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 used to determine the percolation threshold and critical exponents such as β, γ, and ν. The results showed that the percolation threshold was accurately determined for all lattice types, although the critical exponents showed some deviations, especially for γ. Increasing the number of simulations improved the accuracy of the critical exponents. The findings provide insights into percolation theory and suggest that running more simulations can help refine results. Future work may explore larger systems and alternative algorithms for more efficient simulations.

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 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. I created lattices with the different values for $N = [10000, 40000, 90000, 160000, 250000, 490000, 810000, 1000000]$.

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 quick sketch of the square lattice. I made similar sketches for the triangular and hexagonal lattices as well. The periodic boundary conditions were implemented by using the modulo operator, as this automatically reaches back around when we reach the boundaries.

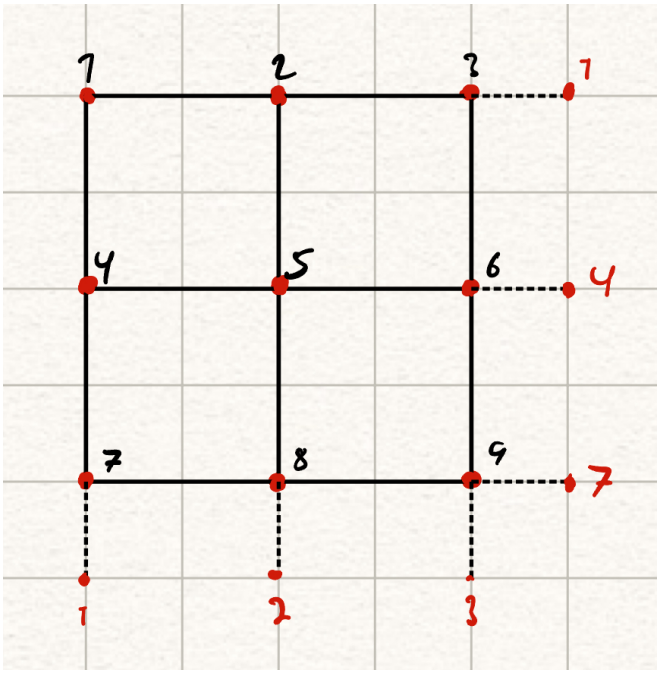

FIG. 1. Sketch of the square lattice. For this we would for example have a bond [1 2], and also [3 1], because of periodic boundary conditions.

After creating the lists of bonds, they had to be shuffled randomly, 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. 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 algorithm I implemented processes all M bonds in a single pass, significantly reducing computational complexity compared to classical methods. 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. 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 found by identifying the probability q at which the largest cluster size P_∞ follows a power law with the system size $\xi \sim N^{1/2}$. This was done by plotting P_∞ against ξ in a log-log plot for different values of q and using linear regression from Scipy to fit a straight line and find the q where the R^2 was at its maximum in the regression.

The exponent γ, related to the divergence of the average cluster size $\langle s \rangle$, was obtained by analyzing its maximum value across different system sizes. By plotting $\max(\langle s \rangle)$ against ξ in a log-log plot, the exponent ratio γ/ν was found.

Finally, the exponent ν, which characterizes the divergence of the correlation length, was determined by studying how $q_{\max}$, the probability at which $\langle s \rangle$ is maximized, scales with system size. A log-log plot of $|q_{\max} - p_c|$ against ξ provided an estimate for $1/\nu$.

By combining these analyses, I found the critical exponents and verified the scaling laws expected for percolation transitions.

III. RESULTS AND DISCUSSION

Some snapshots were taken to get a visualization of how the system looks, and they are shown in Figure 2. The transition from a) to b) clearly show that there is indeed a critical point where the system suddenly goes to one large cluster.

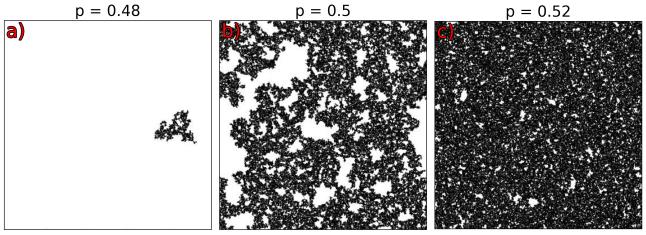

FIG. 2. Snapshots of square lattice percolation for $N = 10000$.

To visualize how the P_∞ and $\langle s \rangle$ behaves, I plotted them both against q after having done 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 and triangular lattices.

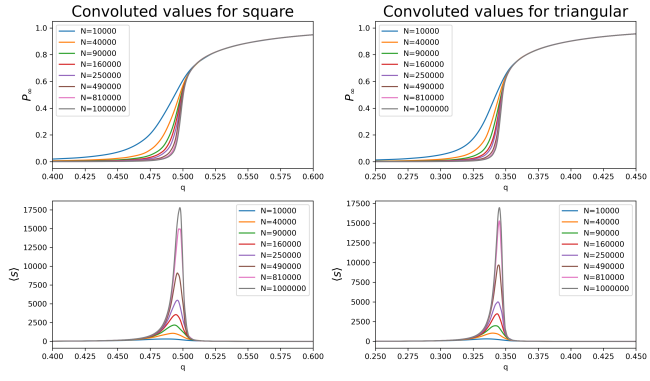

FIG. 3. Plots of P_∞ and $\langle s \rangle$ against q for square and triangular lattice, for different lattice sizes.

Although the plots 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. The results of this are shown in table I. These critical exponents have theoretical exact values for 2D percolation and the deviations are also shown in table I. 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, 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.

After obtaining the original results from 200 simulations I decided to check if running more simulations would have any effect on the results. I therefore ran 1000 simulations for the hexagonal lattice, and as one can see from table I there is a clear increase in accuracy for all the values. This proves that there is indeed a reason to use the time to run more simulations to improve the accuracy.

	Square	**Triangular**	**Hexagonal**	**Hexagonal (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.

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 was that increasing the number of simulations improved the accuracy of the critical exponents, confirming the value of running more simulations to refine the results.

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. 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.

REFERENCES

- Stauffer, D. and Aharony, A. *Introduction to Percolation Theory*, CRC Press, 1994.
- Newman, M. E. J. and Barkley, D. *Networks: An Introduction*, Oxford University Press, 2012.
- Gimenez, A. M. and Herrmann, H. J. *Percolation in the presence of a fraction of defects*, Journal of Statistical Physics, **92**(1-2), 45–67, (1998).
- Broadbent, S. R. and Hammersley, J. M. *Percolation processes*, Mathematical Proceedings of the Cambridge Philosophical Society, **53**(3), 629–641, (1957).
- Hoshen, J. and Kopelman, R. *Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm*, Physical Review B, **14**(8), 3438–3445, (1976).
- Python Software Foundation. *The Python Standard Library: random module*. Available at: https://docs.python.org/3/library/random.html, Accessed: February 17, 2025.

[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.