Parallel Approximate Steady-state Analysis of Large Probabilistic Boolean Networks

Andrzej Mizera
University of Luxembourg
andrzej.mizera@uni.lu

Jun Pang
University of Luxembourg
jun.pang@uni.lu

Qixia Yuan*
University of Luxembourg
qixia.yuan@uni.lu

ABSTRACT

Probabilistic Boolean networks (PBNs) is a widely used computational framework for modelling biological systems. The steady-state dynamics of PBNs is of special interest in the analysis of biological machinery. However, obtaining the steady-state distributions for such systems poses a significant challenge due to the state space explosion problem which arises in the case of large PBNs. The only viable way is to use statistical methods. In the literature, the two-state Markov chain approach and the Skart method have been proposed for the analysis of large PBNs. However, the sample size required by both methods is often huge in the case of large PBNs and generating them is expensive in terms of computation time. Parallelising the sample generation is an ideal way to solve this issue. In this paper, we consider combining the Gelman & Rubin method with either the two-state Markov chain approach or the Skart method for parallelisation. The first method can be used to run multiple independent Markov chains in parallel and to control their convergence to the steady-state while the other two methods can be used to determine the sample size required for computing the steady-state probability of states of interest. Experimental results show that our proposed combinations can reduce time cost of computing stead-state probabilities of large PBNs significantly.

Keywords

probabilistic Boolean networks, Markov chains, parallel computing, steady-state analysis

1. INTRODUCTION

Computational systems biology aims to model and analyse biological systems from a holistic perspective with the use of formal, mathematical reasoning and computational techniques that exploit efficient data structures and algorithms. Computational modelling allows systematisation of available biological knowledge concerning biochemical processes of a biological system and provides formal means for the analysis and understanding of real-life systems. Unfortunately, it often arises that the size of the state space of the system to be considered is so huge that it prohibits the analysis. Thus comprehensive understanding of biological processes requires further development of efficient methods and techniques for formal modelling and analysis of biological systems.

One key aspect in the analysis of biological systems is to understand their long-run behaviour, which is crucial in many contexts, e.g., in the analysis of the long-term influence of one gene on another gene in a gene regulatory network (GRN) [9]. In this work, we concentrate on the steady-state analysis of biological mechanisms, in particular GRNs, cast into the framework of probabilistic Boolean networks. As introduced by Shmulevich et al. [10] (see [17] for a recent survey), PBNs are a probabilistic generalisation of the standard Boolean networks: they not only incorporate rule-based dependencies between genes and allow the systematic study of global network dynamics; but also provide means to deal with uncertainty, which comes naturally in biological processes. The dynamics of PBNs can be studied in the realm of discrete-time Markov chains (DTMCs). Therefore, the rich theories of DTMCs can be applied in the analysis of PBNs.

Given a PBN, one natural and crucial issue is to study the steady-state probabilities of its underlying DTMC, which characterise the long-run behaviour of the corresponding biological system [3]. Much effort has been devoted to analysing the steady-state behaviour of biological systems for better understanding of influences of genes or molecules in the systems [9]. Furthermore, steady-state analysis has been used in gene intervention and external control [13, 12], which is of special interest to cancer research to predict the potential reaction of a patient to treatment.

It has been well studied how to compute the steady-state probabilities of small-size PBNs using numerical methods [14, 16]. However, in the case of large PBNs, their state-space size becomes so huge that the numerical methods, often relying on the transition matrix of the underlying DTMC of the studied network, are not scalable any more. This poses a critical challenge for the steady-state analysis of large PBNs. In fact, approximations with Markov Chain Monte Carlo (MCMC) techniques remain the only feasible method to solve the problem. In [6], 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 [5], and successfully used it for the analysis of large PBNs

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

SAC 2016,April 04-08, 2016, Pisa, Italy

http://dx.doi.org/10.1145/2851613.2851614

with a few thousands of nodes. However, the trajectory required for analysing a large PBN is often very long and generating such long trajectories is expensive in terms of computation time. A natural idea for speeding up this is to perform the trajectory generation in parallel. In this work, we consider combining the Gelman & Rubin method [2] with the two methods considered in [6]. We simulate multiple trajectories in parallel and verify the convergence of the trajectories based on the Gelman & Rubin method. Once convergence is reached according to the Gelman & Rubin method, either the two-state Markov chain approach or the Skart method can be applied to the converged trajectories to estimate the steady-state probability for a set of states that are of interest. We show with experiments that the combinations can significantly reduce the computation time for approximate steady-state analysis of large PBNs.

2. PRELIMINARIES

2.1 Finite discrete-time Markov chains

We define a discrete-time Markov chain (DTMC) as a tuple (S, s_0, P), where S is a finite set of states, $s_0 \in S$ is the initial state, and $P : S \times S \to [0,1]$ is a transition probability matrix. For any two states s and s', an element of $P(s,s')$ defines the probability that a transition is made from state s to state s'. It satisfies that $P(s,s') \geq 0$ for all $s,s' \in S$ and $\sum_{s' \in S} P(s,s') = 1$ for all $s \in S$. A path of length n is a sequence of states $s_0, s_1, \ldots, s_{n-1}$, where $s_i \in S$ for all $i \in \{0,1,\ldots,n-1\}$ and $P(s_i, s_{i+1}) > 0$ for all $i \in \{0,1,\ldots,n-2\}$. State $s' \in S$ is said to be *reachable* from state $s \in S$ if there exists a path from s to s'. A DTMC is said to be *irreducible* if any two states in the state space are reachable from each other. A state of a DTMC is of period d where d equals to the greatest common divisor of the lengths of all paths that start and end in the state. If all states in the state space of a DTMC are of period one, then the DTMC is *aperiodic*. A finite state DTMC that is both irreducible and aperiodic is *ergodic*. Let π be a probability distribution on S. π is called a *stationary distribution* of the DTMC if $\pi = \pi \cdot P$. According to the ergodic theory of DTMC [7], an ergodic DTMC has a unique stationary distribution, being simultaneously its *limiting distribution*. It is also known as the *steady-state distribution* given by $\lim_{n\to\infty} \pi_0 \cdot P^n$, where π_0 is any initial probability distribution on S and P^n is the n times multiplication of the transition matrix P. Therefore, the limiting distribution of an ergodic DTMC is independent of the choice of the initial distribution and it can be estimated by iteratively multiplying P.

2.2 Probabilistic Boolean network

A probabilistic Boolean network $G(V, \mathscr{F})$ is composed of a set of binary-valued variables (also referred to as nodes) $V = (v_1, v_2, \ldots, v_n)$ whose values are governed by a list of sets $\mathscr{F} = (F_1, F_2, \ldots, F_n)$. The set $F_i = \{f_1^i, f_2^i, \ldots, f_{\ell(i)}^i\}$ is defined as a set of possible predictor functions for node v_i, where $i \in \{1,2,\ldots,n\}$ and $\ell(i)$ is the number of possible predictor functions for v_i. Each predictor function f_j^i is a Boolean function defined with respect to a subset of nodes referred to as parent nodes of the node v_i. At a given time point t ($t = 0,1,\ldots$), one predictor function is selected for each of the nodes. We call the combination of all the selected prediction functions at time t a *realisation* of a PBN. Assuming independence among the predictor functions for different nodes, there are $N = \prod_{i=1}^{n} \ell(i)$ possible realisations for a PBN. We denote the realisations as vectors f_k, $k \in \{1,\ldots,N\}$, where the i-th element is the Boolean function selected for node v_i. The realisation at time point t is expressed as $f(t)$. For a node v_i, the selection probability for selecting its jth predictor function is denoted as $c_j^{(i)}$ and it holds that $\sum_{j=1}^{\ell(i)} c_j^{(i)} = 1$ for all $i \in \{1,2,\ldots,n\}$. The state of a PBN at time point t, denoted as $s(t)$, is defined as a collection of all the node values at time t, namely $s(t) = (v_1(t), v_2(t), \ldots, v_n(t))$, where $v_i(t)$ is the value of node v_i at time t. A PBN with n nodes has 2^n possible states and $s(t) \in \{0,1\}^n$ for each t. The transition from $s(t)$ to $s(t+1)$ is conducted by synchronously updating the node values according to the realisation at time t, i.e., $s(t+1) = f(t)(s(t))$.

The concept of perturbations is introduced to PBN by providing a parameter $p \in (0,1)$, which is used to sample a perturbation vector $\gamma(t) = (\gamma_1(t), \gamma_2(t), \ldots, \gamma_n(t))$, where each $\gamma_i(t) \in \{0,1\}$ is a Bernoulli distributed random variable with the parameter p for all t and $i \in \{1,2,\ldots,n\}$. If $\gamma_i(t) = 0$, the next state of a PBN is given by $s(t+1) = f(t)(s(t))$; otherwise, it is determined as $s(t+1) = s(t) \oplus \gamma(t)$, where $\oplus$ is the 'exclusive or' operator for vectors. Perturbations allow to reach an arbitrary state from any other state within one transition in a PBN. Thus the dynamics of a PBN with perturbations can be viewed as an ergodic DTMC over $S = \{0,1\}^n$ [10]. With the ergodic theory of DTMCs [7], it can be concluded that the long run dynamics of a PBN is independent of the choice of its initial state. This allows the estimation of the steady-state behaviour of a PBN by performing simulation from an arbitrary initial state.

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

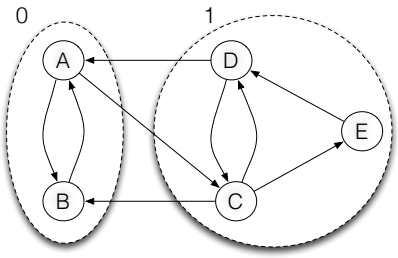
3. STEADY-STATE ANALYSIS OF PBNS

As shown in [6], both the two-state Markov chain approach and the Skart method are effective for analysing steady-state probabilities of a PBN with number of nodes up to a few thousands. We briefly discuss in this section these two methods. An efficient implementation of the two methods for the analysis of large PBNs is available in the ASSA-PBN tool [5].

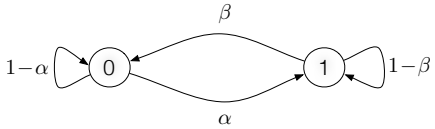
3.1 The two-state Markov chain approach

The two-state Markov chain approach [8] is a method for approximate computation of the steady-state probability for a subset of states of a DTMC. This approach splits the states of an arbitrary DTMC into two parts, referred to as two meta states. One part is composed of the states of interest, numbered 1, and the other part is its complement, numbered 0. Such consideration abstracts an arbitrary DTMC into a 0-1 stochastic process, which can further be approximated by a first-order, two-state DTMC that consists of the two meta states with transition probabilities α and β between them. Fig. 1 illustrates the construction of a two-state Markov chain from a 5-state Markov chain.

The steady-state probability of meta state 1 can be estimated by performing simulation of the original DTMC. This estimation is achieved iteratively using the standard two-state Markov chain approach of [8] to guarantee that the samples used for estimation are drawn from a distribution which differs from the the steady-state distribution at most by ε and that two precision requirements (precision r, and confidence level s) are satisfied. We outline the steps in Algorithm 1. The two arguments m_0 and n_0 are the initial 'burn-in' period and the initial sample size, respectively. In each iteration



(a) Original DTMC



(b) Two-state DTMC

Figure 1: Conceptual illustration of the idea of the two-state Markov chain construction. (a) 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. (b) Projecting the behaviour of the original chain on the two meta states results in a binary (0-1) stochastic process which can be approximated as a first-order, two-state Markov chain.

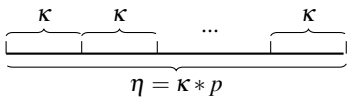
Algorithm 1 The Two-state Markov chain approach

1: **procedure** ESTIMATEPROBABILITY($m_0, n_0, \varepsilon, r, s$)
2: $M := m_0$; $N := n_0$; $l := M+N$;
3: Generate an initial trajectory of length l abstracted to the two meta states.
4: **repeat**
5: Extend the trajectory by $M+N-l$.
6: $l := M+N$;
7: Estimate α, β based on the last N elements of the extended trajectory
8: $M := \left\lceil \log\left(\frac{\varepsilon(\alpha+\beta)}{\max(\alpha,\beta)}\right) / \log\left(|1-\alpha-\beta|\right) \right\rceil$
$N := \left\lceil \frac{\alpha\beta(2-\alpha-\beta)}{(\alpha+\beta)^3} \frac{\left(\Phi^{-1}(\frac{1}{2}(1+s))\right)^2}{r^2} \right\rceil$
9: **until** $M+N \leq l$
10: Estimate the probability of meta state 1 from the last N elements of the trajectory.
11: **end procedure**

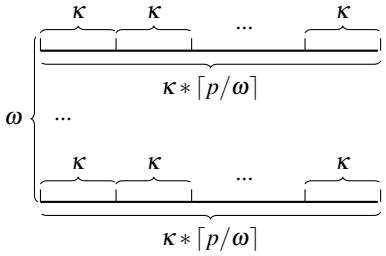
of the algorithm, the 'burn-in' steps M and the actual sample size N are re-estimated. The iteration continues until the estimated sample size $(M+N)$ is not bigger than the current trajectory length. For more details on this approach, a derivation of the formulas for M and N in line 8, and a discussion regarding a proper choice of n_0, we refer to [6].

3.2 The Skart method

Proposed by Tafazzoli et al. [15] in 2008, 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. It divides the simulated trajectory of length η, i.e., $\{X_i : i = 1,2,\ldots,\eta\}$, into p non-overlapping batches, each of size κ. See Figure 2a for an illustration. Assuming p and κ are both



(a) A single chain is divided into p batches



(b) ω chains are divided into p batches

Figure 2: Demonstration of dividing samples into batches for the Skart method. (a) Dividing samples from a single chain into p batches, each of size κ. The chain size $\eta = \kappa * p$. (b) Dividing samples from ω chains into p batches, each of size κ. The size of each chain is $\kappa * \lceil p/\omega \rceil$. The actual number of batches after dividing is $p' = \omega * \lceil p/\omega \rceil$.

Algorithm 2 The Skart method

1: **procedure** ESTIMATECI(H^*, α)
2: $\eta := 1280$; $l := 1024$; *pass* := FALSE;
3: Generate an initial trajectory of length η.
4: Compute the skewness $\hat{B}$ of the last l samples and set batch size κ based on $\hat{B}$.
5: $p := \eta$; $\eta := \kappa * p$;
6: Extend trajectory to η and compute randomness test statistics C
7: **while** Independence test is not passed **do**
8: Adjust batch size κ, number of batches p and spacer [1] size d; $\eta := \kappa * p$
9: Extend trajectory to η and compute randomness test statistics C
10: **end while**
11: $\zeta := d * \kappa$; skip first ζ samples.
12: **repeat**
13: Extend the trajectory to length η if necessary.
14: Compute nonspaced batch means μ and variance estimator Var.
15: Compute skewness and autogression adjusted CI based on Var and α.
16: $H = max\{\mu - CI_{bottom}, CI_{top} - \mu\}$;
17: **if** $H > H^*$ **then**
//check whether the precision requirement is satisfied
18: Adjust batch size κ and number of batches p';
19: $\eta := \kappa * (p' + d)$
//p' does not contain the number of discarded batches
20: **else** *pass* := TRUE;
21: **end if**
22: **until** *pass*
23: **end procedure**

large enough, it guarantees that the batch means are approximately independent and identically distributed (i.i.d) normal random variables. The grand mean of the individual batch means, denoted as

μ, is considered as a point estimator of μ_X, i.e., the steady-state expected value of the process X_i. In practise, some initial batches, known as 'burn-in' steps and denoted as ζ, are discarded to eliminate the initialisation bias when computing the point estimator μ_X. The method then constructs a CI (confidence interval) estimator for μ_X that is centered on μ. The key process of the Skart method is to determine a proper batch size κ, a proper batch number p, and proper number of 'burn-in' steps ζ, so that the computed steady-state estimations are approximately the theoretical ones and the computed CI estimator satisfies certain precision requirements. This is achieved by using randomness test, autocorrelation, and skewness adjustment in an iterative way. We summarise the process of the Skart method in Algorithm 2, and we refer to [15] for a more detailed description. Given two input parameters H^* (precision requirement) and α (confidence level), this algorithm computes a CI estimator $[CI_{bottom}, CI_{top}]$ and a point estimator μ which together satisfy that $H^* < max\{\mu - CI_{bottom}, CI_{top} - \mu\}$ and the real steady-state probability of the system is within the confidence interval with $100(1-\alpha)\%$ probability.

4. PARALLEL STEADY-STATE ANALYSIS OF LARGE PBNS

It often appears that a huge sample size is required to estimate the steady-state probabilities for large PBNs, which can be computationally expensive. In principle, parallelising the sample generation process can be considered an ideal solution to this problem. We propose to combine the Gelman & Rubin method with the two above mentioned methods. The Gelman & Rubin method is used to monitor that all the simulated chains have approximately converged to the steady-state distribution while the other two methods are used to determine the sample size required for computing the steady-state probabilities of the states of interest.

4.1 The Gelman & Rubin method

The Gelman & Rubin method [2] is an approach for monitoring the convergence of multiple chains. It starts from simulating 2ψ steps of $\omega \geq 2$ independent Markov chains in parallel. The first ψ steps of each chain, known as the 'burn-in' period, are discarded from it. The last ψ elements of each chain are used to compute the within-chain (W) and between-chain (B) variance, which are used to estimate the variance of the steady state distribution ($\hat{\sigma}^2$). Next, the potential scale reduction factor $\hat{R}$ is computed with $\hat{\sigma}^2$. $\hat{R}$ indicates the convergence to the steady state distribution. The chains are considered as converged and the algorithm stops if $\hat{R}$ is close to 1; otherwise, ψ is doubled, the trajectories are extended, and $\hat{R}$ is recomputed. We list the steps of this approach in Algorithm 3. For further details of this method and the discussion on the choice of the initial states for the ω chains we refer to [2].

Algorithm 3 The Gelman & Rubin method

procedure GENERATECONVERGEDCHAINS(ω, ψ_0)
  $\psi := \psi_0$;
  Generate in parallel ω trajectories of length 2ψ;
  repeat
    chains($1\ldots\omega$,$1\ldots 2\psi$) := Extend all the ω trajectories to length 2ψ;
    for $i = 1..\omega$ **do**
      $\mu_i :=$ mean of the last ψ values of chain i;
      $s_i :=$ standard deviation of the last ψ values of chain i;
    end for
    $\mu := \frac{1}{\omega}\sum_{i=1}^{\omega}\mu_i$;
    $B := \frac{\psi}{\omega-1}\sum_{i=1}^{\omega}(\mu_i-\mu)^2$; $W := \frac{1}{\omega}\sum_{i=1}^{\omega}s_i^2$;
      //Between and within variance
    $\hat{\sigma}^2 := (1-\frac{1}{\psi})W + \frac{1}{\psi}B$;
      //Estimate the variance of the stationary distribution
    $\hat{R} := \sqrt{\hat{\sigma}^2/W}$;
      //Compute the potential scale reduction factor
    $\psi := 2\cdot\psi$;
  until $\hat{R}$ is close to 1
  return (chains,$\psi/2$);
end procedure

4.2 Parallelising the two-state Markov chain approach

To reduce the time cost of the two-state Markov chain approach in the case of large PBNs, we propose to parallel this approach by providing samples from multiple chains. To achieve this, the Gelman & Rubin method is used to run multiple chains of the original DTMC to assure their convergence to the steady-state distribution. Once convergence is reached, the second halves of the chains are merged into one sample, and the two-state Markov chain approach is applied to estimate N based on the merged sample. Since the convergence is assured, we propose to skip the iterative computation of the 'burn-in' period in the two-state Markov chain approach to maximise the speed-up. The stop criterium for the two-state Markov chain approach becomes that the estimated value N is not bigger than the size of the merged sample. If the stop criterium is not satisfied, the multiple chains are extended in parallel to provide a sample of required length. The above idea would be fully correct, if the fact that the simulated chains of the original Markov chain have converged, would imply that the two-state Markov chain abstraction is also converged to its steady-state distribution. Although our computational experiments indicated that this is often the case (data not shown), it does not hold in general. Therefore, we add an additional step. Once the stop criterium is satisfied, the 'burn-in' period M of the two-state Markov chain is computed. The assumption is verified true if M is not larger than the 'burn-in' period ψ of the Gelman & Rubin method. Otherwise, additional $M-\psi$ elements will be discarded in the beginning of each chain and the tweaked two-state Markov chain part is re-run on the modified sample. The detailed steps of this approach are outlined in Algorithm 4.

When analysing a biological system, we are often interested in more than one set of states, e.g., in the case of a long-run sensitivity analysis of a PBN modelling a biological system. For simplicity, we call the steady-state probability of the states of interest as one *property* and computing this steady-state probability as *checking one property*. Given q different properties, the two-state Markov chain approach needs to be run for q times in order to check all of them. Since the generation process is the most time consuming part in the algorithm, the time cost for checking multiple properties can be reduced significantly if we can reuse the generated samples. We modify Algorithm 4 to allow the reuse of samples for computing the steady-state probabilities of multiple properties. The crucial idea is that the simulated samples are abstracted into different meta states based on these different q properties simultaneously each time an extension of chains is obtained. The calcu-

Algorithm 4 The Parallelised two-state Markov chain approach

1: **procedure** ESTIMATEINPARALLEL($\omega, \psi_0, \varepsilon, r, s$)
2: $(chains, \psi) := generateConvergedChains(\omega, \psi_0)$;
3: $n := 0$; $extend\_by := \psi$; $monitor :=$ FALSE; $ab\_sample :=$ NULL;
4: **repeat**
5: **repeat**
6: $chains :=$ Extend in parallel each chain in chains by $extend\_by$;
7: $sample := chains(1 \dots \omega, (n+\psi+1) \dots (n+\psi+extended\_by))$;
8: $ab\_sample :=$ abstract $sample$ and combine with ab_sample ;
9: $n := n + extend\_by$; $sample\_size := \omega \cdot n$;
10: Estimate α, β from ab_sample;
11: Compute N as Line 8 Algorithm 1;
12: $extend\_by := \lceil (sample\_size - N)/\omega \rceil$;
13: **until** $extend\_by < 0$
14: Compute M as Line 8 Algorithm 1; $monitor :=$ FALSE;
15: **if** $M \geq \psi$ **then**
16: $extend\_by := \psi - M$; $\psi := M$; $monitor :=$ TRUE;
17: **end if**
18: **until** $monitor$
19: Estimate the prob. of meta state 1 from ab_sample;
20: **end procedure**

lations of N for different properties are then performed simultaneously as well, resulting in another level of parallelisation. The next extension length is determined by the minimal value of all the calculated Ns. Using the minimal value could increase the number of extensions; however it can avoid unnecessary abstraction of samples, which is a relatively expensive process. The extension process is stopped when all the calculated Ns are smaller than the current sample size. The steady-state probabilities of all the properties are then calculated based on their corresponding abstracted meta states.

4.3 Parallelising the Skart method

The trajectory required by the Skart method can be very long as well in the case of large networks. We propose to apply a similar strategy as what we have done for the two-state Markov chain approach to reduce the time cost of the Skart method. It is assumed in the Skart method that the number of batches p and the batch size κ are large enough to guarantee that the batch means are approximately i.i.d normal random variables. This assumption still holds when the batches are obtained from different chains given that convergence has been reached in those chains. Therefore, the Skart method can be parallelised by fetching samples from multiple chains which have converged. We use the Gelman & Rubin method to guarantee that different chains have converged and the Skart method to determine the number of batches and the size of each batch in order to estimate the target stationary probability with a given precision. Since the burn-in steps are already discarded by the Gelman & Rubin method, no samples will be truncated when computing the CI in the parallelised version of the Skart method. For efficiency consideration, we further require that the number of batches obtained from each chain is the same. This makes the number of batches used in the parallelised Skart method slightly larger than that in the original Skart method. Figure 2 shows how a single chain and multiple chains are divided into batches. We summarise the parallelisation of the Skart method in Algorithm 5 and highlight the lines where there exists a main difference with respect to the sequential method by adding comments.

Algorithm 5 The Parallelised Skart method

1: **procedure** ESTIMATECIINPARALLEL(H^*, α)
2: $(chains, \psi) := generateConvergedChains(\omega, \psi_0)$;
3: Skip first ψ samples of each chain; $\eta := \lceil 1,280/\omega \rceil * \omega$; $pass :=$ FALSE;
4: Extend all the chains each to length η/ω if necessary
5: Compute the skewness $\hat{B}$ and set batch size κ based on $\hat{B}$
6: $p := \lceil \eta/\omega \rceil * \omega$; $\eta := \kappa * p$; *// batch number is adjusted*
7: Extend all the chains each to η/ω and compute randomness test statistics C
8: **while** Independence test is not passed **do**
9: Adjust κ, number of batches p and spacer size d;
10: $p := \lceil p/\omega \rceil * \omega$; $\eta := \kappa * p$*// batch number is adjusted*
11: Extend all the chains each to η/ω and compute randomness test statistics C
12: **end while**
13: **repeat**
14: Extend all the chains each to length η/ω if necessary
15: Compute nonspaced grand batch mean μ and variance estimator Var
16: Compute skewness and autogression adjusted CI based on Var and α
17: $H = max\{\mu - CI_{bottom}, CI_{top} - \mu\}$
18: **if** $H > H^*$ **then**
19: Adjust batch size κ and number of batches p;
20: $p := \lceil p/\omega \rceil * \omega$; $\eta := \kappa * p$
// do not consider discarding the first d samples
21: **else** $pass :=$ TRUE
22: **end if**
23: **until** $pass$
24: **end procedure**

5. EVALUATION

The above mentioned algorithms have been implemented in the tool ASSA-PBN [5] and the performance of Algorithm 1 and Algorithm 2 have been evaluated in [6]. We show in this section that the proposed two parallel algorithms can significantly reduce the time cost for computing steady-state probabilities of large PBNs in comparison with their sequential versions. We evaluate this first on randomly generated PBNs and then on a PBN for a real biological system. To make the evaluation as fair as possible, the proposed two parallel algorithms are implemented in the same programming language, i.e., Java, as the tool ASSA-PBN uses. All the experiments in this paper are conducted in a high-performance computing (HPC) node, which contains 16 Intel Xeon E7-4850 processors@2GHz. The 16 processors are equally distributed in 4 Bull S6030 boards (servers) and each processor contains 10 cores. This hardware architecture allows us to run a program with the maximum of 40 cores in one board. The initial and the maximum Java Heap Size are set to 2GB and 64GB, respectively.

5.1 Speed-up for checking a single property

We first evaluate the speed-up for checking a single property using the parallelised algorithms, i.e., Algorithm 4 and Algorithm 5. We randomly generate 18 different PBNs with node numbers from the

set $\{80, 100, 200, 500, 1000, 2000\}$ using the tool ASSA-PBN. For each node number, 3 PBNs are generated. We assign the obtained PBNs into three different classes with respect to their density measure $\mathscr{D}$: dense models with density $150-300$, sparse models with density around 10, and in-between models with density $50-100$. The precision and confidence level of all the experiments are set to 10^{-4} and 0.95, respectively. The parameter ε in the two-state Markov chain approach is set to 10^{-10}. We compute steady-state probabilities for the 18 PBNs using Algorithms 1, 2, 4, and 5 and compare their results as well as time costs. The parallelised algorithms are launched with 6 different number of cores in one board ranging in $\{2, 5, 10, 20, 30, 40\}$.

As the models we use are too large to be analysed with numerical methods, it is not possible to check the correctness of the parallelised algorithms using the models' theoretical steady-state probability distributions. Instead we compare the results of the parallelised algorithms with their corresponding sequential algorithms. We collect the two probabilities computed by Algorithms 1 and 4 or by Algorithms 2 and 5 for checking the same property of one model as a pair. In this section, there are 216 pairs of probabilities in total. We expect the difference of the two probabilities in a pair to be less than 2×10^{-4} with the probability of 95%. In all the 216 pairs that we have obtained, the difference is always less than 2×10^{-4}. Moreover, we perform a similar verification for the evaluation results in Section 5.2 and obtain the same observations.

Figure 3a and Figure 3b present the speed-ups when analysing a single property of the 18 PBNs with the parallelised two-state Markov chain approach. Each speed-up is computed using the formula $t_{sequential}/t_{parallel}$, where $t_{sequential}$ is the time cost of the sequential two-state Markov chain approach and $t_{parallel}$ is the time cost of its parallelised version. The parallelised approach is launched with different number of cores ranging in $\{2, 5, 10, 20, 30, 40\}$. We see clearly from these figures that the speed-up is almost proportional to the number of cores. Meanwhile, the speed-ups vary a lot in different models with 40 cores, e.g., we observe a speed-up about 46 in the case of the 2000-node dense model and a speed-up of 22 in the case of the 200-node sparse model.

On the one hand, the required sample size varies in different runs due to the nature of the two-state Markov chain approach. The speed-up can be bigger than the number of cores when the required sample size in the parallelised run is smaller than what is required in the sequential run – this is actually the case where we obtain the speed-up of 46 for the 2000-node dense model. On the contrary, the speed-up can be much smaller than the number of cores when the required sample size in the parallelised run is bigger than that required in the sequential run – this is the case where we obtain the speed-up of 31.7 for the 80-node dense model. To show the affection of sample size, we compute the speed-ups for the parallelised run with 40 cores after eliminating the affection of sample size using the formula $sp * size_p/size_s$, where sp is the original speed-up computed with $t_{sequential}/t_{parallel}$, $size_s$ is the sample size used for the sequential run, and $size_p$ is the sample size used for the parallel run. The results are shown in Table 1 (in the rows labelled with speed-up E). On the other hand, the time cost also varies when checking a property for different models. When the time cost of the sequential run is small, the percentage of time spent in generating samples is also small. As a consequence, the percentage of time the parallelised algorithm can reduce is small as well. Therefore, the speed-up the parallelised algorithm can gain is small when the time cost of the sequential run is small – this is the case where we obtain the speed-up of 22 for the 200-node sparse model. On the contrary, a larger speed-up is easy to obtain when the time cost of the sequential run is big – this is the case where we obtain the speed-up of 46 for the 2000-node dense model. We obtain similar speed-ups with the Skart method and the speed-ups are presented in Figure 3c and Figure 3d.

Besides, we obtain maximum speed-ups with the use of 40 cores under the current hardware condition. To illustrate this, we show in Table 1 more detailed information, i.e., the time costs (in minutes), the actual sample sizes (of millions), the speed-ups, and the speed-ups after eliminating the affection from the sample size. Note that in order to make the results as accurate as possible, all speed-ups are computed using the original time and size values we get from experiments, not the truncated ones shown in Table 1. For the two-state Markov chain approach, the speed-ups are greater than 30 for 14 out of 18 cases and for the Skart method the speed-ups are greater than 30 for 12 out of 18 cases.

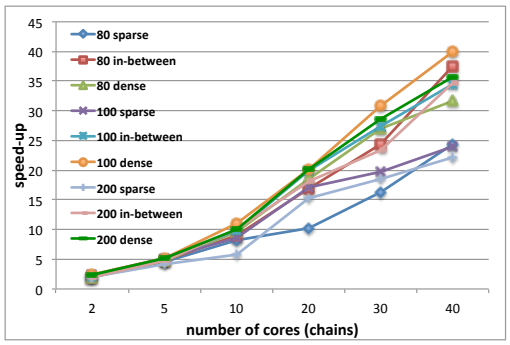
Moreover, we show in the next section that with the use of 40 cores, speed-ups between 19.67 and 33.04 are obtained for a 96-node PBN modelling a real biological system.

5.2 Speed-up for checking multiple properties

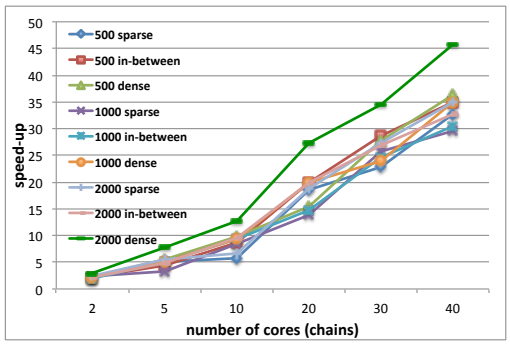
We have performed one influence analysis and two long-run sensitivity analyses of an apoptosis network using the sequential two-state Markov chain approach in [6]. The apoptosis network contains 96 nodes; one of the nodes, i.e., UV, can take on three values and was refined as UV(1) and UV(2) in order to cast the original multi-value model into the binary PBN framework. The 96 nodes with 107 Boolean functions and their parameters, i.e., the selection probabilities of Boolean functions, were fitted to experimental data in [16]. We took the 20 best fitted parameter sets and performed analyses for them. With an efficient implementation of a PBN simulator, we managed to finish this analysis in an affordable amount of time. Nevertheless, the analysis was still very expensive in terms of computation time since the trajectories required were huge and a number of properties needed to be checked.

In this work, we re-perform part of the influence analyses by running the parallelised two-state Markov chain approach for checking 7 properties simultaneously with 40 cores. In the influence analysis, we aim to compute the *long-term influences* on complex2 from each of its parent nodes: RIP-deubi, complex1, and FADD, in accordance with the definition in [11]. We consider both the case of UV(1) and UV(2) and hence we construct 2 PBNs for each of the 20 best fit parameter sets. In total, we need to compute 7 different steady-state probabilities for 40 different PBNs.

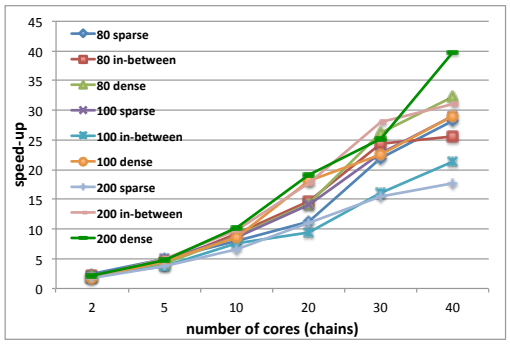
In [6], the two-state Markov chain approach has been applied 280 times to finish the computation. Using the parallelised version, we only need to perform the parallelised two-state Markov chain approach 40 times since 7 properties for one PBN can be checked in one run. In this evaluation, we perform the parallelised two-state Markov chain approach to check the 7 properties of one of the 40 PBNs simultaneously and show in Table 2 the time cost (in minutes), the actual sample size (in millions) we use and the speed-ups we obtain for checking them with the sequential and parallelised algorithms. To make the comparison complete, we also perform the parallelised two-state Markov chain approach to check the 7 properties one by one and show the results in Table 2. The precision r, confidence level s, and steady-state convergence parameter ε in this experiment are set to 10^{-5}, 0.95 and 10^{-10}, respectively. The



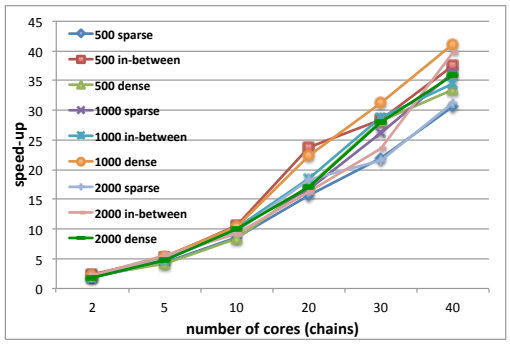
(a) Two-state Markov chain (I): PBNs with 80, 100, 200 nodes.



(b) Two-state Markov chain (II): PBNs with 500, 1000, 2000 nodes.



(c) The Skart method (I): PBNs with 80, 100, 200 nodes.



(d) The Skart method (II): PBNs with 500, 1000, 2000 nodes.

Figure 3: Speed-up of Algorithms 4 and 5. Each curve shows the speed-up as a function of the number of cores used.

node #			80			100			200		
density			sparse	in-between	dense	sparse	in-between	dense	sparse	in-between	dense
two-state	time (m)	seq.	27.4	50.7	30.6	30.4	52.5	119.5	18.0	159.3	171.0
		par.	1.1	1.4	1.0	1.3	1.5	3.0	0.8	4.6	4.8
	size (million)	seq.	84.7	105.3	41.6	70.2	98.8	117.7	25.9	134.6	83.7
		par.	85.2	105.0	49.3	70.3	98.8	117.5	25.9	134.7	83.7
	speed-up		24.3	37.5	31.7	23.9	34.5	40.0	22.1	34.7	35.8
	speed-up E		24.5	37.4	37.7	23.9	34.5	39.9	22.2	34.7	35.8
Skart	time (m)	seq.	29.9	49.2	31.7	30.5	47.6	110.0	16.7	167.3	176.9
		par.	1.1	1.9	1.0	1.1	2.2	3.8	0.9	5.4	4.4
	size (million)	seq.	94.1	110.2	43.5	72.7	89.6	110.1	24.5	141.2	86.9
		par.	78.3	109.7	40.8	69.6	109.7	123.5	29.1	131.1	73.8
	speed-up		28.2	25.5	32.3	28.9	21.3	28.9	17.8	31.0	39.8
	speed-up E		23.4	25.4	30.3	27.7	26.1	32.5	21.1	28.8	33.8

node #			500			1000			2000		
density			sparse	in-between	dense	sparse	in-between	dense	sparse	in-between	dense
two-state	time (m)	seq.	302.5	556.9	410.8	211.3	620.4	1841.7	111.0	460.5	643.1
		par.	9.3	15.9	11.4	7.2	20.3	52.6	3.2	14.0	14.0
	size (million)	seq.	156.1	198.3	113.9	75.8	176.7	297.6	44.8	114.1	115.5
		par.	153.8	204.8	114.0	75.6	176.4	302.1	40.8	113.5	91.8
	speed-up		32.7	34.9	36.2	29.5	30.6	35.0	34.9	32.8	45.8
	speed-up E		32.2	36.0	36.2	29.5	30.5	35.6	31.8	32.6	36.4
Skart	time (m)	seq.	278.5	594.0	394.2	218.7	671.3	2095.5	98.9	467.5	466.9
		par.	9.1	15.9	11.7	6.0	19.5	51.0	3.2	11.7	13.0
	size (million)	seq.	144.1	216.5	114.9	78.7	185.8	292.9	40.1	109.2	94.1
		par.	134.9	208.6	117.4	75.3	184.4	274.0	38.4	103.3	86.4
	speed-up		30.6	37.4	33.6	36.3	34.5	41.1	31.3	39.8	35.8
	speed-up E		28.7	36.0	34.3	34.7	34.3	38.5	29.9	37.7	32.9

Table 1: Speed-up obtained with 40 cores for Algorithms 4 and 5. Seq. and par. are short for sequential and parallel.

sequential			parallelised			speed-up
property #	sample size (million)	time (m)	property #	sample size (million)	time(m)	
1	146.99	53.30	1	147.20	2.31	23.04
2	454.14	174.25	2	461.58	6.81	25.58
3	253.45	97.77	3	253.60	3.86	25.32
4	48.81	16.71	4	50.11	0.73	22.83
5	305.35	120.33	5	335.85	5.38	22.39
6	50.21	17.65	6	51.31	0.90	19.67
7	255.17	99.75	7	263.39	3.02	33.04
total	1563.05	579.75	1-7	452.74	10.96	52.88

Table 2: Performance comparison on checking seven properties. Property # "1-7" means checking seven properties simultaneously.

speed-ups for checking a single property are computed similarly as in Figure 3; while the speed-up for checking the seven properties simultaneously is computed with $\sum_{i=1}^{7} t_i/t_{multi}$, where t_i is the time cost for checking the i-th property with the sequential algorithm and t_{multi} is the time cost for checking the seven properties simultaneously with the parallelised algorithm. From Table 2, the parallelised algorithm obtains a speed-up between 19.67 and 33.04 for checking a single property and a speed-up of 52.88 for checking seven properties. By reuse of generated samples, the sample size is also reduced by 3.45 times from 1563.05 to 452.74 million.

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed to combine the Gelman & Rubin method with two statistical methods, i.e., the two-state Markov chain approach and the Skart method, to reduce the time cost for computing steady-state probabilities of large PBNs. We showed with experiments that the proposed combinations could reduce the time cost of the original sequential methods significantly.

Our parallelised algorithms work well on multiple-core CPU architecture. However, the scalability of multiple-core CPU based parallelisation is often restricted by the CPU architecture since the number of processing units in a CPU is usually small. On the contrary, GPUs often contain thousands of processing units and GPUs achieve high performance when thousands of threads execute concurrently [4]. Parallelising those algorithms with GPU based architecture will potentially lead to further speed-ups.

7. REFERENCES

[1] FOX, B. L., GOLDSMAN, D., AND SWAIN, J. J. Spaced batch means. *Operations Research Letters 10*, 5 (1991), 255–263.

[2] GELMAN, A., AND RUBIN, D. Inference from iterative simulation using multiple sequences. *Statistical Science 7*, 4 (1992), 457–472.

[3] KAUFFMAN, S. A. *The Origins of Order: Self-organization and Selection in Evolution*. Oxford University Press, 1993.

[4] MICIKEVICIUS, P. 3D finite difference computation on GPUs using CUDA. In *Proc. 2nd Workshop on General Purpose Processing on Graphics Processing Units* (2009), ACM, pp. 79–84.

[5] MIZERA, A., PANG, J., AND 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* (2015), LNCS, Springer. Available at http://satoss.uni.lu/software/ASSA-PBN/.

[6] MIZERA, A., PANG, J., AND YUAN, Q. Reviving the two-state markov chain approach (technical report). Available online at http://arxiv.org/abs/1501.01779, 2015.

[7] NORRIS, J. *Markov Chains*. Cambridge University Press, 1998.

[8] RAFTERY, A., AND LEWIS, S. How many iterations in the Gibbs sampler? *Bayesian Statistics 4* (1992), 763–773.

[9] SHMULEVICH, I., DOUGHERTY, E., AND ZHANG, W. From Boolean to probabilistic Boolean networks as models of genetic regulatory networks. *Proceedings of the IEEE 90*, 11 (2002), 1778–1792.

[10] SHMULEVICH, I., AND DOUGHERTY, E. R. *Probabilistic Boolean Networks: The Modeling and Control of Gene Regulatory Networks*. SIAM Press, 2010.

[11] SHMULEVICH, I., DOUGHERTY, E. R., KIM, S., AND ZHANG, W. Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics 18*, 2 (2002), 261–274.

[12] SHMULEVICH, I., DOUGHERTY, E. R., AND ZHANG, W. Control of stationary behavior in probabilistic Boolean networks by means of structural intervention. *Journal of Biological Systems 10*, 04 (2002), 431–445.

[13] SHMULEVICH, I., DOUGHERTY, E. R., AND ZHANG, W. Gene perturbation and intervention in probabilistic Boolean networks. *Bioinformatics 18*, 10 (2002), 1319–1331.

[14] SHMULEVICH, I., GLUHOVSKY, I., HASHIMOTO, R., DOUGHERTY, E., AND ZHANG, W. Steady-state analysis of genetic regulatory networks modelled by probabilistic Boolean networks. *Comparative and Functional Genomics 4*, 6 (2003), 601–608.

[15] TAFAZZOLI, A., WILSON, J., LADA, E., AND STEIGER, N. Skart: A skewness- and autoregression-adjusted batch-means procedure for simulation analysis. In *Proc. 2008 Winter Simulation Conference* (2008), pp. 387–395.

[16] TRAIRATPHISAN, P., MIZERA, A., PANG, J., TANTAR, A.-A., AND SAUTER, T. optPBN: An optimisation toolbox for probabilistic Boolean networks. *PLOS ONE 9*, 7 (2014), e98001.

[17] TRAIRATPHISAN, P., MIZERA, A., PANG, J., TANTAR, A.-A., SCHNEIDER, J., AND SAUTER, T. Recent development and biomedical applications of probabilistic Boolean networks. *Cell Communication and Signaling 11* (2013), 46.