Network Visualizer

Kojin Oshiba, Roman Sigalov

May 3, 2016

1 Motivation

In cs136, we learned about different network models that involve and donâĂŹt involve rational agents. The most interesting part of these models is that each network is formed differently depending on different parameters that we set, e.g. the number of nodes, the probability of forming edges, etc. However, with so many different parameters to be defined, it is hard for individuals to imagine how the networks will look like, especially when there are numerous number of nodes. Thus, we have decided to create a website that would visualise these different types of networks so that people can play around with different parameters visually and intuitively learn about the interesting features of different types of networks with different parameters.

2 Project Guideline

Our project is split into three parts:

1. Python code to create different models with different parameters and output a network graph.
2. A website (frontend: HTML/CSS/JS, backend: Python) that displays different networks depending on different models and parameters users select.
3. Analysis of each model with different parameters using the website to verify that our implementation reflect the common characteristics of different models.

In this report, chapter 3 covers the first two, and chapter 4 covers the last analysis part.

3 Implementation

3.1 Python Network Generator

Generation of random graph models was straightforward and involved the same steps, described in the textbook. We implemented 3 different random network models: Erdos-Renyi, Preferrential Attachement and Copying model.

The most interesting part is the network formation games. In the class we covered three different type of such games: Stay Connected, Direct Reachability and Bilateral Connection. We implemented the first and the last one since the second one is not interesting from the point view of out method of implementation.

In the class we considered Nash Euilibrium of such networks for different parameters. We decided not to solve for NE for several reasons. First, it is generally intractible. Second, real agents do not behave according to Nash equilibrium. Hence, instead, we implemented a greedy algorithm that build network step by step:

1) The random connected graph is formed in the beginning using Watts and Strogatz model with small number of nodes.

2) Nodes arrive one after another and decide whether to form a connection the following way:
   i. All existing nodes are sorted by utility of initial connection in descending order.
   ii. In stay connected-game new agent connect to the node that will give him the maximum utility. In Bilateral-Connection game new agent will propose to form an edge to the node that will give the highest utility and if there is a mutual agreement then an edge is formed.

3) New agents loop through the remaining nodes in the ranking and form a connection only if the marginal utility of additional connection will be positive. For Bilateral Connection game new agent keeps proposing to nodes that will have positive marginal utility of connecting to.

This solution is not going to be a NE, however, for some parameters it can be close (about different parameters can be seen in Analysis part). Therefore, we make an attempt to get game network closer to the NE by using optimization of behavior of existing agents which is done the following way. All agents in random order are allowed to reconsider all of their connection (in random order) and if it will be beneficial to drop an edge (marginal utility of dropping an edge is positive). Using such repeated best response optimization we can get each network closer to the NE one and you will see that in some cases it actually becomes a NE network (more on this in Analysis section)

3.2 Network Visualizer Website

Our website has three pages; home, about and visualizer.

1. Home

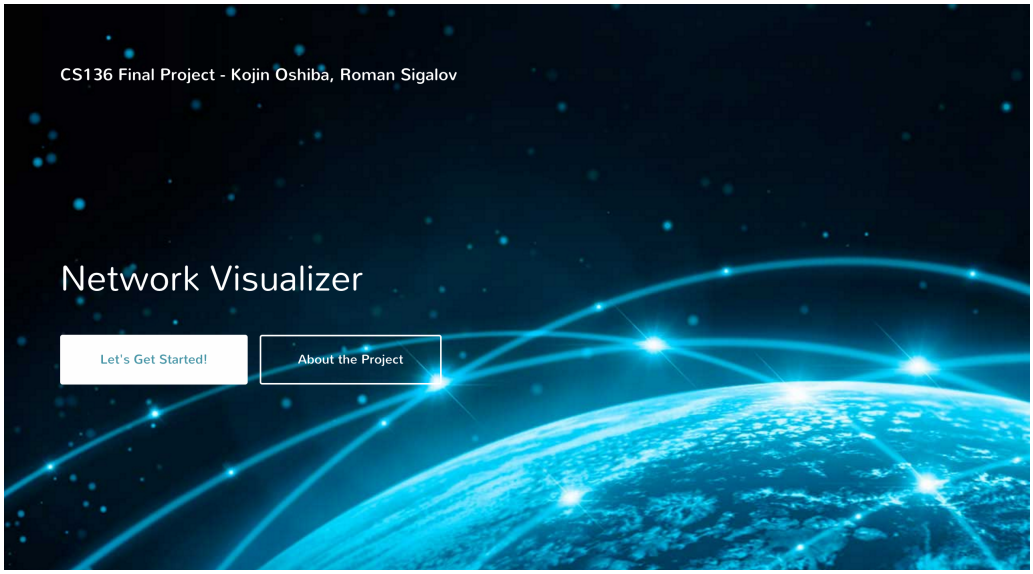


Figure 3.1: Home

There are not many technical details to it, but we did spend decent amount of time making the fonts look nicer, make the buttons and the title fade in when the page is loaded, and so on.

2. About

This is just a page with our project report.

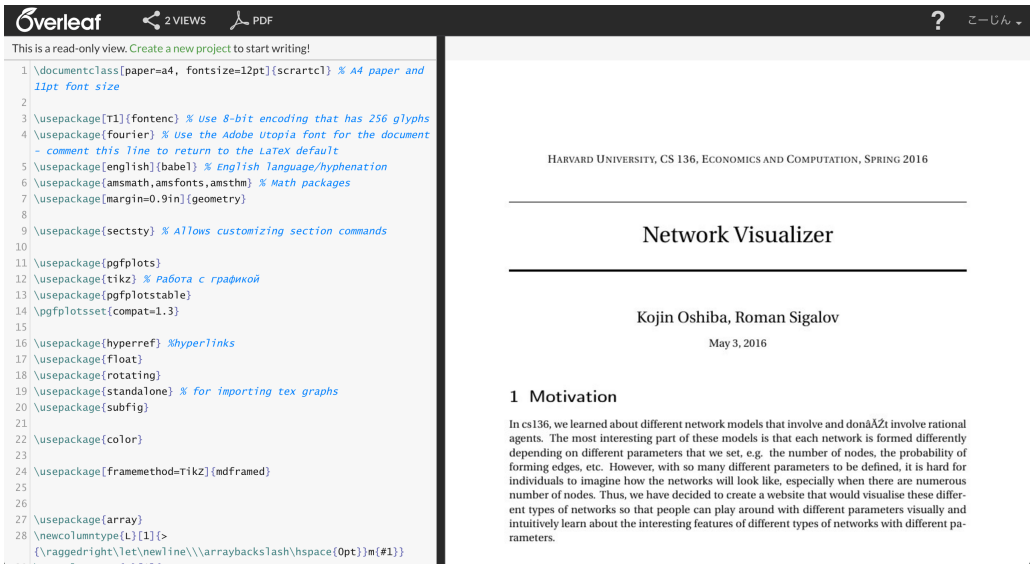


Figure 3.2: The report is uploaded on overleaf.

3. Visualizer

   In this page, users first select the type of model they want to visualize. Then, they set the parameters that are associated with the model, and the corresponding network appears on the right side. In addition, we will also display relevant information to the model: the degree distribution and clustering coefficient.

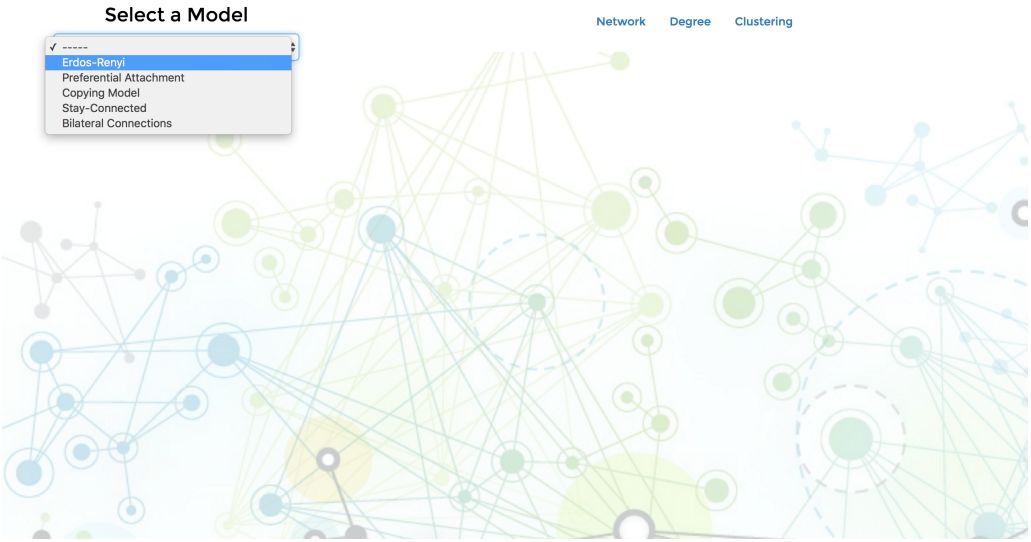


Figure 3.3: User first selects a model.

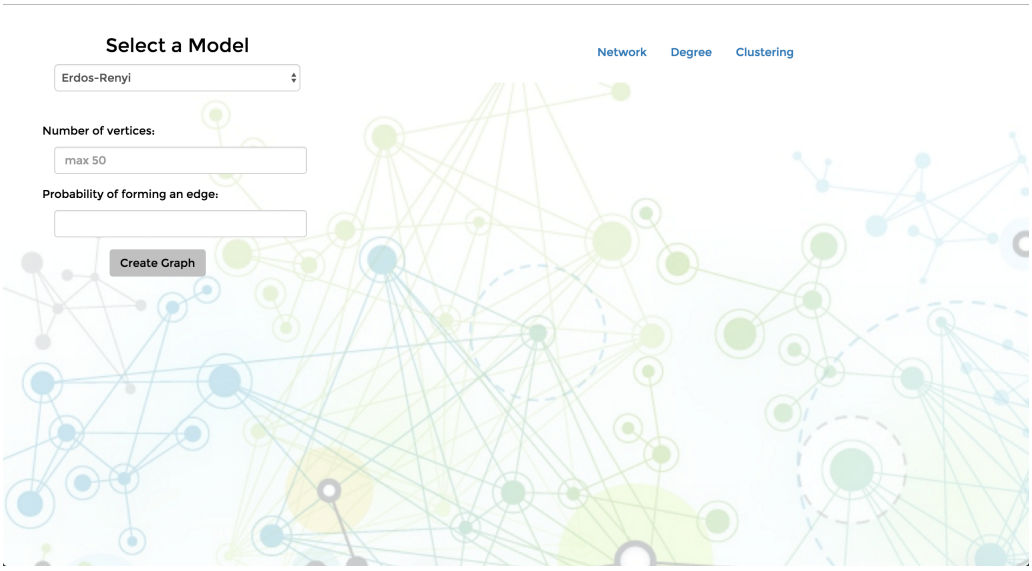


Figure 3.4: Based on the model chosen, parameters that need to be specified will pop up.

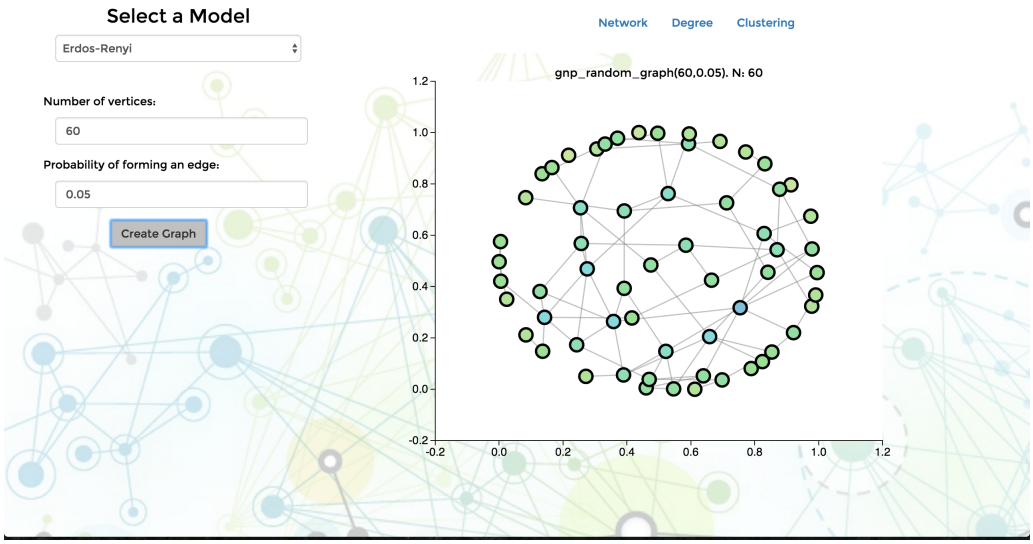


Figure 3.5: "Create Graph" will display the network.

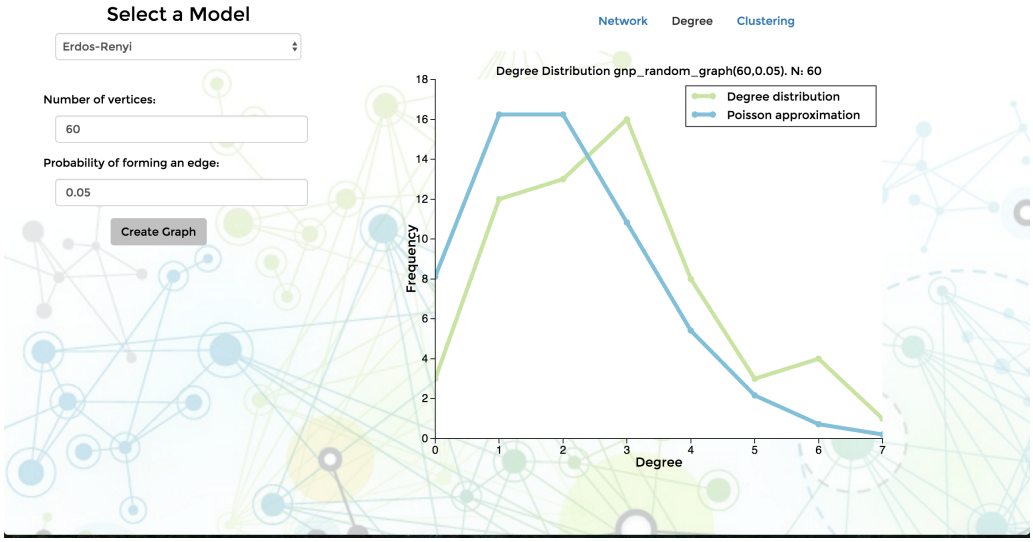


Figure 3.6: Users can also view the degree distribution of the network generated.

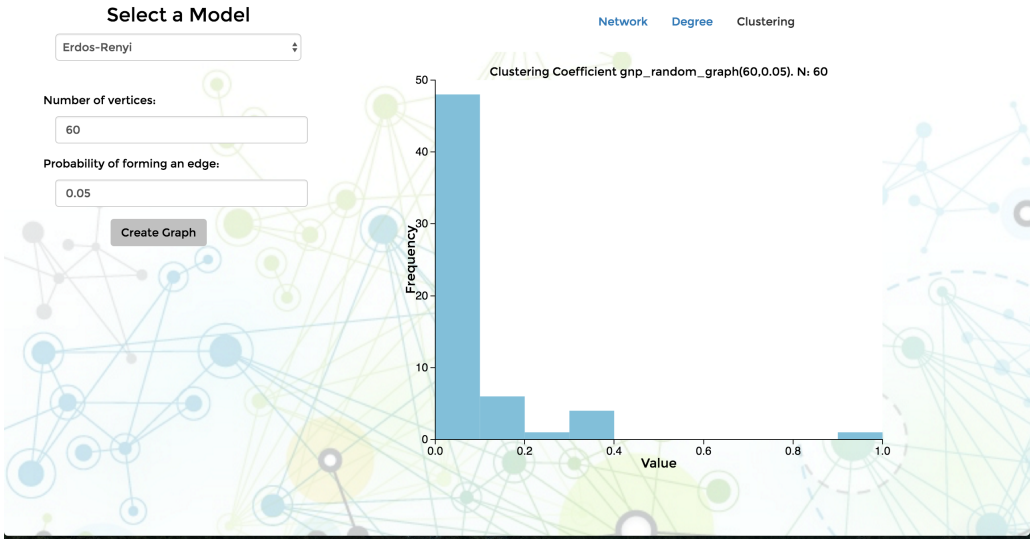


Figure 3.7: Users can also view the clustering coefficient distribution of the network generated.

This is the key page of our website. There are several functions that we implemented here:

- A jquery function which displays a form depending on the model selected.
- A JSON/ajax function to display the created network on the same page as the form without reloading the page.
- Buttons to show the network, the degree distribution and the clustering coefficient distribution on the same position of the page.
- Setting max and min for each parameter so that users won't violate the requirements (e.g. probabilities being between 0 and 1, number of vertices being less than 30 or 50 and not something too big like a million).

3.3 Running Site on Local Machine

The following steps will guide you through running our site (which, unfortunately, we didn't have time to deploy. Sorry for you inconvenience. We will deploy it after the finals...) on local machine. In order to run it you need:

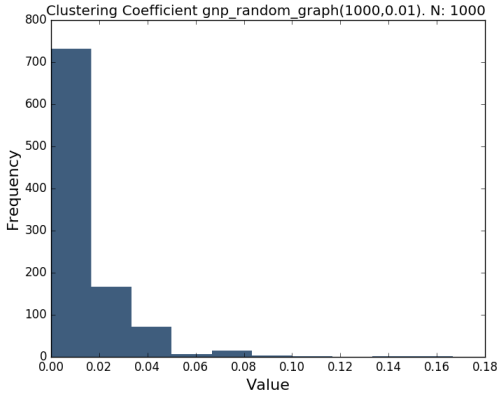
1. Have Python 2.7 installed.
2. Install flask.
3. Install the following Python libraries: **numpy, pandas, networkx, random, operator, math, collections, scipy, mpld3, matplotlib, colour** using **pip install <library>**
4. Run using command **python visualizer.py**

4 Analysis

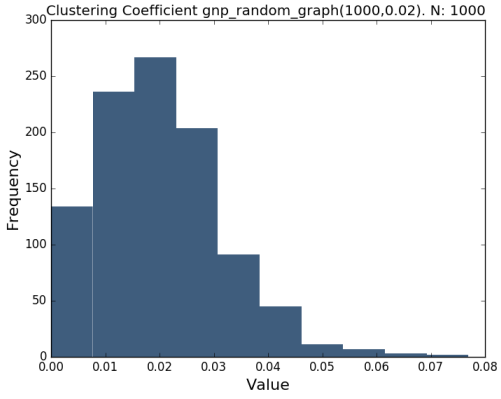
4.1 Random Network Models

From the Problem Set 9 we know some properties of Erdos-Renyi and Prefferential Attachement random models. Therefore, we are not going to repeat it here, but we are going to show additional characteristic: clustering coefficient. On the Figure 4.1 you can see distribution of clustering coefficient for Erdos-Renyi random graph with 1000 nodes and different probabilities of edge occurrence.

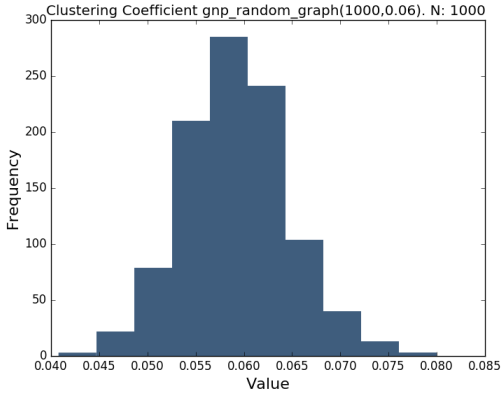
Figure 4.1: Clustering coefficient distribution for Erdos-Renyi Model on 1000 nodes



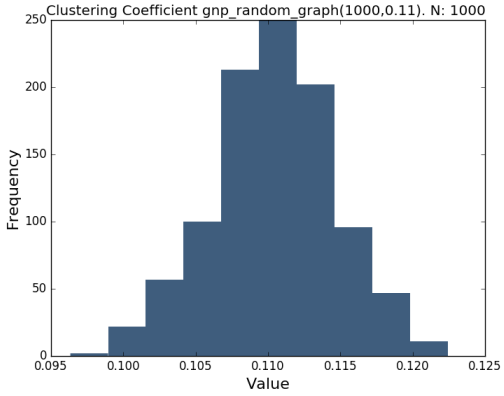
(a) p = 0.01



(b) Cost = 0.02



(c) Cost = 0.06



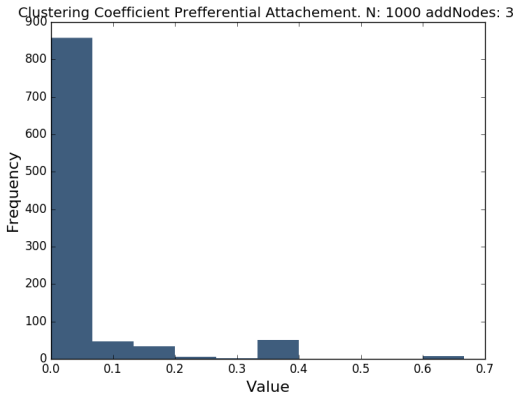
(d) Cost = 0.11

For low p distribution is highly right skewed because there is a small probability of occurrence of a triangle in a graph and therefore clustering coefficient is small. As probability increases the distribution reminds a bell curve with mean at the probability of edge and light tails which provides support to the fact this model cannot generate high clusterization.

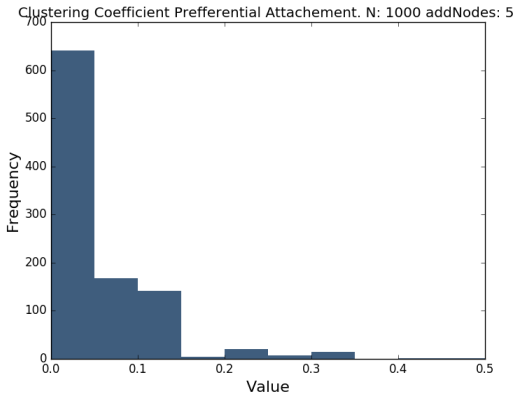
On Figure 4.2you can see distribution of clustering coefficient for Preferential Attachment model and different number of connections on each step. The general picture is similar to that of Erdos-Renyi. Though, we can see heavy tails which are most likely correspond to high-degree edges which are a property of a PA model. However, the curve becomes more like bell curve as number increases.

Finally we are going to analyze the Copying model. On the Figure 4.3 you can see degree distribution for the following four sets of parameters:

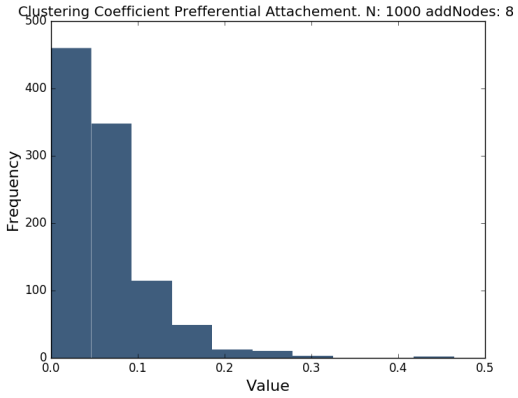
Figure 4.2: Clustering coefficient distribution for Prefferential Attachement Model on 1000 nodes with different number of nodes added at each step.



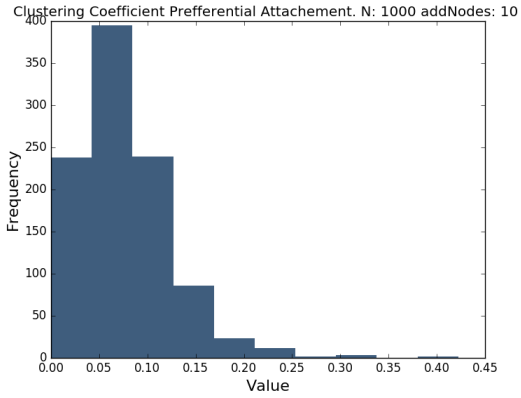
(a) m = 3



(b) m = 5



(c) m = 8



(d) m = 10

	(1)	(2)	(3)	(4)
NSet	24	8	16	8
NNeighbor	8	16	16	24
PSet	0.1	0.05	0.1	0.05
PNeighbor	0.1	0.15	0.1	0.15

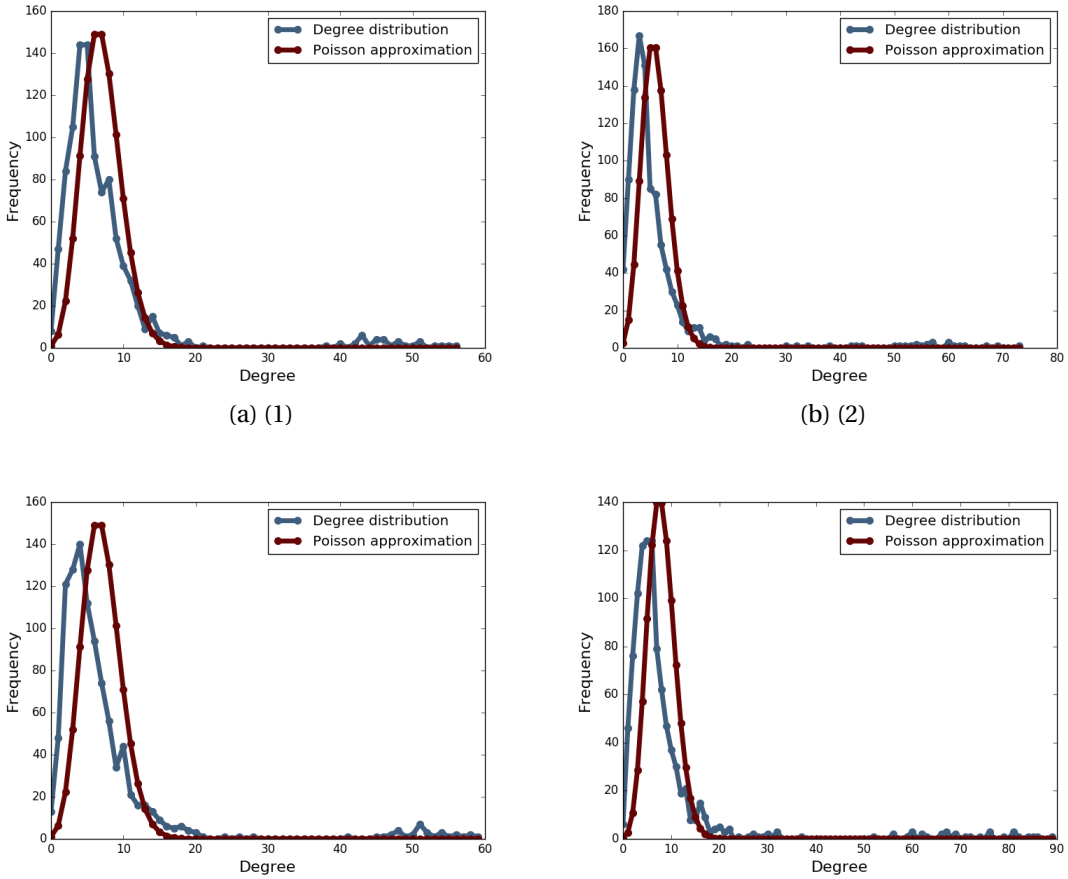
All of these distributions have the same common property, they are skewed to the left compared to Poisson approximation and they have a lot of point of the right tail. Unlike in Prefferential attachement model where these nodes occur randomly on right tails here we have concentration of high degree nodes which point to clusters.

Clustering coefficient distribution for the model above is presented on Figure 4.4. Here we again can see that some nodes have unusually high clustering coefficients compared to the general population. We can see the presence of clusters in this type of model which wasn't the case in tow previous ones: ER and PA.

4.2 Network Formation Games

Stay Connected. There is only one degree of freedom in this model and it is the cost to form an edge. Hence it was a variable that we varied in our analysis. On Figure 4.5 you see generated networks for different cost of forming an edge.

Figure 4.3: Degree distribution for Copying mode Model
on 1000 nodes with different parameters



(a) (1) (b) (2)

(c) (3) (d) (4)

As can be seen the network is gradually converging to Nash Equilibrium which is in case of $C > 1$ is a star graph as the cost increases. When cost is close to 1 the network is highly connected and barely remind a star graph, though, we can see an edge with a high degree and many peripheral nodes with low degree. If cost equals 2 it is easy to see a hub which has a very high degree. When cost equals to 4 the graph is almost a star graph and there are several anomalies caused by initial random graph generation.

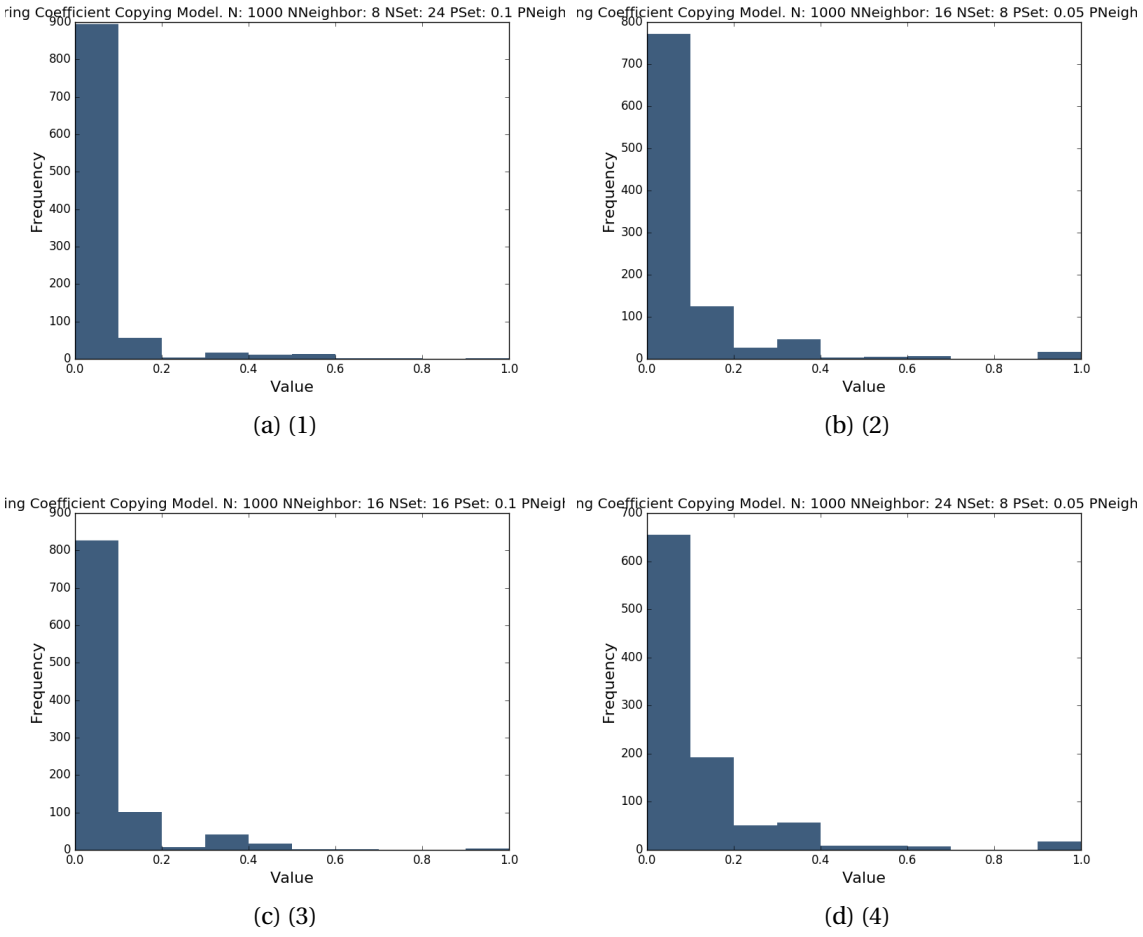
Next we need to do optimization of this graph. We ran 5 iteration of a process described in section 3.1 which gives a visible movement towards NE. All of these networks after optimization are presented in Figure 4.6.

After optimization the network with the lowest cost barely changed but we still can see that the "central hub" became more distinguished from other nodes and has now even higher degree. The most interesting case happened with the second network with cost 2. There were previously 1 big hub and 3 small ones. After optimization 2 small hubs disappeared completely and there are only one big and one small hub. If we allow for more iterations of optimization this small hub would have eventually disappeared. Lastly, network with the highest cost became a Nash Equilibrium star network.

The main conclusion is that even greedy generation of network and subsequent optimization is able to produce a graph which is close to the Nash Equilibrium graph.

Bilateral Connection. In Figures 4.7 and 4.8 you can find results of greedy generation of bi-

Figure 4.4: Clustering coefficient distribution for Copying mode Model
on 1000 nodes with different parameters



(a) (1) (b) (2)

(c) (3) (d) (4)

lateral connection game before and after optimization of 5 iterations, respectively. Parameters correspond to the following grid:

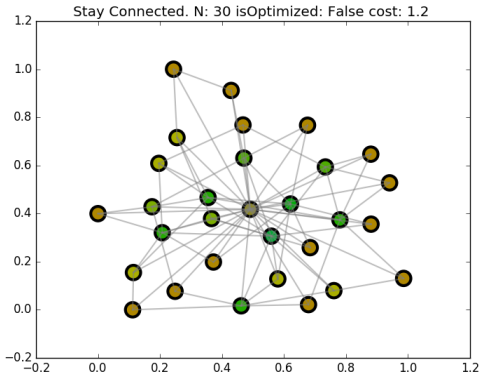
	cost			
delta	0.2	0.4	0.8	1.2
0.8	(1)	(5)	(9)	(13)
0.7	(2)	(6)	(10)	(14)
0.6	(3)	(7)	(11)	(15)
0.5	(4)	(8)	(12)	(16)

Before optimization networks (2), (3) and (4) form almost complete graphs. During optimization agents decide that it will be beneficial to add all possible edges due to low costs.

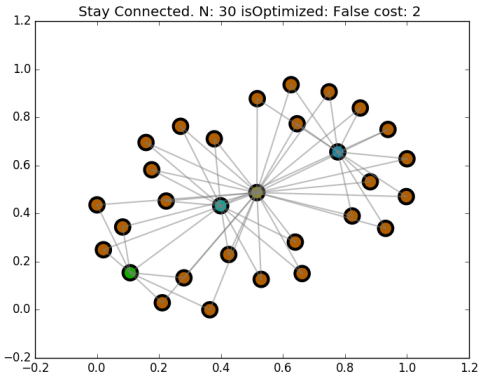
The second column doesn't change after optimization process. It means that each player's action is the best response to other players' actions. But we know that the strategy profile which consists of best responses is a NE. Therefore in second column we have different NE (actually only two types).

In thirds column in network (9) agents have similar degrees and after optimization they decide to add much more edges between them, however, the graph doesn't become complete, probably we need more iterations of optimization to achieve this result. Networks (10) - (16) after optimization become a graph without edges: with high discounting agents prefer not to connect with different agents.

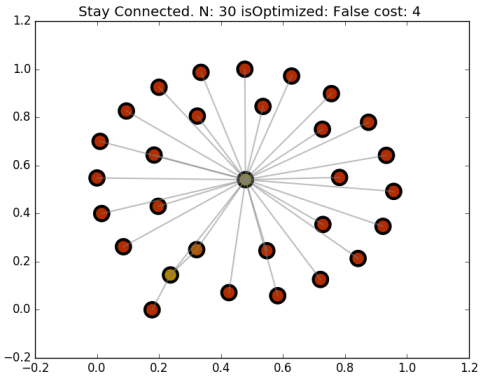
Figure 4.5: Stay Connected greedy generation for 30 nodes. Not optimized.



(a) Cost = 1.2

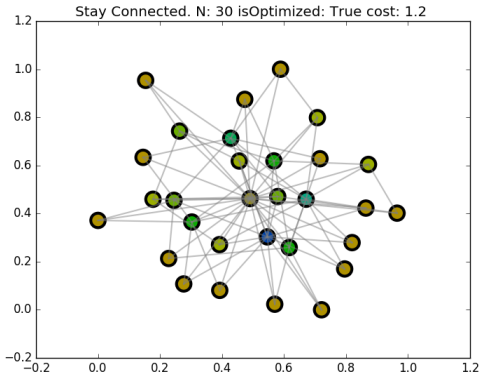


(b) Cost = 2

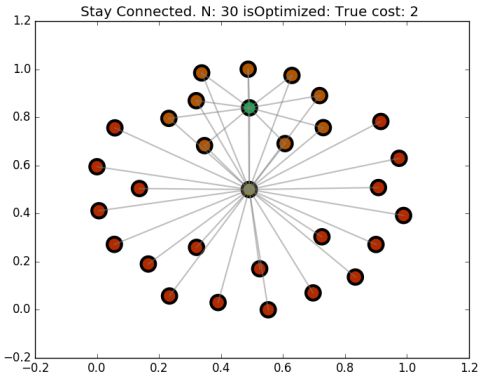


(c) Cost = 4

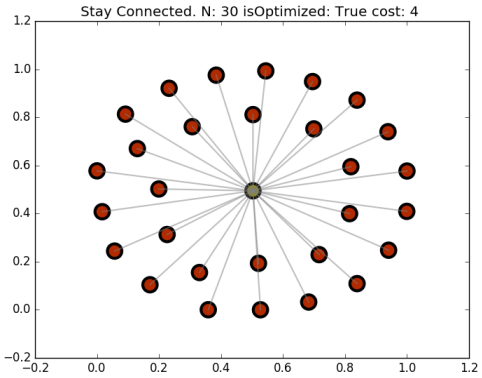
Figure 4.6: Stay Connected greedy generation for 30 nodes. Optimized.



(a) Cost = 1.2



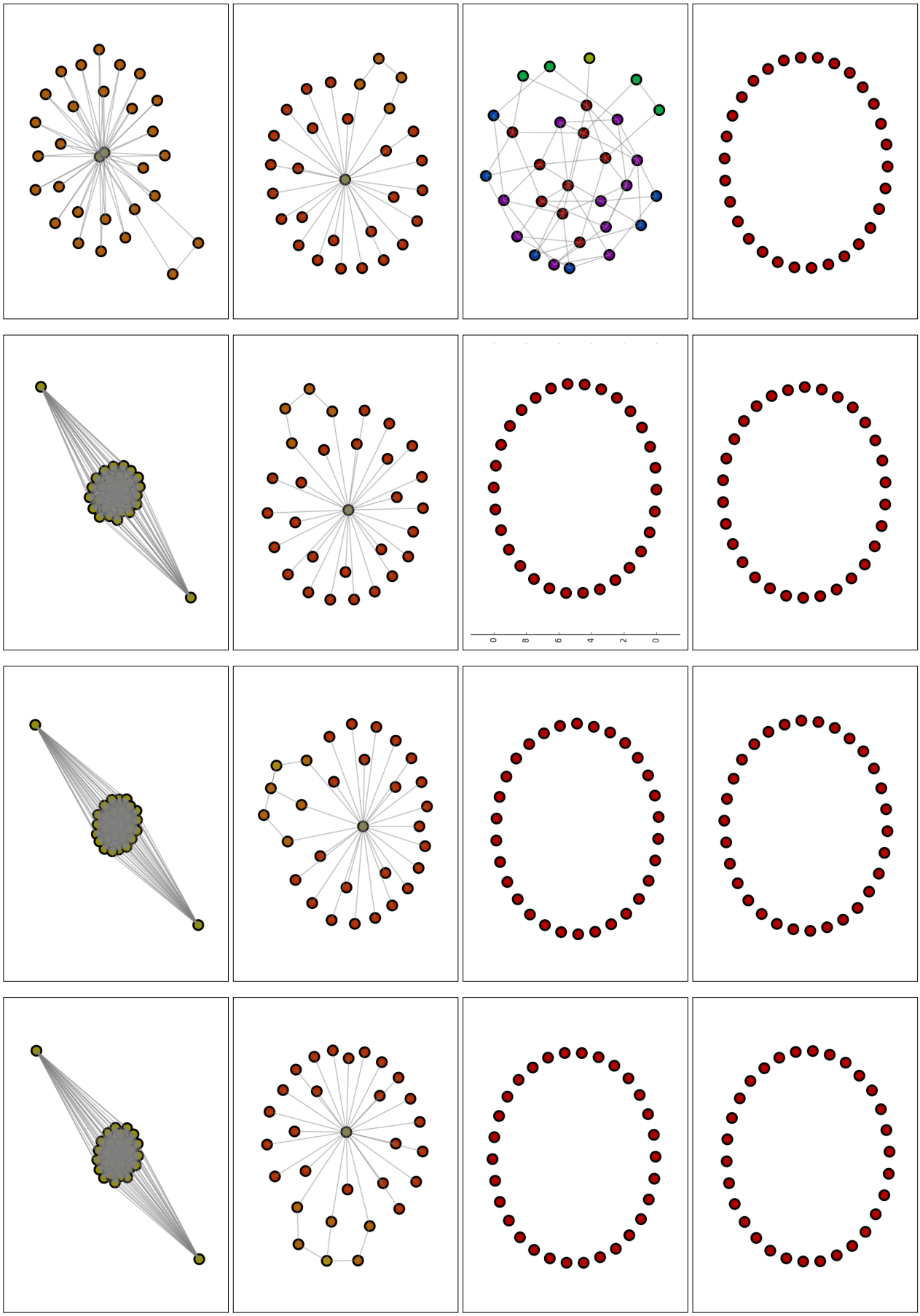
(b) Cost = 2



(c) Cost = 4

Figure 4.7: Bilateral Connection before Optimization

Figure 4.8: Bilateral Connection after Optimization



5 Reflection on Our Work

Overall, we are satisfied with the fact that we implemented everything specified in the proposal as "must-do". Although it is not highly relevant to the material covered in class, all the frontend work that we've done also allowed us to learn quite a lot about visualization, especially since we've hardly used HTML/CSS/JS before. We analyzed all networks covered in class and now have much better understanding of how different parameters influence the model. We introduced greedy algorithm and found that it can be a good approximation for Nash Equilibrium and, for some parameters, we can easily optimize the network to correspond to a Nash Equilibrium. However, we couldn't implement things specified as "ambitious", e.g. several other models that we didn't learn in class and finding real world examples for each of the model. Moreover, all the models that we've implemented here can visualize up to networks with 50-100 vertices which is a bit low to learn features about real world networks with a billion vertices. In general, we have learned how long it takes even for the greedy algorithm to run as the number of vertices increases. One last thing that we regret we couldn't achieve in time is the deployment of the website. Although we did devote quite a lot of time on searching around how to deploy flask app to heroku and AWS, we couldn't manage to make successfully do so.