DETERMING THE RELATIONSHIP BETWEEN A SQUARE LATTICE’S SIZE AND ITS PERCOLATION THRESHOLD

Zoe Toigo

HYPOTHESIS

The site percolation threshold exists, and for a square lattice, it will be greater than 50%. Furthermore, the value of the percolation threshold will be independent of lattice size.

**INTRODUCTION**

Percolation theory deals with fluid flow in random media [1]. The media is filled according to a specified concentration, or occupation probability, *p.* The sites can be labeled occupied or emptied, and each labeling event is independent of the previous one. In this investigation, the media was a square lattice, and the lattice vertices, or sites, were the relevant entities. Each site has nearest neighbors, directly to the north, south, east, and west. A cluster on the lattice is a set of occupied sites mutually connected as nearest neighbors. A spanning cluster, or a percolating cluster, includes at least one site on two opposite boundaries. The percolation threshold, or critical probability, is the minimum occupation probability at which one spanning cluster will be formed. Spanning clusters are extremely rare below the percolation threshold, and become more common above the percolation threshold. Thus, a phase transition occurs at the percolation threshold[2].

The purpose of this investigation was to determine whether the site percolation threshold for a square lattice exists, its value, and its dependency on lattice size. Monte Carlo analysis, which involves varying the parameters of a simulation within given constraints and using the results to characterize a system, was used to determine the percolation threshold [3]. The simulation programmed for this investigation randomly percolated a square lattice of a given size, labeled clusters of sites, and determined whether the lattice contained a spanning cluster. The methods described in this report were unique because they took advantage of open-source, third-party Python libraries to create visual displays of the percolated lattice.

Percolation theory has diverse physical applications, such as modeling transport in porous media, “conducting materials, the fractality of coastlines, networks, turbulence, magnetic models, growth models, retention capacity and watersheds of landscapes.” Percolation also exhibits relations between a graph’s probabilistic and algebraic properties [4].

**METHODOLOGY**

Python was chosen as the programming language for executing this investigation because of its compatibility with displaying graphics, its straightforward syntax, and the

availability of open-source, third party libraries to add additional capabilities to the script. For example, the NumPy library was downloaded and implemented to organize results into a two-dimensional array, which could then be easily imported into Microsoft Excel for data analysis and plotting. Similarly, the Pygame library was utilized to create a visual display of the lattice with color-coded clusters.

**Programming Logic**

The code for this investigation was written in three main stages: creating the randomly percolated lattice, labeling the clusters of connected sites, and adding a loop to repeat trials and export the results to a text file.

In the code developed during the first stage, the user was first required to enter the desired size of the lattice and its occupation probability, the percentage of sites to be randomly filled. After initiating a lattice of zeros, a nested for loop was used to visit each site in the lattice once. At each site, a random number between one and one hundred was generated. If the random number was less than the chosen occupation probability, then the value of the site was changed from zero to one. To represent this lattice graphically, Pygame was used

to generate a white lattice and fill any site that remained a zero with the color black. The numeric version of the lattice and the graphic version of the lattice were filled simultaneously, so only one nested for loop was used during this stage.

In the second stage, a “tree method” was used to find clusters of sites. Four new functions were written to label connected clusters: the “neighbors” function, the “update\_lat” function, the “find\_root” function, and the percolation function. These functions will be explained in detail in the following paragraphs.

First, the “draw\_lat” function from the previous stage was called, and a blank, two-column array called candidates was created. One column in the array corresponded to an x-coordinate, and the other to a y-coordinate on the lattice. Next, within an infinite loop, the “find\_roots” function was called. This function looped through sites in the lattice, searching for a site equal to one. A value of one indicated that the site had not already been assigned to a cluster. The function returned the x and y values of the found site, as well as a Boolean that indicated whether or not an undealt with site was found. If the “find\_root” function returned false, then the program exited the infinite loop. Otherwise, it moved on to execute the “neighbors” function.

The “neighbors” function checked the sites to the north, east, south, and west of the site in the candidates list. If one of the neighbors had a value of one, it was added to the candidates list. Because the neighbors received the candidates array as an input but also returned the updated candidates list, it would identify clusters in a tree-like fashion. Each site in the candidates list was a branching off point. The neighbors function ran until it all the neighbors of each site in the cluster had were equal to zero, one, or the cluster label.

The variable “x” was chosen to label the clusters such that all sites in the cluster had the same label. While the value of x was less than the length candidates array, the “neighbors” function ran, and then incremented “x” by one. “x” was initiated at 0 so the candidate in row “x” of the candidates array would be checked for neighbors. However, new clusters were labeled with a value of x+2 so that the sites labeled 0 and 1 would remain untouched.

The “update\_lat” function displayed clusters visually on the lattice. Within this function, three random numbers were generated to come up with an RGB value for the cluster’s color. Any site with a value of x+2 was changed to the randomly. Since the “update\_lat” function was called within the “neighbors” function, only the current cluster being labeled could be given a color. Figures 1 and 2 show sample lattices with color-coded clusters.

****

FIGURE 1

Sample lattice that does not percolate. Size is 30x30. p =0.50.



FIGURE 2

Sample lattice that percolates. Size is 30x30. p =0.59.

To determine whether a lattice contained a percolating cluster, the top row, bottom row, far left column, and far right column of sites were checked. If the same cluster label existed on two opposite sides, then a percolating cluster existed.

Finally, during the third stage of code development, two for-loops were added to execute fifty trials for each value of p from 1 to 100. The trial for-loop encompassed all existing code from stage two. It was then nested inside the p-value loop. Some changes to the existing code from stage two were also added. Since creating a visual display for each trial would be unnecessarily time consuming, any lines of code that referenced the Pygame library were removed. A timer, added to track the time in seconds needed to run the script. Finally, the results were compiled into an array through NumPy. In the array, the results of each of the fifty trials were listed for each value of p. This array was then exported to a text file. Note that the code did not loop through possible lattice sizes. Therefore, in order to test lattice sizes from 100 to 10,000 and increment the size by , the program had to be executed ten times. Each time, the size of the desired lattice was entered manually, and as a result, ten text files were produced in total.

**Data Analysis**

Each text file was loaded to a separate Microsoft Excel sheet. Next, the average and standard deviation of the results for one occupation probability were calculated using functions provided in excel. The average represented the percentage of lattices with the same size and occupation probability that successfully percolated.

The percolation threshold was defined as. Therefore, is also equal to the inflection point in a graph of the occupation probability vs the percentage of trials that percolated. To calculate the inflection point, the instantons first derivative was calculated for each point using the equation:

where y(p) is percentage of successful percolations for each concentration, p. The inflection point is the point at which the first derivative has a maximum value.

**RESULTS AND DISCUSSION**

In this investigation, the average site percolation threshold of the ten chosen lattice sizes was 0.60, with a standard deviation of 0.01549. Average occupation probabilities for each lattice size tested are shown in Plot 2. Because of the low standard deviation, it was determined that the site percolation threshold is independent of lattice size for a square lattice. Thus, the hypothesis was correct.

Regardless of size, the maximum standard deviation was always between 49 and 50. Additionally, it was the occupation probability equal to .01 less than the percolation threshold that consistently had the highest standard deviation. This was characteristic of a phase transition, for which there exists a critical probability, or threshold, below which spanning clusters are extremely rare and above which they become increasingly common. Furthermore, it is known that the most variance in data will occur around the critical probability [2]. This indicates that the calculations led to the correct percolation threshold.

As the size of the lattice increased, the sharpness of the phase transition also increased. This was observed in Plot 1.

PLOT 1

Lattice Size vs. Occupation Probability. This plot shows the progression of the phase transition for each lattice size. The

percentage of lattices that included spanning clusters for each occupation probability are plotted.

PLOT 2

Lattice Size vs. Occupation Probability. This plot demonstrates that the site percolation threshold of a square lattice is independent of size.

Quantitatively, the sharpness of the transition can be represented by the range of occupation probabilities that did not equal zero. The 100 site lattice had the greatest range of 35, and the 10,000 site lattice had the smallest range of 4. The lattice size was related to the sharpness of the phase transition through a logarithmic trend line, as shown in Plot 3. The line’s R2 value of 0.9667 shows that it is reliable.

PLOT 3

This plot demonstrates that the relationship between lattice size and sharpness of phase transition, which was quantified as the range of concentrations with standard deviation greater than zero. The relationship is logarithmic.

The greatest weakness in the methods described in this report was the inefficiency of the code. It was expected that the larger size lattices would take longer to run the required number of trials. However, the 10,000 site lattice took 75 minutes to complete 50 trials for each occupation probability. Therefore, the code would need to be revised for applications involving large lattices. The most likely reason for this inefficiency in the code was the way that the candidates array was built. New candidates were simply added to the end of the array. Therefore, each time the “neighbors” function ran, unnecessary candidates were checked. If this investigation were repeated, the candidates that had previously been assigned to a cluster would need to be overwritten.

**CONCLUSION**

Before this investigation began, it was hypothesized that the site percolation threshold exists, and for a square lattice, it will be greater than 50%. Furthermore, the value of the percolation threshold was predicted to be independent of lattice size. To test this hypothesis, Monte Carlo simulations were executed in Python to randomly percolate the lattice, label clusters, and determine whether the lattice percolated. For each lattice, fifty trials were run for each possible occupationally probability. After averaging the percolation threshold for each size lattice, it was determined that the percolation threshold, the minimum concentration at which a percolating cluster will be formed, is equal to 0.60 and is independent of lattice size. Thus, the hypothesis was correct. Therefore, because it accomplished the purpose state above, this investigation was successful.

**REFERENCES**

[1] K. Christensen. (2002). “Percolation Theory.” *Statistical Physics II.* (online document). <http://www.mit.edu/~levitov/8.334/notes/percol_notes.pdf>

[2] R. Gonsalvdes. (2013). “The Percolation Phase Transition.” *University of Buffalo.* (online document). <http://www.physics.buffalo.edu/phy410-505-2009/topic6/lec-6-1.pdf>

.

[3] (2016). “Monte Carlo Simulation.” *MathWorks*. (webpage). <http://www.mathworks.com/discovery/monte-carlo-simulation.html>

[4] A. Saberi. (2015). “Recent advances in percolation theory and its applications.” *Physics Reports.* (online article.) <http://web.a.ebscohost.com/ehost/command/detail?sid=ee9ad09c-fe9c-4275-b934-c33591da1f84%40sessionmgr4004&vid=4&hid=4112>

**ADDITIONAL SOURCES**

(2009). “Downloads.” *Pygame*. (webpage). <http://pygame.org/download.shtml>

(2016). “Obtaining NumPy & SciPy libraries.” *SciPy.org.* (webpage). <http://www.scipy.org/scipylib/download.html>

J. Johnson. (2011). “List of Color Names with RGB and Hex Values.”  *99 Colors.* (webpage). <http://www.99colors.net/color-names>

(2016). “[random](https://docs.python.org/2/library/random.html#module-random) — Generate pseudo-random numbers”. *The Python Standard Library*. (webpage). <https://docs.python.org/2/library/random.html>

**ACKNOWLEDGMENTS**

Much gratitude is due to Dr. Balazs and Natalie Austin for their guidance in writing pseudo code for this investigation, as well as answering any questions that arose. Several classmates also provided critical advice. The “tree method” used was based on Andy’s recursive tree model. Kaylene Stocking suggested an alternative to my ineffective code. Finally, Dean Miller provided useful advice for editing my lab report. I would like to thank these students for their role in expanding my understanding of programming and helping me to turn in my best work.