Fast Simulation of Probabilistic Boolean Networks

Andrzej Mizera[1], Jun Pang[1,2], and Qixia Yuan[1]*

[1] Faculty of Science, Technology and Communication , University of Luxembourg
[2] Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg
firstname.lastname@uni.lu

Abstract. As an important mathematical framework, probabilistic Boolean networks (PBNs) are widely used for modelling and analysing biological systems. PBNs are suited for modelling large biological systems, which more and more often arise in systems biology. However, the large system size poses a significant challenge to the analysis of PBNs, in particular, to the crucial analysis of their steady-state behaviour. Numerical methods for performing steady-state analyses suffer from the state-space explosion problem, which makes the utilisation of statistical methods the only viable approach. However, such methods require long simulations of PBNs, rendering the simulation speed a crucial efficiency factor. For large PBNs and high estimation precision requirements, a slow simulation speed becomes an obstacle. In this paper, we propose a structure-based method for fast simulation of PBNs. This method first performs a network reduction operation and then divides nodes into groups for parallel simulation. Experimental results show that our method can lead to an approximately 10 times speedup for computing steady-state probabilities of a real-life biological network.

1 Introduction

Systems biology aims to model and analyse biological systems from a holistic perspective in order to provide a comprehensive, system-level understanding of cellular behaviour. Computational modelling of a biological system plays a key role in systems biology. It connects the field of traditional biology with mathematics and computational science, providing a way to organize and formalize available biological knowledge in a mathematical model and to identify missing biological information using formal means. Together with biochemical techniques, computational modelling promotes the holistic understanding of real-life biological systems, leading to the study of large biological systems. This brings a significant challenge to computational modelling in terms of the state-space size of the system under study. Among the existing modelling frameworks, probabilistic Boolean networks (PBNs) is well-suited for modelling large-size biological systems. It is first introduced by Shmulevich et al. [1, 2] as a probabilistic generalisation of the standard Boolean networks (BNs) to model gene regulatory networks (GRNs). The framework of PBNs incorporates rule-based dependencies between genes and allows the systematic study of global network dynamics; meanwhile, it is capable of dealing with uncertainty, which naturally occurs at different levels in the study of biological systems.

*Supported by the National Research Fund, Luxembourg (grant 7814267).

Focusing on the wiring of a network, PBNs is essentially designed for revealing the long-run (steady-state) behaviour of a biological system. Comprehensive understanding of the long-run behaviour is vital in many contexts. For example, attractors of a GRN are considered to characterise cellular phenotypes [3]. There have been a lot of studies in analysing the long-run behaviour of biological systems for better understanding the influences of genes or molecules in the systems [4]. Moreover, steady-state analyses have been used in gene intervention and external control [5, 6], which is of special interest to cancer therapists to predict the potential reaction of a patient to treatment. In the context of PBNs, many efforts have been devoted to computing their steady-state probabilities. In [7, 8], efficient numerical methods are provided for computing the steady-state probabilities of small-size PBNs. Those methods utilise an important characteristics of PBNs, i.e., a PBN can be viewed as a discrete-time Markov chain (DTMC) and its dynamics can be studied with the use of the rich theory of DTMCs. The key idea of those methods relies on the computation of the transition matrix of the underlying DTMC of the studied PBN. They perform well for small-size PBNs. However, in the case of large-size PBNs, the state-space size becomes so huge that the numerical methods are not scalable any more.

Many efforts are then spent on addressing the challenge of the huge state-space in large-size PBNs. In fact, the use of statistical methods and Monte Carlo methods remain the only feasible approach to address the problem. In those methods, the simulation speed is an important factor in the performance of these approaches. For large PBNs and long trajectories, a slow simulation speed could render these methods infeasible as well. In our previous work [9], we have considered the two-state Markov chain approach and the Skart method for approximate analysis of large PBNs. Taking special care of efficient simulation, we have implemented these two methods in the tool ASSA-PBN [10] and successfully used it for the analysis of large PBNs with a few thousands of nodes. However, the required time cost is still expected to be reduced. This requirement is of great importance for the construction of a model, e.g., parameter estimation, and for a more precise and deep analysis of the system. In this work, we propose a structure-based method to speed up the simulation process. The method is based on analysing the structure of a PBN and consists of two key ideas: first, it removes the unnecessary nodes in the network to reduce its size; secondly, it divides the nodes into groups and performs simulation for nodes in a group simultaneously. We show with experiments that our structure-based method can significantly reduce the computation time for approximate steady-state analyses of large PBNs. To the best of our knowledge, our proposed method is the first one to apply structure-based analyses for speeding up the simulation of a PBN.

2 Preliminaries

2.1 Probabilistic Boolean networks (PBNs)

A PBN $G(X,F)$ models elements of a biological system with a set of binary-valued nodes $X = \{x_1, x_2, \ldots, x_n\}$. For each node $x_i \in X$, the update of its value is guided by a set of *predictor functions* $F_i = \{f_1^{(i)}, f_2^{(i)}, \ldots, f_{\ell(i)}^{(i)}\}$, where $\ell(i)$ is the number of predictor functions for node x_i. Each $f_j^{(i)}$ is a Boolean function whose inputs are a subset of nodes, referred to as *parent nodes* of x_i. For each node x_i, one of its predictor functions will be selected to update the value of x_i at each time point t. This selection is in accordance with a probability distribution $C_i = (c_1^{(i)}, c_2^{(i)}, \dots, c_{\ell(i)}^{(i)})$, where the individual probabilities are the *selection probabilities* for the respective elements of F_i and they sum to 1. The value of node x_i at time point t is denoted as $x_i(t)$ and the state of the PBN at time point t is denoted as $s(t) = (x_1(t), x_2(t), \dots, x_n(t))$. The state space of the PBN is $S = \{0,1\}^n$ and it is of size 2^n. There are several variants of PBNs with respect to the selection of predictor functions and the synchronisation of nodes update. In this paper, we consider the *independent synchronous* PBNs, i.e., the choice of predictor functions for each node is made independently and the values of all the nodes are updated synchronously. The transition from state $s(t)$ to state $s(t+1)$ is performed by randomly selecting a predictor function for each node x_i from F_i and by applying those selected predictor functions to update the values of all the nodes synchronously. We denote $f(t)$ the combination of all the selected predictor functions at time point t. The transition of state $s(t)$ to $s(t+1)$ can then be denoted as $s(t+1) = f(t)(s(t))$.

Perturbations of a biological system are introduced by a perturbation rate $p \in (0,1)$ in a PBN. The dynamics of a PBN is guided with both perturbations and predictor functions: at each time point t, the value of each node x_i is flipped with probability p; and if no flip happens, the value of each node x_i is updated with selected predictor functions synchronously. Let $\gamma(t) = (\gamma_1(t), \gamma_2(t), \dots, \gamma_n(t))$, where $\gamma_i(t) \in \{0,1\}$ and $\mathbb{P}(\gamma_i(t) = 1) = p$ for all t and $i \in \{1, 2, \dots, n\}$. The transition of $s(t)$ to $s(t+1)$ in PBNs with perturbations is given by $s(t+1) = s(t) \oplus \gamma(t)$ if $\gamma(t) \neq 0$, and $s(t+1) = f(t)(s(t))$ otherwise, where $\oplus$ is the element-wise exclusive or operator for vectors. According to this equation, perturbations allow the system to move from a state to any other state in one transition, hence render the underlying Markov chain irreducible and aperiodic. Thus, the dynamics of a PBN with perturbations can be viewed as an ergodic DTMC [1]. Based on the ergodic theory, the long-run dynamics of a PBN with perturbations is governed by a unique limiting distribution, convergence to which is independent of the choice of the initial state.

The density of a PBN is measured with its predictor function number and parent nodes number. For a PBN G, its density is defined as $\mathscr{D}(G) = \frac{1}{n}\sum_{i=1}^{N_F} \phi(i)$, where n is the number of nodes in G, N_F is the total number of predictor functions in G, and $\phi(i)$ is the number of parent nodes for the ith predictor function.

2.2 Simulating a PBN

A PBN can be simulated via two steps based on its definition. First, perturbation is verified for each individual node and its value is flipped if there is a perturbation. Second, if no perturbation happens for any of the nodes, the network state is updated by selecting predictor functions for all the nodes and applying them. For efficiency reason, the selection of predictor functions for each node x_i is performed with the *alias method* [11], which allows to make a selection among choices in constant time irrespective of the number of choices. The alias method requires the construction of an *alias table* of size proportional to the number of choices, based on the selection probabilities of C_i.

3 Structure-based Parallelisation

The simulation method described in the above section requires to check perturbations, make a selection and perform updating a node for n times in each step. In the case of large PBNs and huge trajectory (sample) size, the simulation time cost can become prohibitive. Intuitively, the simulation time can be reduced if the n-time operations can be speeded up, for which we propose two solutions. One is to perform *network reduction* such that the total number of nodes is reduced. The other is to perform *node-grouping* in order to parallelise the process for checking perturbations, making selections, and updating nodes. For the first solution, we analyse the PBN structure to identify those nodes that can be removed and remove them to reduce the network size; while for the second solution, we analyse the PBN structure to divide nodes into groups and perform the operations for nodes in a group simultaneously. We combine the two solutions together and refer to this simulation technique as *structure-based parallelisation*. We formalise the two solutions in the following three steps: the first solution is described in Step 1 and the second solution is described in Steps 2 and 3.

Step 1. Remove unnecessary nodes from the PBN.
Step 2. Parallelise the perturbation process.
Step 3. Parallelise updating a PBN state with predictor functions.

We describe these three steps in the following subsections.

3.1 Removing unnecessary nodes

We first identify those nodes that can be removed and perform network reduction. When simulating a PBN without perturbations, if a node does not affect any other node in the PBN, the states of all other nodes will not be affected after removing this node. If this node is *not of interest* of the analysis, e.g., we are not interested in analysing its steady-state, then this node is dispensable in a PBN without perturbations. We refer to such a dispensable node as a leaf node in a PBN and define it as follow:

Definition 1 (Leaf node). *A node in a PBN is a* leaf node *(or* leaf *for short) if and only if either (1) it is not of interest and has no child nodes or (2) it is not of interest and has no other children after iteratively removing all its child nodes which are leaf nodes.*

According to the above definition, leaf nodes can be simply removed without affecting the simulation of the remaining nodes in a PBN without perturbations. In the case of a PBN with perturbations, perturbations in the leaf nodes need to be considered. Updating states with Boolean functions will only be performed when there is no perturbation in both the leaf nodes and the non-leaf nodes. Perturbations of the leaf nodes can be checked in constant time irrespective of the number of leaf nodes as describe in Algorithm 1. The input p is the perturbation probability for each node and ℓ is the number of leaf nodes in the PBN. Then, the probability that no perturbation happens in all the leaf nodes is given by $t = (1-p)^{\ell}$. With the consideration of their perturbations, the leaf nodes can be removed without affecting the simulation of the non-leaf nodes also in a PBN with perturbations. Since the leaves are not of interest, results of analyses performed on the simulated trajectories of the reduced network, i.e., containing only

Algorithm 1 Checking perturbations of leaf nodes in a PBN

```
procedure CheckLeafNodes($p, \ell$)
    $t = pow(1-p, \ell)$;        // the probability that no perturbation happens in leaves
    if $rand() > t$ then return true;
    else return false;
    end if
end procedure
```

non-leaf nodes, will be the same as performed on trajectories of the original network, i.e., containing all the nodes.

3.2 Performing perturbations in parallel

The second step of our method speeds up the process of determining perturbations. Normally, perturbations are checked for nodes one by one. In order to speed up the simulation of a PBN, we perform perturbations for k nodes simultaneously instead of one by one. For those k nodes, there are 2^k different perturbation situations. We calculate the probability for each situation and construct an alias table based on the resulting distribution. With the alias table, we make a choice c among 2^k choices and perturb the corresponding nodes based on the choice. The choice c is an integer in $[0, 2^k)$ and for the whole network the perturbation can then be performed k nodes by k nodes using the logical bitwise *exclusive or* operation, denoted $|$. To save memory, the alias table can be reused for all the groups since the perturbation probability p for each node is the same. It might happen that the number of nodes in the last perturbation round will be less than k nodes. Assume there is k' nodes in the last round and $k' < k$. For those k' nodes, we can reuse the same alias table to make the selection in order to save memory. After getting the choice c, we perform $c = c\&m$, where & is a bitwise *and* operation and m is a mask constructed by setting the first k' bits of m's binary representation to 1 and the remaining bits to 0.

Theorem 1. *The above process for determining perturbations for the last k' nodes guarantees that the probability for each of the k' nodes to be perturbed is still p.*

Proof. Without loss of generality, we assume that in the last k' nodes, t nodes should be perturbed and the positions of the t nodes are fixed. The probability for those t fixed nodes to be perturbed is $p^t(1-p)^{k'-t}$. When we make a selection from the alias table for k nodes, there are $2^{k-k'}$ different choices corresponding to the case that t fixed position nodes in the last k' nodes are perturbed. The sum of the probabilities of the $2^{k-k'}$ different choices is $[p^t(1-p)^{k'-t}] \cdot \sum_{i=0}^{k-k'} p^i(1-p)^{k-k'-i} = p^t(1-p)^{k'-t}$. □

We present the procedures for constructing groups and performing perturbations based on the groups in Algorithm 2, where n is the given number of nodes,[3] k is the maximum number of nodes that can be perturbed simultaneously and s is the PBN's

[3] In our methods, it is clear that Step 2 and Step 3 are independent of Step 1. Thus, we consistently use n to denote the number of nodes in a PBN.

Algorithm 2 The group perturbation algorithm

```
 1: procedure PreparePerturbation($n, k$)
 2:     $g = \lceil n/k \rceil$;  $k = \lceil n/g \rceil$;  $k' = n - k * (g - 1)$;
 3:     construct the alias table $\mathbb{A}_p$; $mask = 0$; $i = 0$;
 4:     repeat  $mask = mask \mid (1 \ll i)$; $i++$;
 5:     until $i = k'$;
 6:     return $[\mathbb{A}_p, mask]$;
 7: end procedure
 8: procedure Perturbation($\mathbb{A}_p, mask, s$)
 9:     $perturbed = false$;
10:     for $(i = 0; i < g; i++)$ do
11:         $c = Next(\mathbb{A}_p)$;          //$Next(\mathbb{A}_p)$ returns a random integer based on $\mathbb{A}_p$
12:         if $c \neq 0$ then
13:             $s = s \oplus (c \ll (i * k))$;      //shift c to flip only the bits (nodes) of current group
14:             $perturbed = true$;
15:         end if
16:     end for
17:     $c = Next(\mathbb{A}_p)$ & $mask$;
18:     if $c \neq 0$ then
19:         $s = s \oplus (c \ll (i * k))$;  $perturbed = true$;
20:     end if
21:     return $[s, perturbed]$;
22: end procedure
```

current state which is represented by an integer. To obtain more balanced groups, k can be decreased in line 2. As perturbing one node equals to flipping one bit of s, perturbing nodes in a group is performed via a logical bitwise *exclusive or* operation, denoted $\oplus$ (see line 13 of Algorithm 2). Perturbing k nodes simultaneously requires 2^k double numbers to store the probabilities of 2^k different choices. The size of k is therefore restricted by the available memory.[4]

3.3 Updating nodes in parallel

The last step to speed up PBN simulation is to update a number of nodes simultaneously in accordance with their predictor functions. For this step, we need an initialisation process to divide the n nodes into m groups and construct *combined predictor functions* for each group. After this initialisation, we can select a combined predictor function for each group based on a sampled random number and apply this combined function to update the nodes in the group simultaneously.

We first describe how predictor functions of two nodes are combined. The combination of functions for more than two nodes can be performed iteratively. Let x_α and x_β be the two nodes to be considered. Their predictor functions are denoted as

[4] For the experiments, we set k to 16 and k could be bigger as long as the memory allows. However, a larger k requires larger table to store the 2^k probabilities and the performance of a CPU drops when accessing an element of a much larger table due to the large cache miss rate.

$F_\alpha = \{f_1^{(\alpha)}, f_2^{(\alpha)}, \ldots, f_{\ell(\alpha)}^{(\alpha)}\}$ and $F_\beta = \{f_1^{(\beta)}, f_2^{(\beta)}, \ldots, f_{\ell(\beta)}^{(\beta)}\}$. Further, the corresponding selection probability distributions are denoted as $C_\alpha = \{c_1^{(\alpha)}, c_2^{(\alpha)}, \ldots, c_{\ell(\alpha)}^{\alpha}\}$ and $C_\beta = \{c_1^{(\beta)}, c_2^{(\beta)}, \ldots, c_{\ell(\beta)}^{\beta}\}$. After the grouping, due to the assumed independence, the number of combined predictor functions is $\ell(\alpha) * \ell(\beta)$. We denote the set of combined predictor functions as $\bar{F}_{\alpha\beta} = \{f_1^{(\alpha)} \cdot f_1^{(\beta)}, f_1^{(\alpha)} \cdot f_2^{(\beta)}, \ldots, f_{\ell(\alpha)}^{(\alpha)} \cdot f_{\ell(\beta)}^{(\beta)}\}$, where for $i \in [1, \ell(\alpha)]$ and $j \in [1, \ell(\beta)]$, $f_i^{(\alpha)} \cdot f_j^{(\beta)}$ is a combined predictor function that takes the input nodes of functions $f_i^{(\alpha)}$ and $f_j^{(\beta)}$ as its input and combines the Boolean output of functions $f_i^{(\alpha)}$ and $f_j^{(\beta)}$ into integers as output. The combined integers range in $[0,3]$ and their 2-bit binary representations (from right to left) represent the values of nodes x_α and x_β. The selection probability for function $f_i^{(\alpha)} \cdot f_j^{(\beta)}$ is $c_i^{(\alpha)} * c_j^{(\beta)}$. It holds that $\sum_{i=1}^{\ell(\alpha)} \sum_{j=1}^{\ell(\beta)} c_i^{(\alpha)} * c_j^{(\beta)} = 1$. With the selection probabilities, we can compute the alias table for each group so that the selection of combined predictor function in each group can be performed in constant time.

We now describe how to divide the nodes into groups. Our aim is to have as few groups as possible so that the updating of all the nodes can be finished in as few rounds as possible. However, fewer groups lead to many more nodes in a group, which will result in a huge number of combined predictor functions in the group. Therefore, the number of groups has to been chosen properly so that the number of groups is as small as possible, while the combined predictor functions can be stored within the memory limit of the computer performing the simulation. Besides, nodes with only one predictor function should be considered separately since selections of predictor functions for those nodes are not needed. In the rest of this section, we first formulate the problem for dividing nodes with more than one predictor function and give our solution afterwards; then we discuss how to treat nodes with only one predictor function.

Problem formulation. Let S be a list of n items $\{\mu_1, \mu_2, \ldots, \mu_n\}$. For $i \in [1,n]$, item μ_i represents a node in a PBN with n nodes. Its weight is assigned by a function $\omega(\mu_i)$, which returns the number of predictor functions of node μ_i. We aim to find a minimum integer m to distribute the nodes into m groups such that the sum of the combined predictor functions numbers of the m groups will not exceed a memory limit θ. This is equivalent to finding a minimum m and an m-partition $S_1, S_2, \ldots, S_m$ of S, i.e., $S = S_1 \cup S_2 \cup \cdots \cup S_m$ and $S_k \cap S_\ell = \emptyset$ for $k, \ell \in \{1, 2, \ldots, m\}$, such that $\sum_{i=1}^{m} \left(\prod_{\mu_j \in S_i} \omega(\mu_j) \right) \leq \theta$.

Solution. The problem in fact has two outputs: an integer m and an m-partition. We first try to estimate a potential value of m, i.e., the lower bound of m that could lead to an m-partition of S which satisfies $\sum_{i=1}^{m} \left(\prod_{\mu_j \in S_i} \omega(\mu_j) \right) \leq \theta$. With this estimate, we then try to find an m-partition satisfying the above requirements.

Denote the *weight* of a sub-list S_i as w_i, where $w_i = \prod_{\mu_j \in S_i} \omega(\mu_j)$. The inequality in the problem description can be rewritten as $\sum_{i=1}^{m} w_i \leq \theta$. We first compute the minimum value of $\hat{m}$, denoted as $\hat{m}_{min}$, satisfying the following inequality:

$$\hat{m} \cdot \sqrt[\hat{m}]{\prod_{i=1}^{n} \omega(\mu_i)} \leq \theta. \tag{1}$$

Algorithm 3 The greedy algorithm

```
procedure FINDPARTITIONS($S$,$m$)
    sort $S$ with descending orders based on the weights of items in $S$;
    initialise $A$, an array of $m$ lists;    //initially, each $A[i]$ is an empty list
    for $(j = 0; j < S.size(); j++)$ do    //$S.size()$ returns the number of items in $S$
        among the $m$ elements of $A$,    //the weight of $A[i]$ is $w_i = \prod_{\mu_j \in A[i]} \omega(\mu_j)$
            find the one with the smallest weight and add $S[j]$ to it;
    end for
    return $A$;
end procedure
```

Theorem 2. *$\hat{m}_{min}$ is the lower bound on m that allows a partition to satisfy $\sum_{i=1}^{m} w_i \leq \theta$.*

Proof. We proceed by showing that for any $k \in \{1,2,\ldots,\hat{m}_{min}-1\}$, $\hat{m}_{min}-k$ will make the inequality unsatisfied, i.e., $\sum_{i=1}^{\hat{m}_{min}-k} w_i^{'} > \theta$, where $w_i^{'}$ is the weight of the ith sub-list in an arbitrary partition of S into $\hat{m}_{min}-k$ sub-lists. Since $\hat{m}_{min}$ is the minimum value of $\hat{m}$ that satisfies Inequality (1), we have $(\hat{m}_{min}-k) \cdot \sqrt[(\hat{m}_{min}-k)]{\prod_{i=1}^{n} \omega(\mu_i)} > \theta$. Hence,

$$(\hat{m}_{min}-k) \cdot \sqrt[(\hat{m}_{min}-k)]{\prod_{i=1}^{\hat{m}_{min}-k} w_i^{'}} > \theta. \tag{2}$$

Based on the inequality relating arithmetic and geometric means, we have

$$\sum_{i=1}^{\hat{m}_{min}-k} w_i^{'} \geq (\hat{m}_{min}-k) \cdot \sqrt[(\hat{m}_{min}-k)]{\prod_{i=1}^{\hat{m}_{min}-k} w_i^{'}}. \tag{3}$$

Combining Inequality (2) with Inequality (3) gives $\sum_{i=1}^{\hat{m}_{min}-k} w_i^{'} > \theta$. □

Starting from the lower bound, we try to find a partition of S into m sub-lists that satisfies $\sum_{i=1}^{m} w_i \leq \theta$. Since the arithmetic and geometric means of non-negative real numbers are equal if and only if every number is the same, we get the heuristic that the weight of the m sub-lists should be as equal as possible so that the sum of the weights is as small as possible. Our problem then becomes similar to the NP-hard multi-way number partition problem: to divide a given set of integers into a collection of subsets, so that the sum of the numbers in each subset are as nearly equal as possible. We adapt the greedy algorithm (see Algorithm 3 for details) for solving the multi-way number partition problem, by modifying the sum to multiplication, in order to solve our partition problem.[5] If the m-partition we find satisfies the requirement $\sum_{i=1}^{m} w_i \leq \theta$, then we get a solution to our problem. Otherwise, we need to increase m by one and try to find a new m-partition. We repeat this process until the condition $\sum_{i=1}^{m} w_i \leq \theta$ is satisfied. The whole partition process for all the nodes is described in Algorithm 4.

[5] There exist other algorithms to solve the multi-way number partition problem and we choose the greedy algorithm for its efficiency.

Algorithm 4 Partition n nodes into groups.

```
 1: procedure PARTITION($G, \theta$)
 2:     compute two lists $S$ and $S'$ based on $G$;    //$S'$ contains nodes with one predictor function
 3:     compute the lower bound $\hat{m}$ according to Inequality (1);   $m = \hat{m}$;
 4:     repeat
 5:         $A_1 = $ FINDPARTITIONS($S, m$);
 6:         $sum = \sum_{i=1}^{m} \left(\prod_{\mu_j \in A_1[i]} \omega(\mu_j)\right)$;   $m = m + 1$;          //compute the sum of weights
 7:     until $sum < \theta$;
 8:     divide $S'$ into $A_2$;  //using modified Algorithm 3: in each iteration, a node is put in a list
 9:     merge $A_1$ and $A_2$ into $A$;        //which shares most common parent nodes with this node
10:     return $A$;
11: end procedure
```

Nodes with only one predictor function are treated in line 8. We divide such nodes into groups based on their parent nodes, i.e., we put nodes sharing the most common parents into the same group. In this way, the combined predictor function size can be as small as possible such that the limited memory can handle more nodes in a group. The number of nodes in a group is also restricted by the combined predictor function size, i.e., the number of parent nodes in this group.[6] The partition is performed with an algorithm similar to Algorithm 3. The difference is that in each iteration we always add a node into a group which shares most common parent nodes with this node.

3.4 The new simulation method

We describe our new method for simulating PBNs in Algorithm 5. The procedure PREPARATION describes the whole preparation process of the three steps (network reduction for Step 1, and node-grouping for Step 2 and Step 3). The three inputs of the procedure PREPARATION are the PBN network G, the memory limit θ, and the maximum number k of nodes that can be put in a group for perturbation. The PREPARATION procedure performs network reduction and node grouping. The reduced network and the grouped nodes information are then provided for the PARALLELSIMULATION procedure via seven parameters: $\mathbb{A}_p$ and *mask* are the alias table and mask used for performing perturbations of non-leaf nodes as explained in Algorithm 2; l is the number of leaf nodes; p is the perturbation rate; $\mathbb{A}$ is an array containing the alias tables for predictor functions in all groups; F is an array containing predictor functions of all groups; and *cum* is an array storing the cumulative number of nodes in each group, i.e., $cum[0] = 0$ and $cum[i] = \sum_{j=0}^{i-1} \tau_j$ for $i \in [1, m]$, where m is the number of groups and τ_j is the number of nodes in group j. Procedure PARALLELSIMULATION simulates one step of a PBN by first checking perturbation and then updating PBNs with combined predictor functions. Perturbations for leaf nodes and non-leaf nodes have been explained

[6] In our experiments, the maximum number of parent nodes in one group is set to 18. Similar to the value of k in Step 2, the number can be larger as long as the memory can handle. However, the penalty from large cache miss rate will diminish the benefits by having fewer groups when the number of parent nodes is too large.

Algorithm 5 Structure-based PBN simulation.

```
 1: procedure PREPARATION($G, \theta, k$)
 2:     perform network reduction for $G$ and store the reduced network in $G'$;
 3:     get the number of nodes $n$ and perturbation probability $p$ from $G$;
 4:     get the number of nodes $n'$ from $G'$;  $\ell = n - n'$;
 5:     $[\mathbb{A}_p, mask]$ =PREPAREPERTURBATION($n', k$);
 6:     $PA$ =PARTITION($G', \theta$);
 7:     for each group in $PA$, compute its combined functions and put them as a list in array $F$,
 8:         and compute its alias table in array $\mathbb{A}$;
 9:     compute $cum$ as $cum[0] = 0$ and $cum[i] = \sum_{j=0}^{i-1} \tau_j$ for $i \in [1, m]$, where $m$ is the number
10:         of groups in $PA$ and $\tau_j$ is the number of nodes in group $j$;
11:     return $[\mathbb{A}_p, mask, \ell, p, \mathbb{A}, F, cum]$;
12: end procedure
13: procedure PARALLELSIMULATION($\mathbb{A}_p, mask, \mathbb{A}, F, cum, \ell, p, s$)
14:     $[s, perturbed]$ =PERTURBATION($\mathbb{A}_p, mask, s$);          //perform perturbations by group
15:     if $perturbed$ || CHECKLEAFNODES($p, \ell$) then return $s$; //check perturbations of leaves
16:     else  $s' = 0$; $count = size(\mathbb{A})$;        //$size(\mathbb{A})$ returns the number of elements in array $\mathbb{A}$
17:         for $(i = 0; i < count - 1; i++)$ do
18:             $index = Next(\mathbb{A}[i])$; //select a random integer based on the alias table of group $i$
19:             $f = F[i].get(index)$;             //obtain the predictor function at the given index
20:             $v = f[s]$;                       //$f[s]$ returns the integer output of $f$ based on state $s$
21:             $s' = s' \mid (v \ll cum[i])$;      //bit shift $v$ to update only nodes in the current group
22:         end for
23:     end if
24:     return $s'$;
25: end procedure
```

in Algorithms 1 and 2. We now explain how nodes in a group are simultaneously updated with combined predictor function. It is performed via the following three steps: 1) a random combined predictor function is selected from F based on the alias table A; 2) the output of the combined predictor function is obtained according to the current state s; 3) the nodes in this group are updated based on the output of the combined predictor function. To save memory, states are stored as integers and updating a group of nodes is implemented via a logical bitwise *or* operation. To guarantee that the update is performed on the required nodes, a shift operation is needed on the output of the selected function (line 21). The number of bits to be shifted for the current group is in fact the cumulative number of nodes of all its previous groups, which is stored in the array *cum*.

4 Evaluation

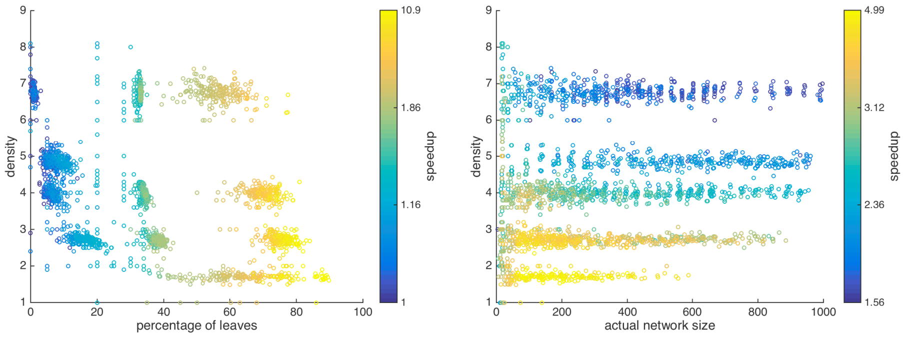
The evaluation of our new simulation method is performed on both randomly generated networks and a real-life biological network. All the experiments are performed on high performance computing (HPC) machines, each of which contains a CPU of Intel Xeon X5675 @ 3.07 GHz. The program is written in Java and the initial and maximum Java virtual machine heap size is set to 4GB and 5.89GB, respectively.

4.1 Randomly generated networks

With the evaluation on randomly generated networks, we aim not only to show the efficiency of our method, but also to answer how much speedup our method is likely to provide for a given PBN.

The first step of our new simulation method performs a network reduction technique, which is different from the node-grouping techniques in the later two steps. Therefore, we evaluate the contribution of the first step and the other two steps to the performance of our new simulation method separately. We consider the original simulation method as the reference method and we name it $Method_{ref}$. The simulation method applying the network reduction technique is referred to as $Method_{reduction}$ and the simulation method applying both the network reduction and node-grouping techniques as $Method_{new}$. $Method_{reduction}$ and $Method_{new}$ require pre-processing of the PBN under study, which leads to a certain computational overhead. However, the proportion of the pre-processing time in the whole computation decreases with the increase of the sample size. In our evaluation, we first focus on comparisons without taking pre-processing into account to evaluate the maximum potential performance of our new simulation method; we then show how different sample sizes will affect the performance when pre-processing is considered.

How does our method perform? Intuitively, the speedup due to the network reduction technique is influenced by how much a network can be reduced and the performance of node-grouping is influenced by both the density and size of a given network. Hence, the evaluation is performed on a large number of randomly generated PBNs covering different types of networks. In total, we use 2307 randomly generated PBNs with different percentages of leaves ranging between 0% and 90%; different densities ranging between 1 and 8.1; and different network sizes from the set $\{20,50,100,150,200,250,300,350,400,450,500,550,600,650,700,750,800,850,900,950,1000\}$. The networks are generated randomly using the tool ASSA-PBN [10], by providing the following information: the number of nodes, the maximum (minimum) number of predictor functions for the nodes, and the maximum (minimum) number of parent nodes for the predictor functions. Thus, the generation of these networks' density and percentage of leaves cannot be fully controlled. In other words, density and percentage of leaves for these 2307 PBNs are not uniformly distributed. We simulate 400 million steps for each of the 2307 PBNs with the three different simulation methods and compare their time costs. For the network reduction technique the speedups are calculated as the ratio between the time of $Method_{reduction}$ and the time of $Method_{ref}$, where the pre-processing time of the former method is excluded. The obtained speedups are between 1.00 and 10.90. For node-grouping, the speedups are calculated as the ratio between the time of $Method_{new}$ and the time of $Method_{reduction}$ without considering the required pre-processing times. We have obtained speedups between 1.56 and 4.99. We plot in Figure 1 the speedups of the network reduction and node-grouping techniques with respect to their related parameters. For the speedups achieved with network reduction, the related parameters are the percentage of leaves and the density. In fact, there is little influence from density to the speedup resulting from network reduction as the speedups do not change much with the different densities (see Figure 1a). The determinant factor is the percentage of leaves. The more leaves a PBN has, the more speedup we can obtain for the network. For



(a) Simulation time of $Method_{reduction}$ over simulation time of $Method_{ref}$.

(b) Simulation time of $Method_{new}$ over simulation time of $Method_{reduction}$.

Fig. 1: Speedups obtained with network reduction and node-grouping techniques. The pre-processing time is excluded from the analysis.

the speedups obtained from node-grouping, the related parameters are the density and the network size after network reduction, i.e., the number of non-leave nodes. Based on Figure 1b, the speedup with node-grouping is mainly determined by the network density: a smaller network density could result in a larger speedup contributed from the node-grouping technique. This is mainly due to the fact that sparse network has a relatively small number of predictor functions in each node and therefore, the nodes will be partitioned into fewer groups. Moreover, while the performance of network reduction is largely influenced by the percentage of leaves, the node-grouping technique tends to provide a rather stable speedup. Even for large dense networks, the technique can reduce the time cost almost by half.

The combination of these two techniques results in speedups (time of $Method_{new}$ over time of $Method_{ref}$) between 1.74 and 41.92. We plot in Figure 2 the speedups in terms of the percentage of leaves and density. The figure shows a very good performance of our new method on sparse networks with large percentage of leaves.

What is the influence of sample size? We continue to evaluate the influence of sample size on our proposed new PBN simulation method. The pre-processing time for the network reduction step is relatively very small. Therefore, our evaluation focuses on the influence of the total pre-processing time of all the three steps on the speedup of $Method_{new}$ with respect to $Method_{ref}$. We select 9 representative PBNs from the above 2307 PBNs, with respect to their densities, percentages of leaves and the speedups we have obtained. We simulate the 9 PBNs for different sample sizes using both $Method_{ref}$ and $Method_{new}$. We show the average pre-processing time of $Method_{new}$ and the obtained speedups with $Method_{new}$ (taking into account pre-processing time costs) with different sample sizes in Table 1. As expected, with the increase of the sample size, the influence of pre-processing time becomes smaller and the speedup increases. In fact, in some cases, the pre-processing time is relatively so small that its influence becomes negligible, e.g., for networks 7 and 8, where the sample size is equal or greater than 100

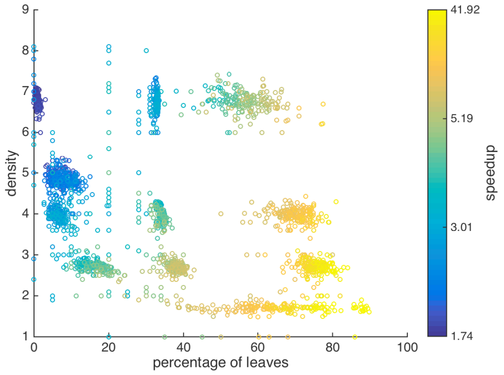


Fig. 2: Speedups of $Method_{new}$ with respect to $Method_{ref}$.

network #	size	percentage of leaves	density	average pre-processing time (second)	speedup with different sample sizes (million)			
					1	10	100	400
1	900	1.11	6.72	28.12	0.65	1.49	1.71	1.73
2	950	0.84	6.96	32.35	0.59	1.47	1.73	1.75
3	1000	0.30	7.00	33.72	0.58	1.45	1.71	1.73
4	600	67.83	4.25	162.21	0.13	1.08	4.51	6.89
5	800	68.38	3.94	43.17	0.66	3.05	6.75	7.69
6	900	68.00	3.89	36.58	0.69	3.56	6.90	7.70
7	450	89.78	1.60	0.23	21.44	37.59	41.62	41.84
8	550	88.55	1.72	0.24	20.26	35.94	36.47	36.62
9	1000	89.10	1.75	1.08	10.04	31.83	35.09	37.19

Table 1: Influence of sample sizes on the speedups of $Method_{new}$ with respect to $Method_{ref}$.

million. Moreover, often with a sample size larger than 10 million, the effort spent in pre-processing can be compensated by the saved sampling time (simulation speedup).

Performance prediction. To predict the speedup of our method for a given network, we apply regression techniques on the results of the 2307 PBNs to fit a prediction model. We use the normalised percentage of leaves and the network density as the predictor variables and the speedup of $Method_{new}$ with respect to $Method_{ref}$ as the response variable in the regression model. We do not consider network size as based on the plotted figures it does not directly affect the speedup. In the end, we obtain a polynomial regression model shown in Equation (4), which can fit 90.9% of the data:

$$y = b_1 + b_2 * x_1 + b_3 * x_1^2 + b_4 * x_2 + b_5 * x_2^2, \tag{4}$$

where $[b_1, b_2, b_3, b_4, b_5] = [2.89, 2.71, 2.40, -1.65, 0.71]$, y represents the speedup, x_1 represents the percentage of leaves and x_2 represents the network density. The result of

#	$Method_{ref}$			$Method_{new}$				speedup
	sample size (million)	time (m)	probability	pre-processing time (s)	sample size (million)	total time (m)	probability	
1	147.50	9.51	0.003243	4.57	147.82	1.05	0.003236	9.09
2	452.35	28.65	0.990049	3.10	452.25	2.79	0.990058	10.28
3	253.85	14.88	0.005583	3.42	253.99	1.74	0.005587	8.54
4	49.52	2.96	0.001087	3.38	50.39	0.36	0.001078	8.31
5	315.06	17.73	0.993293	4.40	305.43	2.05	0.993298	8.39
6	62.22	3.69	0.001088	3.13	50.28	0.39	0.001087	7.67
7	255.88	16.74	0.005621	4.01	256.61	1.70	0.005623	9.88

Table 2: Performance of $Method_{ref}$ and $Method_{new}$ on an apoptosis network.

a 10-fold cross-validation of this model supports this prediction rate. Hence, we believe this model does not overfit the given data. Based on this model, we can predict how much speedup is likely to be obtained with our proposed method for a given PBN.

4.2 An apoptosis network

In this section, we evaluate our method on a real-life biological network, i.e., an apoptosis network of 91 nodes [12]. This network has a density of 1.78 and 37.5% of the nodes are leaves, which is suitable for applying our method to gain speedups. The network has been analysed in [9]. In one of the analyses, i.e., the long-term influences [13] on complex2 from each of its parent nodes: RIP-deubi, complex1, and FADD, seven steady-state probabilities of the network need to be computed. In this evaluation, we compute the seven steady-state probabilities using our proposed structure-based simulation method ($Method_{new}$) and compare it with the original simulation method ($Method_{ref}$). The precision and confidence level of all the computations, as required by the two-state Markov chain approach [14], are set to 10^{-5} and 0.95, respectively. The results of this computation are shown in Table 2. The computed probabilities using both methods are comparable, i.e., for the same set of states, the differences of the computed probabilities are within the precision requirements. The sample sizes required by both methods for computing the same steady-state probabilities are very close to each other. Note that the speedups are computed based on the accurate data, which are slightly different from the truncated and rounded data shown in Table 2. We have obtained speedups ($Method_{new}$ with respect to $Method_{ref}$) between 7.67 and 10.28 for computing those seven probabilities. In total, the time cost is reduced from 1.5 hours to about 10 minutes.

5 Conclusion

We propose a structure-based method for speeding up simulations of PBNs. Using network reduction and node-grouping techniques, our method can significantly improve the simulation speed of PBNs. We show with experiments that our method is especially efficient in the case of analysing sparse networks with a large number of leaf nodes.

The node-grouping technique gains speedups by using more memory. Theoretically, as long as the memory can handle, the group number can be made as small as possible. However, this causes two issues in practice. First, the pre-processing time increases dramatically with the group number decreasing. Second, the performance of the method drops a lot when operating on large memories due to the increase of cache miss rate. Therefore, in our experiments we do not explore all the available memory to maximise the groups. Reducing the pre-processing time cost and the cache miss rate would be two future works to further improve the performance of our method. We plan to apply our method for the analysis of real-life large biological networks.

References

1. Shmulevich, I., Dougherty, E.R.: Probabilistic Boolean Networks: The Modeling and Control of Gene Regulatory Networks. SIAM Press (2010)
2. Trairatphisan, P., Mizera, A., Pang, J., Tantar, A.A., Schneider, J., Sauter, T.: Recent development and biomedical applications of probabilistic Boolean networks. Cell Communication and Signaling **11** (2013) 46
3. Kauffman, S.A.: Homeostasis and differentiation in random genetic control networks. Nature **224** (1969) 177–178
4. Shmulevich, I., Dougherty, E.R., Zhang, W.: From boolean to probabilistic boolean networks as models of genetic regulatory networks. Proceedings of the IEEE **90**(11) (2002) 1778–1792
5. Shmulevich, I., Dougherty, E.R., Zhang, W.: Gene perturbation and intervention in probabilistic Boolean networks. Bioinformatics **18**(10) (2002) 1319–1331
6. Shmulevich, I., Dougherty, E.R., Zhang, W.: Control of stationary behavior in probabilistic Boolean networks by means of structural intervention. Journal of Biological Systems **10**(04) (2002) 431–445
7. Shmulevich, I., Gluhovsky, I., Hashimoto, R., Dougherty, E., Zhang, W.: Steady-state analysis of genetic regulatory networks modelled by probabilistic Boolean networks. Comparative and Functional Genomics **4**(6) (2003) 601–608
8. Trairatphisan, P., Mizera, A., Pang, J., Tantar, A.A., Sauter, T.: optPBN: An optimisation toolbox for probabilistic boolean networks. PLOS ONE **9**(7) (2014)
9. Mizera, A., Pang, J., Yuan, Q.: Reviving the two-state markov chain approach (technical report). Available online at `http://arxiv.org/abs/1501.01779` (2015)
10. Mizera, A., Pang, J., Yuan, Q.: ASSA-PBN: a tool for approximate steady-state analysis of large probabilistic Boolean networks. In: Proc. 13th International Symposium on Automated Technology for Verification and Analysis. LNCS, Springer (2015) Available at `http://satoss.uni.lu/software/ASSA-PBN/`.
11. Walker, A.: An efficient method for generating discrete random variables with general distributions. ACM Transactions on Mathematical Software **3**(3) (1977) 253–256
12. Schlatter, R., Schmich, K., Vizcarra, I.A., Scheurich, P., Sauter, T., Borner, C., Ederer, M., Merfort, I., Sawodny, O.: ON/OFF and beyond - a Boolean model of apoptosis. PLOS Computational Biology **5**(12) (2009) e1000595
13. Shmulevich, I., Dougherty, E.R., Kim, S., Zhang, W.: Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks. Bioinformatics **18**(2) (2002) 261–274
14. Raftery, A.E., Lewis, S.: How many iterations in the Gibbs sampler? Bayesian Statistics **4** (1992) 763–773