CS414 Homework 2 Report

Uğur ÖZTUNÇ 28176

Colab Notebook Link

https://drive.google.com/file/d/15G3S8kwVfnFD99rm57DBWHYddYl1kn7P/view?usp=sharing

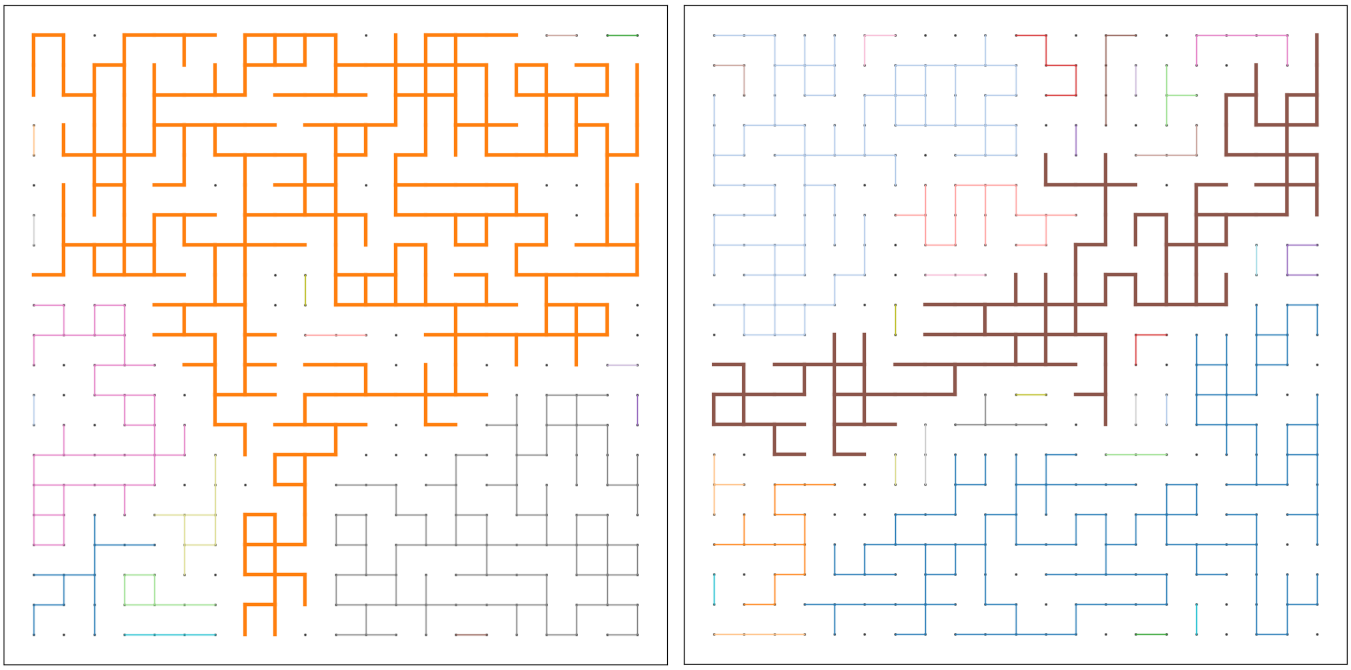
1) Fundamental Functions:

a) create_bonds(): For creating bonds, I have basically iterated all edges by using double for loop with N value and create a bond if a random number in range [0,1] is smaller than connection probability. The format of nodes, bonds and clusters are explained detailly in notebook.

b) find_clusters(): In order to find which nodes are connected, I got help from networkX. I have created a graph and defined all bonds to that graph. After putting all bonds, I have used nx.connected_components() function to clusterize the nodes. After clusterizing, the clusters are returned after beign sorted by their size.

c) check_percolation(): Since the clusters list is sorted by cluster sizes, I used a method which starts looking the clusters list reversed, in order to save time in case of big N values. Logic of 'looking' is like that: if it comes across a node that is at the left or right edge in a cluster, function looks for the other. Likewise, if it comes across a node that is at the top or bottom of the lattice, it looks for the other. If it founds, returns true; if these conditions are not met in any set, it returns false.

Here is a visualizations of two percolating lattice:



2) Experiments Part:

In order to see different statistics for different p values while NxN is varying, I have defined a couple of functions which utilize previous ones:

i) hasPercolatingCluster_and_AverageFiniteSize():

This function is a modified version of check_percolation(), which also calculates average finite cluster size. It returns both average finite cluster size and isPercolating boolean.

ii) simulate_for_NxN():

This function performs simulations for a square lattice of size N using different p values. It takes the number of simulations (num_simulations) and a list of p values (p_values) as inputs. During each simulation, bonds are created based on the given dimensions and p value. Clusters are then identified using the find_clusters function. The function determines if a percolating cluster exists and calculates the average finite size. The function collects the frequency of percolation, percolation probabilities, average finite sizes, and average finite size percentages for each p value. These values are stored in separate lists and returned at the end of the function.

iii) simulateExperiment():

This function basically runs simulate_for_NxN() function for different N values and stores the statistics that comes from that function. It also calculates mean and standart deviation of percolation probabilities of different NxN lattices. At the end, it returns: percolation probabilities, mean, std, and average finite cluster sizes.

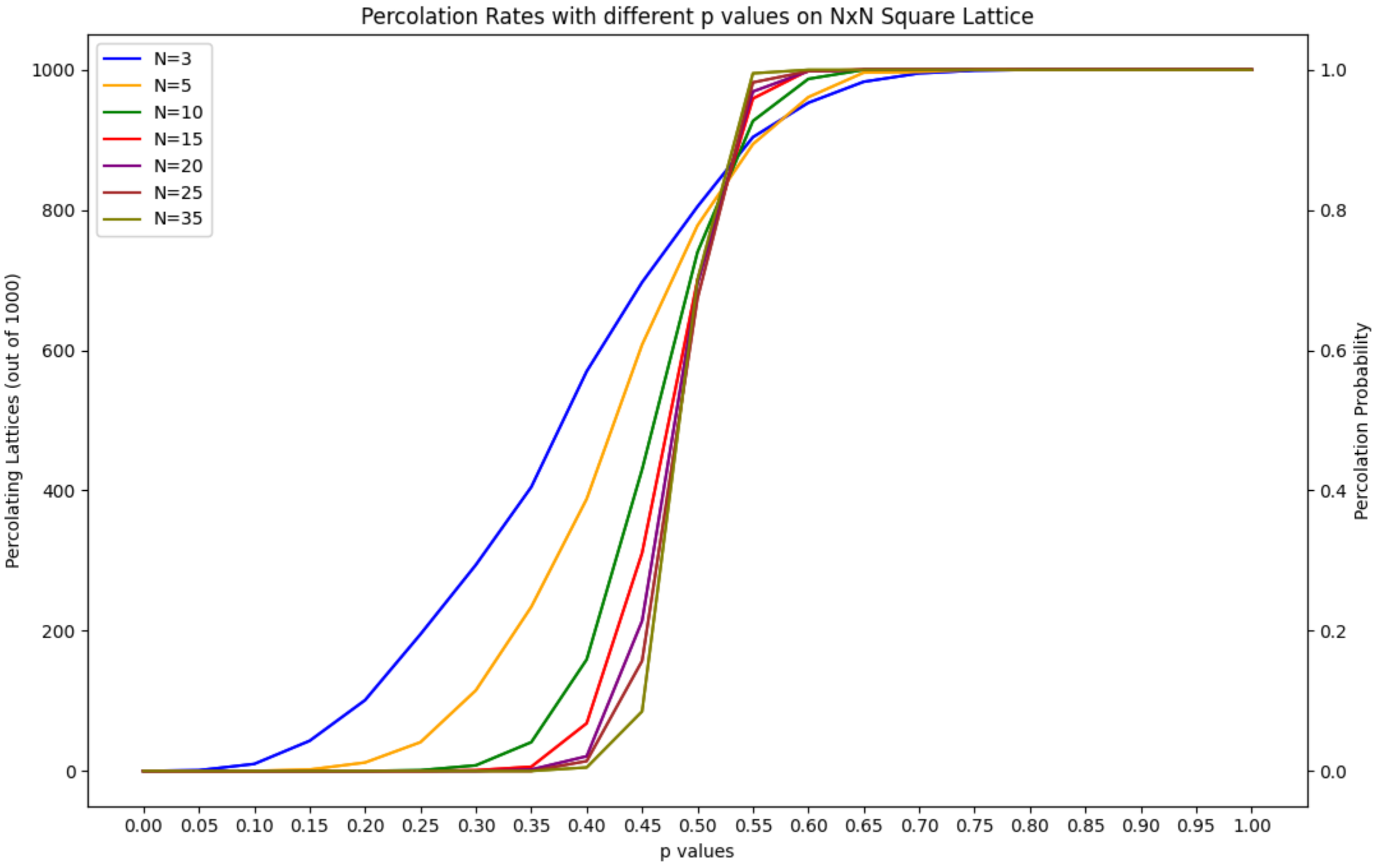
iv) Plotting functions:

I have defined 3 functions for plotting the statitstics comes from simulatExperiment() function: first one is for plotting percolation probabilities, second one is for plotting mean and std, and the third one is for plotting average finite sizes.

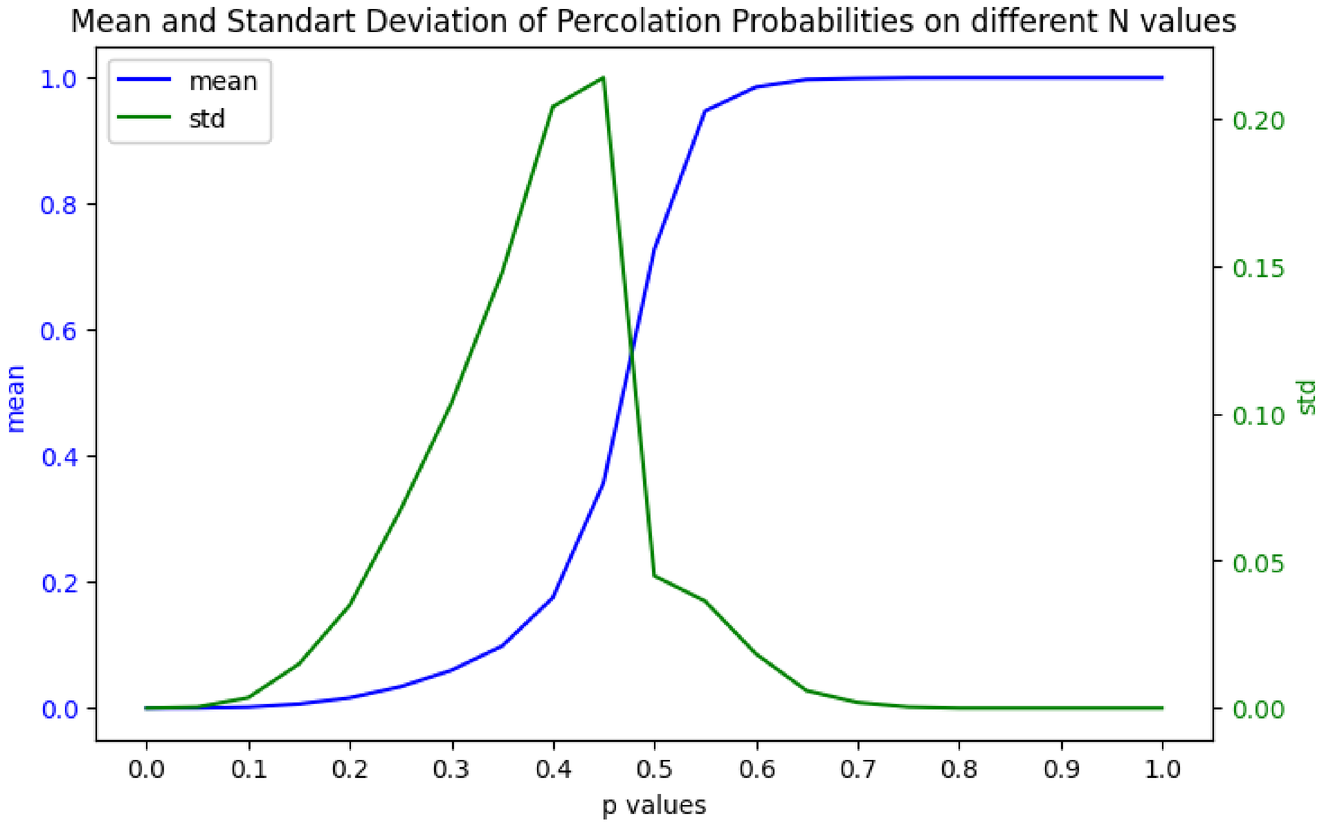
By using these functions, I have created an experiment with:

- N values = 3, 5, 10, 15, 20, 25, 35
- p values = from 0 to 1 with step size 0.05
- each simulation is repeated 1000 times

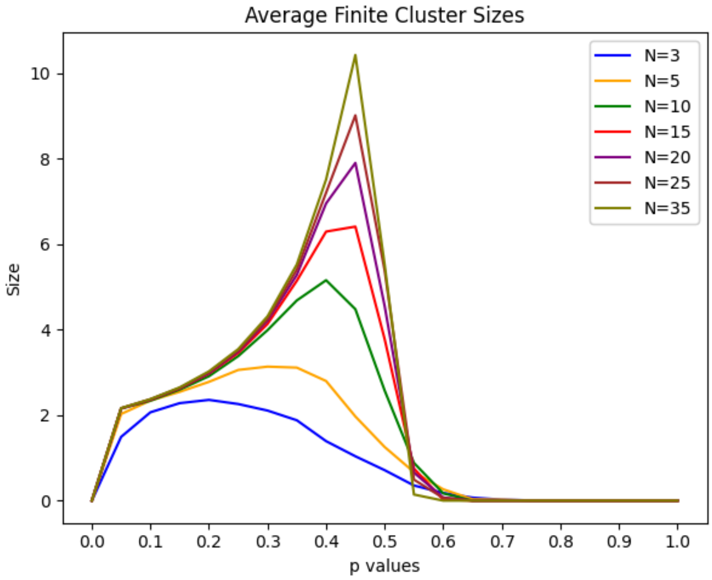
Here are the results:

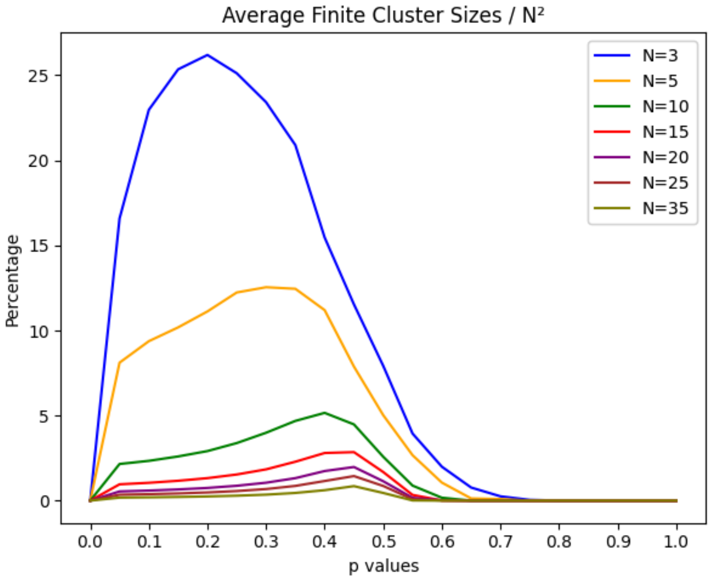


Percolation Rates with different p values on NxN Square Lattice



Mean and Standart Deviation of Percolation Probabilities on different N values





- Looking at the results, it is seen that there is a probability explosion when the p value exceeds a value between 0.50 and 0.70. Also, it is quite interesting that the lattices with higher N values are more likely to reach high probabilty values rapidly after the critical value between 0.50 and 0.70, while it is opposite before the critical p value. By looking the standar deviations, it can be said that in we might found a critical p value around 0.7, that can generalize for different N values, since the std reaches 0 around 0.7.

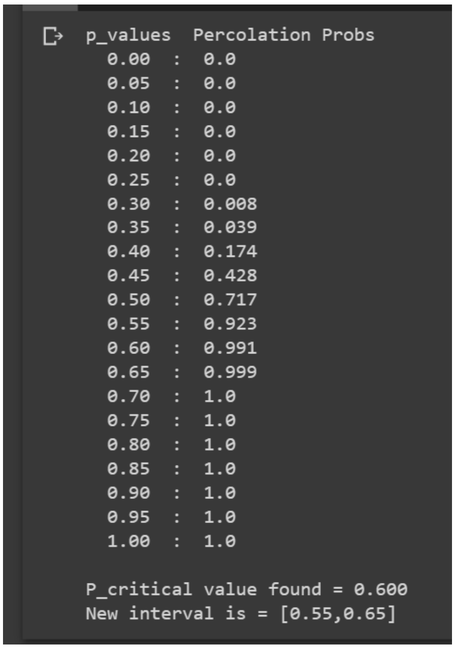
3) Estimation Part:

In this part, the goal is to estimate the critical percolation threshold (p_c) for different N values, specifically N = [5, 10, 15, 20, 25]. The estimation process involves iterative simulations and adjustments of the p values range. Initially, simulations are performed for different p values to identify the critical value (p_c) where percolation probabilities surpass a predefined threshold (0.97). The range of p values is then narrowed down to converge to p_c, and the process is repeated until p_c remains unchanged up to the fourth decimal place for five consecutive iterations.

After obtaining the estimated p_c values for each N value, the means of these estimates are plotted. Finally, the validity of the estimated p_c values is tested by conducting simulations using the interval [p_c, 1] for the p values.

Here is a one epoch of demonstration of the mathematical method that I developed to converge p_c:

- N = 10
- p values = from 0 to 1 with step size 0.05
- each simulation is repeated 1000 times
- Threshold probability value for determining p_c = 0.97



```
p_values  Percolation Probs
   0.00  :  0.0
   0.05  :  0.0
   0.10  :  0.0
   0.15  :  0.0
   0.20  :  0.0
   0.25  :  0.0
   0.30  :  0.008
   0.35  :  0.039
   0.40  :  0.174
   0.45  :  0.428
   0.50  :  0.717
   0.55  :  0.923
   0.60  :  0.991
   0.65  :  0.999
   0.70  :  1.0
   0.75  :  1.0
   0.80  :  1.0
   0.85  :  1.0
   0.90  :  1.0
   0.95  :  1.0
   1.00  :  1.0

P_critical value found = 0.600
New interval is = [0.55,0.65]
```

4) Estimating p_c for N = [5, 10, 15, 20, 25]:

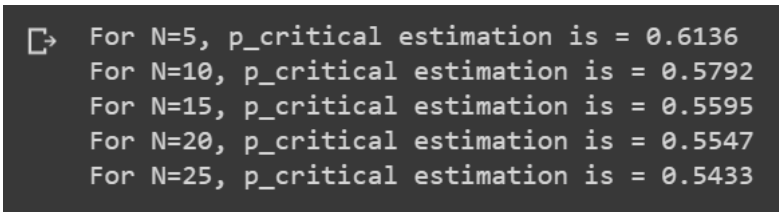
I chose the N values this way because there will be tons of simulations going on and as the N value increases, the time required to perform the estimations increases exponentially. Even like this, it takes too long to produce results.

I have defined the following function for estimating p_c:

v) converge_to_p_critical_and_AverageFiniteSize():

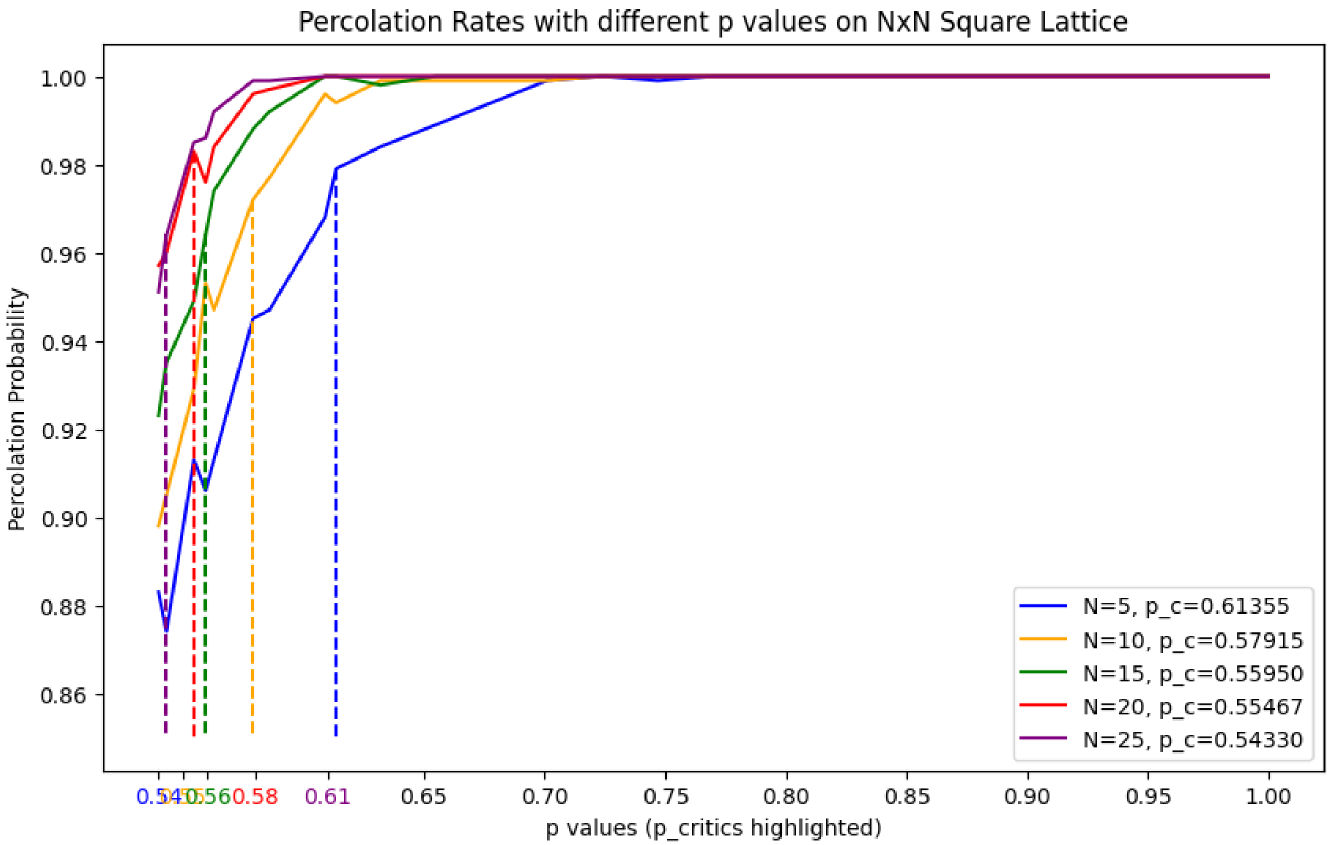
This function tries to estimate the critical percolation probability for a given N value. It uses an iterative process to converge to the p_critical value with a desired threshold. The function starts with an initial p_range and repeatedly performs simulations with evenly spaced p_values within the range. It identifies the index of the p_value that meets the threshold criterion and updates the p_range accordingly. The process continues until the p_critical value remains constant up to the fourth decimal place for five consecutive iterations. The estimated p_critical value is then returned as the result.

By using this function, I have conducted experiments on N = [5, 10, 15, 20, 25] values and make estimations. The p_critic values I have reached are as follows:



```
For N=5, p_critical estimation is = 0.6136
For N=10, p_critical estimation is = 0.5792
For N=15, p_critical estimation is = 0.5595
For N=20, p_critical estimation is = 0.5547
For N=25, p_critical estimation is = 0.5433
```

After reaching these p_critical values, I have conducted a final experiment to see whether they are accurate or not:



Based on the observations from the results, increasing the number of simulation repeats appears to lead to more accurate results. Despite the relatively small number of simulations, the obtained values are deemed acceptable. Additionally, a notable finding is that there seems to be a positive correlation between the N value and the corresponding p_critical value, suggesting that as the N value increases, the p_critical value tends to be larger.