Multi-threaded Approach for Generation of Random Boolean Networks

Nikolay Shegunov[1], Armyanov Peter[1] and Ivanov Ivan[2]

[1] Faculty of Mathematics and Informatics, University of Sofia, Sofia, Bulgaria

[2] Veterinary Physiology & Pharmacology, Texas A&M University, College Station, USA

parmyanov@fmi.uni-sofia.bg

Abstract. The concept of Boolean networks (BNs), as a way to model a gene regulatory network, is often used to represent the dynamical properties of gene-gene interactions. Although this dynamical system is a simplified model of gene-gene interactions, it is particularly useful when the participating in the model genes exhibit a switch-like behavior. The successful application of this model to real data depends on our understanding of the model's properties. A typical study of BNs properties/features requires a reliable and fast generation of BNs that satisfy a specific set of constraints. In this paper, we describe an efficient multi-threaded approach for generating of random BNs on N genes. Because the state space of the model grows exponentially with N, the generation of a large number of BNs that satisfy certain constraints requires a huge computational effort. With the recent advances of computer science, the problem of efficiently generating a large sample of such networks and study the statistics of their features becomes even more relevant. Our novel multi-threaded approach for simulation and generation of BNs allows estimating the model's mean attractor length, the average Boolean function bias and network connectivity of large samples of randomly generated networks in a parallel manner.

Keywords: random boolean networks, parallel generation, multi-threading.

1 Introduction

Stuart Kaufman pioneered the idea to use Boolean networks (BNs) as a model of a gene regulatory network in 1963. Boolean networks are an example of a discrete dynamical system. The model consists of a set of genes/nodes, that take one of the possible discrete values 0 or 1, and connections between them. Each gene value is updated synchronously according to its assigned Boolean function and the current state of the model. Each such Boolean function is consistent with the number of edges entering the respective gene in the gene connectivity

directed graph. The Boolean model is also often referred to as the $N-K$ model where N is the number of genes and K represents the connectivity of the network [2]. Studying the behavior of the corresponding dynamical system can uncover fundamental principles of regulation of living systems or assist in the development of new gene-targeted therapies.

In this paper, we focus on random Boolean networks where the model instances are randomly selected from the set of all possible networks of a given $N-K$ class. The ability to generate large samples of random BNs allows for proper statistical studies of the model's general features, e.g. average Boolean function bias and network connectivity or attractor length [1]. Of a particular interest are those networks that have a relatively low connectivity - at the range of square root of the number of the participating genes, where a transition between a chaotic and ordered regime states occurs [2]. The generation and statistical description of these complex random systems is quite challenging. It requires a huge computational effort. For example, consider a network with only 64 genes. BNs from this general class have 2^{64} states and $2^{2^{64}}$ possible Boolean functions, which emphasizes the importance of a careful computer memory and CPU management when studying the general structural and dynamical properties of the class. Our approach to address these issues is based on a shared memory model. It speed up of the simulation of random BNs and can be easily extended to a distributed memory.

The paper is organized as follows:

- The general mathematical setting of the $N-K$ model including the definition of the state transition diagram is presented and discussed in section 2;
- Section 3 focuses on the generation of random BNs and the properties of that task that allow for parallel implementation;
- Finally, section 4 presents result from our limited simulations.

2 Mathematical setting

A Boolean network is a pair *(V, F)* where V can be mathematically represented by a directed graph. In that graph, each node represents a gene. Each gene has only two possible states: either enabled (1) or disabled (0). Each gene can be influenced by other genes and the second component F of the model accounts for this. F is a vector of Boolean functions where each one of those functions is assigned to a unique node, i.e. F has as many components as the number of genes in V. Each respective Boolean function in F takes the values of the genes that have directed edges in V pointing to the gene function assigned to and returns the state of the assigned gene given the current states of those inputs. Each gene in V is updated synchronously according to the functions in F. As discrete objects, the Boolean

functions are completely described by their truth tables. Thus, randomly selecting a Boolean truth table is equivalent to randomly selecting a Boolean function. Since the truth table for the Boolean function that guides the state of a given gene is randomly selected it may turn out that some of the input parameters (genes) are not essential variables for that Boolean function, i.e. they do not influence the function's output. Hence, the connectivity K parameter of the model is interpreted as the maximum number of influencing genes or incoming connections to any node in V. Viewing the Boolean networks as discrete dynamical systems, one can represent them as state transition diagrams. Each network has 2^N possible states, where N is the number of genes. A state of the network is a vector of length N where each position i of this vector represents the state of gene i. Each state of the system transitions deterministically to its successor according to the Boolean functions in F. The totality of such transitions is a directed graph - the state transition diagram of the model. Being a directed graph, the state transition diagram can be easily encoded as a matrix - the state transition matrix where each row has all but one of its entries equal to 0. The only non-zero entry equals 1, which indicates the relation between the respective states in the dynamic of the model.

Using this general setting of random Boolean networks, we consider a set of tree fundamental model statistics in our simulations: (i) average function bias, (ii) average connectivity, and (iii) average attractor length. These statistics can be computed using the network's state transition diagram; however, randomly generating the state transition diagram is quite challenging because of the constraints on the structure of the $N{-}K$ model. Therefore, our simulation study first focuses on $N{-}K$ networks generation, and then computes the corresponding transition matrix. The approach is as follows:

1. Select a non-negative integer value for the parameter N.
2. Uniformly randomly pick a non-negative integer number for the connectivity K in the range of (0, N].
3. For the selected $N{-}K$ class of networks randomly generate Boolean functions, according to the connectivity parameter K.
4. Finally, represent each generated network as a transition matrix and compute the statistics (i), (ii), and (iii).

3 Parallel implementation

Our implementation consists of several steps. The first one is the generation of a random $N{-}K$ BN. The next step focuses on the calculation of the state-transition matrix. Then the three statistics of interest are calculated - the average network connectivity, average function bias and attractor length. These steps are then repeated to generate a large random sample from the respective $N{-}K$ class and

results for the three statistics are aggregated. During this repeated sampling, it may happen, that a generated sample have the same statistical properties as previously generated sample. Having two BNs with the same values for our three properties of interest does not necessarily imply that those BNs are identical. It could mean also that they are isomorphic, i.e. one of them can be transformed into the other by a proper relabeling of the genes and rearranging the Boolean functions. While the problem of identifying isomorphic BNs in the randomly generated sample is important on its own, we decided to allow the generation of BNs with identical statistics (i), (ii), and (iii) for our current simulation because our main objective is to study the parallel implementation of the generation algorithm.

Each one of the steps, except finding the average attractor length, can be easily implemented in a parallel manner. The most time-consuming part is the generation of random Boolean functions that satisfy the constraints of a given $N{-}K$ class. Because of the synchronous update scheme of the network state the Boolean functions are considered independent from each other and their generation can be split into an arbitrary amount of parallel jobs. Similarly, calculation of the specific network connectivity and function bias can be split into independent jobs. The algorithm for calculating the attractor length is not trivial for parallelization; however, our simulations show that it runs much faster than all other steps and its parallelization does not appear to lead to a notable speedup. Taking all of these points into consideration, we implemented a code, that executes each step on several threads and the network samples are processed sequentially to compute: (i) average function bias, (ii) average connectivity, and (iii) average attractor length. Our simulations show that for networks, with N greater than 25, the parallel implementation speedup is nearly proportional to the number of threads used. Of course, the maximum number of threads have to be less than the number of available CPU cores. It is important to note that the proposed approach is not suitable for relatively small networks. It is not suitable for a computing system with a distributed memory is used. We propose a different approach for such cases. In particular, each step of our algorithm is executed on the same thread, but the individual samples/BNs are processed in parallel on separate threads. This modification of the implementation leads to speedup, nearly proportional to the number of CPU cores when generating a large number of samples from a given $N{-}K$ class. It is clear that this approach requires that each thread have to build its own network model, which consumes large part of the shared system memory. Thus, we recommend out modified approach for experiments with large number of samples drawn from a $N{-}K$ classes with a relatively small N; or when calculations are executed on system with distributed memory, e.g. using an MPI framework. Currently, we have implemented only the shared-memory approach, using a thread pool where the tasks are split in one of the two possible ways:

- Proportional to their size. This is relatively easy to implement with

OpenMP or with classical threads, organized in a pool. Note, that this approach is suitable when all the jobs have similar length.

- Create small, independent jobs and place them in a queue where each thread of the pool takes a job from, processes it and then proceeds to take another one. This approach is suitable in cases of significant differences in the jobs' length.

The steps of network generation and calculating the function bias result in jobs with same size for a set of given network parameters. However, the size of the job for the step of calculating the network connectivity can vary significantly for different sizes of network samples. Our simulations seem to confirm this: using 24 threads to generate 10000 samples, the difference between the slowest and the fastest thread is approximately 4.03 seconds - close to the average time for computing a single sample - 4.05. Interestingly, the thread with least number of samples processed 413 samples while the thread with largest number of samples processes 422 samples. Note, that the average number of samples, if distributed equally, is 416.

4 Results

Fig. 1 presents the results from a simulation of 90000 random samples for Boolean networks with size N=20. Based on the randomness of the generation and the large size of the sample, one would expect approximately normal distribution of the parameters of the networks in the sample, a feature that is clearly manifested. Networks with average bias close to 0.5 have approximately the same number of zeros and ones in their Boolean functions truth tables and exhibit larger connectivity. In addition, the binary logic dictates that the results should be symmetrical with respect to the 0.5 bias. Similarly, the average attractor length is also strongly coupled with the bias. Function bias close to 0.5 means larger probability of a connection between genes, and potentially longer cycles in the state transition graph.

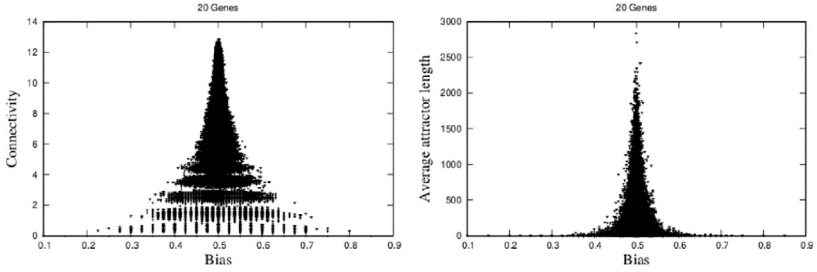

Fig. 1. 20 Gene RBN generation, 90000 samples

Fig. 2 presents our results concerning the strong scaling behavior of the problem. The proposed algorithm achieves a good scaling behavior taking into account the small sample size. The diagonal black line gives the potential theoretical speedup, while the red and the blue lines represent the actual experimental speedup. Clearly, the tread pool approach provides very good scaling of the problem. However, the random nature of the network sampling leads to a great variability in the time for computing the attractor length and connectivity. Thus, a pre-determined load on a thread is not feasible. The disadvantages, of this approach are the price for creating and spawning threads and the lack of a guarantee that the jobs will be optimally distributed.

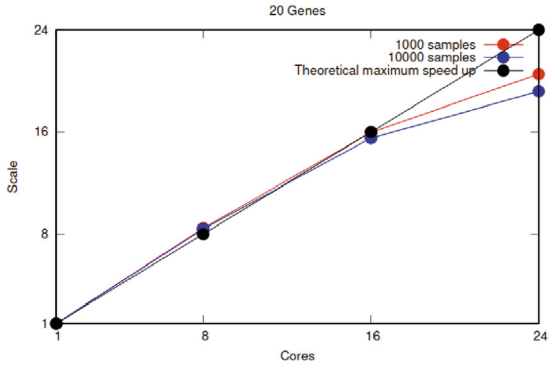

Fig. 2. Parallel performance

5 Conclusions

The proposed parallel approach for generation of random BNs, i.e. the job queue parallelism, gives promising results as our limited simulation study shows. Our experiments suggest that simply distributing the samples equally among all threads is not sufficient - it does not provide reliable allocation of the resources due to the variable job length. We believe that an approach based on a job queue parallelism should also be considered for implementations involving distributed memory.

Our simulations confirm the expectation that the generation of the random truth tables tends to be an expensive operation: the time spent for it dominates the computational time for BNs with a relatively small number of genes. One could potentially try to speed up the implementation by using the GPU architecture and moving the job of generation the networks to the GPUs.

References

1. Gershenson, C.: Introduction to Random Boolean Networks. arXiv:nlin/0408006.
2. Kauffman, S. A.: Origins of Order: Self-Organization and Selection in Evolution. Oxford Univ. Press, Oxford (1993).
3. Shmulevich I., Dougherty E.R., Kim S., Zhang W.: Probabilistic Boolean Networks: a rule-based uncertainty model for gene regulatory networks. Bioinformatics 18 (2) : pp 261-274. (2002)
4. Pal R., Ivanov I., Datta A., Bittner M.L., Dougherty E.R: Generating Boolean networks with a prescribed attractor structure. Bioinformatics 21 (21) : pp 4021-4025. (2005)