Assignment 1 TFY4235: Percolation

Haakon Monclair

ABSTRACT

This project investigates percolation theory through computational simulations of bond percolation on square, triangular, and hexagonal lattices. Monte Carlo simulations were conducted to determine the percolation threshold p_c and critical exponents such as β, γ, and ν. The results revealed that the percolation threshold p_c was accurately determined for all lattice types, with only small deviations from theoretical values. However, the critical exponents showed larger deviations, particularly for γ, which was more sensitive to finite-size effects and statistical fluctuations. Notably, increasing the number of simulations for the hexagonal lattices led to more precise estimates of the critical exponents, indicating that higher simulation counts reduce statistical noise. These findings highlight the importance of sufficient sampling in Monte Carlo simulations. Future work may focus on scaling to larger systems, exploring alternative algorithms for more efficient simulations, and increasing the number of simulations for all lattice types to further improve accuracy.

I. INTRODUCTION

Percolation theory is a fundamental concept in statistical physics, with applications spanning materials science, epidemiology, and network theory. It describes the behavior of connected clusters in a random medium and serves as a model to understand phase transitions and critical phenomena. One of the key motivations for studying percolation is its relevance in understanding the robustness of networks, fluid flow through porous media, and even the spread of diseases in populations[1,2].

Percolation has been extensively studied both analytically and through computational simulations. The critical point at which a cluster expands is of particular interest, as it marks a phase transition in the system[3]. Lattice-based models, such as square, triangular, and hexagonal lattices, provide a simple but powerful framework for studying percolation behavior[4,5]. By analyzing the scaling properties near the percolation threshold, one can extract universal critical exponents that characterize the system[1].

In this project, I have implemented computational simulations using Python to study percolation in different lattice structures. By performing Monte Carlo simulations and finite-size scaling analysis, I have determined the percolation threshold and extracted critical exponents such as β, γ, and ν. The results are compared with theoretical predictions to assess the accuracy of the simulations.

This report is structured as follows: Section 2 describes the models and methods used in the simulations. Section 3 presents and discusses the results and their analysis. Finally, Section 4 summarizes the key conclusions of this work.

II. MODELS AND METHODS

For this study of percolation, I have used bond percolation to simulate the systems. This is where we have a network consisting of N sites and M bonds between the sites. For each step in the simulation, we activate one random bond between the sites, until all bonds are activated. The first step in this procedure was to create the lattices.

A. The lattices

I created a square, triangular and hexagonal lattice for this project. All of these had N sites and M bonds following periodic boundary conditions, i.e. that the lattice *wraps* around itself. Values between 10000 and 1000000 were used for N.

The lattices were made by creating a list consisting of all the different possible bonds in a lattice. To do this it was helpful to create drawings of the lattice and think about which bonds we have. Figure 1 shows a visualization of the square lattice. The periodic boundary conditions were implemented by using the modulo operator, as this automatically reaches back around when we reach the boundaries.

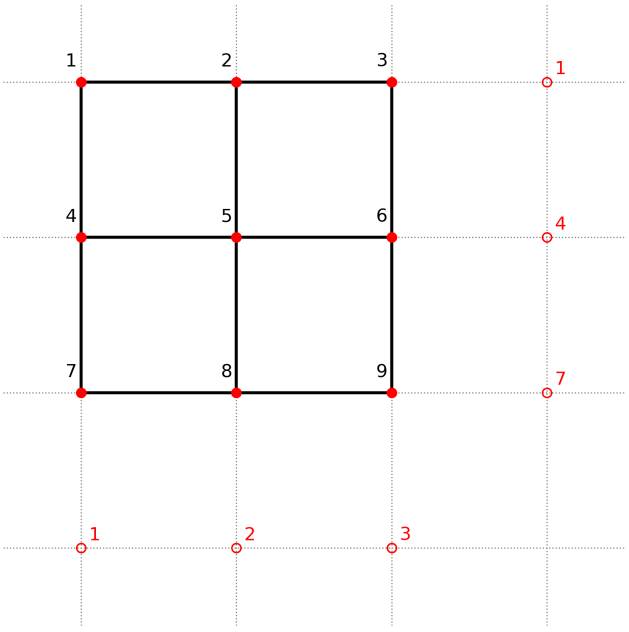

FIG. 1. Visualization of the square lattice. For this we would for example have a bond [1 2], and also [3 1], because of periodic boundary conditions.

After creating the lists of bonds, they had to be shuffled ran-

domly, to simulate the random activation of bonds. To shuffle the list I followed an algorithm described in the assignment paper. It goes as follows: First generate a random number r, uniformly distributed between 2 and M. Swap the first bond with the r-th bond (remember to swap both elements in the row). Next, generate another random number r, this time between 3 and M, and swap the second bond with the r-th bond. This process was repeated M times.

The random numbers were generated from the random library in Python. This library generates pseudo-random numbers from the Mersenne Twister generator which is a largely tested random number generator which has a period of $2^{19937}-1$[6]. Although it is deterministic, it fits well for this kind of project.

B. Percolating the system

After shuffling the bond lists, I began activating the bonds in the newly shuffled list. This was done by using two functions: *findRoot* and a *link* function. The findRoot function is a recursive function which finds the root node of a cluster by using the union find algorithm[7]. This was used in the link function which finds the root of the two newly bonded sites and links the whole system by keeping track of the largest cluster and the largest cluster root node.

While iterating through the bond list, linking sites, I always kept track of two parameters to be used later on

$$P_\infty = s_\infty/N, \tag{1}$$

$$< s >= \sum_i s_i^2/(N-s_\infty), \tag{2}$$

where N is the number of sites, s_i is the size of the selected cluster, and s_∞ is the size of the largest cluster.

Running the simulation once provides some initial results which were gathered into snapshots of the system, but to gain meaningful insights, I employed a Monte Carlo simulation. In this approach, I shuffled the bond lists with a different random seed for each run. The system was simulated 200 times for each lattice size, using seeds ranging from 0 to 199. After completing all the simulations, I calculated the average values for P_∞, P_∞^2, and $\langle s\rangle$. I also ran one simulation of the hexagonal lattice with 1000 simulations.

This simulation took quite a long time to execute, especially for the triangular lattices, which have $3\cdot N$ bonds, resulting in increased computational time. To optimize performance and reduce runtime, I utilized the ‘@jit‘ decorator from the ‘NumPy‘-based ‘numba‘ library, with the ‘nopython=True‘ option. This ensures that the function is compiled in "no Python mode," where the code is converted into highly optimized machine code. This approach significantly speeds up the execution of the simulation, particularly for operations involving large arrays and loops, by avoiding the overhead of Python's dynamic typing and interpretation. As a result, the simulation runs much faster and is more efficient, especially for larger systems.

C. Convolution

To efficiently analyze percolation properties, the implemented algorithm processes all M bonds in a single pass, offering a significant reduction in computational complexity compared to classical approaches that iterate over all N sites. However, since each bond configuration contributes differently to the overall behavior, a direct measurement is not sufficient. To obtain reliable results, it's necessary to place our data within the appropriate statistical framework.

This is achieved through a convolution with a binomial probability distribution, which accounts for all possible ways n bonds can be activated in a system with M bonds. By weighting each measurement accordingly, we ensure that our results reflect the expected behavior over many realizations. The formula used is

$$Q(q) = \sum_{n=0}^{M} \binom{M}{n} q^n (1-q)^{M-n} Q_n \tag{3}$$

where the Q_n´s are the mean values for $P_\infty(q)$, $P_\infty^2(q)$, and $\langle s\rangle(q)$. I ran this with 10000 q values evenly spaced between 0 and 1.

There were several challenges in implementing this calculation efficiently in Python. The first issue was computing the binomial coefficients $\binom{M}{n}$ for large values of M, which quickly led to overflow. To handle this, I computed the logarithm of the coefficients and later took the exponent of them to obtain the final results. Additionally, to improve efficiency, I precomputed all binomial coefficients before entering the main loop.

The second challenge was the computational speed. Using nested Python loops—one for iterating over Q_n and another for summing over n—proved to be extremely slow. To fix this, I replaced the loops with NumPy array operations, which drastically improved the performance and allowed the program to run much faster.

D. Finite size scaling analysis

To determine the critical percolation threshold p_c and the critical exponents β, γ, and ν, I used finite-size scaling (FSS) to analyze the behavior of the system near the phase transition. Close to the transition point, the variables P_∞, $\langle s\rangle$, and the correlation length ξ scale according to the following relationships:

$$P_\infty \propto (q-p_c)^\beta \tag{4}$$

$$\langle s\rangle \propto |q-p_c|^{-\gamma} \tag{5}$$

$$\xi \propto |q-p_c|^{-\nu} \tag{6}$$

Since phase transitions are only well-defined in infinite systems, we approximate them by studying lattices of different finite sizes and extrapolating the results.

The percolation threshold p_c was determined by identifying the probability q at which the largest cluster size P_∞ follows a power law with system size $\xi \sim N^{1/2}$. This was done by plotting P_∞ against ξ in a log-log plot for various q, using linear regression to find the q where R^2 was maximized. The slope of the fit, $-\beta/\nu$, was then used to calculate β.

The exponent γ, related to the divergence of the average cluster size $\langle s \rangle$, was obtained by a log-log plot of $\max(\langle s \rangle)$ versus ξ and finding the exponent ratio γ/ν.

Finally, the exponent ν, characterizing the divergence of the correlation length, was determined by studying how $q_{\max}$ scales with system size. A log-log plot of $|q_{\max} - p_c|$ versus ξ provided an estimate for $-1/\nu$.

These analyses were used to calculate the critical exponents and verify the scaling laws for percolation transitions.

III. RESULTS AND DISCUSSION

To visualize the percolation process, snapshots of all the lattice types were taken and are shown in Figure 2. These images clearly illustrate the sharp transition occurring near the critical probability $p_c$for each lattice structure. One can also observe that the triangular and hexagonal lattices appear slightly dragged or distorted. However, this is purely a result of the visualization implementation, which places the lattice sites onto a square grid for plotting purposes.

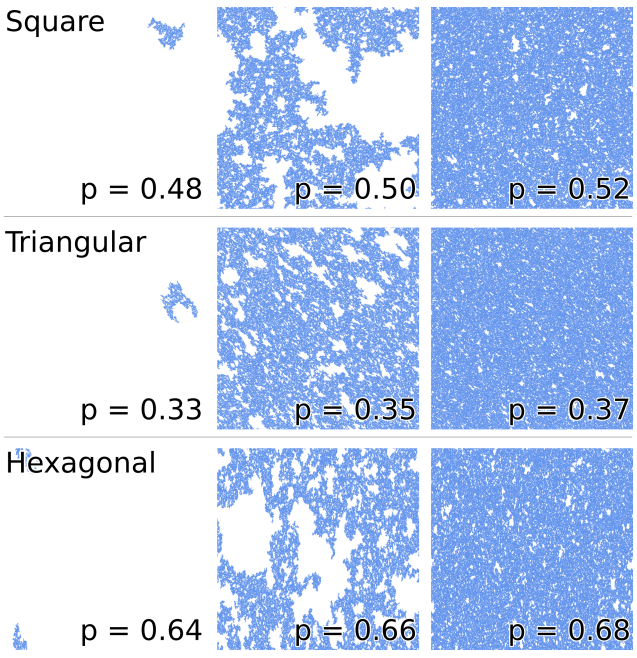

FIG. 2. Snapshots of percolation with $N = 1000000$ and seed = 50. Each row corresponds to a different lattice types, and the respective p values are indicated within the plots.

To visualize how P_∞ and $\langle s \rangle$ behaves, I plotted them both against q after doing the convolution. I did this for all lattice shapes, and the plots look similar, however they are spiking at different q values. Figure 3 displays these plots for the square lattice.

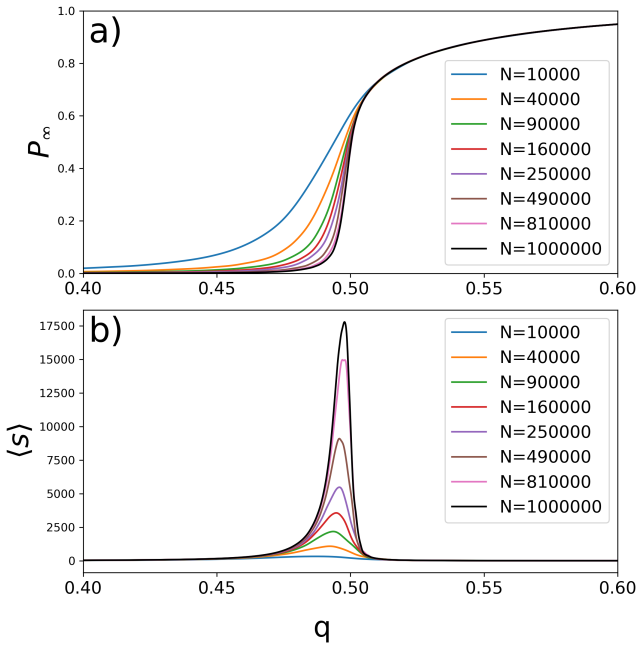

FIG. 3. The figure presents the results for the square lattice. Panel a) shows P_∞ while panel b) displays $\langle s \rangle$, both as functions of q around the critical point.

Although the plots from Figure 3 give us an idea of where the critical probability lies, I still had to calculate them and the critical exponents accurately as described in section II D.

Figure 4 shows the data points and the linear regression fit for the square lattice at the value of q that maximized the R^2 value in the regression. This value of q gives the critical probability p_c. Similar plots were obtained for the other lattice types.

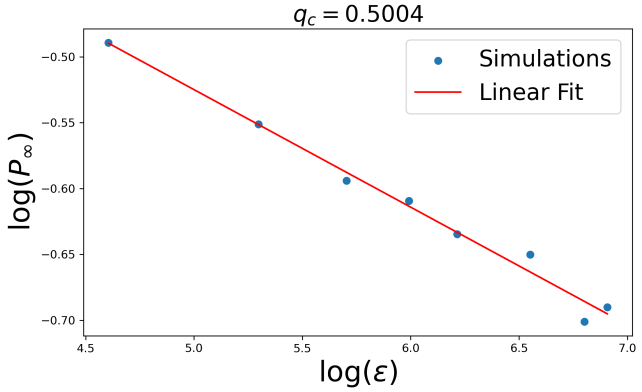

FIG. 4. Log-log plot of P_∞ versus $\xi \sim N^{1/2}$ for the square lattice at the critical value of q that yields the best linear fit.

Figure 5 shows the regression fit for the hexagonal lattice using both 200 and 1000 simulations. The data points from

the 1000 simulations align more closely with the fitted line, indicating improved accuracy and reduced statistical noise.

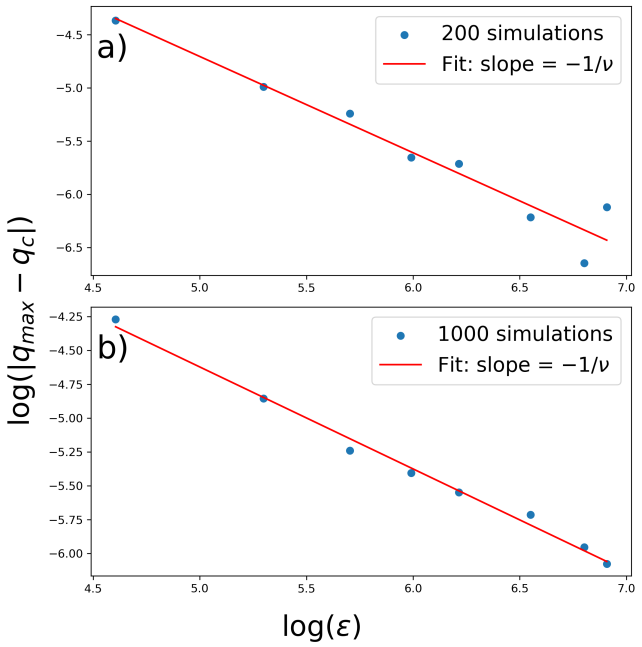

FIG. 5. Log-log plots of $|q_{\max} - p_c|$ versus ξ for the hexagonal lattice, used to estimate the critical exponent ν. Panel a) shows the linear regression based on 200 simulations, while panel b) shows the result from 1000 simulations.

The results of all the calculations are presented in table I. These critical exponents have theoretical exact values for 2D percolation and the deviations from these are also shown in table I.

	Square	Triangular	Hexagon	Hex (1000 sims)
p_c	0.5004 (0.07%)	0.3471 (0.05%)	0.6520 (0.11%)	0.6526 (0.02%)
β	0.1392 (0.22%)	0.1497 (7.8%)	0.1723 (24%)	0.1503 (8.2%)
γ	2.710 (13%)	2.175 (8.9%)	1.920 (20%)	2.312 (3.2%)
ν	1.559 (17%)	1.246 (6.5%)	1.106 (17%)	1.328 (0.41%)

TABLE I. Critical probability p_c, critical exponents β, γ, and ν for different lattice types. Deviations from theoretical values are given in parentheses.

From this we can see a couple of things. Firstly one can see that all the values for p_c are very precise, however the other critical exponents are a lot less accurate. We can also see that it seems that an increase in accuracy of one exponent, may lead to a decrease for the others, as seen from the square lattice, where β is very precise, while the others have large deviations.

The deviations in the critical exponents may arise due to factors such as finite-size effects, statistical fluctuations and potential rounding errors that propagate through the program.

A noteworthy point is the improvement in accuracy observed for the hexagonal lattice when increasing the number of simulations from 200 to 1000. This serves as a simple form of convergence analysis, demonstrating that increasing the number of realizations can significantly reduce statistical noise and yield more reliable estimates of critical exponents. This is particularly evident in the improved fits seen in the regression plots in Figure 5 and the reduced deviation from theoretical values for the hexagonal lattice in Table I.

IV. CONCLUSIONS

In this project, I implemented Monte Carlo simulations to study bond percolation on different lattice structures: square, triangular, and hexagonal. By applying finite-size scaling analysis, I determined the percolation threshold and found critical exponents such as β, γ, and ν for each lattice type. The results showed that the critical probability p_c was accurately determined for all lattice types, with deviations being relatively small. However, the critical exponents exhibited larger deviations, particularly for the exponent γ, which suggests that statistical fluctuations and finite-size effects might have influenced the accuracy of the estimates.

An important observation during the analysis was the effect of increasing the number of simulations. When the number of realizations for the hexagonal lattice was increased from 200 to 1000, the resulting critical exponents became noticeably more accurate. This serves as a form of convergence analysis, demonstrating that higher sampling rates reduce statistical noise and improve the reliability of the estimates. It also emphasizes the importance of sufficient sampling in numerical studies of critical phenomena.

This work provides a detailed exploration of percolation theory using computational methods, offering insights into the behavior of connected clusters near the percolation threshold. Future work could include further investigation into the scaling behavior for larger systems, as well as the application of other algorithms for more efficient and accurate percolation simulations. Additionally, running a larger number of simulations for all lattice types would improve the precision of the critical exponents and reduce statistical fluctuations, leading to more reliable results. One could also investigate different lattice types than the ones in this project. Lastly, alternative approaches such as parallel computing or optimization algorithms could be explored to reduce computational time and improve the efficiency of large-scale simulations.

[1] D. Stauffer and A. Aharony, "Introduction to percolation theory," CRC Press (1994).
[2] M. E. J. Newman and D. Barkley, "Networks: An introduction," Oxford University Press (2012).
[3] A. M. Gimenez and H. J. Herrmann, "Percolation in the presence of a fraction of defects," Journal of Statistical Physics **92**, 45–67 (1998).
[4] S. R. Broadbent and J. M. Hammersley, "Percolation processes," Mathematical Proceedings of the Cambridge Philosophical Society **53**, 629–641 (1957).
[5] J. Hoshen and R. Kopelman, "Percolation and cluster distribution. i. cluster multiple labeling technique and critical concentration algorithm," Physical Review B **14**, 3438–3445 (1976).
[6] P. S. Foundation, "The python standard library: random module," https://docs.python.org/3/library/random.html (2025), accessed: February 17, 2025.
[7] GeeksforGeeks, "Disjoint set (union-find)," https://www.geeksforgeeks.org/union-find/ (2015), accessed: 2025-04-22.