Generating Null Models for Large-Scale Networks on GPU

Huan Li, Gang Lu[a], Junxia Guo

College of Information Science & Technology, Beijing University of Chemical Technology, Beijing, 100029, China

[a]email: sizheng@126.com

Keywords: complex network; null model; GPU; parallel algorithm;

Abstract. A network generated by randomly rewiring the edges of an original network on some constraint conditions is called the null model of the original network. It's a useful tool for revealing some mechanisms affecting the topology of networks. As the scales of networks become larger and larger, time consumption of generating null models increases. How to randomly rewire the edges of a large-scale network quickly becomes an urgent. In this paper, the generating algorithms for 0K, 1K and 2K null models of networks are implemented on GPU, which have not been done yet before. The experimental results show that the parallel algorithms greatly reduce the time consumption. Generating null models for large-scale networks on GPU is an efficient solution for study on null models of large-scale networks.

Introduction

Complex networks vary in scales and structures. Null models of neworks are always created as reference using numerical algorithms and mathematical methods based on statistical theory. It was proposed by ecologists initially, and then molecular biology scientists Maslov and Slppen [1] clearly stated the concept of null models of complex networks. A null model [2][3] of a complex network is a randomized network which has the same number of nodes and some same characteristics as the original network. The strategy is to deliberately exclude the mechanism being tested and decide whether the null model has the same characteristics as the original network. The specificity and flexibility provided by null models often cannot be supplied by general statistical tests in data analysis.

A null model of a network can be generated by randomly rewiring the edges of the network under some constraint conditions. According to the number of constraint conditions, null models can be classified into different orders:

(1) **0K-null model:** randomized network which has the same number of nodes and edges as the original network.

(2) **1K-null model:** randomized network which has the same number of nodes and degree distribution as the original network.

(3) **2K-null model:** randomized network which has the same number of nodes and joint degree distribution as the original network.

Null models with higher orders can be defined by adding more constraint conditions. However, as more and more constraint conditions are added, the number of edges can be rewired will decrease. Consequently, the generated randomized networks will become more and more similar to the original networks. Quan Chen et al. point out that if there are L edges in a network, the basic steps of random rewiring algorithms for generating a randomized network need to be iterated for $4L$ times [4]. In addition, in order to get statistically stable results, many randomized networks need to be generated for an original network. As a result, generating reasonable randomized null models for large-scale networks with millions, tens of millions or even more edges in acceptable time periods will become impossible without certain techniques. As far as the authors know, all the existing random rewiring algorithms of null models are implemented sequentially on CPU. There are no parallel ones on CPU or on GPU.

The rest of this paper is organized as follows: The basic random rewiring algorithms for

generating null models of networks are introduced. The algorithms are then implemented on GPU. Finally, the performance of the parallel algorithms is evaluated by comparing the implementations on CPU and GPU.

Random Rewiring Algorithm for Networks

A null model of a network is generated by randomly rewiring the edges of the network under some conditions. For example, to construct a randomized network model with the same degree sequence as the original network, we can make the positions of edges as random as possible on the condition that the degree of each node is preserved. If the edge between node v_i and node v_j is called as $e_{i,j}$, the random rewiring algorithms [5][6] can be stated as follows:

(1) Random rewiring algorithm for 0K null model. In each step, randomly delete an edge $e_{m,n}$ from the original network and then randomly add a new edge $e_{p,q}$ if it doesn't exist.

(2) *Random rewiring algorithm for 1K null model*. In each step, randomly pick two edges from the original network, which are $e_{m,n}$ and $e_{p,q}$. If $e_{m,q}$ and $e_{p,n}$ do not exist, $e_{m,n}$ and $e_{p,q}$ are deleted, and then $e_{m,q}$ and $e_{p,n}$ are created.

(3) Random rewiring algorithm for 2K null model. In each step, a constraint condition which says the nodes v_n and v_q must have the same degree is added to 1K null model.

Figure 1 shows the rewiring process of the three algorithms.

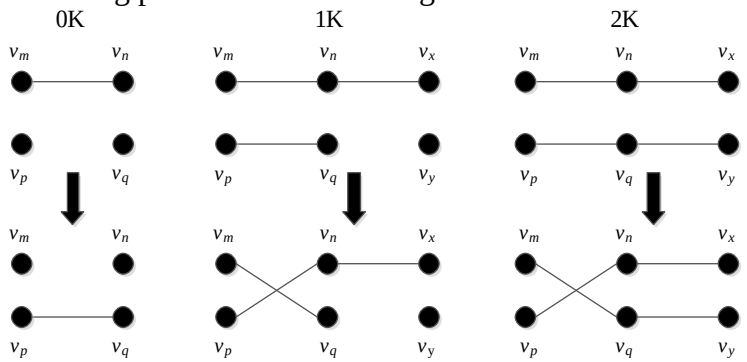

Fig. 1. Random rewiring algorithms for 0K, 1K and 2K null models

The above random rewiring algorithms can be extended to directed networks easily, such as by preserving both in-degree and out-degree of each node to generate 1K null model.

Maslov's matlab programs use an adjacency matrix to store a network. For storing a network with N nodes, an N by N adjacency matrix is needed. However, an adjacency matrix will occupy a lot of memory for a large scale network. It also wastes memory for storing a lot of zeros because complex networks are usually sparse networks. So Maslov's algorithms is only applicable to some small-scale of networks. In this paper, vectors are used in C++ to store data and implement both serial and parallel algorithms on CPU, and parallel algorithms on GPU as well.

Implementing the Random Rewiring Algorithm on GPU

Given a directed graph $G = (V, E)$, we have V as the nodes set, E as the edges set, $M = |V|$ as the number of nodes, and $L = |E|$ as the number of edges.

In order to focus on the algorithms for randomly rewiring, but not be trapped in the details of programming and debugging, C++ AMP is selected rather than other techniques such as CUDA or OpenCL. That makes it easy to write codes for GPU just by adding some features basing on classical C++.

In fact, random rewiring algorithms of null models have great requirements for sequence, every rewiring result will become next rewiring's basement. So there're two key problems to be solved when migrating the algorithms onto GPU: (1) Multiple threads may try to disconnect the same edge. (2) Multiple threads may try to create the same edge. All the following parallel algorithms on GPU for every order null model of networks have to solve the two problems.

A. Generating 0K Null Models of Networks on GPU

Firstly, we backup the graph's edges, and then randomly select a series of different edges to be rewired into a vector called *raneg*. After that, each edge in *raneg* is assigned to a thread on GPU. During the rewiring, every thread randomly selects two different vertices, if there is no edge between them, rewiring is then done. When all threads have finished rewiring, we serially search duplicated edges in the new edge set. If duplicated new edges are found, we retain the edge that appeared for the first time and replace the left by their backup. As the new set of edges are backup, previous duplicated rewirings need to be randomly rewired again by repeating above steps until there are no more duplicated new edged generated in the same iteration.

In this way, the number of edges of the network and the average degree of the network won't be changed. For a network with L edges, L different edges can be selected at most at a time. If N is larger than L, it is necessary to run the above algorithm repeatedly with no more than L rewirings in each iteration.

B. Generating 1K Null Models of Networks on GPU

For the 1K null model, two edges are randomly selected to be crossly rewired in each step. If multiple threads randomly select the same edge, the degree distribution of the network will be changed after rewiring, which will break the constraint condition of preserving degree distribution for 1K null model. To avoid this situation, in our algorithm, a series of edges to be rewired are randomly and non-repetitively selected from the original network and then grouped into nonoverlapping pairs at random. After that, different pairs of edges are assigned to different threads for 1K rewiring. Like 0K algorithm, different threads also may generate same new edges here. As it is done in 0K algorithm, duplicated new edges are serially found out. Except one of them, the left are restored by their backups. The restored duplicated edges need to be rewired at random repeatedly until no duplicated edges are generated. Since the edges need to be grouped into pairs, random rewiring times N must be smaller than or equal to $L/2$. When N is larger than that, it is just rewired for $L/2$ times in one iteration, until the left rewiring times is smaller than or equal to $L/2$.

C. Generating 2K Null Models of Networks on GPU

2K null model has one more constraint condition than the 1K one, which is node v_n and node v_q must have the same degree, as Figure1 shows. Algorithm for 1K null model can be easily extended for 2K null model by adding condition that node v_n and node v_q must have the same degree, so no more tautology here.

Test results

A Wikipedia vote relationship network[1] is chosen for experiments here. There're 7115 nodes and 103689 edges in this network. The nodes of this network stand for Wikipedia users. An edge from node i to node j means user i votes for user j. The experiments are taken on a computer with a Intel i7-4770 CPU, 32GB memory, a NVIDIA GTX480 GPU, and 64-bit Windows Server 2008 system.

The execution time of serial and parallel algorithms for 0K, 1K and 2K null models is tested by randomly rewiring the original network for 50000 to 450000 times. Every test is taken for 5 times, and the average time spent is taken as the result. The results are shown in Figure 2. The horizontal and vertical axis represent random rewiring times and time spent in seconds respectively. To make the figure more clear, the values on vertical axis are logarithmic.

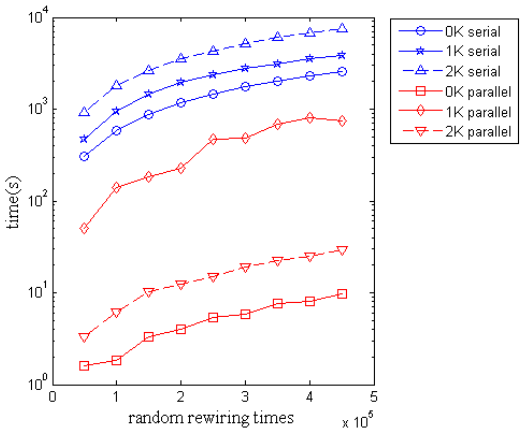

Fig. 2. Comparison of time consumption between serial and parallel algorithms

As the figure shows, the time consumption of serial algorithms grows linearly as random rewiring times increase. As the constraint conditions become stricter, the CPU needs to do more judgments, which makes the algorithms slower.

Among the three parallel algorithms on GPU, the one for 1K null models is slower than the 2K one. That's because it needs several iterations for duplicated new edges. Meanwhile, the strict constraint conditions of 2K null model result in less edges to be rewired, which makes the 2K algorithm faster. When randomly rewiring for 400000 times, which is about 4 times of the number of edges, the speed-up ratios of parallel algorithms on GPU for 0K, 1K and 2K null models can reach 283, 4 and 276 respectively.

To further demonstrate the superiority of GPU, the same parallel algorithms are also implemented on CPU. The experimenting computer has an i7-4770 CPU with 4 cores and 8 threads in total. All of the 8 threads are used in the experiments. Five public datasets, which are p2p-Gnutella (P2P), wiki_Vote (WV), email-EuAll (EM), amazon (AM), and web-stanford (WS), are used to take the experiments. Table I lists the numbers of nodes, edges, and random rewiring times of the five netwoks. The random rewiring times are set to be 4 times of number of the edges. The time consumptions of serial algorithms on CPU, parallel algorithms on CPU, and parallel algorithms on GPU are compared. Table II, Table III and Table IV list the results of 0K, 1K and 2K null models respectively. C-S means serial algorithms on CPU, C-P means parallel algorithms on CPU, and G-P means parallel algorithms on GPU. N/A means the time spent is beyond 24 hours.

As the tables show, the classical serial algorithms on CPU can not finish in 24 hours for networks larger than wiki_vote. Though parallelizing them on CPU can get several times speed-up, it still can not finish in 24 hours for amazon or web-stanford network. Parallel algorithms on GPU, however, mostly can finish in less than 1 hour for all the five datasets, and it only takes less than 2 hours for web-stanford's 0K and 1K null model.

TABLE I. THE INFORMATION OF THE DATASETS

dataset	nodes	edges	# of random rewiring
P2P	6,301	20,777	83,108
WV	8,298	103,689	414,756
EM	265,214	420,045	1,680,180
AM	262,111	1,234,877	4,939,508
WS	281,903	2,312,497	9,249,988

TABLE II. GENERATION TIME OF 0K NULL MODEL

dataset	*C-S*	*C-P*	*G-P*
P2P	4m	30.021s	2.855s
WV	38m2s	9m	18.209s
EM	N/A	2h59m53s	1m7s
AM	N/A	N/A	21m1s
WS	N/A	N/A	38m55s

TABLE III. GENERATION TIME OF 1K NULL MODEL

dataset	*C-S*	*C-P*	*G-P*
P2P	4m35s	58.193s	16.493s
WV	1h1m10s	48m3s	13m31s
EM	N/A	N/A	15m46s
AM	N/A	N/A	35m
WS	N/A	N/A	1h43m30s

TABLE IV. GENERATION TIME OF 2K NULL MODEL

dataset	*C-S*	*C-P*	*G-P*
P2P	4m53s	35.348s	2.923s
WV	1h54m18s	12m	25.001s
EM	N/A	3h30m32s	2m22s
AM	N/A	N/A	25m9s
WS	N/A	N/A	1h5m36s

Conclusion

By single-threaded serial algorithms on CPU, the time consumption of generating null models for complex networks increases rapidly as the scales of the networks expand. In this paper, new algorithms for generating null models based on existing graphs by GPU are proposed. The algorithms are able to generate 0K, 1K, and 2K null models. The experimental results show that the parallel algorithms on GPU are much faster than single-threaded serial algorithms on CPU. However, the memory of GPU is always limited, while the scales of networks are not. Therefore, it will be interesting work to design and implement similar algorithms on multiple GPUs. Loading data by segmentation for even larger scale complex networks which cannot be loaded into memory of a GPU at one time will be interesting as well.

Acknowledgement

In this paper, the research was sponsored by Beijing Higher Education Young Elite Teacher Project (Project No. YETP0506).

References

[1] Sergei Maslov and Kim Sneppen. Specificity and Stability in Topology of Protein networks [J]. Science. 2002, 296 (5569):910–913.

[2] Nicholas. J. Gotelli and Werner Ulrich. Statistical Challenges in Null Model Analysis [J]. Oikos. 2012 121 (2) 171–180.

[3] Priya Mahadevan, Dmitri Krioukov, Kevin Fall, and Amin Vahdat. Systematic Topology Analysis and Generation Using Degree Correlations [J]. ACM SIGCOMM Computer Communication Reveiw, 2006 36 (4) 135–146.

[4] Quan Chen, Jianmei Yang and Jinqun Zeng. Null Model and Its Application in the Research of Complex Networks [J]. Complex systems and consumption science, 2013 10 (1) 8-17.

[5] Yabing Liu and Xiaofan Wang. Community Structure Analysis of Complex Network based on Random Rewiring [J]. Microcomputer Applications, 2010 26 (11) 29-32.

[6] Priya Mahadevan, Calvin Hubble, Dmitri Krioukov, Bradley Huffaker, and Amin Vahdat. Orbis: Rescaling Degree Correlations to Generate Annotated Internet Topologies [J]. ACM SIGCOMM Computer Communication Reveiw, 2007 37 (4) 325–336.