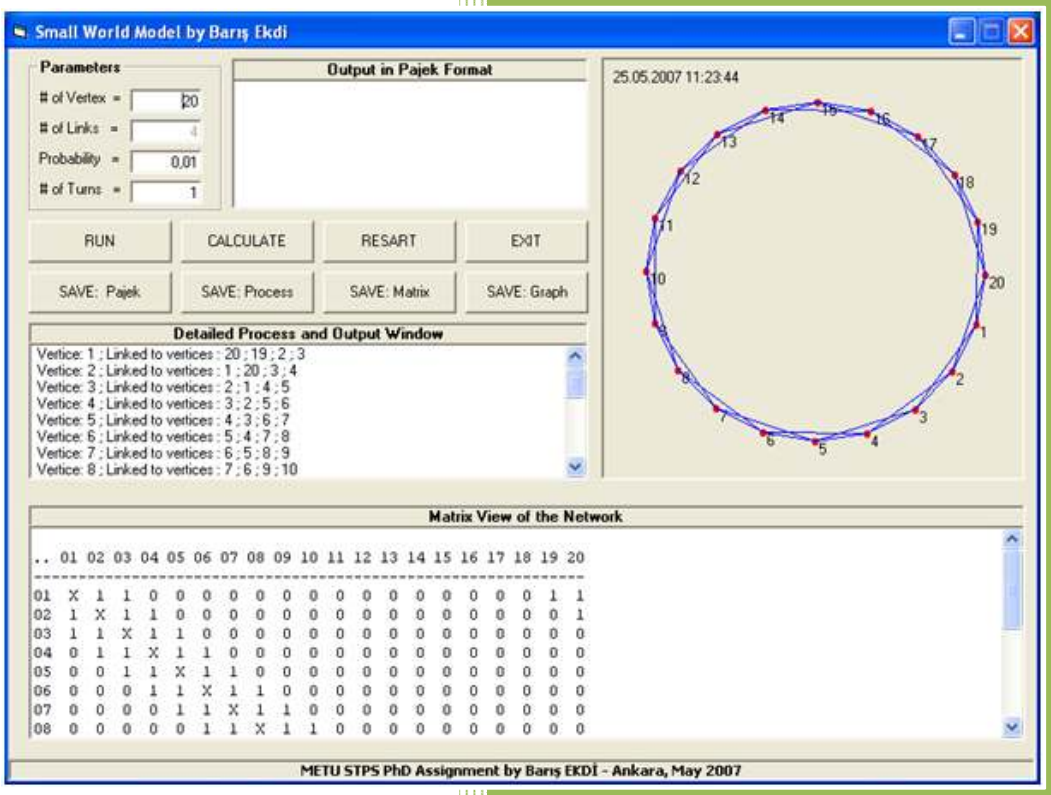
2007

WATTS & STROGATZ SMALL WORLD MODEL



Barış Ekdi

5/15/2007

WATTS & STROGATZ SMALL WORLD MODEL

by

BARIŞ EKDİ

Phd Candidate @ METU - STPS

- **Basics of the Model & the Simulation**
- **Main Features of the Program & the Pseudo-Algorithm**
- **Output of the Simulation and Comments on the Simulation Results**

Ankara, May 2007

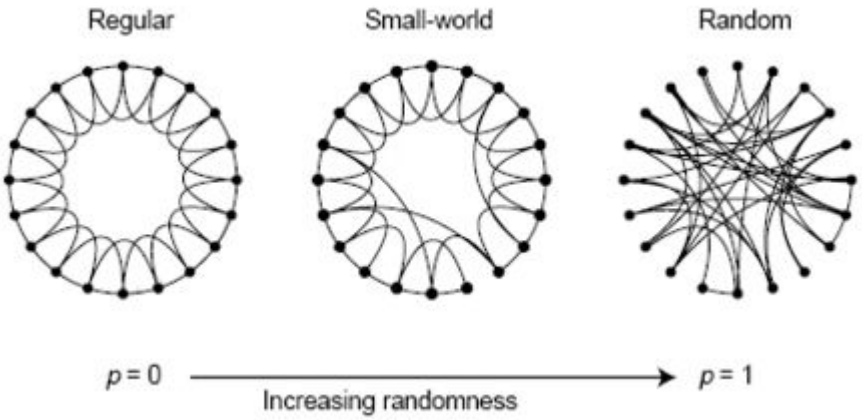
1. THE MODEL: Watts & Strogatz Small World Model

A "*small world*" is described as a network in which closely intra-connected groups are connected with each other through inter-group shortcuts. In ***Collective Dynamics of Small-World Networks*** Watts and Strogatz introduce a small world model with short average path length and high clustering coefficient.

The W&S Model is built by rewiring a regular lattice to create long-range edges or shortcuts in the network. The model takes a single parameter "*p*" and interpolates between a regular lattice and a random network as *p* varies; according to the following algorithm:

1. **Start with order :** We start with a regular ring lattice in which there are N nodes each connected with K of its neighbors (K / 2 on each side).
2. **Randomize :** Now we randomly add long-range connections through the following procedure:
    a. Visit each node.
    b. At a given node, visit each link.
    c. Rewire this connection by moving it to another randomly selected node (uniform probability while avoiding self-connection and link duplication) with probability p.

This process introduces non-lattice edges which are likely to connect nodes that were distant in the original lattice. By varying *p* we can interpolate between a regular lattice ($p = 0$) and a random network ($p = 1$):

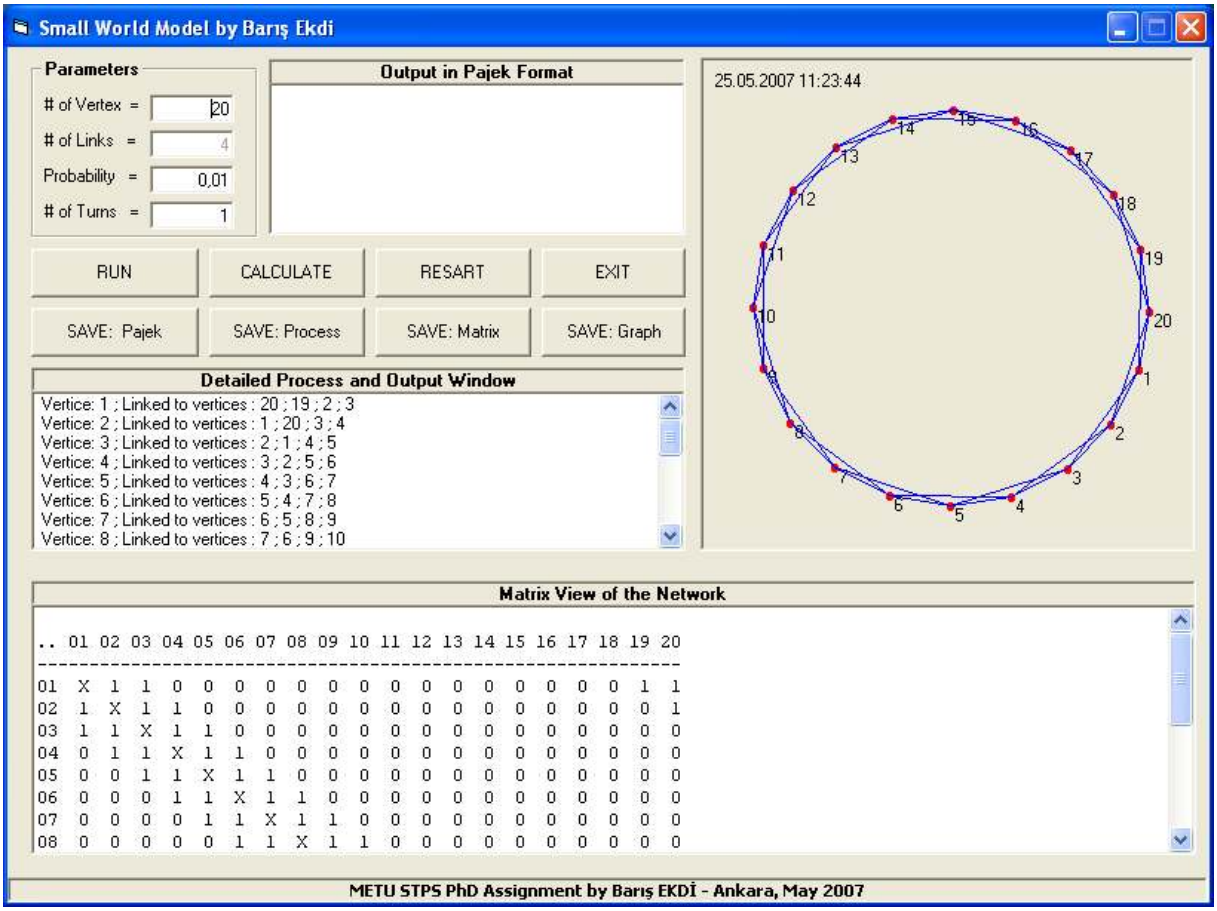


The underlying lattice structure of the WS model produces a locally clustered network, and the long-range links introduced by the randomization procedure dramatically reduce the diameter of the network, even when very few such links are introduced.

2. THE PROGRAM

2.1. Main Features of the Program

The Program is written using Microsoft Visual Basic 6.0. The startup screenshot of the program is as follows:



Parameters Frame: A graphical user interface (GUI) is used for getting the required parameters from the user, in order to run the simulation. However, the user can also use the default parameters (Number of Vertices=20; Probability=0.01; Number of Turns to Run the Simulation = 1) loaded while starting the program. As the program started, a graph is drawn according to those values, and re-drawn whenever the user changes the number of vertices.

- There are also three output windows apart from the one which is used for the graphic.

Output in Pajek Format: This window stays empty until the simulation run. The results of the simulation are shown in Pajek format in order to check the accuracy. So, the user can save the results and use in Pajek.

Detailed Process and Output Window: This window is used in order to inform the user about almost each step of the process the program is executing. Therefore the user can check the calculations and save it for further use.

Matrix View of the Network Window: Used to show the connections between the vertices. If there is a connection between to vertices, (i.e. 3 and 5) the value at intersection of the row 3 and column 5 will be "1" otherwise it will be "0".

- There are eight buttons provided to control the program:

RUN: Gets the parameters and starts the simulation (Explained in detail in the next subsection of this paper: "*How it Works*") and prints the results and processes onto *Output in Pajek Format Window, Detailed Process and Output Window, Matrix View of the Network Window* accordingly.

CALCULATE: Calculates the individual cliquishness of the vertices and average cliquishness of the net. (See "How It Works" for the details.)

RESTART: Initializes the values, variables and clears the windows for another simulation.

EXIT: Exits the program.

SAVE_Pajek: Saves the output in *Output in Pajek Format Window* to drive C:\ in a Pajek readable format. Date and time stamp is added to the filename with the ".NET" extension.

SAVE_Process: Saves the output in *Detailed Process and Output Window* to drive C:\. Date and time stamp is added to the filename with the ".TXT" extension.

SAVE_Matrix: Saves the output in *Matrix View of the Network Window* to drive C:\. Date and time stamp is added to the filename with the ".TXT" extension.

SAVE_Graph: Saves the graphic of the network to drive C:\ in bitmap format. Date and time stamp is added to the filename with the ".BMP" extension.

2.2. How It Works? Pseudo-Algorithm of the Program

A simple pseudo-algorithm is given below in order to explain how the program runs:

1. Load default parameters and draw default graph when the program is started – using the subs <INITALIZE> and <PRINT_MATRIX>
2. Wait for user input. If the user;
   2.1. Changes the number of the vertices redraw the graph and print the new values into relative windows; using the subs <INITALIZE> and <PRINT_MATRIX>
   2.2. Hits RUN button go to sub <SIM_START >
   2.3. Hits CALCULATE button go to sub <CALCULATE_CLIQUISHNESS>
   2.4. Hits RESTART button, clear the windows and graph; go to sub <INITALIZE> for getting the default values and redraw and fill the windows, using subs <PRINTMATRIX>, <DRAW_NODES>, <DRAW_EDGES>
   2.5. Hits EXIT button END the program.
   2.6. Hits SAVE:Pajek button go to sub <SAVE_PAJEKFORMAT>
   2.7. Hits SAVE:Process button go to sub <SAVE_PROCESS>
   2.8. Hits SAVE:Matrix button go to sub <SAVE_MATRIX>
   2.9. Hits SAVE:Graph button go to sub <SAVE_GRAPH>

2.3. Main subprograms:

1. <*Sim_Start* >

1.1. Get / check user defined parameters: the probability (*ProbabSet*); number of vertices (*NofNodes*); number of the turns the simulation will run (*NofTurns*)

1.2. For i=1 to *NofTurns*

For each vertice (*vFocused*) from no.1 to the last one (*NofNodes*) (in clockwise turn);

- Set a random probability: 0<ProbabRnd<1
- Compare it with user's parameter

- If ProbabRnd < Probabset Then

  Break the Existing link of the node focused;

  Get a random node number;

  *{NewNode = Int(Rnd * (NofNodes - 1)) + 1}*

  Connect the node focused with this *NewNode*;

End If

Next vertice

Next i

1.3. Print the results to relevant windows and draw the graph.

2. <Calculate_Cliquishness>

2.1. *Calculate_NumberofNeighbours* :

2.1.1. For each of the node (*i=1 to Nofnodes*) ;

2.1.1.1. count the number of the neighbors of the node

{ Node(i).NofNeighbours }

2.1.1.2. and record the names of each of the neighbors (j) to the property of the node: *{ Node(i).Neighbours(j) }*

2.2. Calculate the cliquishness of the neighbors of each node:

2.2.1. For each of the node (*i =1 to NofNodes*) calculate the maximum edges potentially available for the node and cliquishness of the node as follows:

2.2.1.1. Get the number and names of the neighbors of the node

2.2.1.2. For each of those neighbors

- Check whether the first neighbor of Node(i) [NodeB now] is also neighbor to Node(i)'s second neighbor...
- If so, Clique = Clique + 1

2.2.1.3. For each of the node (*i =1 to NofNodes*) calculate the maximum edges potentially available for the node…

*{MaxEdges = Node(i).NofNeighbours * (Node(i).NofNeighbours - 1) / 2}*

2.2.1.4. if MaxEdges>0 then calculate cliquishness of Node(i)

{ Node(i).Cliquisness = Clique / MaxEdges }

2.3. Calculate average cliquishness of the network

2.3.1. Sum the cliquishness values of the each of the nodes (i=1 to NofNodes)

{For i = 1 To NofNodes

Total_Cliquishness = Total_Cliquishness + Node(i).Cliquisness

Next i }

2.3.2. Divide it by number of total nodes in the net.

{ Av_Cliquishness = CDec(Total_Cliquishness / NofNodes) }

3. THE OUTPUT

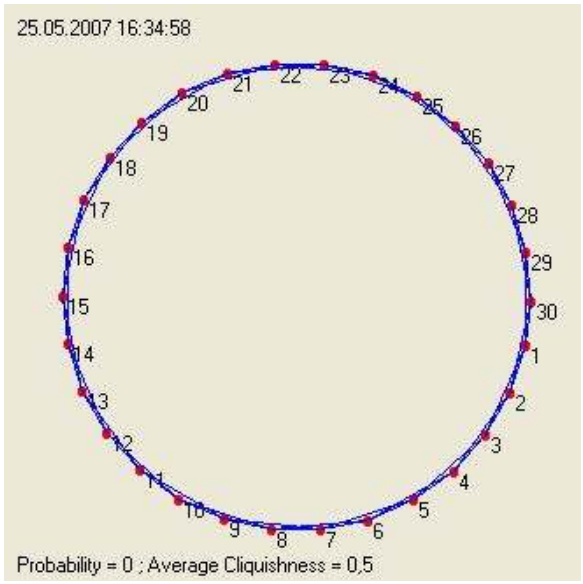
The Output

In order to test the model and the accuracy of the program, the program is run 14 times with;

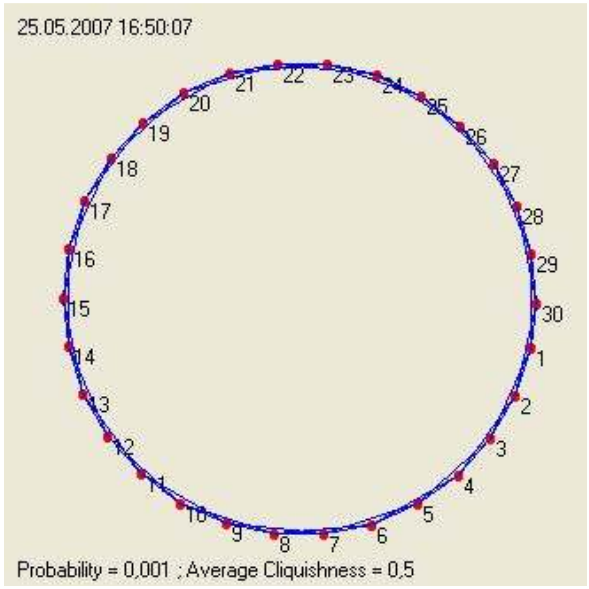
- 30 nodes (vertices) and
- 4 initial (closest – 2 left and 2 right) links for each node
- For 1 turns at each time (for each value of p)
- With the parameters p={0.001, 0.009, 0.01, 0.05, 0.09, 0.1, 0.3, 0.7, 0.9}.

Then the results are also viewed in Pajek in order to calculate the average path length.

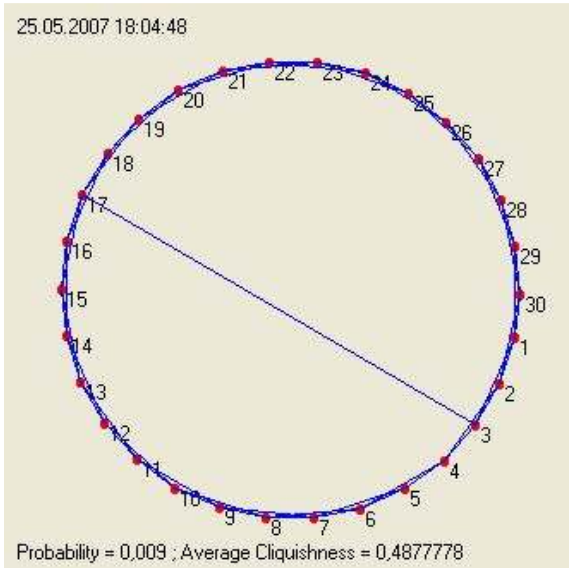
The outputs of those simulations are as follows:



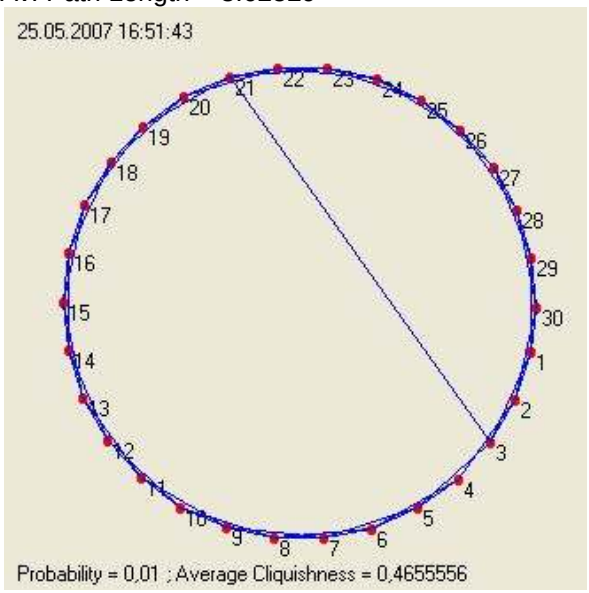
Av. Path Length = 4.13793



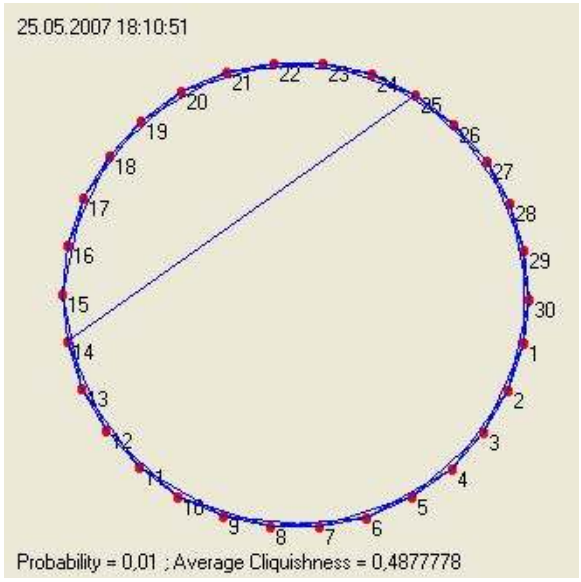
Av. Path Length = 3.62529



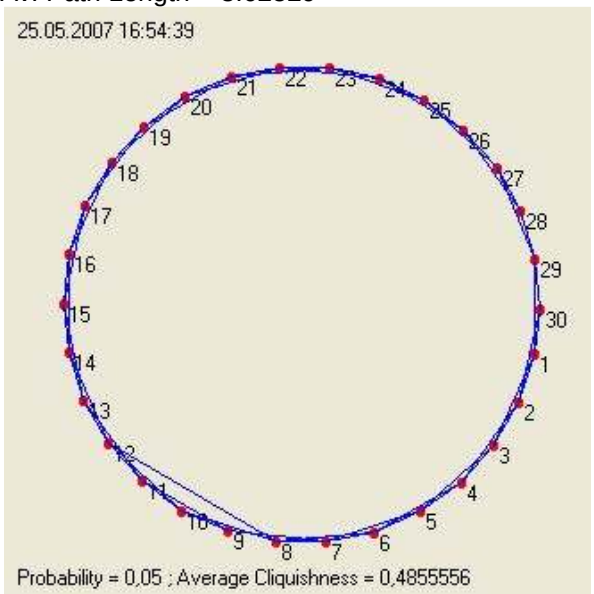
Av. Path Length = 3.64828



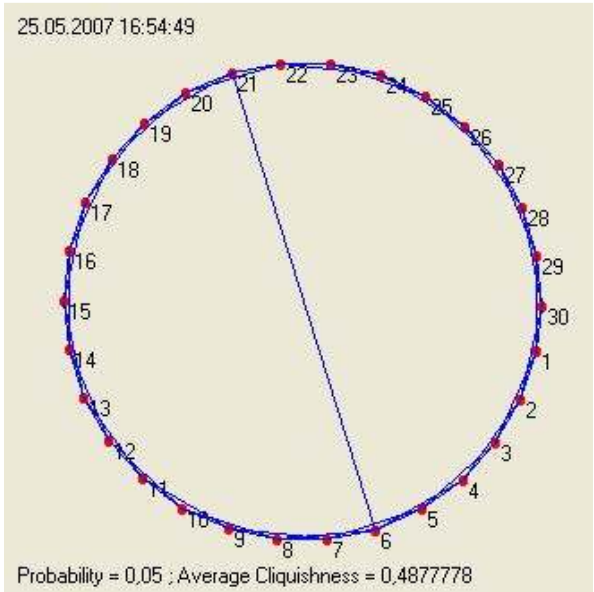
Av. Path Length = 3.62529



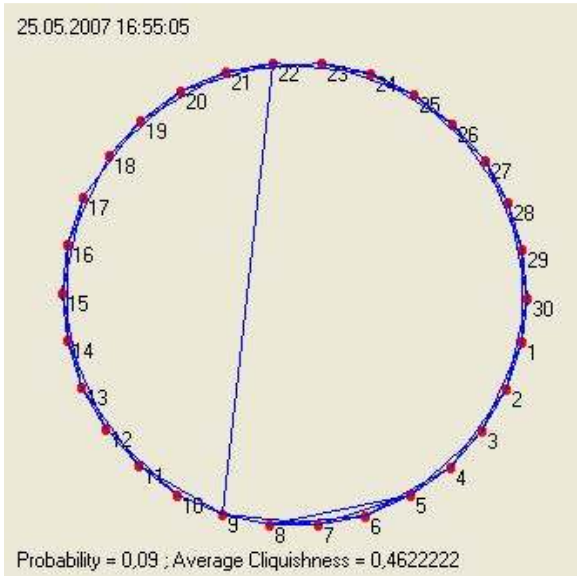
Av. Path Length = 3.68966



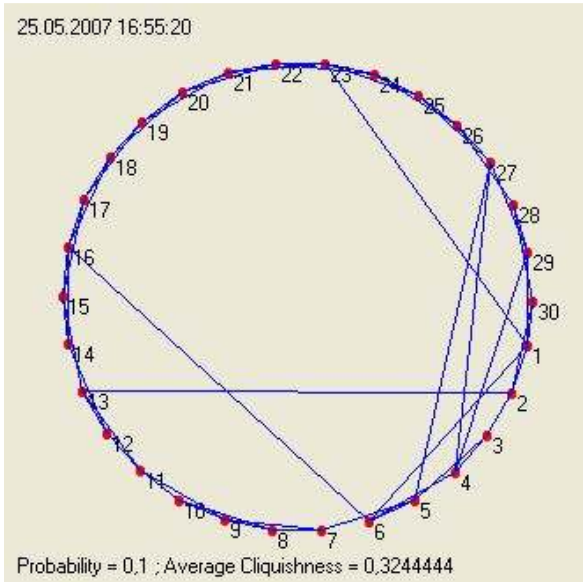
Av. Path Length = 4.05057



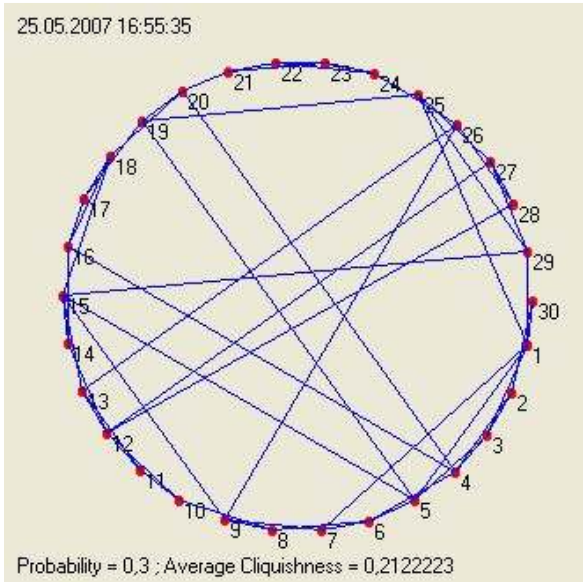
Av. Path Length = 3.65517



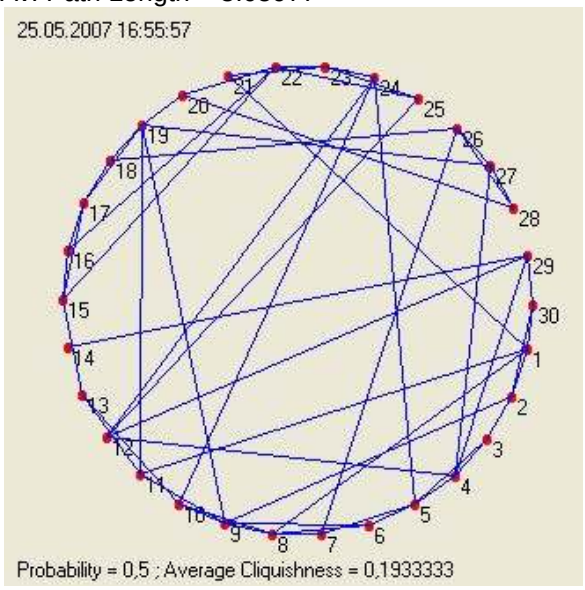
Av. Path Length = 3.65517



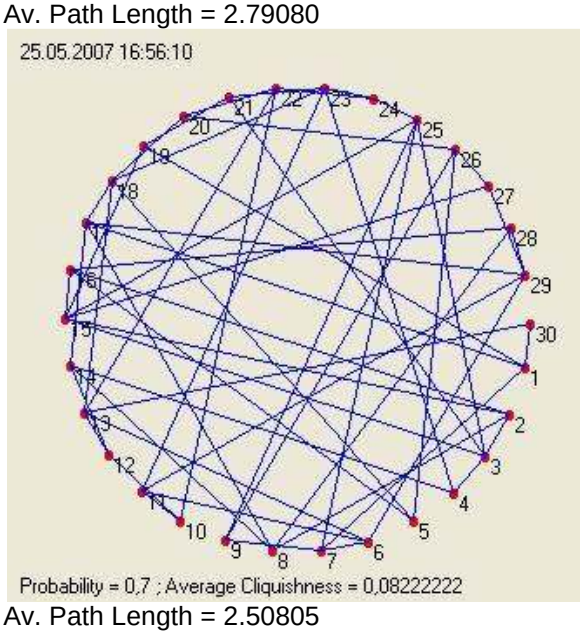
Av. Path Length = 3.05977



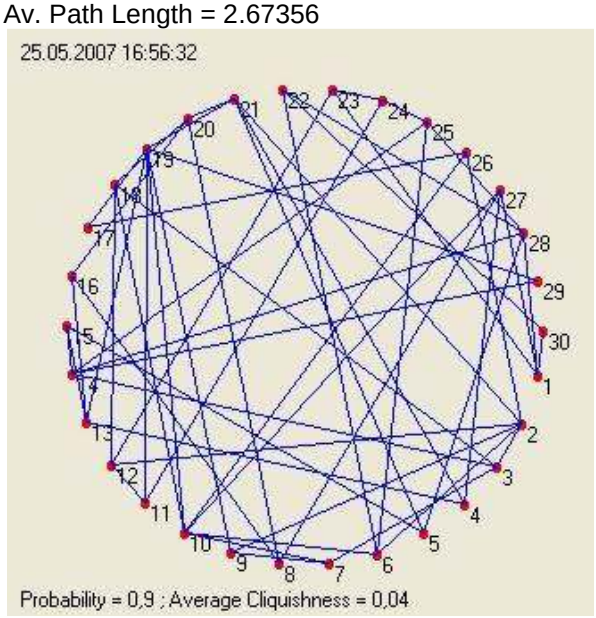
Av. Path Length = 2.79080



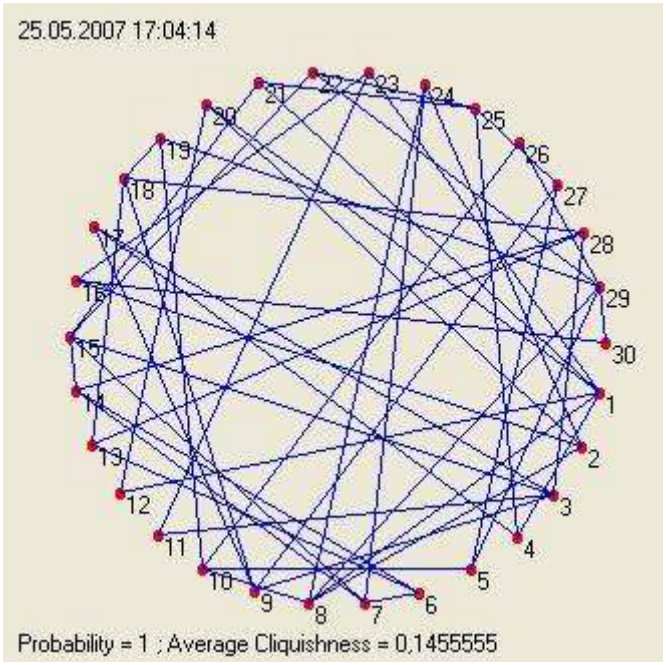
Av. Path Length = 2.67356



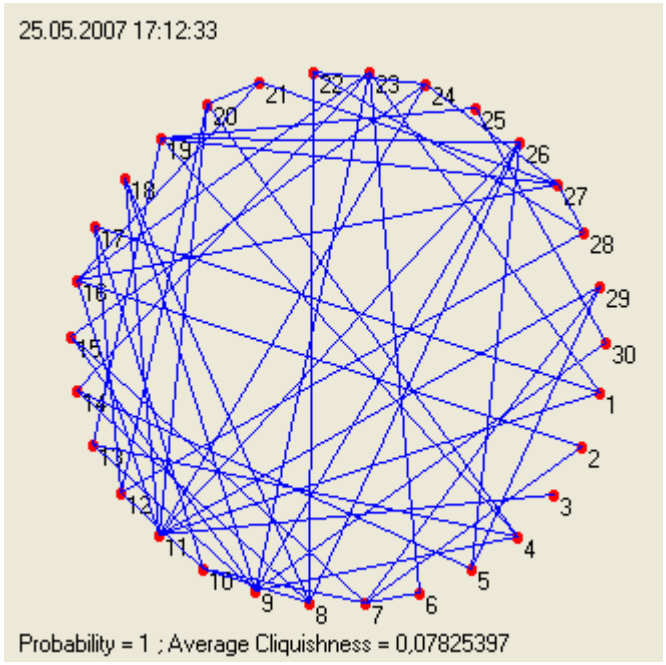
Av. Path Length = 2.50805



Av. Path Length = 2.54253



Av. Path Length = 2.69195



Av. Path Length = 2.50115

The results of the 14 simulations that run suggest that as the parameter "p" increases; (i) the average cliquishness –so that the clustering - and (ii) the average path length in the network decreases. In addition, parallel to the increase of parameter "p", the regular network becomes "small world" at the first instance and then turns into a random one:

Probability (*p*)	Average PL	Cliquishness	Network Type
0	4.13793	0.5000000	Regular
0.001	3.62529	0.5000000	Regular
0.009	3.64828	0.4877778	Small World
0.01	3.62529	0.4655556	Small World
0.01	3.68966	0.4877778	Small World
0.05	4.05057	0.4855556	Small World
0.05	3.65517	0.4877778	Small World
0.09	3.65517	0.4622222	Small World
0.1	3.05977	0.3244444	Small World → Random
0.3	2.79080	0.2122223	Small World → Random
0.5	2.67356	0.1933333	Random
0.7	2.50805	0.8222222	Random
0.9	2.54253	0.0400000	Random
1	2.69195	0.1455555	Random
1	2.50115	0.0782537	Random

-o0 THE END 0o-