Creating Networks with Given Degree Distribution with the Multifractal Network Generator

Árpád Horváth
Alba Regia University Centre of Óbuda University,
H-8000 Székesfehérvár, Budai út 45., Hungary
Email: horvath.arpad@arek.uni-obuda.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. A promising 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 have generate networks with given degree distribution. 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, end finally, we create links between the nodes using the link probability measure.

A. Generating measure and 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.

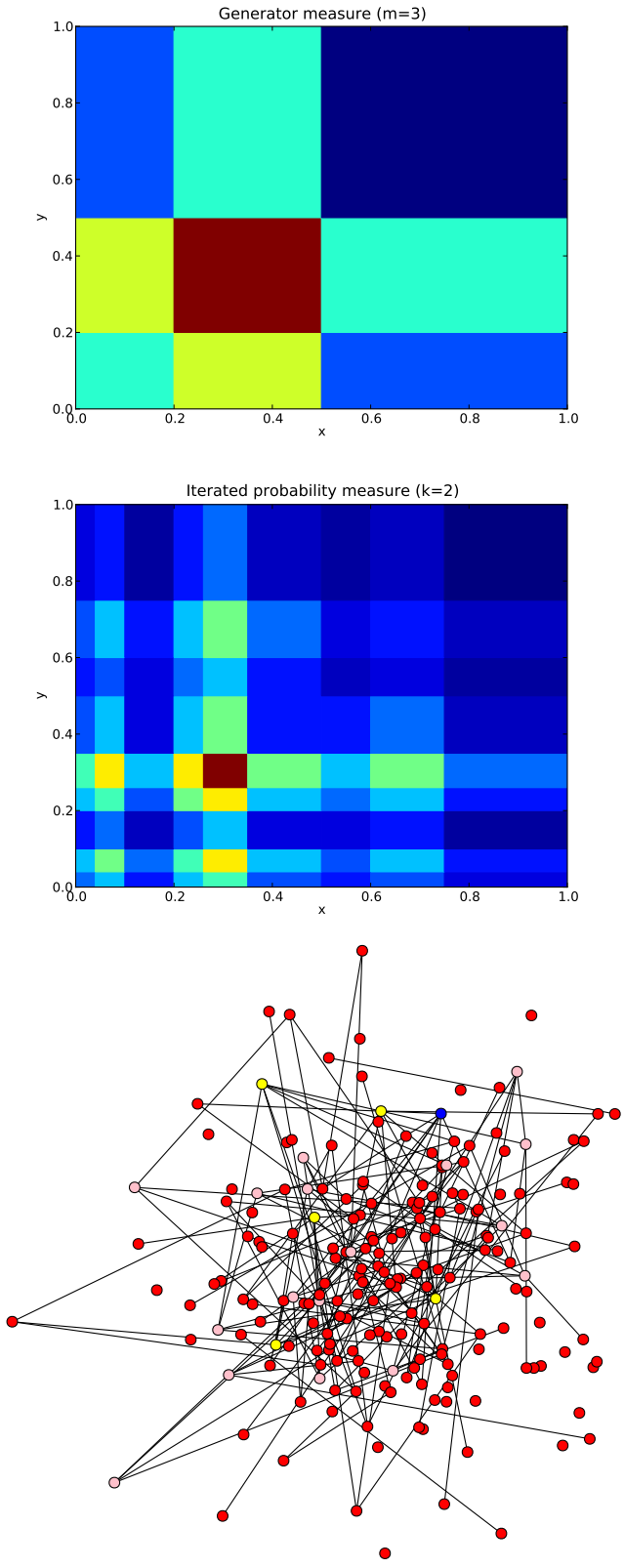


Figure 1. A generator measure with the division points 0.2 and 0.5, and the iterated measure we use as link probability measure, and the network generated from the latter measure. We can see that in the place of a rectangle of the generating measure there is 3×3 rectangles in the iterated probability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

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 probabilities after the K-th iteration of the generator measure defined in eqn. (1).

C. Adjusting the generating function

We define a target property we want to achieve and an energy function (a non-negative function) 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, and the K number of iteration. 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 calculate the energy belongs to the link probability measure. If this E' energy 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 energy. If the $E' > E$, than we accept the new generating measure with the probability

$$P(T) = \exp\left(-\frac{E' - E}{T}\right), \tag{3}$$

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. 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://arek.uni-obuda.hu/cxnet`.

The mfng includes the `ProbMeasure` class, the `Generator` class and some property classes.

A `ProbMeasure` instance contains the probabilities and the division points. It includes a function 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. There is two steps with the same temperature T. In one step the generator changes the probabilities, in the second 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.

The `Generator` class stores the properties as a list of instances of property classes. These classes require an `energy` method to give back a value which is used to rate the goodness of the link probability measure. If more than one property is added, the sum of the energies will be used.

A. *Changing the division points and the probabilities*

In one step our program changes one 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:

$$d_0 = 0, d_1, d_2, \ldots, d_{m-1}, d_m = 1 \tag{4}$$

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

$$\Delta_i(p) = \begin{cases} l\dfrac{\left(p - \frac{a}{l}\right)^n}{\left(1 - \frac{a}{l}\right)^{n-1}}, & \text{if } p > \frac{a}{l} \\ \\ l\dfrac{\left(p - \frac{a}{l}\right)^n}{\left(-\frac{a}{l}\right)^{n-1}}, & \text{if } p \leq \frac{a}{l} \end{cases}. \tag{5}$$

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).

The changing of the probabilities has several steps. First the program chooses one of the element of the probability matrix randomly. Then it chooses a random value from a uniform distribution on the $[0.9, 1.1[$ interval and it multiplies the probability with this value. If it is not in the main diagonal, we need to do this with the symmetric element as well. In the last step the program normalizes the probability matrix.

IV. Generating a network with a given degree distribution

The degree distribution $p(k)$ is a function of degree k giving the probability of a node having the degree k. This can be calculated as

$$p(k) = \sum_{i=0}^{m^K} p_i(k) l_i, \tag{6}$$

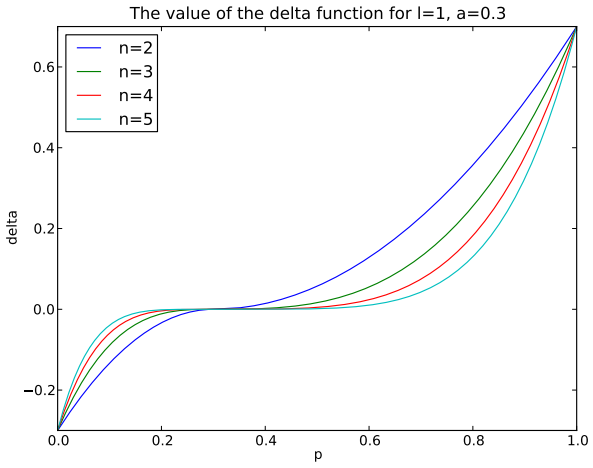


Figure 2. The $\Delta_i(p)$ function used for relocating a division point. If we use one inner division point ($m = 2$) and its actual value is 0.3, the parameters l and a are $l = 1$ and $a = 0.3$. We plotted the function with these parameters and with the exponents $n = 2, 3, 4, 5$. The value of the 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.

where

$$p_i(k) = \frac{\langle k_i \rangle^k}{k!} e^{-\langle k_i \rangle}, \tag{7}$$

$$\langle k_i \rangle = N \sum_j p_{ij} l_j, \tag{8}$$

$\langle k_i \rangle$ is the expected degree of a node in the ith interval and $l_i = d_{i+1} - d_i$ is the length of the ith interval.

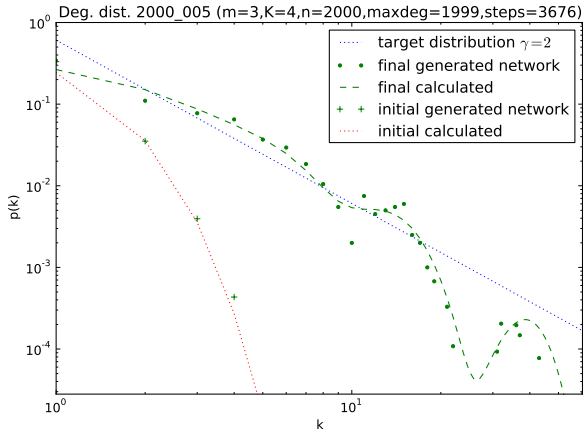
We can define the energy of a link probability measure as

$$E = \sum_{k=k_{min}}^{k_{max}} \frac{|p^*(k) - p(k)|}{p(k)}, \tag{9}$$

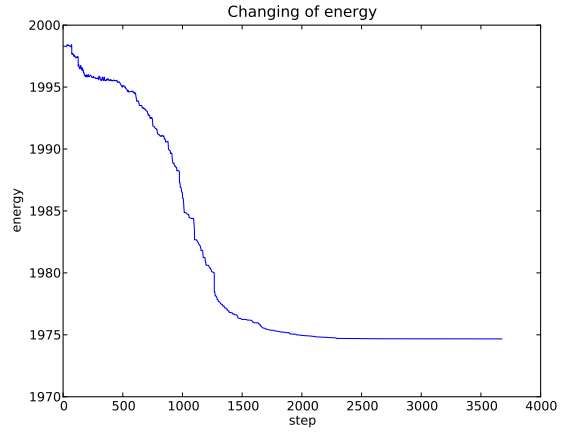
where $p^*(k)$ is the degree distribution of the actual link probability measure, and $p(k)$ is the target degree distribution.

We defined two types of properties in connection with the degree distribution. In the first one the target is given by a function of degree k. In the second one the target is given as a degree distribution with logarithmic binning, so we could create *clones* of real networks like Internet and the coauthorship network of the researchers of the network science. In the figures 3 and 4 we can see the results of the generations with the two types of properties respectively.

We used binning to plot the $p(k)$ degree distributions. The degree distribution $p(k)$ times the bin width, $p(k)(k_{i+1} - k_i)$ for $k \in [k_i, k_{i+1}[$, gives the probability that the degree of a randomly chosen connection falls in the range $[k_i, k_{i+1}[$ (of course, only integer degrees are possible and not all integer values are actually realized in a given network).

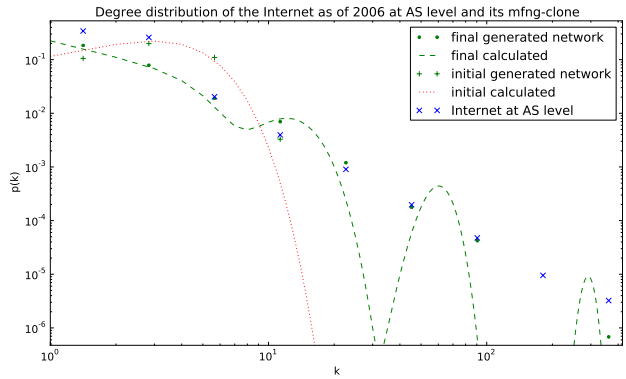


(a)

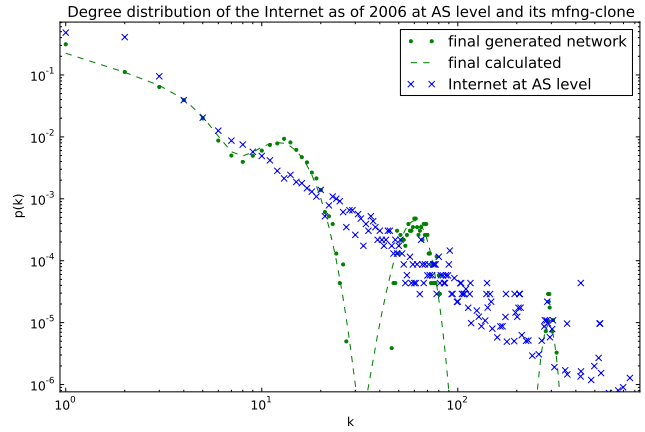


(b)

Figure 3. The results of a generation of a 2000-node-size network with a power-law function target with the exponent -2. In the subfigure (a) we can see the target degree distribution with blue dotted line, the initial degree distribution calculated from the initial probability measure with red dotted line, the final degree distribution calculated from the last accepted probability measure. With dots there are the degree distribution of networks generated from the initial probability measure and from the last accepted one. In the subfigure (b) can be seen the changing of the energy as the function of the step number.



(a)



(b)

Figure 4. The degree distribution of the Internet as of 2006 represented at autonomous system level [2], and its clone. The clone was created by the multifractal network generator. The target of the clone was the degree distribution of the Internet with logarithmic binning. In the subpicture (a) the degree distributions of the networks are plotted with logarithmic binning, in the subpicture (b) with on-demand binning. The degree distributions of the Internet was plotted with x marks, the degree distributions of a network generated from the final generator measure with red dots, the degree distributions of a network generated from the initial generator measure with $+$ marks. The degree distributions calculated from the initial and final generator measure was drawn with dotted and dashed lines respectively.

For the *logarithmic binning* we used the intervals

$$[1,2[, \quad [2,4[, \quad ,[4,8[, \quad , \ldots , \quad ,[2^{n-1},2^n[. \qquad (10)$$

With the logarithmic binning we have a very coarse description of the network. In the subfigure 4 (a) we can see, that the logarithmic binned degree agrees well with that of the Internet.

To get a finest description from the networks we developed another bin smearing method, that we call *on-demand binning*. With this method we choose the bins as follows. If the occurrences of three degree values k_i, k_j and k_l are non-zero, but there is no other degree values with non-zero occurrence between k_i and k_j and as well between k_j and k_l, then we used the geometric means $k_{j\,\min} = \sqrt{k_i k_j}$ and $k_{j\,\max} = \sqrt{k_j k_l}$ to define the bin boundaries around the value k_j. Furthermore, we set the degree distribution value $p(k_j)$ to $p^*(k_j)/(k_{j\,\max} - k_{j\,\min})$, where $p^*(k_j)$ is the probability, that a randomly chosen node has k_j degree.

In the subfigure 4 (b) we can see, that the clone of the original network has oscillation in the degree distribution, while the original network (the Internet) not. The points of

the degree distribution of the Internet are near a straight line if we use logarithmic scales on both axes, it has proven to be scale-free.

Perhaps using the on-demand binned degree distribution as target, the clones of the original networks will be closer to the original ones. If we will have such a better generator, it would be worth to investigate, whether the oscillation in the degree distribution of the generated probability measure has any effects on the properties of the network (diameter, average geodesic distance, clustering coefficient) or not.

In our realization of the multifractal network generator, as being written in Python, the generation of a network clone with a given degree distribution is quite slow. The generation of the Internet clone took more than 3 hours. This is partly because the Internet as a complex network has many nodes ($N = 22963$) and the maximal degree is large ($k_{max} = 2390$) so using logarithmic binning we need to calculate the degree distribution of the generator measure from 1 to $2^{12} - 1 = 4095 > k_{max}$ in each step. The speed is also depends on the size of the link probability measure. The generator measure in our runs had $m = 3$ intervals on each axis, and we had used $K = 4$ iterations, so the probability matrix of the link probability measure had $m^K \times m^K = 81 \times 81$ elements. We decreased the temperature from $T_0 = 0.2$ to $T_{lim} = 2 \times 10^{-5}$ multiplying with $f_T = 0.997$ in each double steps so we had 6132 steps. When we changed the division points, we used the exponent $n = 1$ but this has no notable effects on running speed.

The exponent has no notable effects on the final energy as we can see from Tab. I. All of the means and its uncertainties agrees with the arithmetic means of all the final energies 7.55. Perhaps the final energies in the generation with exponent $n = 1$ has larger uncertainties. It would not be surprising because with this exponent the division points are moved with larger amount in average as with larger exponents, so the generation could have fortune or misfortune to find better measure or not, but we have too few sample to conclude.

We used some of the possibilities of the numpy module to gain more speed. Perhaps there are undiscovered possibilities to speed up running in pure Python, but we think, that a significant acceleration would be possible for us, if the calculation of the degree distribution from the generator measure and the iteration of the generator measure would be rewritten in C or C++. In this case we wanted to preserve the Python binding, because the Python is very convenient language to use, and has good bindings to C and C++.

V. Conclusion

We have a method to get close to a given degree distribution. There are far faster methods to create networks with a given degree sequence. I think that the significance of the multifractal network generator method in the possibility, that we can generate networks with more than one given properties. With a appropriate energy function we would be able to generate a network that has a given degree distribution and has a given clustering coefficient or perhaps with the given degree distribution the clustering coefficient of the nodes have a dependency from the degree as in the hierarchical networks [8].

n	$\langle E_f(n) \rangle$	n_n
1	7.670 ± 1.158	4
2	7.035 ± 0.493	5
3	6.995 ± 0.409	2
4	7.637 ± 0.106	4
5	7.596 ± 1.085	3
6	8.125 ± 0.781	3
7	7.730 ± 0.465	5
8	7.640	1

Table I. The final energies of the Internet clone generations ($m = 3, K = 4, T_0 = 0.2, T_{lim} = 2 \times 10^{-5}, f_T = 0.997$, 6132 steps) The energy of the initial generator measure was $E_i = 9.830$. n is the exponent of the Δ function, $\langle E_f(n) \rangle$ is the arithmetic mean of the final energies in the generations where the exponent was n, the value after the $\pm$ sign has been calculated as the arithmetic mean of the $|E_f(n) - \langle E_f(n) \rangle|$ values for all of the final energies $E_f(n)$ coming from generations with the exponent n and n_n is the number of generations with the exponent n.

The multifractal network generator is worth to show to the computer engineer students.

To make the multifractal network generator program usable for real life problems, the running time must be reduced.

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] A. Horváth and Z. Trócsányi, "Multifractal network generator with igraph," *AIS 2010, Symposium on Applied Informatics and Related Areas*, 2010.

[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.

[8] E. Ravasz and A. Barabasi, "Hierarchical organization in complex networks," *PHYSICAL REVIEW E*, vol. 67, no. 026112, FEB 2003.