Parallel Simulation of Adaptive Random Boolean Networks

Klavdiya Bochenina[1], Kirill Kuvshinov[1], Piotr Gorsky[2], and Janusz Holyst[12]

[1] ITMO University, St. Petersburg, Russian Federation
[2] Warsaw University of Technology, Warsaw, Poland

Abstract
TBD

Keywords: TBD

1 Introduction

One of goals of complex networks analysis is a better understanding, description and prediction of the behavior of real-world systems. A starting point are usually simplified models that have few parameters and driven by random dynamics. The example of such a model is a random Boolean network (RBN) [1, 2, 3, 4].

RBNs were initially introduced to describe gene regulatory networks [5, 6]. But as RBNs are generic they were also applied to other fields, e.g. neural networks and social networks [2].

An RBN is a directed network consisting of N nodes. A node i ($i = 1, ..., N$) has three attributes: (a) a Boolean state $\sigma_i(t) \in \{0, 1\}$, (b) a sequence $\boldsymbol{N_i}$ containing all nodes $(\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_{|\boldsymbol{N_i}|}})$ having a connection to the node i and (c) a randomly chosen Boolean function $f_i : \{0, 1\}^{|N_i|} \to \{0, 1\}$. Note that we assume a network with no loops, that is $i \notin \boldsymbol{N_i}$. Arguments of the function f_i are states of the nodes from $\boldsymbol{N_i}$. Boolean functions determine nodes' states in the next time step:

$$\sigma_i(t+1) = f_i(\sigma_{i_1}(t), \sigma_{i_2}(t), \ldots, \sigma_{i_{|\boldsymbol{N_i}|}}(t)), \tag{1}$$

Each Boolean function is randomly chosen. It means that a node n_i has $2^{2^{|N_i|}}$ possible different Boolean functions and one of them is chosen at random. Taking also into consideration all network topologies, the size of the network ensemble is enormous. That is why RBNs are generic and despite their simplicity they can model many complex systems [2].

We assume all the nodes are updated according to the Eq. (1) at the same time. Let $\boldsymbol{S}(t) = \{\sigma_1, \sigma_2, \ldots, \sigma_N\}$ define a state of the whole network. Having N nodes in the network, network state space is 2^N-dimensional. It can be huge, but it is finite. The update rule (1) is deterministic, thus after a number of transient network states, the system will eventually reach an attractor (i.e. a periodic orbit of network state $\boldsymbol{S}(t)$).

As RBNs are applied to model biological structures, it is interesting to study the evolution of RBNs. Core of the evolution process is a constant searching for [1] the most optimal configuration [4, 7, 8]. An RBN modification that incorporates system evolution is *adaptive* RBN (ARBN) [9, 10, 11, 12]. In RBNs nodes can be divided into two groups of active and frozen nodes. Nodes of the former group change their state during network attractor. On the other hand, the frozen nodes keep their state. The basis of ARBN is a natural idea that overactive nodes should be quietened down and underactive (frozen) ones should be stimulated. This concept is included in an activity-dependent rewiring rule (ADRR) [13], which is as follows: if the nodes mean state $< \sigma_i >$ during the network attractor is 0 or 1, then this node is considered to be frozen and one new incoming edge is added to this node. Otherwise, this node is considered to be active and one of its incoming edges is randomly chosen and deleted.

A single simulation of ARBN evolution consists of an a priori-defined number of epochs. In each epoch the network attractor is found (evolution of nodes' states) and incoming connections of randomly chosen node are updated according to the ADRR (evolution of network structure). In the simulation results it is observed [9, 12], that after a number of transient epochs, the system reaches a dynamical equilibrium and oscillates around steady-state levels of measured values (e.g. mean connectivity).

In this paper, we perform ARBN simulations for larger networks with up to 1 000 nodes. In such structures the difficulty is in finding the attractor because transient and attractor lengths grow quickly with network size. In previous work there have been only few results with such network sizes. In original paper [9] the full model analysis is for the network of size up to 400 nodes[2]. In our previous work, where we analyze modular Boolean networks with separate ARBNs connected by a number of interlinks [12], the studied maximal network size is 80 nodes. Here, we do not include modularity into ARBN model. This research is a step towards large scale simulations of modular ARBNs [12]. To investigate large ARBNs, we design a parallel algorithm for simulation of ARBN's evolution and combine it with several algorithmic improvements.

The remaining part of the paper is organized as follows. Section 2 gives an overview of a related work on algorithmics of RBN modeling. Section 3 presents a description of algorithms and techniques which were used to speed up simulations (including an algorithm for parallel simulation of adaptive RBNs). Section 4 (experimental investigation) has two-fold purpose: (i) study the performance of the parallel algorithm and the influence of proposed modifications on simulation results, and (ii) study the dynamics of evolution of non-modular ARBNs up to 1000 nodes.

2 Related Work

3 Algorithms to Speed up Simulation of ARBNs

Modeling the evolution of ARBN larger than several dozens of nodes is hampered by a significant growth of transient and attractor lengths. This growth extremely prolongs the time of a repeated calculation of nodes' states during search for an attractor (which is the most time consuming part of a modeling). To speed up modeling for large ARBNs, we propose to combine algorithms and techniques described in this Section. We use existing sequential algorithms for attractor's

[1] Tu sie nie zgadzam, ze powinno by of. Istnieja 2 zwroty: 'search for' oraz 'in search of'. Ten drugi znaczy 'w poszukiwaniu'.

[2] TODO: Later it might be emphasized that LiuBassler results must have skipped most (significant number) of the attractors.

search with a limitation on a maximum number of iterations as a basis of the parallel algorithm, and extend ADDR to support larger number of nodes to be rewired.

I. Sequential heuristics for attractor's search. The naive implementation of an algorithm for attractor's search is to compare a current state $\mathbf{S}(t)$ to all the previous states at each iteration. This requires $O(l^2)$ time and $O(l)$ memory where l is a sum of attractor and transient lengths. To speed up searching for attractors in comparison with the naive implementation a number of sequential heuristics have been proposed. These algorithms do not perform the exhaustive comparisons of states. Here we briefly describe and discuss two sequential algorithms which have been used in previous studies on ARBN [9, 12]. We also use them as basic algorithms for a parallel algorithm introduced in this paper.

Algorithm 1. This is the algorithm that was used in the original paper introducing Adaptive RBNs [9]. It is based on a set of checkpoints $\mathbf{T} = \{T_0, T_1, T_2, ...T_n\}$. If at some time point t current state $\mathbf{S}(t)$ equals to the state at the checkpoint $\mathbf{S}(T_i)$, an attractor is considered to be found, and some nodes get rewired according to the ADRR presented above. If the program reaches the next checkpoint (except the last one), and $\mathbf{S}(t) \neq \mathbf{S}(T_i) \forall t \in [T_i, T_{i+1})$, all the information gathered between the checkpoints is discarded as it most probably corresponds to a transient period rather than the attractor, and T_{i+1} becomes the new checkpoint. The longest attractor that can be found using this algorithm equals to the maximum distance between the checkpoints. However, if the distance between the checkpoints is too large, more state updates then necessary are performed. Thus, the values of the checkpoints are crucial to the overall algorithm performance and correctness.

Algorithm 2. (The description of Knuth algorithm)

II. Updating states of nodes in parallel. Another way to reduce an execution time of modeling is to parallelize calculations of nodes' states at a particular iteration. In this study, we propose a GPGPU algorithm which speeds up sequential heuristics of type (I). As it does not influence the logic of a search, this algorithm can be used in a combination with any sequential heuristic. The main idea of the algorithm is to update states of different nodes in a network in parallel on distinct Graphic Processor Units (GPUs). More formal description is presented in Algorithm 1.

The performance of the GPU implementation highly depends on the Boolean network representation it uses. Since a node with inputs $\mathbf{N_i}$ has $2^{|\mathbf{N_i}|}$ possible combinations of inputs' state, it is a common approach to store a Boolean function of the node as $1 \times 2^{|\mathbf{N_i}|}$ vector $\mathbf{b_i}$. We can represent these vectors as $N \times 2^{\max |\mathbf{N_i}|}$ matrix B.

Having Boolean functions stored this way, one can obtain the next state of the network using one sparse matrix-vector multiplication (Eq. (2)) and one gather operation (Eq. (3)).

$$\mathbf{v} = A \times \mathbf{S}(t) \tag{2}$$

$$\sigma_i(t+1) = B_{i,\mathbf{v}_i} \tag{3}$$

These operations are well-studied and efficient algorithms for GPGPU exist [14][15].

The matrix A is constructed as defined in Eq. (4), (5). The special case in Eq. (5) ensures that nodes with no in-connections do not change their states.

```
GenerateNetwork();
epoch ← 0;
while epoch < epochsCount do
    gpuNetwork ← ConvertNetworkToGpuRepresentation();
    ZeroInitialize(stateSum);
    CopyToGpu(gpuNetwork, currentState, stateSum);
    while attractor not found do
        UpdateStateOnGpu(currentState, stateSum);
    end
    CopyFromGpu(stateSum);
    nodesToRewire ← ChooseRandomNodes(N_rew);
    ActivityDependentRewiring(nodesToRewire, stateSum);
    epoch ← epoch + 1;
end
```

Algorithm 1: General algorithm of network evolution with GPU acceleration

$$A_{ij} = \begin{cases} 2^k, & \text{if } i \text{ is a } k\text{-th input of } j \\ 0, & \text{otherwise} \end{cases} \quad \forall i \neq j \tag{4}$$

$$A_{ii} = \begin{cases} 1, & \text{if } |\mathbf{N}_i| = 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

III. Limiting the number of state updates. For RBN in a critical state, lengths of attractors and transients tend to have a power-law distribution with an exponent of -1 [16]. This implies that a number of attractors with a given length is in inverse proportion to that length, i.e. a major part of attractors has comparatively small lengths. During evolution of ARBN, the goal is to find an attractor at each epoch, and then modify the topology of a network according to a state of a randomly chosen node on the attractor. While searching for an attractor, one can add a limitation on a maximum number of iterations which will be examined for a single epoch. In Liu-Bassler algorithm this limitation is a part of the algorithm itself (as it has predefined checkpoints) while in Knuth's algorithm is can be added artificially. The logic behind this approach is that lengthy attractors are rare, so a limitation can be set in a way that will not significantly influence the dynamic of convergence to a steady state. In this study we address the question: to what extent such limiting influence the qualitative results of simulation? In other words, we study how the results of ARBN evolution will differ if we eliminate from consideration a part of lengthy attractors (extending the results which were reported in [12]).

IV. Rewiring different number of nodes per epoch. I (heuristics for attractor's search), II (parallel algorithm of updating the states) and III (limiting the number of state updates) are aimed to reduce an execution time of a single epoch. The total execution time is equal to the summation of times for all epochs, and the total number of epochs should be sufficient to reach the steady state. Therefore, another way to speed up the modeling process is to reduce a number of epochs which is required to achieve steady state. In this paper, we propose to perform it by a modification of ADRR intoduced in [9]. Instead of rewiring a single node per

epoch, we suggest to use several nodes as it could speed up the convergence to a steady state (especially for large networks) and, as a result, significantly decrease total time of simulation.

4 Experimental Study

The modified algorithm of ARBN evolution consists of a set of consecutive epochs. In each epoch the network attractor is found and N_{rew} nodes are rewired according to the modified ADRR. The maximum length of attractors μ_{max} is limited ether by predefined checkpoints in Liu-Bassler algorithm, or artificially in Knuth's algorithm in order to speed up the computations as described in Section 3. Due to this limitation only a fraction R of the attractors is found. During the evolution process the system reaches a dynamical equilibrium, which can be observed by measuring network's mean in-degree connectivity K, defined in Eq. 6. As the network evolves, K tends to some steady-state value K_{ss}.

$$K(t) = N^{-1} \sum_{i=1}^{N} |\mathbf{N}_i(t)| \tag{6}$$

Intuitively, Liu-Bassler algorithm of attractor search would have shown better performance than the Knuth's one, as the latter requires at least 3μ state updates to find an attractor of length μ. But in order to use Liu-Bassler algorithm one needs to choose the checkpoints correctly, which requires an a-priori knowledge of attractor and transient period distributions. This distribution depends on a network size, connectivity and structure (e. g. modular networks tend to have longer (shorter?) attractors than non-modular ones [ref]). We obtained an empirical distribution of attractor and transient lengths for different kinds of networks in our preliminary experiments with Knuth's algorithm. The checkpoints $T = [100, 200, 1000, 2000, ...\mu_{max}, 2\mu_{max}]$ seem to be the best trade-off between the number of state updates and the percentage of attractors found. However, with these checkpoints the difference in the number of state updates between the two algorithms is negligible. We used Knuth's algorithm in our future experiments, as it is more general and requires no a-priori knowledge of the network's behaviour.

Both algorithms were implemented using C++ programming language. The implementation is cross-platform and requires a GPU with CUDA compute capability at least 1.2. The representation of RBNs' state update function defined in Eq. 2, 3 allows for experimenting with different SpMV algorithms. These algorithms mainly aim at maximizing memory throughput and data locality, thus reducing the overall GPU kernel execution time. We chose compressed sparse-row (CSR) matrix storage format and implemented an sparse matrix-vector multiplication kernel as described in [14], as it shows the best performance on unstructured matrices.

To test the applicability of the approaches presented in Section 3, a set of experiments was performed. The experiments were conducted on a cluster consisting of 20 nodes with NVidia GeForce GT 640 graphic cards. In this study we considered only the networks with initial in-degree connectivity $K_{ini} = 2$. The results were averaged over 10 realizations of the evolution process.

Speedup obtained by GPU acceleration. We measured the execution times of both sequential and parallel versions of state updates. We used Liu-Bassler algorithm with checkpoints $T = \{100, 200, 1000, 2000, ..., 10^7, 2 \cdot 10^7\}$, $\mu_{max} = 10^7$. To decrease the number of random samples required to compare the versions, we did not stop the epoch as soon as an attractor was found. Instead, we performed all $2 \cdot 10^7$ state updates per epoch. Such simulation corresponds to the worst-case scenario when no attractors were found, i. e. $R = 0\%$. The results

presented in Figure 1 indicate that the GPU implementation is more efficient than the serial one for networks consisting of more than 200 nodes.

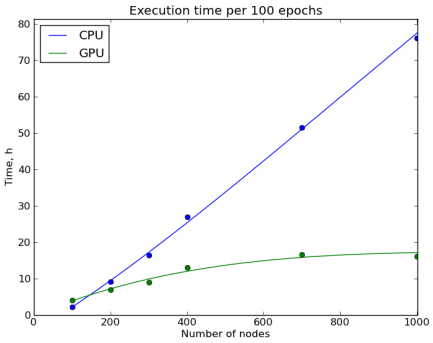

Figure 1: Execution time of the GPU algorithm compared to the serial one

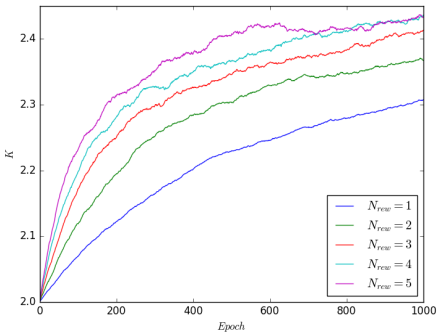

Figure 2: Evolution of mean connectivity for different number of nodes rewired each epoch; $N = 1000$

Rewiring more than one node per epoch.

	Nodes count				
μ_{max}	100	200	300	500	750
$5 \cdot 10^4$	99.77	88.19	71.19	45.48	29.62
$2.5 \cdot 10^5$	99.99	95.92	81.24	56.10	35.96
$5 \cdot 10^5$	100.00	97.43	86.58	59.36	38.73
$2.5 \cdot 10^6$	100.00	99.14	92.99	68.74	46.46
$5 \cdot 10^6$	100.00	99.53	94.40	73.00	49.22
$1 \cdot 10^7$	100.00	99.75	96.08	77.73	53.81

Table 1: Percentage R of attractors found during the evolution process with attractor length limited by μ_{max}

	Nodes count				
μ_{max}	100	200	300	500	750
$5 \cdot 10^4$	2.64	2.59	2.58	2.52	2.47
$2.5 \cdot 10^5$	2.66	2.60	2.56	2.51	2.48
$5 \cdot 10^5$	2.68	2.59	2.55	2.51	2.48
$2.5 \cdot 10^6$	2.69	2.60	2.55	2.51	2.48
$5 \cdot 10^6$	2.69	2.59	2.55	2.52	2.48
$1 \cdot 10^7$	2.68	2.62	2.55	2.51	2.48

Table 2: Mean steady state connectivity K_{ss} obtained as a result of evolution process with attractor length limited by μ_{max}

Limiting the maximum attractor length.

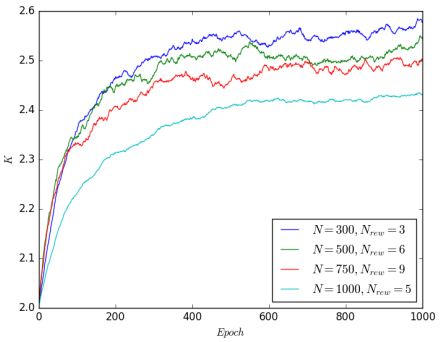

Figure 3: Evolution of mean network connectivity

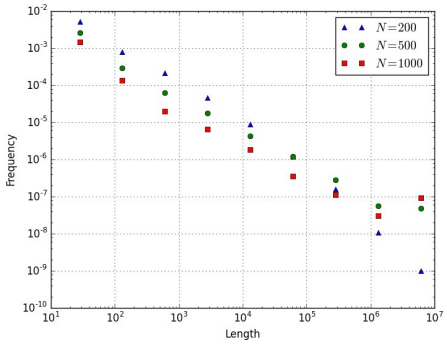

Figure 4: Empirical distribution of attractor lengths

5 Conclusion

References

[1] Carlos Gershenson. Introduction to Random Boolean Networks. In *Workshop and Tutorial Proceedings, Ninth International Conference on the Simulation and Synthesis of Living Systems (ALife IX*, pages 160–173, 2004.

[2] Barbara Drossel. Random boolean networks. *Rev. Nonlinear Dyn. Complex.*, 1:69–110, 2008.

[3] Daizhan Cheng, Hongsheng Qi, and Zhiqiang Li. Random Boolean Networks. In *Analysis and control of Boolean networks: a semi-tensor product approach*, pages 431–450. Springer London, 2011.

[4] Maximino Aldana, Susan Coppersmith, and Leo P Kadanoff. Boolean dynamics with random couplings. In *Perspectives and Problems in Nolinear Science*, pages 23–89. Springer, 2003.

[5] Stuart A Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of theoretical biology*, 22(3):437–467, 1969.

[6] Stuart Kauffman. The ensemble approach to understand genetic regulatory networks. *Phys. A*, 340(4):733–740, 2004.

[7] Ricard V Solé and Susanna C Manrubia. Extinction and self-organized criticality in a model of large-scale evolution. *Physical Review E*, 54(1):R42, 1996.

[8] Mart\'\in G Zimmermann, V\'\ictor M Egu\'\iluz, and Maxi San Miguel. Coevolution of dynamical states and interactions in dynamic networks. *Physical Review E*, 69(6):65102, 2004.

[9] Min Liu and Kevin E Bassler. Emergent criticality from coevolution in random boolean networks. *Phys. Rev. E*, 74(4):41910, 2006.

[10] Thimo Rohlf and Stefan Bornholdt. Self-organized criticality and adaptation in discrete dynamical networks. In *Adaptive Networks*, pages 73–106. Springer, 2009.

[11] Taichi Haruna and Sayaka Tanaka. On the Relationship between Local Rewiring Rules and Stationary Out-degree Distributions in Adaptive Random Boolean Network Models. In *ALIFE 14: The Fourteenth Conference on the Synthesis and Simulation of Living Systems*, volume 14, pages 420–426, 2014.

[12] Piotr J. Górski, Agnieszka Czaplicka, and Janusz A. Hołyst. Coevolution of information processing and topology in hierarchical adaptive random Boolean networks. *European Physical Journal B*, 89(2):1–9, 2016.

[13] Stefan Bornholdt and Thimo Rohlf. Topological evolution of dynamical networks: Global criticality from local dynamics. *Phys. Rev. Lett.*, 84(26):6114, 2000.

[14] Nathan Bell and Michael Garland. Efficient sparse matrix-vector multiplication on cuda. Technical report, Nvidia Technical Report NVR-2008-004, Nvidia Corporation, 2008.

[15] Bingsheng He, Naga K Govindaraju, Qiong Luo, and Burton Smith. Efficient gather and scatter operations on graphics processors. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, page 46. ACM, 2007.

[16] Florian Greil and Kevin E Bassler. Attractor period distribution for critical boolean networks. *arXiv preprint arXiv:0911.2481*, 2009.