The cxnet software package for investigation of complex networks

Árpád Horváth
Alba Regia University Centre of Óbuda University,
H-8000 Székesfehérvár, Budai út 45., Hungary
Email: horvath.arpad@arek.uni-obuda.hu
and
Zoltán Trócsányi
University of Debrecen and Institute of Nuclear Research of the Hungarian Academy of Sciences
H-4010 Debrecen P.O.Box 105, Hungary
Email: Z.Trocsanyi@atomki.hu

***Abstract*—One of the aim of the recent investigations of complex networks is to explore the reason behind the astonishing similarity of different networks. We can compare real world networks with the network models to test our theories about the evolution of networks. There are cases, when we do not have evolutionary models for describing the most important properties of a network. In these cases we can use optimization. We have examined one of the methods for optimization based on multifractal networks, and have written a program to realize it. In our presentation we will introduce the method and the results produced by our program.**

I. Introduction

Networks are often used as a synonym of the graphs in the field of sociology and some other fields of the science. Complex networks are very large networks with a structure difficult to describe in details. The aim of the science of the complex networks is to study the general properties of real networks. The earliest investigations of networks happened in the field of sociology, when they studied the acquaintance network of people.

There is a lot of networks in the fields of engineering and informatics, such as the World Wide Web, the Internet, whose investigation has brought networks in the forefront of research recently. In biology and medicine the network of protein interactions, the food chain or to forecast the spreading of a disease, the acquaintance and sexual networks are important. Properties of many networks, network models and methods of the investigations have been summarized in several papers [1], [2].

If we want to to create networks with arbitrary properties, we usually make optimization that means we change the network to get closer and closer to the properties we want to achieve. An efficient way of achieving this optimization is the multifractal network generator [3]. We have implemented this method based on the igraph module of the Python language [4][5]. We describe here the method, our program and some results.

II. Multifractal network generation

The method of generating networks with the usage of multifractals is described in detail in the article of Palla et al [3].

In the multifractal network generation we define a generating (probability) measure on the $[0,1[\times[0,1[$ unit square, then we create a link probability measure with the iteration of the generating measure, an finally, we create links between the nodes using the link probability measure.

A. Generating function and the link probability measure

We divide both of the x and y axes into m not necessarily equal intervals to define a generating measure. The intervals on the x and y axes must have the same division points. With this division we created m^2 rectangles on the unit square. We assign probabilities p_{ij} to each of the rectangles in a symmetric fashion,

$$p_{ij} = p_{ji}\,, \quad \sum_{i,j=0}^{m-1} p_{ij} = 1.$$

The probability assigned to the rectangle at the origin is denoted by the p_{00} and that at the opposite corner is $p_{m-1,m-1}$.

The kth iteration of the generating measure means a unit square divided into rectangles with assigned probabilities as in the generating measure, but with $m^k \times m^k$ rectangles. By definition the first iteration ($k = 1$) gives the generating measure itself.

For the case of $k > 1$, we obtain the division points from the division points of the $(k-1)$st iteration by dividing each of its intervals into m subintervals, where the length of subintervals are proportional to the length of intervals of the original generating measure.

The $p_{ij}(k)$ probabilities of the k-th iteration can be calculated as

$$p_{ij}(k) = \prod_{q=1}^{k} p_{i_q j_q}, \tag{1}$$

where

$$i_q = \left\lfloor \frac{i \mod m^{k-q+1}}{m^{k-q}} \right\rfloor. \tag{2}$$

The notation ($i \mod d$) means the remainder of the integer division i/d and $\lfloor x \rfloor$ denotes the floor (integer part) of x. Analogous equation to (2) gives j_q as well.

B. Generating the network

The generation of networks proceeds in two steps. First, we need to iterate the generating measure to get the link probability measure, and than generate the network from the link probability measure obtained after the iterations. The latter goes as follows.

We choose the number of iterations (k) and the number of nodes (N) in the network. If one axis of the generator measure is divided into m intervals with the division points, there will be m^k intervals in one axis of the link probability measure. We assign to each node with index l ($l \in [1, N]$, integer) an r_l random value from a uniform distribution on the $[0, 1[$ interval. We determine the i_l index of the interval where r_l is located ($i_l \in [0, m^k - 1]$).

We now go through the node pairs of the network. Let l_1 and l_2 be the indexes of the nodes of the pair with corresponding r_{l_1} and r_{l_2} values. Then we link the pair with the probability $p_{i_{l_1} i_{l_2}}(k)$, where $p_{ij}(k)$ are the probabilites after the k-th iteration of the generator measure defined in eqn. (1).

C. Adjusting the generating function

We define one or more target property we want to achieve and an energy function (a non-negative function) for each target property that measures the goodness of the created network. The smaller energy, the closer the network to the one with the target property.

We need to fix the m numbers of intervals on one axis, the k number of iteration, the N_s number of the generated networks in a step, and a f_{Ns} factor to the cases, when the generator measure is a good candidate to be accepted. In the method there is a decreasing T temperature variable as well. In our program we start with equal probabilities and equal interval lengths on axes. In each step we relocate either a division point, or change one of the probabilities. We create the new link probability measure and we generate more networks. The method we describe below has two steps: i) we calculate the E' energy to this network set, and ii) accept or reject the new generator measure. If we accept, we store the generator measure and its energy.

If we have N_p target properties, we calculate the energy of a network as the sum of the energies of this networks for all of the properties:

$$E_i = \sum_{p=1}^{N_p} E_{i,p}, \tag{3}$$

then we can calculate the energy of a network set as the average energy of the networks:

$$E'(N) = \frac{1}{N} \sum_{i=1}^{N} E_i. \tag{4}$$

When we have averaged the energy $E'_0 = E'(N_s)$ of the first N_s networks, we have to decide to generate more networks in

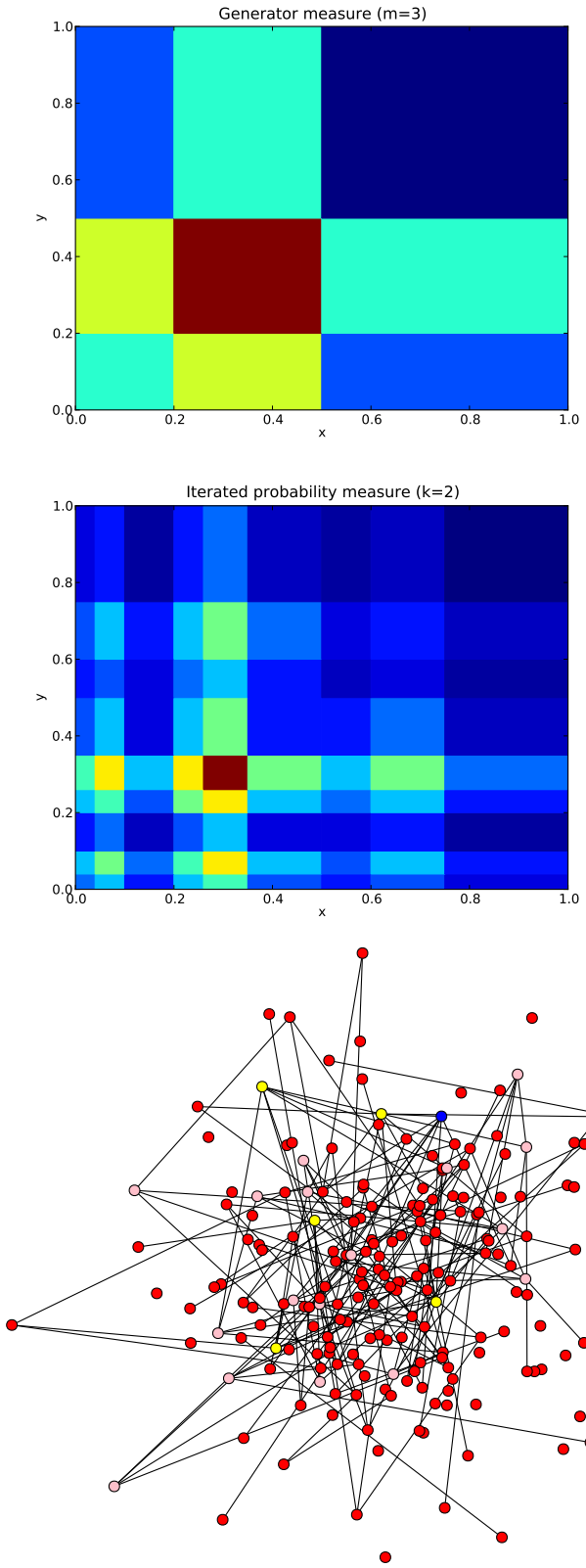

Figure 1. A generator measure with the division points 0.2 and 0.5, and the iterated measure we use as link probability measure, and the network generated from the latter measure. We can see that in the place of a rectangle of the generating measure there is 3×3 rectangles in the iterated probability measure. Network nodes are colored by degree (8: blue, 6 and 7: yellow, 4 and 5: pink, smaller degree: red)

this step or not. If the energy is smaller then the sum of the E energy belonging to the network of the existing generating measure E and the temperature,

$$E'_0 < E + T, \tag{5}$$

we will generate more networks to get altogether $f_N s \times N_s$ networks to calculate a more accurate $E' = E'(f_{Ns} \times N_s)$ energy for the generator measure, else we use the E'_0 as the E' energy of the generator measure.

If E' is smaller than that belonging to the network of the existing generating measure E, than we change the generating measure to the new one, and store the average energy. If the $E' > E$, than we accept the new generating measure with the probability

$$P(T) = \exp\left(-\frac{E' - E}{T}\right), \tag{6}$$

and reject with $1 - P(T)$ probability. The arbitrary parameter T plays the role of temperature (in units of the energy).

If we decrease the temperature slowly, the generating process allows for possible to escape from local minima. The smaller the temperature, the more changes will be rejected and the network converges to one with the target property.

III. The MFNG part of CXNET

There is an existing implementation of the multifractal network generator written in C++ [3] without the option of setting target properties if we do not have the source code. In a previous work, we developed a software, called cxnet, in Python to investigate complex networks and bring them into the higher education [6], [7]. In order to be able to set target properties of the generated network that in a second step can be analyzed with the cxnet package, we developed our version of mfng as a part of the `cxnet` module of cxnet software package. The documentation of cxnet can be reached from the page `http://mail.roik.bmf.hu/cxnet`.

The mfng includes the `ProbMeasure` class, the `Generator` class, the `Property` class and its derived classes.

A `ProbMeasure` instance contains the probabilities and the division points. It includes a method to iterate the measure returning with a new `ProbMeasure` instance. It implements two other methods to create a network with a given number of nodes, and to plot the `ProbMeasure` with Pylab.

`mfng` generates a probability measure for networks with the given properties. In each step of the generation there is two half steps. In the first half step the generator changes the probabilities, in the second half step changes the division points.

The `Generator` class stores the main settings of the generations, and the properties we want to achieve. The main settings are the initial and final temperature, the temperature factor with which the temperature is multiplied in each step, the number of generated networks in each step, the m parameter of the initial probability measure and the k number of iteration.

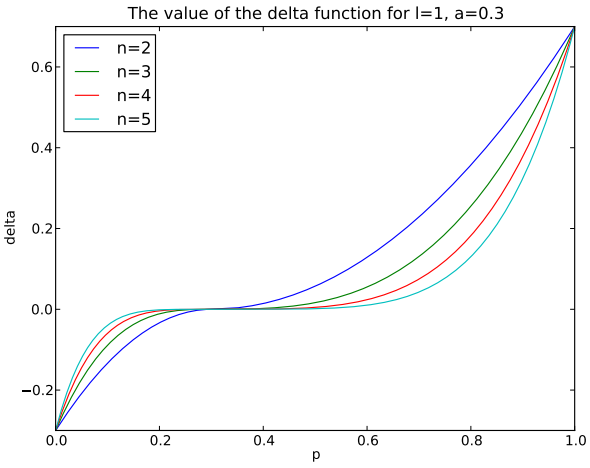

Figure 2. The $\Delta_i(p)$ function used for relocating a division point. If we use one inner division point ($m = 2$) and its actual value is 0.3, the parameters l and a are $l = 1$ and $a = 0.3$. We plotted the function with these parameters and with the exponents $n = 2, 3, 4, 5$. The value of the function will be in the interval $[-a, l - a[= [-0.3, 0.7[$ and as the n increases the value of the function will be more likely to be close to zero if p is a random value.

The `Generator` class stores the properties as instances of derived classes of the `Property` class. These classes require an `energy` method to give back a value which is used to rate the goodness of the network generated from the link probability measure.

If a single property is added to the Generator instance, the decision will depend on the average of the energies of the generated networks from the iterated probability measures. If more than one property is added, the sum of the average energies will be used.

A. Changing the division points and probabilities

In one step our program changes on of the probabilities first, then one of the division points.

To change the division points we add zero and one to the list of the division points, so the division points are:

$$d_0 = 0, d_1, d_2, \ldots, d_{m-1}, d_m = 1 \tag{7}$$

Then the program chooses randomly one of the inner division points with the index $i \in [1, m-1]$, and chooses a p random value from the uniform distribution on the $[0, 1[$ interval. We relocate the chosen division point to $d_i + \Delta_i(p)$, where

$$\Delta_i(p) = \begin{cases} l\dfrac{\left(p - \frac{a}{l}\right)^n}{\left(1 - \frac{a}{l}\right)^{n-1}}, & \text{if } p > \frac{a}{l} \\ \\ l\dfrac{\left(p - \frac{a}{l}\right)^n}{\left(-\frac{a}{l}\right)^{n-1}}, & \text{if } p \leq \frac{a}{l} \end{cases}. \tag{8}$$

Here, $l = d_{i+1} - d_{i-1}$, and $a = d_i - d_{i-1}$ and $n > 1$. If we increase the n parameter, the new division point is more likely to stay in the proximity of the original division point (Fig 2).

IV. Results

We defined two target properties as two classes, and assigned an energy function to each. This function has one input, a certain property of the network, and returns a positive value. We can add one or more properties to the generator instance. The MaxDegree property has one parameter, the maximum degree k_{max} we want to achieve. Its energy function calculates the k'_{max} maximal degree of the given network, and returns the value

$$E = \frac{|k'_{max} - k_{max}|}{k_{max}}.$$

The AverageDegree property is similar to the MaxDegree property, but instead of the maximum degree we use the arithmetic mean of degrees of the nodes.

We made two series of simulations. In the first one, we set a single target property, a given maximal degree. In the second one we set a maximal degree and an average degree simultaneously.

In the first series of simulations we used the parameters $m = 2$, $k = 3$ and the number of generated networks was $N = 200$. We generated 10 networks from each generator measure to calculate the energy. We started with temperature $T_0 = 0.8$ and stopped at $T_{limit} = 0.0005$. We multiplied the temperature with T_{factor} in each step. We used the values $T_{factor} = 0.999$, 0.998 and 0.996. We chose $n = 2$ in the exponent in the Δ_i function to change the division points. The target of the maximal degree was 15.

The results are in the Table I. E_i is the energy of the first network generation with the initial generator measure, E_f is the final energy with the last accepted generator measure, p_{00}, p_{10} and p_{11} are the 3 independent elements of the probability matrix ($p_{10} = p_{01}$), d_1 is the only one division point.

In the generations, the values of the main diagonal vanish, so the other two values approach 0.5. The division points are in the $[0.18, 0.26]$ or in the $[0.72, 0.77]$ intervals. These two intervals are close to be symmetric with respect to 0.5.

To measure the goodness of the optimization, we generated 100 networks from each of the results of the simulations, and we measured the arithmetic mean and the sample standard deviation of the maximum degree and average degree of the network. These values are denoted by $\langle k_{max} \rangle$ and $\langle k_{avg} \rangle$ in the Table I. This measure was carried out for the initial generator measure (with equal probabilities and equal interval length) as well. (The networks generated from the initial generator measure are random networks with link probability $p = 0.25^k = 0.0156$, giving an estimated value $p(N-1) = 3.109$ for the arithmetic mean of the degrees.) The values we got is in the last row of the table. As we can see the k_{max} increased significantly and the average degree decreased.

In the simulation we generated only 10 networks for each generator measure. The ten values of the maximal degrees can be significantly larger than the average of the 100 values, and so the energy function can be more closer to zero as in the case we would generated more networks. For example, the maximal degrees of the networks created from the last accepted generator measure, corresponding to the E_f value given in the first row of Table I, was

$$16,\ 15,\ 14,\ 14,\ 16,\ 15,\ 15,\ 15,\ 16,\ 15.$$

Each element is bigger then the arithmetic mean of the maximal degrees (12.8) of the 100 network generated later. These large values have given an energy very close to zero. To make better result, we need to make more measures during the simulation, at least for the generator measures we want to accept.

In the second series of generations we used the same parameters as in the first one, but we set two properties simultaneously: the average degree to 3.1 and the maximum degree to 15. The results are in the Table II.

In the first simulation, where the target was only the maximal degree 15, the average degree decreased from near 3 to near 1. In this second series of simulations we wanted to keep constant the average degree during the simulation with increasing maximal degree. In these simulations the maximal degree increased with a smaller amount toward the target value as in the previous ones, but the average degree remained near to the original value.

V. Conclusion

The multifractal network generator based on the igraph module is able to generate networks with given properties. The usage of the program suits for the teaching of complex networks in higher education. Presently, the program is in the testing phase.

This work was supported by the TÁMOP 4.2.1./B-09/1/KONV-2010-0007 project.

References

[1] R. Albert and A. Barabasi, "Statistical mechanics of complex networks," *REVIEWS OF MODERN PHYSICS*, vol. 74, no. 1, pp. 47–97, JAN 2002. [Online]. Available: http://arxiv.org/abs/cond-mat/0106096

[2] M. E. J. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, p. 167, 2003. [Online]. Available: http://arxiv.org/abs/cond-mat/0303516

[3] G. Palla, L. Lovász, and T. Vicsek, "Multifractal network generator," *Proceedings of the National Academy of Sciences*, vol. 107, no. 17, 2010.

[4] G. Csárdi and T. Nepusz, "Igraph," 2003–. [Online]. Available: http://igraph.sourceforge.net/

[5] ——, "The igraph software package for complex network research," *InterJournal Complex Systems*, p. 1695, 2006.

[6] A. Horváth, "Studying complex networks with cxnet," *Acta Physica Debrecina*, vol. XLIV, 2010.

[7] A. Horváth and Z. Trócsányi, "Complex networks in the curriculum of computer engineers," *IEEE Proceedings of the 8th International Symposium on Applied Machine Intelligence and Informatics*, 2010.

Table I
THE RESULTS OF THE GENERATION WITH $k_{max} = 15$

T_{factor}	E_i	E_f	p_{00}	p_{10}	p_{11}	d_1	$\langle k_{max} \rangle$	$\langle k_{avg} \rangle$
0.999	0.413	0.0733	5.24×10^{-16}	0.50000	3.03×10^{-14}	0.257	12.8 ± 3.35749	1.0502 ± 0.230682
0.999	0.440	0.0600	2.16×10^{-13}	0.50000	1.76×10^{-14}	0.730	13.19 ± 4.65083	0.6844 ± 0.211495
0.999	0.407	0.0667	1.36×10^{-15}	0.50000	4.51×10^{-16}	0.221	12.64 ± 3.12216	1.2734 ± 0.270985
0.999	0.387	0.0533	1.19×10^{-14}	0.50000	2.79×10^{-14}	0.757	12.63 ± 3.63389	1.0092 ± 0.217524
0.999	0.440	0.0667	6.67×10^{-16}	0.50000	2.01×10^{-15}	0.184	12.86 ± 2.67808	1.5588 ± 0.26123
0.999	0.373	0.0333	7.94×10^{-16}	0.50000	4.44×10^{-16}	0.775	12.32 ± 2.95379	1.3636 ± 0.264684
0.998	0.387	0.0667	6.28×10^{-9}	0.50000	2.03×10^{-8}	0.763	12.8 ± 3.56753	1.1964 ± 0.247172
0.998	0.420	0.0733	2.90×10^{-8}	0.50000	6.62×10^{-9}	0.743	12.79 ± 3.15042	1.3904 ± 0.248409
0.996	0.407	0.0400	7.94×10^{-5}	0.49975	4.14×10^{-4}	0.761	13.59 ± 2.69716	1.2226 ± 0.20976
0.996	0.387	0.0600	2.43×10^{-5}	0.49995	6.95×10^{-5}	0.251	12.93 ± 2.83291	1.2933 ± 0.235552
Original generator measure			0.25	0.25	0.25	0.5	8.83 ± 1.1106	3.0846 ± 0.179525

Table II
THE RESULTS OF THE GENERATION WITH $\langle k \rangle = 3.1$ AND $k_{max} = 15$

T_{factor}	E_i	E_f	p_{00}	p_{10}	p_{11}	d_1	$\langle k_{max} \rangle$	$\langle k_{avg} \rangle$
0.999	0.455	0.3156	7.16×10^{-18}	0.50000	3.56×10^{-16}	0.573	9.12 ± 1.18305	2.9106 ± 0.203573
0.999	0.470	0.3094	1.76×10^{-15}	0.50000	2.91×10^{-14}	0.439	9.05 ± 1.29002	2.948 ± 0.17363
0.999	0.445	0.3111	2.28×10^{-17}	0.50000	2.14×10^{-15}	0.380	9.91 ± 1.40054	2.6088 ± 0.234082
0.999	0.479	0.3457	1.76×10^{-17}	0.50000	1.21×10^{-16}	0.383	9.99 ± 1.45987	2.6236 ± 0.255705
0.999	0.503	0.3187	1.36×10^{-16}	0.50000	2.62×10^{-17}	0.413	9.41 ± 1.09263	2.7985 ± 0.24648
0.999	0.470	0.3049	1.29×10^{-16}	0.50000	1.26×10^{-16}	0.426	9.44 ± 1.33576	2.9242 ± 0.170188
0.998	0.490	0.3476	6.73×10^{-7}	0.50000	5.69×10^{-8}	0.374	10.06 ± 1.72808	2.5689 ± 0.244044
0.998	0.406	0.3323	1.89×10^{-8}	0.50000	3.41×10^{-8}	0.401	$9.84 \pm 1.3005,$	2.7728 ± 0.217437
0.996	0.489	0.3592	3.31×10^{-4}	0.49980	7.27×10^{-5}	0.460	9.11 ± 1.14499	3.0522 ± 0.193368
0.996	0.472	0.3672	4.20×10^{-5}	0.49997	9.12×10^{-6}	0.539	8.95 ± 1.08595	3.0554 ± 0.19343
Original generator measure			0.25	0.25	0.25	0.5	8.83 ± 1.1106	3.0846 ± 0.179525