.90-3.81%
3.81-5.71%
5.71-7.61%
7.61-9.52%
9.52-11.42%
11.42-13.33%
13.33-15.23%
15.23-17.13%
17.13-19.04%
19.04-20.94%
20.94-22.84%
22.84-24.75%
24.75-26.65%
26.65-28.55%
28.55-30.46%

Fig. 8: The fitness heat map presented after performing parameter estimation in ASSA-PBN.

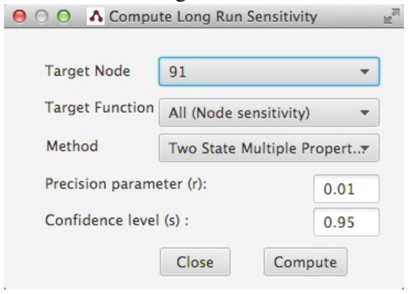
the next sampled state of the PBN belongs to the set of states of interest and stores this information only. Therefore, a new trajectory is required when computing the steady-state probability for a new set of states of interest. ASSA-PBN implements computation of steady-state probabilities of several sets of states in parallel with the two-state Markov chain approach [18], allowing the reuse of a generated trajectory. The crucial idea is that each time the next state of the PBN is generated, it is processed for all state sets of interest simultaneously. Different sets require trajectories of different lengths and the lengths are determined dynamically through an iterative process. Whenever the trajectory is long enough for obtaining the steady-state probability estimate for a certain set of states, the estimate is computed and the set will not be considered in subsequent iterations.

Figure 9a and Figure 9b show the interfaces of long-run influence of a gene on another gene and average long-run sensitivity of a node/predictor function, respectively. The first two elements of the interfaces allow the user to specify the details of the analysis to be performed while the other three parameters, i.e. the method, the precision, and the confidence level, govern the computation of the required steady-state probabilities.


Compute Long Run Influence (G...

Target Node: 91
Parent Node: 26
Method: Two State Multiple..
Precision parameter (r): 0.01
Confidence level (s) : 0.95

(a) Long-run influence of a gene on another gene interface


Compute Long Run Sensitivity

Target Node: 91
Target Function: All (Node sensitivity)
Method: Two State Multiple Propert..
Precision parameter (r): 0.01
Confidence level (s) : 0.95

(b) Average long-run sensitivity of a node/predictor function interface

Fig. 9: Interface of long-run analyses in ASSA-PBN.

6.4 Towards parameter identifiability analysis

In the current version ASSA-PBN implements the first part of the general approach of [29] to analyse arbitrary models for structural and practical identifiability. The approach is based on the concept of *profile likelihood* (PL). In this approach the fit of a model to experimental data is measured by an objective function which is the weighted sum of squared residuals

$$\chi^2(\theta) = \sum_{k=1}^{m}\sum_{l=1}^{d}\left(\frac{y_{k,l} - \hat{y}_{k,l}(\theta)}{\sigma_{k,l}}\right)^2 \qquad (12)$$

where θ is a vector of model parameter values, $y_{k,l}$ denotes m steady-state measurements for individual mutants of the model for each observable l, $\hat{y}_{k,l}(\theta)$ is the l-th observable as predicted by the mutant model k with parameter values θ, and $\sigma_{k,l}$ are the corresponding measurement errors. It is further assumed that the parameters are estimated to find $\hat{\theta} = \arg\min[\chi^2(\theta)]$. It can be shown that for normally distributed observational noise this corresponds to the maximum likelihood estimate (MLE) of θ as in this case $\chi^2(\theta) = \text{constant} - 2 \cdot \log(L(\theta))$, where $L(\theta)$ is the likelihood. In [29], the finite sample confidence intervals are considered, so called *likelihood-based confidence intervals*, defined by a confidence region $\{\theta \mid \chi^2(\theta) - \chi^2(\hat{\theta}) < \Delta_\alpha\}$ with $\Delta_\alpha = \chi^2(\alpha, \mathrm{df})$ whose borders represent confidence intervals [30].

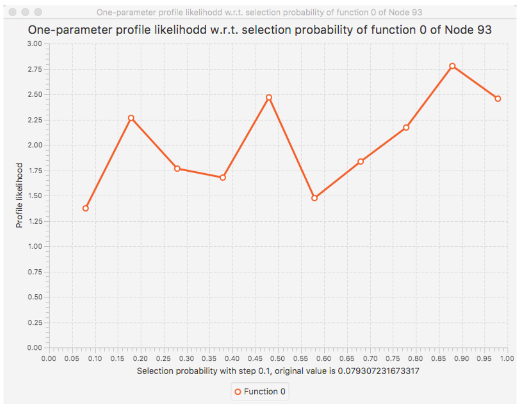


Fig. 10: Plot of a profile likelihood computed in ASSA-PBN.

In the formula above, Δ_α is the α quantile of the χ^2-distribution with df degrees of freedom and represents with $\mathrm{df} = 1$ and

$\text{df} = \dim(\theta)$ pointwise and simultaneous, respectively, confidence intervals with confidence level α.

A parameter θ_i is said to be *identifiable* if the confidence interval $[\sigma_i^-, \sigma_i^+]$ of its estimate $\hat{\theta}_i$ is finite. Two types of parameter non-identifiability are commonly considered. *Structural non-identifiability* arises from a redundant parametrisation manifested as a functional relation between ambiguous parameters that represents a manifold with constant χ^2 value in parameter space. A structural non-identifiability is related to model structure independent of experimental data. For a single parameter it is indicated by flat profile likelihood. A structurally identifiable parameter may still be *practically non-identifiable*, the second type of non-identifiability, due to the amount and quality of experimental data. By the definition of [29], a parameter is practically non-identifiable if the likelihood-based confidence region is infinitely extended, i.e. the increase in χ^2 stays below the threshold Δ_α, in the increasing and/or decreasing direction of θ_i although the likelihood has a unique minimum for this parameter.

The identification of potential structural or practical non-identifiability is based on the exploration of the parameter space in the direction of the least increase in χ^2. For this purpose the profile likelihood χ^2_{PL} is calculated for each parameter individually as $\chi^2_{\text{PL}}(\theta_i) = \min_{\theta_{j \neq i}}[\chi^2_{\text{PL}}]$ by re-optimisation of χ^2 with respect to all other parameters, for each value of θ_i.

The current version of ASSA-PBN facilitates the computation and visualisation of the profile likelihood for a specified parameter. However, since information on measurement errors is not considered in the current version, all $\sigma_{k,l}$ are set to 1 and the finite likelihood-based confidence intervals are not computed. The still missing elements are planned to be implemented in the next releases of ASSA-PBN. An example of a profile likelihood plot computed in ASSA-PBN is shown Figure 10.

7 A BIOLOGICAL CASE STUDY

In this section, we demonstrate how to analyse a PBN model with the use of the functionalities of ASSA-PBN. As a case study we consider a PBN model of an apoptosis network originally presented in [4] and shown in Figure 11. The PBN model comprises 91 nodes (state-space of size 2^{91}) and 107 Boolean functions.

Parameter estimation. To illustrate the parameter estimation function of ASSA-PBN, we perform parameter estimation with PSO and DE methods implemented to optimise the PBN model of the apoptosis network. Our aim is to fit the steady-state probabilities of three output nodes, i.e. apoptosis, C3ap17, and NFκB with respect to the experiment data. The estimated parameters are the selection probabilities of Boolean functions of the following nodes: IKKa, I_kBa, I_KBe, complex2, NF_kB, C8a, C3ap17, and C3ap17_2, which connect to the three output nodes. Six PBN files are used for the optimisation, each determined by fixing an input node's value according to one experimental condition. Meanwhile, we provide the experiment data file, which specifies the validated steady-state probabilities of apoptosis, C3ap17, and NFκB under each experimental condition. As shown in Figure 7, we first load the six PBN models of the apoptosis network under different experimental conditions. Then we specify the nodes to be fitted and assign the experimental data file. For both PSO and DE methods, the precision and confidence level are set to 0.01 and 0.95, respectively. We use 50 particles/agents and set the number of iterations to 100. The parameter estimation starts from the best-fit parameter set obtained in [4]. After 100 iterations, neither DE nor PSO method managed to find a different set of parameter values from the one in [4] that would result in a better fit. We notice however that the fit score value for the same set of parameter values differed slightly from the one in [4] due to the nature of statistical estimation (data not shown). The fitness heat map of PSO optimisation results is shown in Figure 8. Observables 1-3 represent apoptosis, C3ap17, and NFκB, respectively. Models 1-6 represent the PBN models under different experimental conditions. As can be seen from the graph, most of the steady-state probabilities of the three nodes in the fitted model are very close to the steady-state probabilities under different experimental conditions. The value of NFκB with respect to Model 5 is still high, therefore, further research is still needed to improve the PBN model of the apoptosis network. The results validate the reliability and efficiency of the parameter estimation function of ASSA-PBN.

Long-run influence and sensitivity analysis. Due to the inefficient generation of long trajectories in the optPBN toolbox, quantitative analysis of the apoptosis network was suggested but not conducted in [4]. To continue the in-depth analysis of the apoptosis model, we use the parameter set of the apoptosis PBN model in [4] for long-run influence and sensitivity analysis and the computation of steady-state probabilities.

As shown in Figure 11, RIP-deubi can activate complex2 (co2) in two ways: 1) by a positive feedback loop from activated C8* and P $\rightarrow$ tBid $\rightarrow$ Bax $\rightarrow$ smac $\rightarrow$ RIP-deubi $\rightarrow$ co2 $\rightarrow$ C8*-co2 $\rightarrow$ C8*, or 2) by the positive signal from UV-B irradiation (input nodes UV(1) or UV(2)) $\rightarrow$ Bax $\rightarrow$ smac $\rightarrow$ RIP-deubi $\rightarrow$ co2. The activation of the first way is dependent on the stimulation of the type 2 receptor (T2R). The second way is dependent on the activation of complex1 (co1), which requires the stimulation of the TNF receptor-1. As a consequence, co2 can be activated by RIP-deubi only in the condition of co-stimulation by TNF and either UV(1) or UV(2).

We proceed to demonstrate how to analyse with ASSA-PBN the importance of the RIP-deubi interaction with co2 with respect to the long-run behaviour of the system in the context of co-stimulation of TNF and either UV(1) or UV(2). Hence, the analysis is performed under two conditions: 1) co-stimulation of TNF and UV(1), and 2) co-stimulation of TNF and UV(2). Node co2 has three parent nodes: co1, FADD, and RIP-deubi. It has two independent predictor functions: co2 = co1 $\wedge$ FADD and co2 = co1 $\wedge$ FADD $\wedge$ RIP-deubi, whose selection probabilities are represented as $c_1^{(co2)}$ and $c_2^{(co2)}$, respectively. The analysis requires the computation of various steady-state probabilities. The two-state Markov chain approach is applied to perform the computations for the model. The confidence level s and the precision parameter r are set to 0.95 and 0.0001, respectively. The computation of the whole steady-state distribution for the apoptosis PBN model is practically intractable, since it would require the estimation of the steady-state probabilities for 2^{91} states. Therefore, we consider the joint steady-state probabilities for (apoptosis, C3ap17, NFκB). With our efficient implementation, ASSA-PBN can easily handle trajectories of length exceeding 2×10^9 for this case study.

First, we compute the long-run influence on co2 from each of its parent nodes: RIP-deubi, co1, and FADD, in accordance with the long-run influence definition presented in Section 2.2. The results are summarised in Table 1. The results indicate that compared to the influence of co1 or FADD on co2, the influence of RIP-deubi is smaller but not negligible.

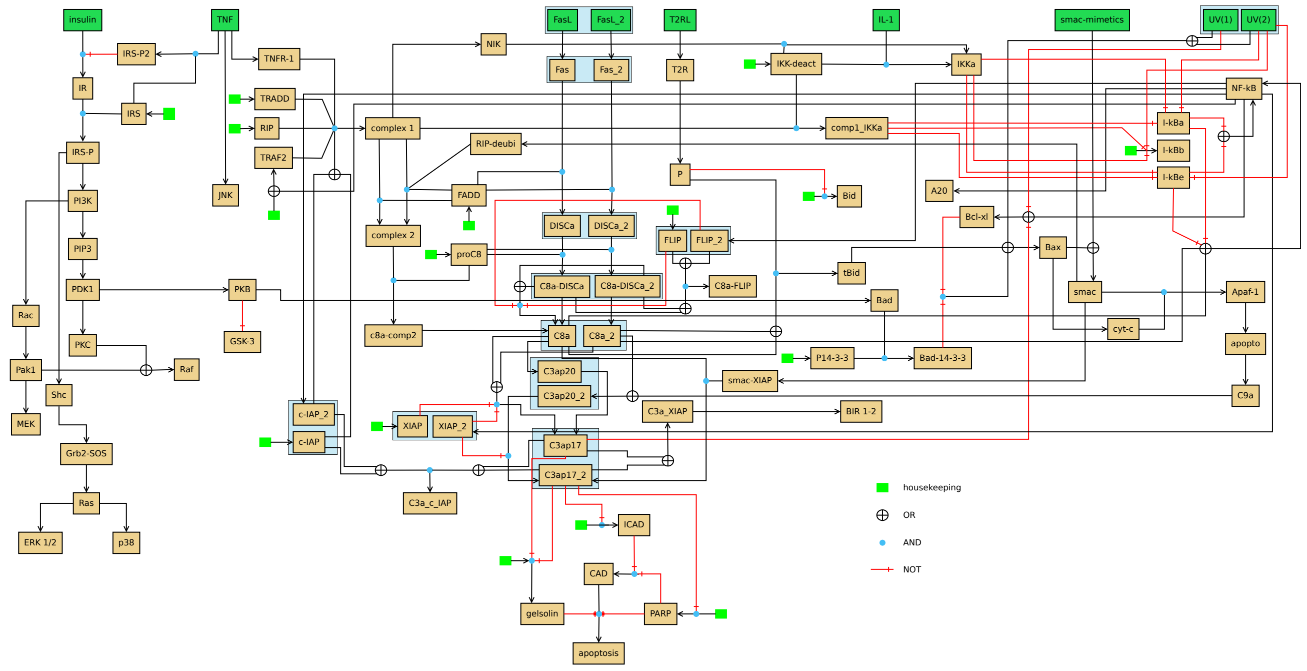


Fig. 11: The wiring of the PBN model of apoptosis originally presented in [4].

TABLE 1: Long-run influences of co1, FADD, and RIP-duebi on co2 under the co-stimulation of both TNF and UV(1) or UV(2).

	TNF and UV(1)	TNF and UV(2)
I_{co1}	0.9980	0.9980
I_{FADD}	0.9935	0.9937
$I_{\text{RIP-deubi}}$	0.2615	0.2615

TABLE 2: Long-run sensitivities w.r.t selection probability perturbations of the second Boolean function of co2.

$c_2^{(co2)}$	TNF and UV(1)	TNF and UV(2)
$+5\%$	0.0005	0.0002
-5%	0.0005	0.0004
0	0.0010	0.0011

Then, we compute various long-run sensitivities with respect to selection probability perturbation of the second predictor function of co2. In particular, we perturb the selection probability $c_2^{(co2)}$ by $\pm5\%$, in accordance with Definition 2.4. Additionally, we compute the sensitivity where $c_2^{(co2)}$ is set to 0. In consequence, $c_1^{(co2)}$ is set to 1. ASSA-PBN computes the steady-state probabilities with two-state Markov chain approach as in the case of long-run influence computation. Under the conditions of co-stimulation of TNF and UV(1) and co-stimulation of TNF and UV(2), we get the results shown in Table 2. Since the sensitivities are very small in all cases, the model is insensitive to small perturbations of the value of $c_2^{(co2)}$. Moreover, the removal of the second predictor function for co2 does not cause any drastic changes in the joint steady-state distribution for (apoptosis, C3ap17, NF-κB).

Steady-state probabilities computation. The BN/PBN toolbox for MATLAB® [5] can only apply numerical methods to compute the steady-state probabilities for PBNs. Because the numerical methods depend on the state transition matrix, obtaining which is expensive both in terms of memory usage and computational time, they are only applicable to small-size PBNs. ASSA-PBN provides not only three statistical methods besides the numerical methods, but also three parallel techniques to speed up the computation of steady-state probabilities. Compared to the BN/PBN toolbox, ASSA-PBN is much more effective in analysing large-scale PBNs. To evaluate the performance of the three parallel techniques, we consider the sequential two-state Markov chain approach as the benchmark and perform steady-state probability computations for the apoptosis PBN model with all these four methods. To make the evaluation as fair as possible, the algorithms except GPU-based parallelisation are implemented in the same programming language, i.e. Java. The GPU-based parallelisation is written in a combination of both Java and C. The latter language is used to program operations on GPUs due to the fact that no suitable Java library is currently provided for programming operations on NVIDIA® GPUs. The experiments with GPU-based parallelisation are performed on a high-performance computing (HPC) platform, of which each machine contains a CPU of Intel Xeon E5-2680 v3@2.5 GHz and an NVIDIA Tesla K80 Graphic Card with 2496 cores@824MHz. The other experiments are conducted in a HPC node, which contains 2 Intel Xeon X5675@3.07 GHz and 12 cores. This hardware architecture allows us to run a program with maximum of 12 cores in one board.

Since the apoptosis PBN network is too large to compute the whole steady-state distribution, we only consider the states of co1, FADD, and RIP-deubi. We focus on the steady-state probabilities of seven subsets of states, in which the states of the three nodes co1, FADD, and RIP-deubi are 011, 111, 101, 110, $*11$, $*10$, and $*01$, respectively. The three-character strings represent steady-state values of the three nodes in the order co1, FADD, and RIP-deubi, where 0, 1, and $*$ represent the values inactive, active, and any, respectively. The confidence level s and the precision parameter r of the two-state Markov chain approach are set to 0.95 and 5×10^{-6}, respectively.[1]

1. The value of the precision parameter r is chosen as 5×10^{-6} for the purpose of demonstrating the high performance of ASSA-PBN.

TABLE 3: Comparison of the sequential and three parallel steady-state computations on the apoptosis network.

subset of states	computed steady-state probabilities				sample size (million)			
	sequential	structure-based	CPU-based	GPU-based	sequential	structure-based	CPU-based	GPU-based
011	0.003237	0.003235	0.003236	0.003237	589.03	589.16	588.60	590.77
111	0.990051	0.990041	0.990048	0.990046	1809.68	1811.77	1809.90	1811.71
101	0.005588	0.005591	0.005588	0.005590	1015.19	1016.22	1014.61	1021.07
110	0.001084	0.001081	0.001081	0.001080	198.42	198.45	209.18	200.12
*11	0.993287	0.993284	0.993286	0.993288	1226.48	1224.37	1374.54	1241.06
*10	0.001085	0.001090	0.001085	0.001087	197.79	200.05	198.74	206.37
*01	0.005613	0.005622	0.005618	0.005624	1021.69	1023.99	1077.50	1039.35

TABLE 4: Speed-ups of three parallel steady-state computations on the apoptosis network.

subset of states	time (s)				speed-up		
	sequential	structure-based	CPU-based	GPU-based	structure-based	CPU-based	GPU-based
011	2229.46	223.65	505.08	9.28	9.97	4.41	240.95
111	6360.66	673.57	1576.96	28.08	9.45	4.03	226.75
101	3884.14	375.96	935.85	15.89	10.34	4.15	245.90
110	651.85	78.09	148.51	3.27	8.35	4.63	201.11
*11	4139.12	459.19	708.18	19.30	9.00	6.55	217.02
*10	609.19	78.40	144.57	3.36	7.86	4.23	189.24
*01	3932.02	397.57	560.24	16.17	9.91	7.40	247.35

We run the sequential two-state Markov chain approach and the three parallel versions to compute the steady-state probabilities of the seven subsets of states for the PBN model of apoptosis. Table 3 shows the the computed steady-state probability and the actual sample size (in millions) for each subset of states with the four methods. The results show that the difference between steady-state probabilities computed by the four methods is within the pre-specified precision. This demonstrates the correctness of our parallel techniques. The right part of Table 3 shows that the required sample sizes for different methods are also very similar.

The computational time cost (in seconds) and the speed-ups of these parallel techniques with respect to the sequential two-state Markov chain approach are shown in Table 4. The speed-ups for computing each subset of states are calculated with the formula speed-up $= \frac{s_{pa}/t_{pa}}{s_{se}/t_{se}}$, where s_{pa} and t_{pa} are the sample size and time cost of the parallel technique, s_{se} and t_{se} are the sample size and time cost of the sequential two-state Markov chain approach, respectively. It is obvious that the sequential two-state Markov chain needs much longer time than the parallel techniques to compute the steady-state probabilities. The density of the apoptosis network is 1.78 and 37.5% of the nodes are leaves, which makes this network suitable for the structure-based parallelisation technique. According to the speed-ups in Table 4, the structure-based technique gains speed-ups in the range 7.86–10.34, compared to the sequential two-state Markov chain approach. CPU/GPU-based parallel techniques generate samples with multiple cores in parallel, thus could reduce the required computational time significantly. As shown in Table 4, the CPU-based and GPU-based parallel techniques gain a speed-up of 4.03–7.40 and 189.24–247.35, respectively. Overall, the structure-based, CPU-based, and GPU-based parallel techniques reduce the total computational time for computing all steady-state probabilities from 6.02 hours to 38.11 minutes, 76.32 minutes, and 1.59 minutes, respectively.

8 Conclusion and Future Work

In this paper, we have presented ASSA-PBN, a software toolbox for modelling, simulation, and analysis of PBNs. The toolbox and the user guide are available at http://satoss.uni.lu/software/ASSA-PBN/. ASSA-PBN provides efficient statistical methods to approximately compute steady-state probabilities of large PBNs. In particular, ASSA-PBN provides three parallel techniques to speed up the computation of steady-state probabilities. These approaches enable ASSA-PBN to analyse large PBNs. In addition, ASSA-PBN also provides support for in-depth analyses of PBNs including parameter estimation, long-run influence and sensitivity analysis, computation of one-parameter profile likelihoods, and the visualisation of one-parameter profile likelihoods.

In the future, we plan to extend the GPU-based parallelisation technique for asynchronous PBNs as well. Next, we aim to add support for providing information on measurement errors and enable full structural and practical parameter identifiability analysis. In addition, user-friendly improvements, such as support for the standard Systems Biology Markup Language (SBML) and graphical editing and visualisation of PBN models, will be introduced in the future releases of ASSA-PBN.

Acknowledgements

Qixia Yuan was supported by the National Research Fund, Luxembourg (grant 7814267). Cui Su was supported by the research project SEC-PBN funded by the University of Luxembourg. This work was also partially supported by ANR-FNR project AlgoReCell (INTER/ANR/15/11191283). Experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg [31] (http://hpc.uni.lu). Andrzej Mizera contributed to this work while holding a postdoctoral researcher position at the Computer Science and Communications Research Unit, University of Luxembourg.

References

[1] I. Shmulevich, E. R. Dougherty, and W. Zhang, "From Boolean to probabilistic Boolean networks as models of genetic regulatory networks," *Proceedings of the IEEE*, vol. 90, no. 11, pp. 1778–1792, 2002.
[2] P. Trairatphisan, A. Mizera, J. Pang, A.-A. Tantar, J. Schneider, and T. Sauter, "Recent development and biomedical applications of probabilistic Boolean networks," *Cell Communication and Signaling*, vol. 11, p. 46, 2013.
[3] I. Shmulevich, I. Gluhovsky, R. Hashimoto, E. Dougherty, and W. Zhang, "Steady-state analysis of genetic regulatory networks modelled by probabilistic Boolean networks," *Comparative and Functional Genomics*, vol. 4, no. 6, pp. 601–608, 2003.

[4] P. Trairatphisan, A. Mizera, J. Pang, A. A. Tantar, and T. Sauter, "optPBN: An optimisation toolbox for probabilistic Boolean networks," *PLOS ONE*, vol. 9, no. 7, p. e98001, 2014.

[5] H. Lähdesmäki and I. Shmulevich, "BN/PBN Toolbox," http://code.google.com/p/pbn-matlab-toolbox, 2009, accessed 2017 March 24.

[6] J.-M. Vincent and C. Marchand, "On the exact simulation of functionals of stationary Markov chains." *Linear Algebra and its Applications.*, vol. 385, pp. 285–310, 2004.

[7] A. E. Raftery and S. Lewis, "How many iterations in the Gibbs sampler?" *Bayesian Statistics*, vol. 4, pp. 763–773, 1992.

[8] A. Mizera, J. Pang, and Q. Yuan, "ASSA-PBN: An approximate steady-state analyser for probabilistic Boolean networks," in *Proc. 13th International Symposium on Automated Technology for Verification and Analysis*, ser. LNCS, vol. 9364. Springer, 2015, pp. 214–220.

[9] A. Tafazzoli, J. Wilson, E. Lada, and N. Steiger, "Skart: A skewness- and autoregression-adjusted batch-means procedure for simulation analysis," in *Proc. 2008 Winter Simulation Conference*, 2008, pp. 387–395.

[10] A. Mizera, J. Pang, and Q. Yuan, "ASSA-PBN 2.0: A software tool for probabilistic Boolean networks," in *Proc. 14th International Conference on Computational Methods in Systems Biology*, ser. LNCS, vol. 9859. Springer, 2016, pp. 309–315.

[11] J. R. Norris, *Markov Chains*. Cambridge University Press, 1998.

[12] P. Zhu and J. Han, "Asynchronous stochastic Boolean networks as gene network models," *Journal of Computational Biology*, vol. 21, no. 10, pp. 771–783, 2014.

[13] I. Shmulevich and E. R. Dougherty, *Probabilistic Boolean Networks: The Modeling and Control of Gene Regulatory Networks*. Society for Industrial and Applied Mathematics, 2010.

[14] I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang, "Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks," *Bioinformatics*, vol. 18, no. 2, pp. 261–274, 2002.

[15] X. Qian and E. R. Dougherty, "On the long-run sensitivity of probabilistic Boolean networks," *Journal of Theoretical Biology*, vol. 257, no. 4, pp. 560–577, 2009.

[16] A. Walker, "An efficient method for generating discrete random variables with general distributions." *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 253–256, 1977.

[17] A. Mizera, J. Pang, and Q. Yuan, "Fast simulation of probabilistic Boolean networks," in *Proc. 14th International Conference on Computational Methods in Systems Biology*, ser. LNCS, vol. 9859. Springer, 2016, pp. 216–231.

[18] ——, "Parallel approximate steady-state analysis of large probabilistic Boolean networks," in *Proc. 31st Annual ACM Symposium on Applied Computing*. ACM, 2016, pp. 1–8.

[19] ——, "GPU-accelerated steady-state computation of large probabilistic boolean networks," in *Proc. 2nd International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, ser. LNCS, vol. 9984. Springer, 2016, pp. 50–66.

[20] J. Propp and D. Wilson, "Exact sampling with coupled Markov chains and applications to statistical mechanics," *Random Structures & Algorithms*, vol. 9, no. 1, pp. 223–252, 1996.

[21] D. El Rabih and N. Pekergin, "Statistical model checking using perfect simulation," in *Proc. 7th Symposium on Automated Technology for Verification and Analysis*, ser. LNCS, vol. 5799. Springer, 2009, pp. 120–134.

[22] A. Mizera, J. Pang, and Q. Yuan, "Reviving the two-state Markov chain approach," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2017, to appear.

[23] A. Tafazzoli, J. R. Wilson, E. K. Lada, and N. M. Steiger, "Skart: A skewness- and autoregression-adjusted batch-means procedure for simulation analysis," in *Proc. 2008 Winter Simulation Conference*, 2008, pp. 387–395.

[24] C. G. Moles, P. Mendes, and J. R. Banga, "Parameter estimation in biochemical pathways: a comparison of global optimization methods," *Genome Research*, vol. 13, no. 11, pp. 2467–2474, 2003.

[25] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE International Conference on Neural Networks*, 1995, pp. 1942–1948.

[26] R. Storn, "On the usage of differential evolution for function optimization," in *Proc. Biennial Conference of the North American Fuzzy Information Processing Society*, 1996, pp. 519–523.

[27] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

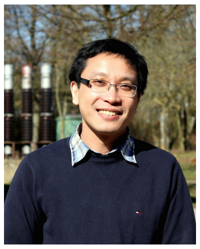
[28] A. Alfi and H. Modares, "System identification and control using adaptive particle swarm optimization," *Applied Mathematical Modelling*, vol. 35, no. 3, pp. 1210–1221, 2011.

[29] A. Raue, C. Kreutz, T. Maiwald, J. Bachmann, M. Schilling, U. Klingmüller, and J. Timmer, "Structural and practical identifiability analysis of partially observed dynamical models by exploiting the profile likelihood," *Bioinformatics*, vol. 25, no. 15, pp. 1923–1929, 2009.

[30] W. Q. Meeker and L. A. Escobar, "Teaching about approximate confidence regions based on maximum likelihood estimation," *The American Statistician*, vol. 49, no. 1, pp. 48–53, 1995.

[31] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos, "Management of an academic HPC cluster: The UL experience," in *Proc. 12th International Conference on High Performance Computing & Simulation*. IEEE CS, 2014, pp. 959–967.

Dr. Andrzej Mizera received the MSc degree in Computer Science from the University of Warsaw, Poland in 2005. He then obtained his PhD in Computer Science with minor in mathematics in the area of Computational Systems Biology from the Åbo Akademi University, Turku, Finland in 2011. His research interests are related to computational and mathematical modelling of biological systems. He is currently holding a research associate position at the Luxembourg Institute of Health.



Dr. Jun Pang received his PhD in Computer Science from Vrije Universiteit Amsterdam, The Netherlands in 2004. Currently, he is a senior researcher in the Security and Trust of Software Systems research group at the University of Luxembourg. His research interests include formal methods, security and privacy, big data analytics, and computational systems biology.

Cui Su received her MSc degrees in Systems Engineering from Yanshan University in 2016. She is currently a PhD student at the Interdisciplinary Centre for Security, Reliability and Trust at the University of Luxembourg, working on the research project SEC-PBN. Her research interests lie in computational systems biology.

Qixia Yuan received his MSc degrees in Computer Science from both the University of Luxembourg and Shandong University in 2012. He is currently a PhD student at the Computer Science and Communications Research Unit at the University of Luxembourg. His research interests focus on development and application of formal methods in systems biology.