William Joel Sandoval Casas

Evaluation of Percolation Using Software

ENGR 0712: Advanced Engineering Applications for Freshman

January 29th, 2024

Dr. Balazs

Contents

List of Figures

Introduction

When introduced in a feedlot, a parasite may wipe out a certain species in a matter of weeks. Recently, the virus, COVID-19, tested society, spreading across the globe, and killing millions. Similarly, information may diffuse rapidly in a connected body of individuals, especially with present-day technologies.

Vital to understanding how diseases, ideas, and practically everything in the periphery spreads is the idea of percolation. As defined by the reputable ScienceDirect, "percolation is a statistical concept that describes the formation of an infinite cluster of connected particles or pathways" [1]. In other words, percolation describes a pathway that connects point A to point B. When focusing percolation on the field of science, it can be observed to possess various applications and traits, such as its similarity to phase transitions [1]. More specific applications of percolation include, "[defining] the interconnectivity between particles for conductive filler of a specific composite," and "explaining and obviating the problems of fluid flow through a static medium" [1].

To test the success of percolation on a two-dimensional, square-like system, a program was developed using the programming language C that randomly generated pathways with varying matrix sizes, representing the lattice of percolation. After these random pathways are connected, an algorithm, inspired by the Hosher-Kopelman algorithm, would count the clusters of revealed pathways. These clusters would then reveal if that random percentage of revealed tiles produced a successfully percolating matrix, by comparing the values at either extreme of the matrix. This process is described in detail in the "Experimental Procedure" section of the "Methods.

Learning to predict and analyze percolation patterns is vital for an ever-growing society. In the event of another global pandemic, for instance, identifying patterns by which disease spreads would help researchers be one step closer to preventing and eliminating the spread of deadly pathogens. In a health-related scenario, the percentage required for percolation to occur would represent the percentage of people expected for exposure to occur at a rate of one hundred percent, taking into consideration these individuals are in an enclosed space. Simply put, percolation can help predict the concentration of people in an area for them to be exposed to a lethal disease.

Hypothesis

Through thorough discussion, the students in ENGR 0712, Advanced Engineering Applications for Freshman, decided unanimously that, on average, percolation would be reached in a system with less than fifty percent of the pathways exposed. This conclusion was reached because, under perfect conditions, the percentage of filled pathways within the lattice needed for the site to percolate can be represented by the following equation: $percolation(\%) = (side)^{-1} * 100$. In other words, an iteration with perfect luck would percolate first at that specific percentage. In a sample 10x10 grid, one would require, at minimum, ten percent of the lattice to be filled in order to create a direct path across the matrix. Therefore, with a large enough dataset, the minimum percentage required for percolation to occur could drop to one, or even half a percent. Similarly, there would be a percentage where percolation is reached, even if more paths are displayed in the matrix. In a 10x10 matrix, by completely filling in the center middle 8x8 square and adding four tiles to connect the middle square to the opposing borders, a total of sixty-eight tiles would be filled; thus, starting at sixty-eight percent of the matrix's

capacity, the system always percolates. If the percentage was highballed and estimated to be eighty-five, then the median between the min and the max of the 10x10 grid sample distribution would be forty-seven and a half—well below the fifty percent mark.

Additionally, a percolation threshold, or a specified percentage of randomly generated pathways that produce one and only one path across, was believed to be non-existent. The possibility exists that a lattice, when generated, creates two pathways simultaneously. Since it was believed that there would be an instance of a "double" pathway, one is unable to draw the conclusion that a percolation threshold exists.

Lastly, with an increase in lattice size, more accurate results were anticipated. For instance, if a researcher ran a 1x1 matrix, it would first percolate at one hundred percent. Similarly, a 2x2 matrix requires a minimum of fifty percent of occupied paths to percolate. The pattern can be represented with the equation above, where: $percolation(\%) = (side)^{-1} * 100$. As the length of the square-matrix is increased, the minimum percent required for percolation decreases, leading to more accurate data.

Anti-Hypothesis

While evaluating percolation using the hypothesis, the possibility of unexpected results, contrary to the expectations dictated in the hypothesis, must also be considered. Therefore, it must also be considered, that a system percolates once fifty percent or more of the lattice is randomly occupied. Additionally, the possibility must be considered, where a percolation threshold, believed to be non-existent, exists. Further, as lattice size increases, data would become less accurate.

Methods

Experimental Procedure

To begin the data collection process, a program would need to be developed capable of running large iterations of the experiment and recording and exporting data. The programming language C was chosen for its dependability and capabilities of performing operations with matrixes and exporting data to a text file through file pointers. Additionally, C would be run using the Integrated Development Environment (IDE) of Visual Studio Code (VS Code); which uses a Windows Subsystem for Linux (WSL) to operate Linux: Ubuntu. Additional computer specifications include the use of an AMD Ryzen 7 7730U with Radeon Graphics-2.00 GHz processor, 16GB of RAM, and a 64-bit operating system.

Focusing on the C program, the structure proceeds as follows. There exists a while-loop that randomizes float values from 0 to 1, up to three significant figures, and initializes them in every location of a matrix of a pre-determined size. Immediately after, a secondary while-loop is entered that uses the original matrix to determine if a -1 or a 0 is placed in a cluster counting matrix. The integers -1 and 0 represent a value under or equal to the percent threshold, or greater than the percent threshold, respectively. The percentage threshold is increased by one per loop iteration until it reaches a hundred, covering every possible percentage value.

Before the percent threshold increases the loop iteration, the program utilizes the Hoshen-Kopelman cluster counting algorithm in tandem with the cluster counting matrix to label the clusters in the lattice. Figure 1 depicts the process by which a sample 6x6 cluster counting matrix, with -1 and 0 values, would have its clusters counted.

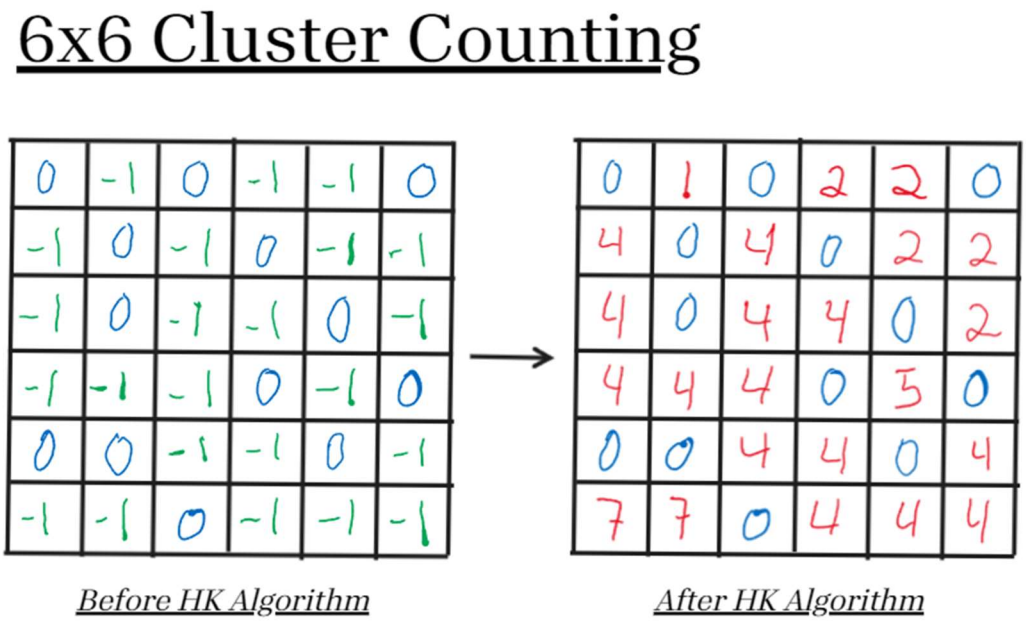

Figure 1: 6x6 Cluster Counting

To determine if a certain percentile value promotes percolation, the cluster counting matrix is placed in for-loops which determines whether the same positive integer is present on either extreme of the board, either top & bottom or left & right. If a cluster of labeled paths is present on both sides, the system successfully percolates at that certain percentage value. Since the clusters are counted and assigned a specific value per cluster grouping, a cluster present on both extremes indicates that a path to facilitate percolation was found. If that is the case, an array records the percentage value which that matrix iteration percolated. At the program's conclusion, data describing the percentage of trials of a certain grid that percolated based on percentage is printed into a ".txt" file for further analysis.

An additional feature of the program allows the user to repeat the coding procedure a desired number of times, where each new iteration regenerates new random values. At the end of the program, the data gathered is adjusted to represent the average of the data compiled based on the number of iterations.

After the data has been collected, it is imported to an Excel sheet, and a graph representative of the dataset is generated. Figures 2 and 3, present in the "Results" and "Discussion" sections, illustrate the data plotted in Excel.

Altering Matrix Dimensions

To fully understand the effect of percolation in an equal-sided matrix, the length of the sides must be increased to collect more accurate and diverse data. To do so, a global variable in the percolation program described in the "Experimental Procedure" section will be varied, changing the dimensions of all matrixes and size-specific operators in the code. The specific dimensions tested for the purposes of this experiment are 10x10, 20x20, 30x30, 100x100, and 200x200. The grids with the dimensions 10x10, 20x20, and 30x30 will run a million times to acquire the most accurate results. Important to note is matrix sizes above 30x30 were executed for only one thousand iterations due to computational logistics; that is, running a million iterations with matrixes larger than 30x30 would require an extensive period of time with available machinery. Setting up a Virtual Machine (VM) in Linux: Ubuntu may facilitate long-term data collection, but the duration of run-time should still be considered to be lengthy.

Results

Representation of Percolation

In par with the experimental procedure, a total of 6 trials were performed, with the 10x10, 20x20, 30x30, 100x100, and 200x200 grid sizes. Additionally, the matrix sizes 10x10, 20x20, and 30x30 are computed a million times, and the large lattice sizes are run for a thousand iterations.

The graph for percolation in a 10x10 grid, seen in Figure 2.1, mirrors a J-Curve. For the first and last, roughly thirty data points, the graph remains stagnant. Starting at thirty-two percent, however, the graph sustains a sharp positive slope. Then, at eighty-nine percent, the slope regresses to zero. When fifty percent of the paths are revealed, only 0,286032 percent of the trials have percolated. On the other hand, the lattice reaches fifty percent percolation between fifty-four to fifty-five percent of revealed paths.

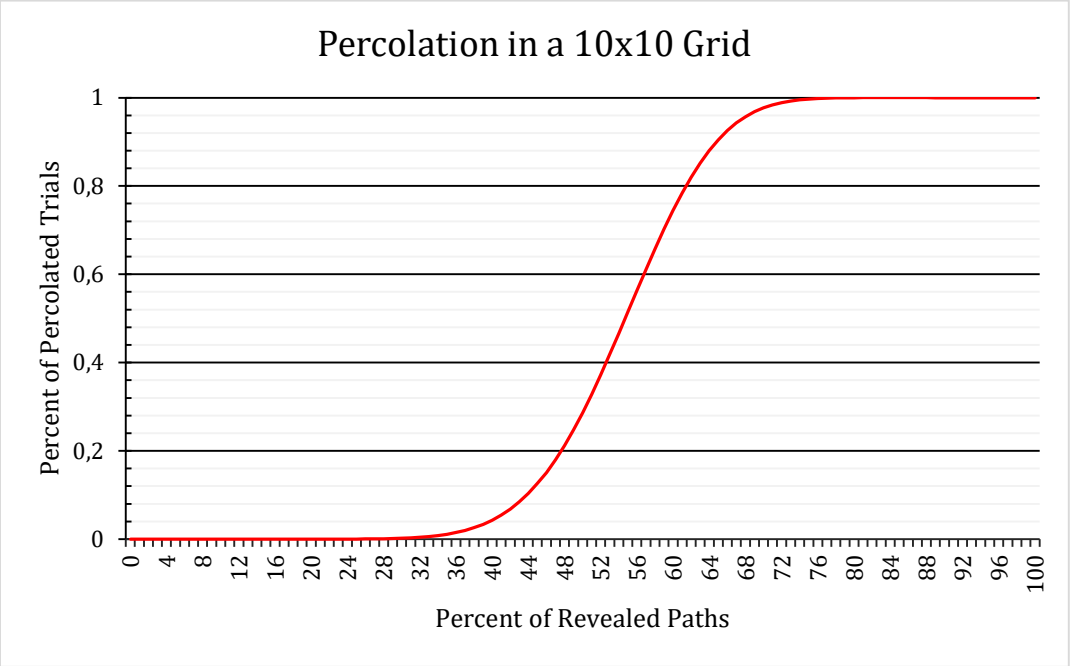

Figure 2.1: Percolation in a 10x10 Grid

Figure 2.2, portraying Percolation in a 20x20 grid, is similar to Figure 2.1. One major difference is that the 20x20 grid possesses a greater, sharper slope. Otherwise, the 20x20 grid starts a positive slope at thirty-four which diminishes drastically at seventy-four. When fifty percent of the paths are revealed, only 0,111734 percent of the trials have percolated. It is only between fifty-six and fifty-seven percent of exposed paths that fifty percent of the trials have percolated.

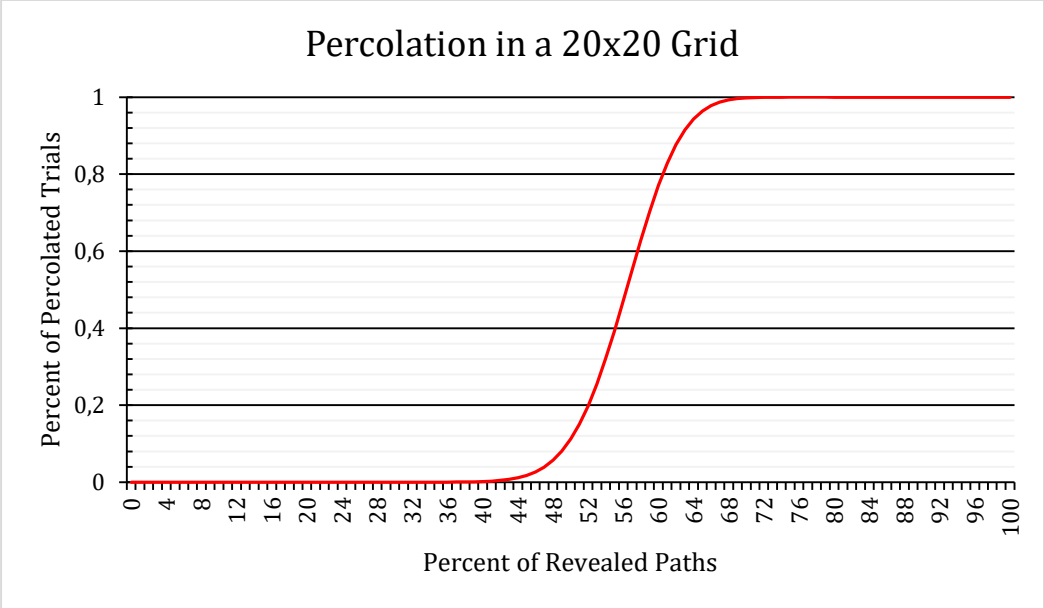

Figure 2.2: Percolation in a 20x20 Grid

The 30x30 grid, depicted in Figure 2.3, increases in slope exponentially at forty percent of revealed paths and returns to a rate of change of 0 at sixty-eight percent of revealed paths. As fifty percent of the paths are exposed, only 0,028952 percent of percolated trials occur, yet fifty percent of the trials percolate between fifty-seven and fifty-eight percent of shown paths.

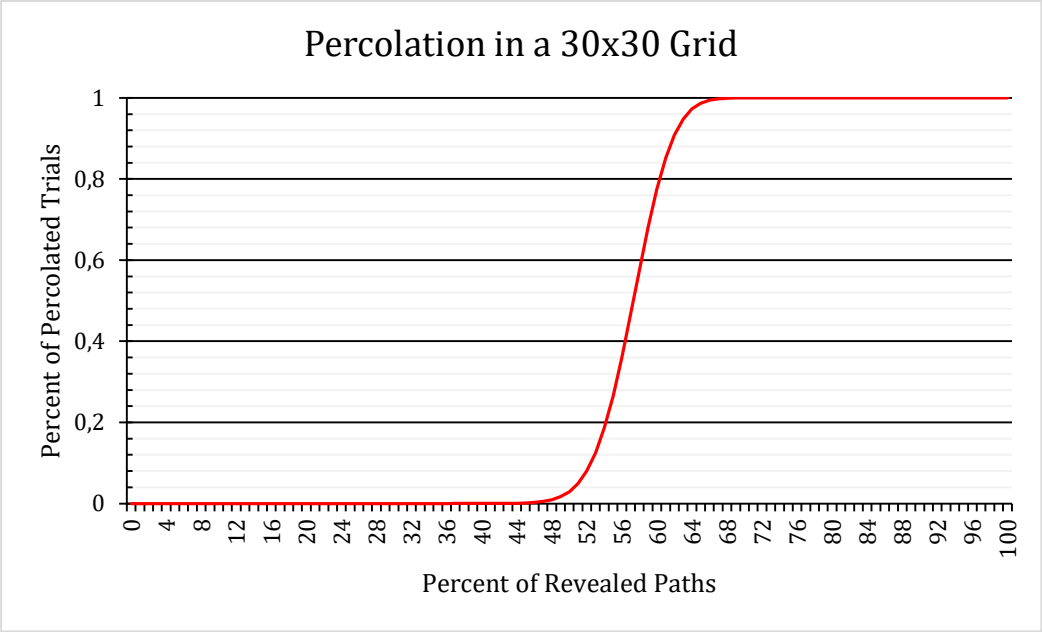

Figure 2.3: Percolation in a 30x30 Grid

Observing lattices with larger dimensions requires more computing power and time per iteration. Figures 2.4-2.5 represent large-scale percolation performed for a thousand iterations. Focusing on the 100x100 grid, Figure 2.4 depicts the development of a sharp positive slope beginning at fifty-two and ending at sixty-four. When fifty percent of the paths are generated, none of the trials percolate. It requires between fifty-eight and fifty-nine percent of the trials to percolate for fifty percent of the paths to be revealed.

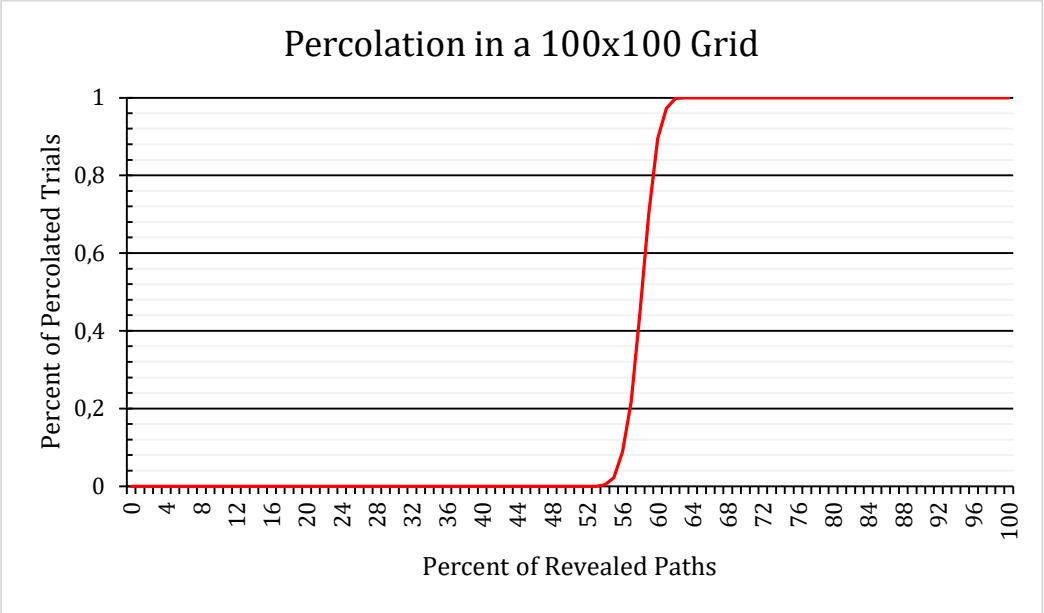

Figure 2.4: Percolation in a 100x100 Grid

Increasing the length of the lattice by a factor of two, Figure 2.5 illustrates a thousand iterations with the 200x200 matrix. This dataset begins a sudden positive slope around fifty-six percent of revealed pathways and returns to a neutral rate of change at sixty-two percent of revealed paths. Percolation occurs above fifty percent of pathways being revealed and fifty percent of the trials percolate between fifty-eight and fifty-nine percent of exposed paths.

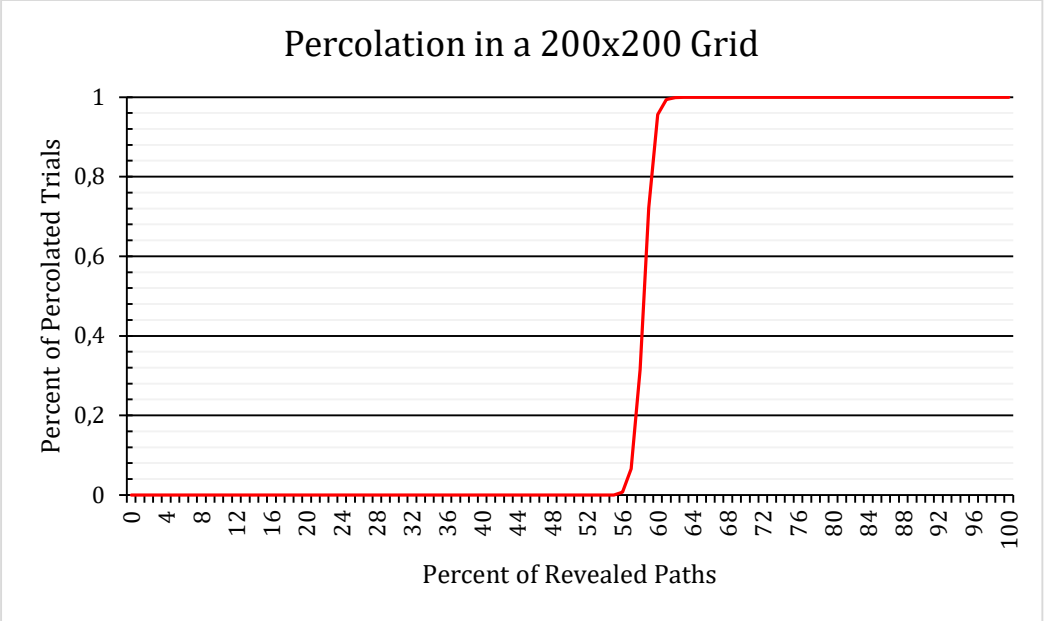

Figure 2.5: Percolation in a 200x200 Grid

Discussions

Assessment of Percolation

Present in the data pictured in Figures 2.1-2.5 are various trials of percolation. Across the board, the data demonstrates that contrary to popular belief, percolation occurs more frequently as fifty percent or more of the lattice is occupied. There are very few trials that percolate before fifty percent of the matrix is randomly generated. In fact, matrixes with a grid size of 100x100 or greater require more than fifty percent of the lattice to be randomly filled to generate values. Additionally, running the data for larger iterations provides more accurate data, as seen through the creation of trends, discussed in the following section.

Learning to predict patterns in percolation is crucial to understanding the concept of random spreading. By being able to predict the exact concentration of people in a certain area

that promotes exposure, disease can be reduced. For instance, stricter guidelines limiting the number of individuals in an enclosed, square-like environment may reduce the spread of a certain disease. Based on this experiment, one could conclude, ignoring other factors affecting the spread of disease, that in a 200ft-by-200ft enclosure, occupancy of fifty-six to sixty-two percent would result in a progressively infected population.

Assessment of the Percolation Threshold

The graphs seen in Figures 2.1-2.5 depict sharp, sudden slopes concentrated in values above the fifty-percent threshold of revealed routes. A trend is developed, particularly seen when comparing the 10x10 and 200x200 matrices. In the 10x10 matrix, the range of positive slope occurs from thirty-two to eighty-nine percent of the generated matrix. By contrast, the 200x200 matrix sustains an upward slope from fifty-six to sixty-two percent of revealed paths. A clearer illustration of this "perpendicular" reaching phenomenon can be seen in Figure 3, a compilation of Figures 2.1-2.5.

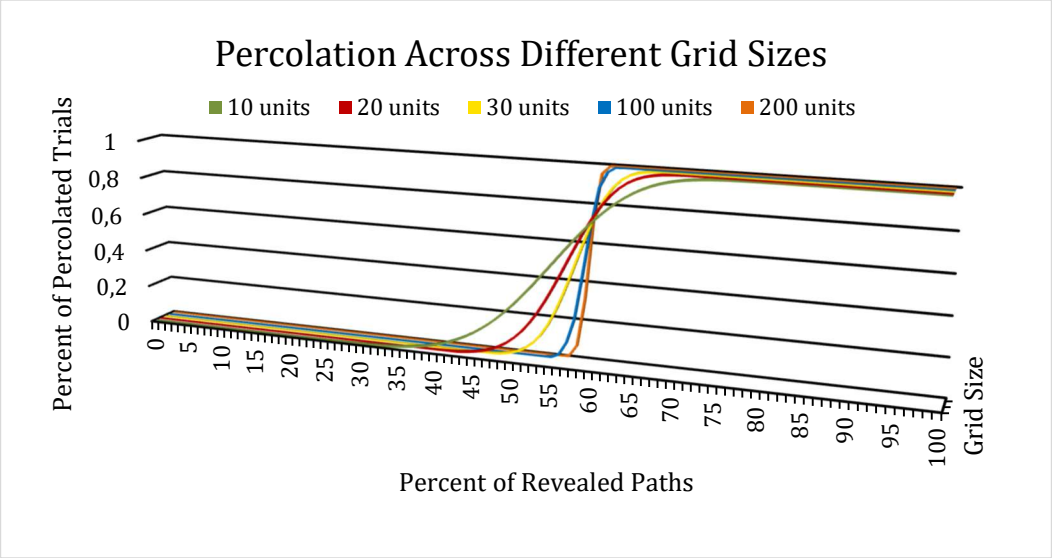

Figure 3: Percolation Across Different Grid Sizes.

It can be concluded that as the grid size increases, there exists one and only one percentage value that incites percolation in the lattice. Contradictory to the preconceived understanding, a percolation threshold, in fact, exists. Furthermore, a percolation threshold may reveal the point at which an entire population becomes infected or which an idea, such as a visual work, spreads mainstream.

Limitations of the Study

Possession of a more powerful computer, capable of running more iterations of the percolation program would result in yet more accurate results. Additionally, coding languages use algorithms to generate random numbers—a generation of values that could be recreated if the algorithm were reverse-engineered [2]. Therefore, random number generators are not truly random. Having a system in place to truly generate random values, based on random measurable patterns, such as thermal factors, may result in improved data [2].

Recommendations

For future recreations, where an individual would like to evaluate percolation using software, it is highly recommended they use a coding language that supports the ability to create visual data, such as graphs, as it may simplify the experiment. While C does support the creation of graphs, the code required to set it up is advanced, to the point it would be simpler to save and export the data to Excel. Additionally, future researchers should maintain a constant number of iterations when testing different matrix sizes to ensure accuracy. Furthermore, it is suggested future researchers map out their code with comments on their respective integrated development environment (IDE), or utilize other forms of pseudocode, to ensure the success and swift

development of their program. Lastly, it is recommended that values printed into a file with the use of file pointers should maintain a large, constant precision.

References

[1] D. Stauffer & A. Aharony. “Percolation.” Elsevier. 2003. Accessed 20.01.2024. https://www.sciencedirect.com/topics/materials-science/

[2] J. Rubin. “Can a computer generate a truly random number?” Massachusetts Institute of Technology School of Engineering. November 1, 2011. Accessed 21.01.2024. https://engineering.mit.edu/engage/ask-an-engineer/can-a-computer-generate-a-truly-random-number/

Additional References

J. Wyszynski. “PathSortingAlgorithm.” 2024. Accessed 22.01.2024.

W. Sandoval Casas. “Balazs_3pm_Percolation_wjs62.cpp.” 2024. Accessed 25.01.2024. https://github.com/sandoval-williamj/ENGR-0712/blob/main/Balazs_3pm_Percolation_wjs62.cpp

K. Christensen. “Percolation Theory.” October 9, 2002.” Accessed 22.01.2024. https://web.mit.edu/ceder/publications/Percolation.pdf

Acknowledgments

I would like to recognize my Advanced Engineering Applications peers for their assistance in this assignment. In particular, I would like to express my gratitude to Jacklyn Wyszynski, who was key in the completion of my Hoshen-Kopelman algorithm. Further, I am thankful for the clarifications on percolations provided by my professor, Dr. Balazs, and my teaching assistant, Dominick Filonowich.