AN ANALYSIS OF 2D CLASSICAL AND MANDELBROT PERCOLATION MODELS

ALAN SAMMARONE

An exploration of percolation 2D models

Institut für Theoretische Physik
Universität Leipzig

Sep 2020

ABSTRACT

The current state of percolation theory is hard to assess. Research often uses different methods, language and notation, is hard to reproduce, and sometimes even provides conflicting results. We aim to give a numerically detailed, reproducible overview of the classic 2D percolation problem, along with data and open source code that allow the reader to easily reproduce the results presented, as well as present a simple numerical analysis of the relatively uncommon model called Mandelbrot percolation. We hope to facilitate further research, as well as provide a starting point for anyone interested in the topic.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— **knuth:1974** [**knuth:1974**]

ACKNOWLEDGEMENTS

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CODE LISTINGS

ACRONYMS

PMF	Probability mass function

Part I

CLASSIC 2D PERCOLATION

1 THEORETICAL ASPECTS

1.1 SETUP

A lattice (G, N, S) is set of nodes G, a neighbour relation N from G to to the power set of G: $N\colon G \to \mathcal{P}(G)$, and a state map $S\colon G \to Q \subseteq \mathbb{N}$. The precise nature of G and N depends on exactly what type of lattice we're considering. For simplicity, our analysis and results here will be limited to a 2D square lattice, which can be either infinite or finite. In this case, G is a subset of the set of ordered pairs of natural numbers:

$$G \subseteq \mathbb{N} \times \mathbb{N} = \{(i, j) \mid i, j \in \mathbb{N}\} \tag{1}$$

So, for example, $(3, 2)$ and $(8843, 0)$ could be elements of G. If we're considering an infinite lattice, then we have

$$G = \mathbb{N} \times \mathbb{N}$$

N specifies a set of neighbours for each element of G. If one views the lattice as a graph, then N contains information about the edges of the graph. In the (infinite) 2D square lattice case, we have

$$\begin{aligned} N\colon G &\to \mathcal{P}(G) \\ (i, j) &\mapsto \{(i-1, j), (i+1, j), (i, j-1), (i, j+1)\} \end{aligned} \tag{2}$$

For every node in G, this map assigns 4 neighbours. If one thinks of the lattice as a grid, then for every node (i, j) this map assigns the left, right, top and bottom nodes as its neighbours.

S simply assigns a "state" in the set Q to each node in G. In standard percolation theory, each node can be in one of two states, so $Q = \{0, 1\}$. These two states are also called dead-/alive, black/white or active/inactive.

A cluster C in the lattice is defined as a subset of connected nodes (i, j) in the lattice with such that $S(i, j) = 1$. Connected here means that for every pair $(i_1, j_1) \in C$ and $(i_2, j_2) \in C$ there exists a sequence of neighbouring nodes with (i_1, j_1) being the first item in the sequence and (i_2, j_2) being the last.

1.2 PERCOLATION

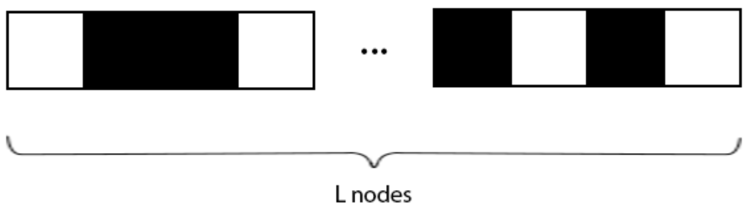

Figure 1: A 1D lattice of size L, with a few dead nodes

Percolation theory refers to the study of the structure of clusters formed in the lattice, given some choice of state map S. In standard percolation theory, one randomly (with probability p) assigns either state 0 or state 1 to each and every node in the lattice. Under these circumstances, the system presents a phase transition, when one varies the occupation probability p. There exists a critical value p_c such that for $p > p_c$, the infinite lattice is said to "percolate", i.e. clusters spanning the whole lattice appear. For the 2D square lattice, the threshold is $p_c = 0.59274621(13)$. This number changes for different choices of G and N.

Figure 1 and Figure 2 provide examples of percolation in 1D and 2D lattices.

In biology, percolation theory has been used to successfully predict the fragmentation of biological virus shells (REF), with the percolation threshold of Hepatitis B virus capsid predicted and detected experimentally (REF). In environmental science, percolation theory has been applied to studies of how environment fragmentation impacts animal habitats (REF) and models of how the plague bacterium Yersinia pestis spreads (REF).

Our goal will be to study the behavior of some quantities of interest associated with percolation models.

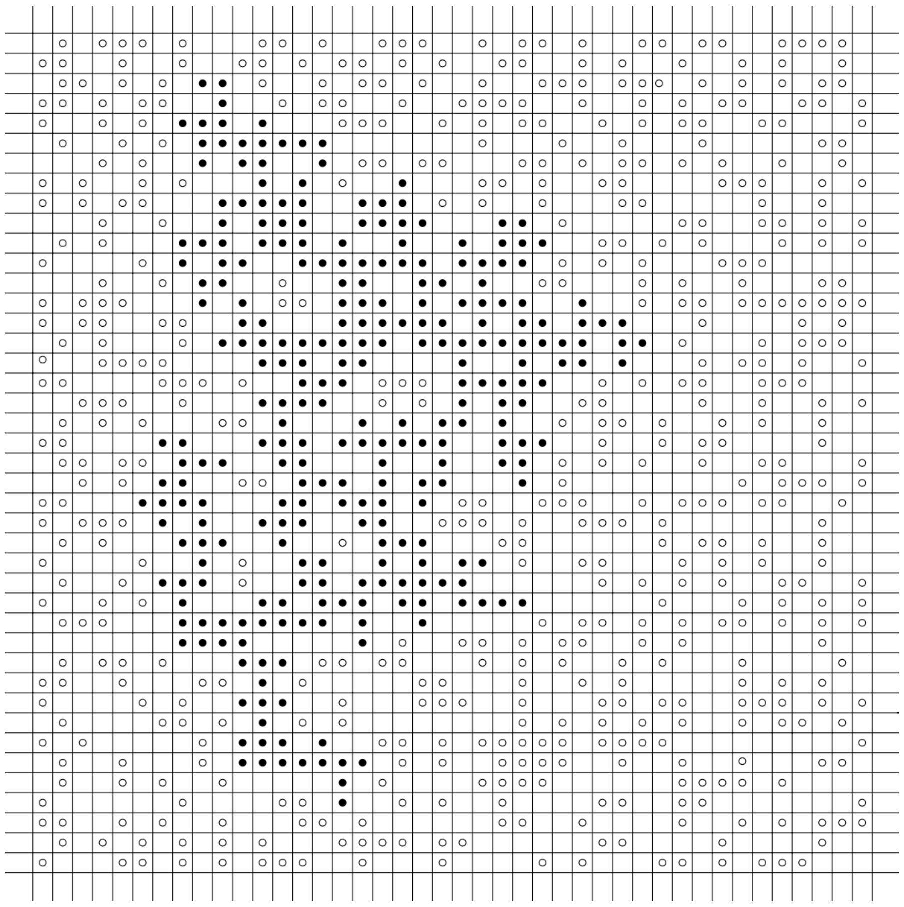

Figure 2: A finite piece of an infinite 2D lattice. The largest cluster is made up of the nodes with circles painted, while the smaller clusters are denoted by open circles.

1.3 PERCOLATING CLUSTER STRENGTH

The first interesting quantity we can look at is the percolating strength $P(p)$. This quantity represents the probability that a randomly picked node belongs to the percolating cluster. Another way to think of this quantity is that it represents the fraction of the whole lattice that is covered by the percolating cluster.

For small values of p, all islands are finite. Starting at any randomly site, it is impossible to get arbitrarily far away by walking only along the connected sites. As we increase p, the largest cluster becomes infinite at a precise value $p = p_c$. When this happens, it is possible to get arbitrarily far from a starting point in the infinite cluster by walking along the connected sites. For $p < p_c$, there are no infinite clusters, so $P(p)$ vanishes. However, for $p > p_c$, $P(p)$ monotonically increases.

For $1 \gg (p - p_c) > 0$, that is, for p greater but close to p_c, we have (REF 1):

$$P(p) \propto (p - p_c)^{\beta}$$

The qualitative behaviour of $P(p)$ can be seen in figure (FIG 1)

β is a universal critical exponent which depends only on the lattice dimension, and not on its geometry. In 2 dimensions, $\beta = \frac{5}{36}$, is the same for the square, hexagonal, or any other geometry (REF 1).

P(p) is one of the order parameters for percolating systems, since it is zero for $p < p_c$, and non-zero for $p > p_c$.

1.4 MEAN CLUSTER SIZE

Another interesting quantity is $\chi(p)$, the mean cluster size, which is the average number of nodes in among finite clusters. One can think of each node in the cluster as having one "unit of mass", in which case the mean cluster size represents the average mass. Just like $P(p)$, for $\chi(p)$, we have

$$\chi(p) \propto |p - p_c|^{-\gamma}, \quad \text{for } |p - p_c| \ll 1$$

The qualitative behaviour of $\chi(p)$ can be seen in Figure 3 .

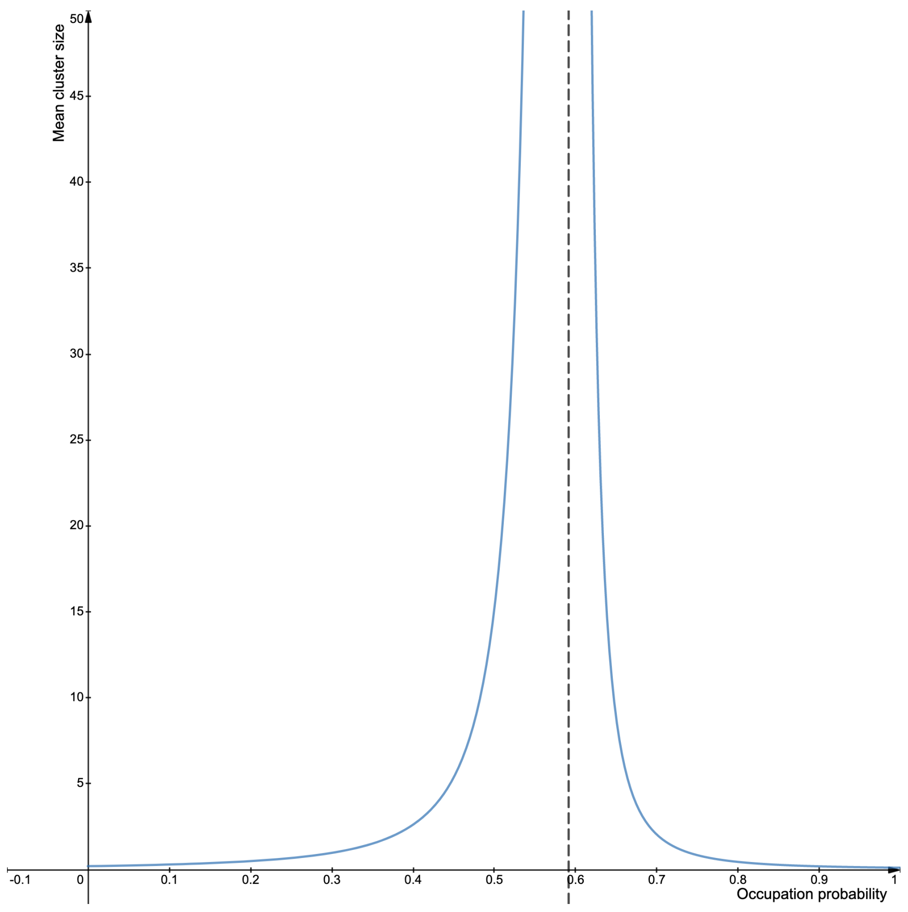

Figure 3: Percolation probability curve for various lattice sizes

Again, analogously to the percolating cluster strength $P(p)$, γ is a universal critical exponent, in the sense that it's independent of the local geometry, and depends only on the dimension of the lattice. In two dimensions, $\gamma = \frac{43}{18}$. One important difference between $\chi(p)$ and $P(p)$, however, is that $\chi(p)$ scales as $|p - p_c|^\gamma$ for both $p < p_c$ and $p > p_c$. The constants of proportionality are different in either side.

1.5 CORRELATION FUNCTION AND CORRELATION LENGTH

The correlation function $g(\vec{r})$ encodes how likely it is that two clusters separated by a displacement vector $\vec{r}$ are part of the same (finite) cluster. In 2D, based on rotational symmetry and in order to simplify the analysis, one often ignores the vector character of $\vec{r}$ and studies only $g(r)$, where $r = |\vec{r}|$.

For large values of r and $p \neq p_c$, the correlation function decays exponentially:

$$g(r) \propto e^{-\frac{r}{\xi(r)}}, \quad \text{for } p \neq p_c, \; r \gg 1$$

$\xi(r)$ is the so called correlation length. One interpretation is that it measures the size of correlations in the lattice, that is, the typical diameter of an island. With this interpretation, one does not need to consider distances larger than ξ: a finite piece of the lattice, larger than ξ, presents the same behavior as an infinite lattice.

This exponential form is only the leading factor in $g(r)$. At $p = p_c$, long range correlations appear, $\xi(p)$ diverges, and a power law leading factor appears:

$$g(r) \propto x^{-\eta}, \quad \text{for } p = p_c, \; r \gg 1$$

Once more, the critical exponent η is universal. In 2D, $\eta = \frac{5}{24}$.

When $p \neq p_c$, the correlation length defines a characteristic length scale. This is not the case when $p = p_c$: there's no characteristic length scale. The lattice is self similar and any finite piece is not enough to describe the behaviour of the system.

Near the threshold p_c, the correlation length, again, behaves like

$$\xi(p) \propto |p - p_c|^{-\nu}, \quad \text{for } |p - p_c| \ll 1$$

with ν the corresponding critical exponent, also universal. In 2D, $\nu = \frac{4}{3}$.

It is interesting to notice here the importance of considering only finite clusters: the correlation length remains finite both below and above the threshold p_c. As we increase p past p_c, finite clusters begin forming connections to the infinite cluster, and so the probability that two nodes belong to the same finite cluster (which is what $g(r)$ measures) decreases, and so does the correlation length.

1.6 CLUSTER SIZE DISTRIBUTION

Perhaps the most important quantity we'll look at is the cluster size distribution, $n(s, p)$. This function describes the s-cluster density at occupation probability p, that is, the number of clusters with size s and probability p divided by the size of the lattice (in the case of infinite lattices, this quantity can be thought of either as an average over many large finite pieces of the infinite lattice, or as a limit as $L \to \infty$. At p_c, the the cluster size follow a power law distribution:

$$n(s, p_c) \propto s^{-\tau}$$

where τ is another universal exponent (in two dimensions, $\tau = \frac{187}{91}$). Once more, the system lacks a characteristic length scale. Near p_c and for large s, $n(s,p)$ is a generalized homogeneous function of $p - p_c$ and $\frac{1}{s}$.

A function of two variables $F(x,y)$ is said to be generalized homogeneous function iff

$$F(c^{\alpha}x, c^{\beta}y) = cF(x,y) \quad \forall(c, \alpha, \beta)$$

In the case of $n(s,p)$, we have

$$n\left(\frac{c^{\alpha}}{s}, (p - p_c)c^{\beta}\right) = cn\left(\frac{1}{s}, p - p_c\right)$$

By choosing $c = s^{\frac{1}{\alpha}}$, one obtains

$$n(1, (p - p_c)c^{\beta/\alpha}) = n((p - p_c)c^{\beta/\alpha}) = s^{\frac{1}{\alpha}} n\left(\frac{1}{s}, p - p_c\right)$$

which allows us to conclude that

$$n(1/s, p - p_c) = s^{-1/\alpha} n\big((p - p_c)c^{\beta/\alpha}\big)$$

What this means is that by measuring $n(s,p)$ in units of $n(s,p_c)$, that is, by taking $\frac{n(s,p)}{n(s,p_c)}$ one does not need to consider it as a function of two variables p and s: it reduces to a function which depends only on the combination $(p - p_c)s^{\sigma}$ for some sigma. In 2D, $\sigma = \frac{36}{91}$, which is also a critical exponent.

One of the reasons why $n(s,p)$ is one of the most interesting quantities to look at is that the order parameter (the percolating cluster strength), the mean finite cluster size, the correlation length and other quantities can all be derived from it. For this reason, $n(s,p)$ is called a generating function.

The percolating cluster strength $P(p)$, for example, can be obtained from

$$P(p) = p - \sum_{s} s\, n(s,p) \tag{3}$$

$n(s,p)$ describes the s-cluster count divided by the size of the lattice. So $s\,n(s,p)$ can be viewed as the total area proportion covered by all clusters of size s. In other other, it represents the probability that by picking a node at random, it will be part of an s cluster. By summing over all (finite) values of s, we get the probability that a randomly picked node belongs to a finite cluster. Since p can be viewed as representing the probability of belonging to any cluster (since any alive cell will, necessarily, belong to a cluster, and any node belonging to a cluster must be alive, these are equivalent descriptions), the difference between p and $\sum_s s\,n(s,p)$ gives us the probability of belonging to an infinite cluster, which is how $P(p)$ is defined.

The mean cluster size $\chi(p)$ is related to $n(s,n)$ through

$$\chi(p) = \sum_s s^2\,n(s,p) \tag{4}$$

As noted before, $s\,n(s,p)$ represents the probability that a randomly picked node belongs to a (finite) s-cluster. Therefore, $\sum_s s^2\,n(s,p)$ represents the expected value of s.

2 CLASSICAL PERCOLATION - EXPERIMENTAL METHODS AND RESULTS

To experimentally verify the results described above, we ran more than **3 billion simulations**, generating approximately **250GB** of data. The code that used to run the simulation and to extract the statistics is available in Github, see Appendix 1 for more details. The full data is also available, see Appendix 2.

2.1 2D LATTICE SIMULATION

We used a $L \times L$ square, periodic lattice, which is topologically equivalent to a torus. Each node has four neighbours: one above it, one to the right, one below, and one to the left. The periodicity is represented by the fact that the right neighbors of the nodes in the rightmost column are the nodes in the leftmost column. Similarly, the left neighbours of the leftmost column are the nodes in the rightmost column, the top neighbours of the topmost row are the nodes in the bottom-most row, and the bottom neighbours of the bottom-most row are nodes in the topmost row.

Each node can be in one of two states: dead or alive. Initially, all nodes are in the alive state. With probability p, we change state of each individual node to dead. We then compute and store the set of clusters found in the lattice. A cluster is a set of connected nodes sharing the same state.

Each cluster can either be a finite or infinite. Since the lattice is periodic, **a cluster is considered infinite if its 1D extension in either direction is greater than or equal to the size of the lattice** L. Otherwise, it is a finite cluster. In other words, we imagine a bounding box around each cluster, and if at least of the dimensions of the box is greater than or equal to L, the cluster is infinite.

We generated around 3000 lattices for each pair (p, L), with p ranging from 0 to 1 (with a higher density near the critical points) and L ranging from 16 to 1024.

2.2 PERCOLATION PROBABILITY

A lattice has percolated if it contains at least one infinite cluster. For each (p, L), we count the number of percolating lattices and divide by the total number of lattices simulated to compute the percolation probability.

In Figure 4, we plot the percolation probability against the occupation probability p for various lattice sizes.

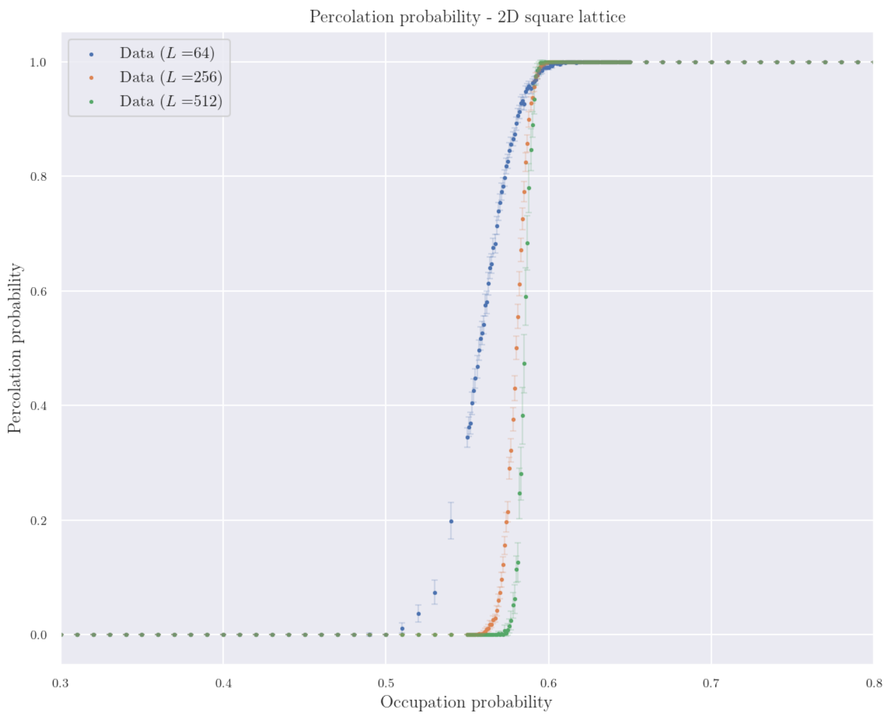

Figure 4: Percolation probability curve for various lattice sizes

By visual inspection, we can hypothesize that the curve is a hyperbolic tangent curve:

$$y(x) = \tanh(x)$$

We can shift and scale this function so that it's centered at $x = \beta$, and such that it's domain matches our data, i.e. $\mathcal{D}(y) = [0, 1]$. We'd also like a parameter that controls how steep the transition is between $y = 0$ and $y = 1$ - this can be done by scaling x by a multiplicative factor α. The function becomes

$$y(x) = \frac{\tanh(\alpha(x - \beta)) + 1}{2}$$

In Figure 5, we can see the the fitted data points. Table 1 contains the precise values.

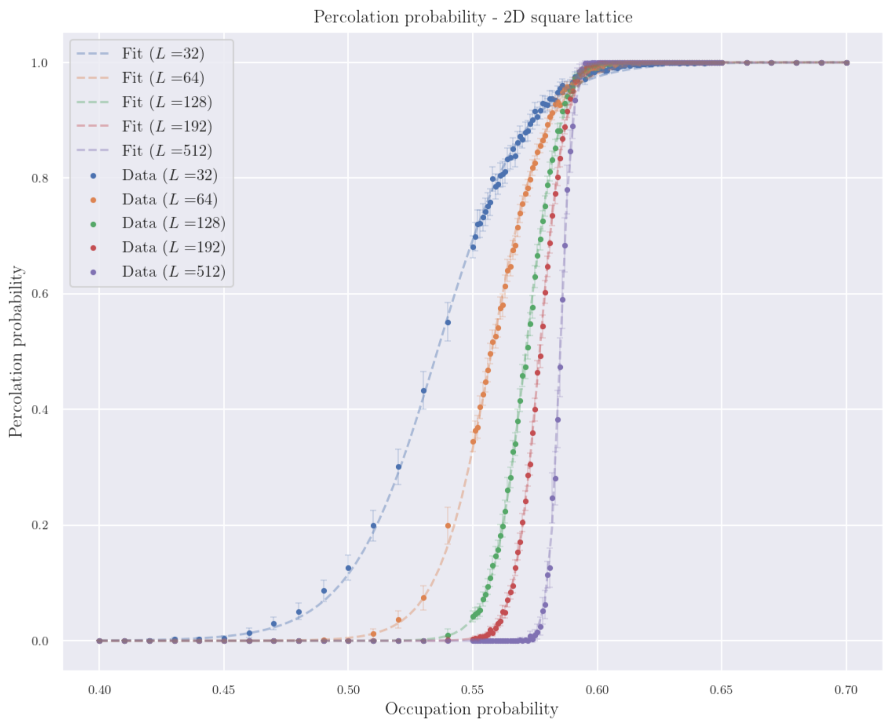

Figure 5: Percolation probability with a hyperbolic tangent LMS fit

We notice that the the bigger the lattice, the faster it jumps from $y = 0$ to $y = 1$. This is consistent with the widely known fact that as L increases, i.e. in the limit $L \to \infty$, the transition becomes a step function at $p = p_c$.

Another interesting fact to notice is that with the structure of the function we're fitting, β represents the point at which the percolation probability first reaches $\frac{1}{2}$.

These two observations suggest that studying the behaviour of α and β as a function of the lattice size L might be interesting. We can see the results obtained in Figure 6, in a so called finite-size scaling analysis.

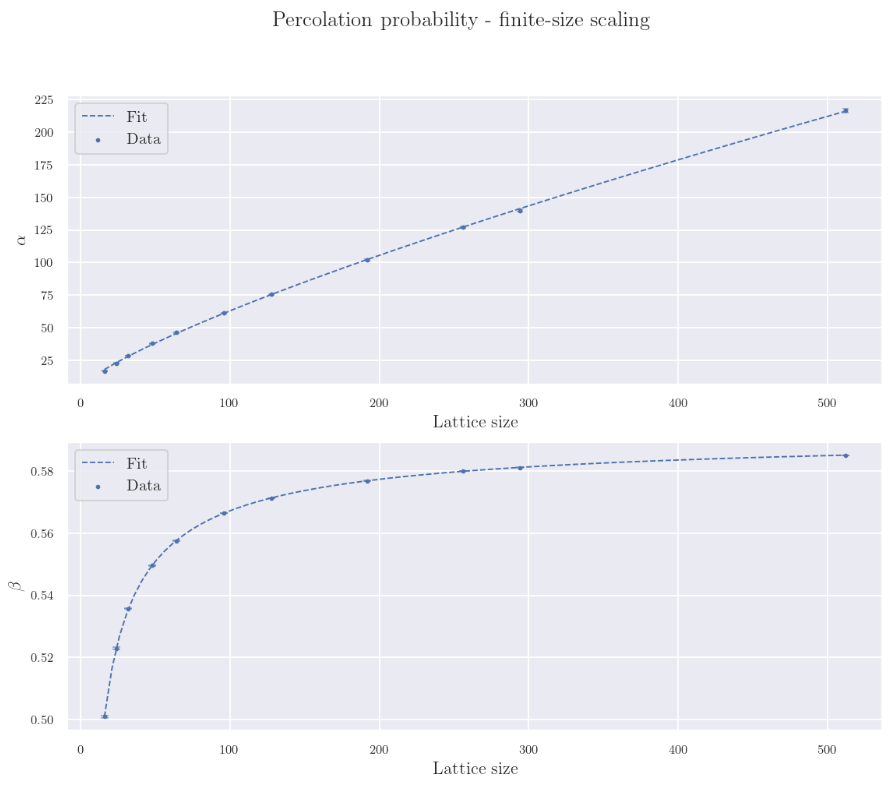

Figure 6: Finite-size scaling analysis of the hyperbolic tangent function

We observe a monotonic growth in α as a function of the lattice size L. We were able to get a reasonable fit by using the ansatz

$$\alpha(L) = aL^n + b$$

The precise values for the constants a, n an b can be seen in Table 2.

Of course,

$$\lim_{L\to\infty} aL^n + b = \infty \qquad a > 0, n > 0$$

Since α controls how steep the hyperbolic tangent curve is, that is, how fast it goes from 0 to 1, this supports the hypothesis that as we increase L, the curve gets arbitrarily close to a step function.

The parameter β, on the other hand, behaves differently. We were able to get a fairly good fit by using the ansatz

$$\beta(L) = c - ae^{\frac{L^n}{b}}$$

The precise values for a, n, b an c can be seen in Table 3.

It is clear that

$$\lim_{L\to\infty} c - ae^{\frac{L^n}{b}} = c$$

Since β represents occupation probability at which the percolation probability reaches 1/2, the limit above allows us to estimate p_c, the critical probability of our system - we do so by interpreting p_c as β in the limit of an infinite lattice. This value comes out to be $\mathbf{0.5907(2)}$.

2.2.1 *Errors*

The error estimates for the percolation probability were computed by treating it as a Bernoulli random variable. We used a 99% confidence interval and the expression

$$\text{Error} = z\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}$$

where z is the value coming from the z-table for the desired confidence interval (numerically, $z = 2.576$), $\bar{p}$ is the mean of the observations, and n is the number of observations[**bertsekas2008introduction**].

2.3 PERCOLATING CLUSTER STRENGTH

The percolating cluster strength represents the probability that a randomly picked node belongs to a infinite cluster. This was numerically approximated by counting the number of nodes belonging to the (possibly multiple) infinite clusters and dividing by the total number of nodes. The results obtained can be seen in Figure 7.

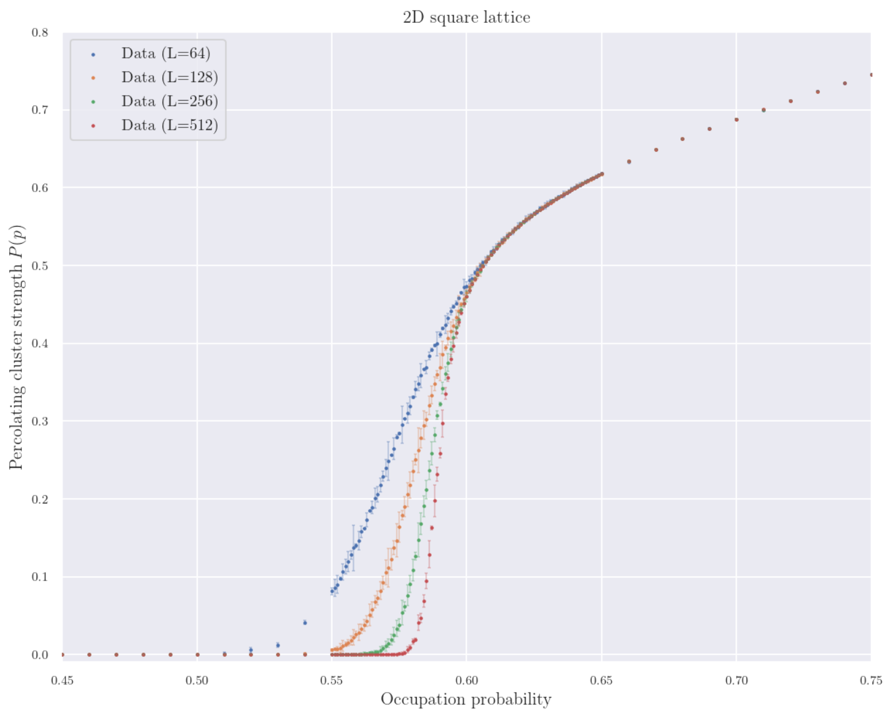

Figure 7: Percolating cluster strength versus occupation probability for various lattice sizes.

As discussed in Section 1.3, we expect to see the percolating cluster strength $P(p)$ go from being zero for $p < p_c$ to a non-zero value for $p > p_c$. This is precisely what we see in Figure 7. It's also possible to notice that the transition becomes more abrupt for larger lattice sizes, which provides evidence for intuition that this transition approaches a step function as $L \to \infty$. Unfortunately, we were not able to find an ansatz that approximates this curve well, and therefore can't provide at this time a convincing numerical estimation for the "abruptness" of this transition or how fast it approaches the step function.

2.3.1 *Errors*

The errors in the calculation of $P(p)$ were estimated by splitting all observations for each p (around 2000) into 20 bins, computing the mean value for $P(p)$ in each bin, and then computing the standard deviation mean values obtained across the different bins.

2.4 MEAN CLUSTER SIZE

The mean cluster size $\chi(p)$ represents the average number of nodes in the finite clusters as defined in Section 2.1. For each simulation, we loop over the finite clusters, adding their sizes. We then divide the sum by the number of finite clusters to get the average in a particular simulation/lattice. Afterwards, we average again, this time over lattices (of course, this averaging is only done over lattices with the same occupation probability p and size L).

The results can be seen in Figure 8.

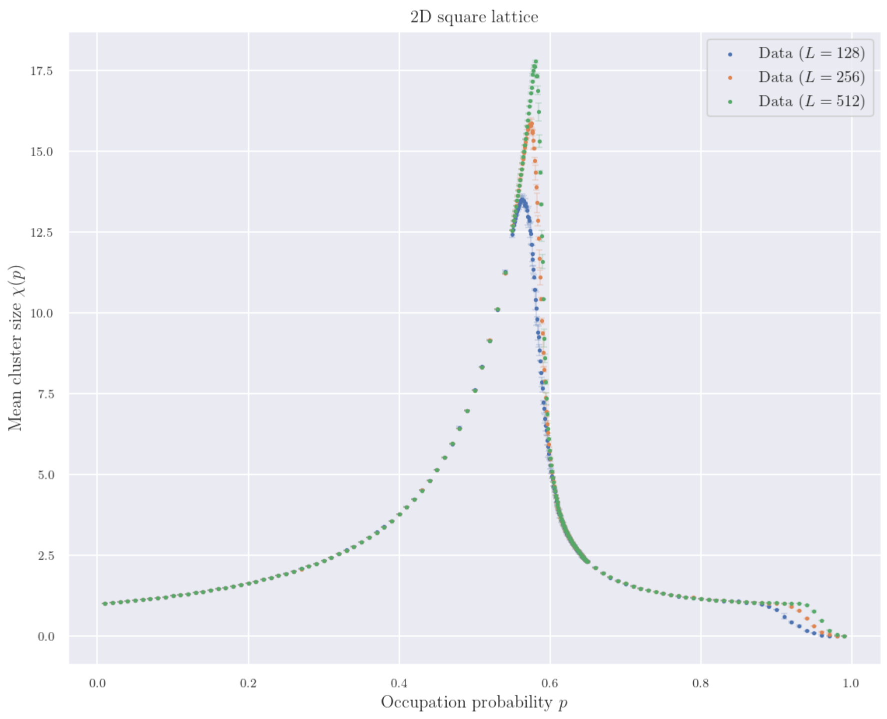

Figure 8: Mean finite cluster size versus occupation probability for various lattice sizes

As expected, we see a monotonic increase in $\chi(p)$ up to a maximum value, corresponding to some sort of critical value, and then a somewhat faster, monotonic decrease after the maximum. There is also an interesting kink close to $p = 0.9$ which we did not investigate further. The presence of the peak immediately presents itself as an opportunity to check some of the facts presented in Section 1.4.

Next, we study the peak position and height for each lattice size. Our goal is to make an educated guess on how these values change as we increase the lattice size. To estimate the peak height and occupation probability, we fitted a parabola to the 10 points closet the numerical peak (the point for which the height is maximum).

Figure 9 shows the peak height H as a function of the lattice size we obtained from our data, and Table 4 shows the precise values obtained.

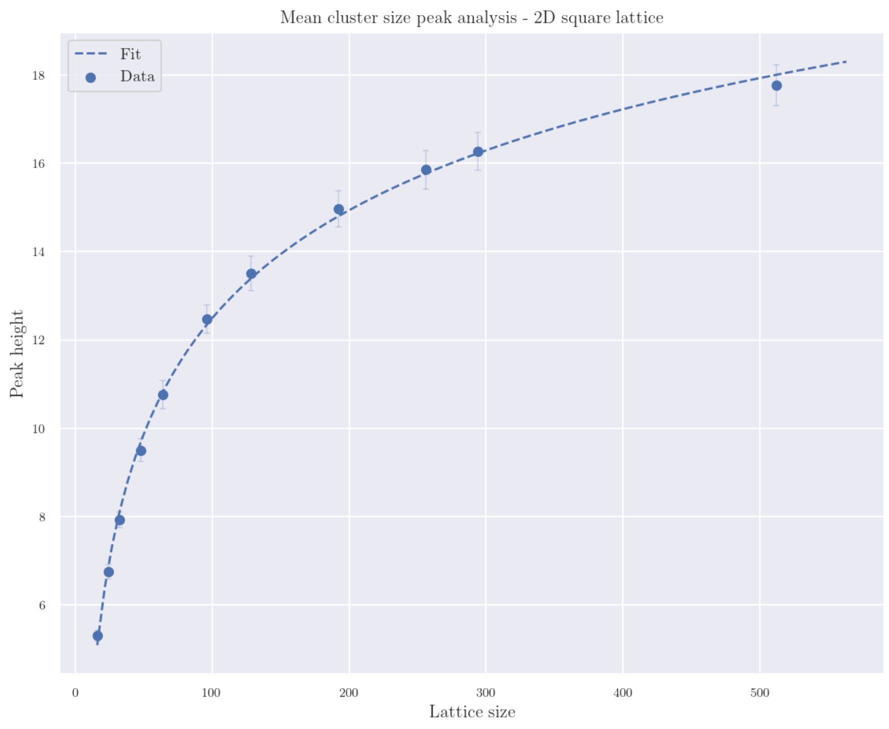

Figure 9: Mean cluster size, peak height for various lattice sizes

As expected, the height grows with the lattice size. The ansatz used to fit the curve is

$$H(L) = a\log(bL)^c + d$$

The precise values of a, b, c and d can be seen in Table 6. Since

$$\lim_{L\to\infty} a\log(bL)^c = \infty$$

This data supports the hypothesis that as we increase L, the peaks height does not converge to any finite height, and instead diverges.

Next, we consider the peak position K as a function of the lattice size L. While the peak height is expected to diverge, as discussed above, one expects that the peak probability would converge the true percolation threshold p_c as L increases. Figure 10 shows the probability at which the peak is located, K, as a function of lattice size, and Table 5 shows the numerical values.

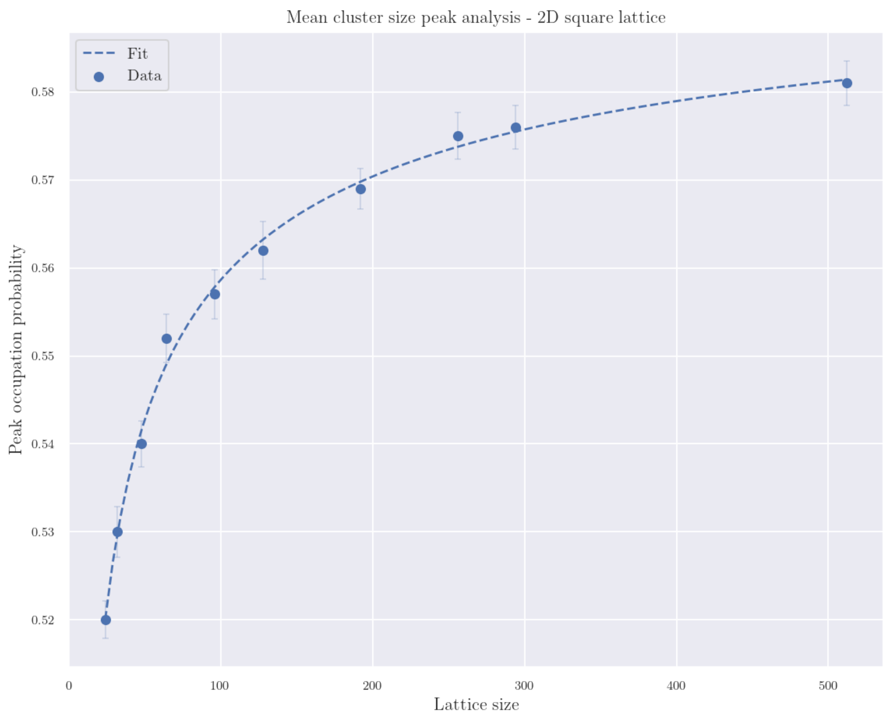

Figure 10: Mean cluster size, peak position for various lattice sizes

The ansatz used in the fit above is again

$$K(L) = c - ae^{\frac{L^n}{b}}$$

So we conclude that this is, as predicted, evidence for the percolation threshold converging to a finite numerical value (namely, c). The precise values for all the fit parameters can be seen in Table 7. Using this method, we estimate p_c to be **0.5986(4)**.

2.4.1 *Errors*

The errors in the calculation of $\xi(p)$ were estimated by splitting all observations for each p (around 2000) into 20 bins, computing the mean value for $P(p)$ in each bin, and then computing the standard deviation mean values obtained across the different bins. For the peak position and height, we errors were computing by taking the square root of the relevant diagonal term in the covariance matrix obtained from the least squares fit.

2.5 CORRELATION FUNCTION AND CORRELATION LENGTH

Lastly, we turn our attention to the study of the correlation length. These are a quantities of great interest in general. Apart from being relevant in the study of many phenomena, they also have an intuitive interpretation: the correlation function, for example, can be used to assess the distance required between nodes for the values to be effectively uncorrelated. This distance is generally called the correlation length.

For each lattice simulation, the correlation function was estimated by selecting a large number (approximately 200000) of random node pairs from the lattice, computing the distance between the nodes in the pair, and then checking whether they belong to the same finite cluster. Averaging over all selected pair a given distance r away, this gives us an estimate of the probability of nodes whose distance is r belonging to the same finite cluster, which is the definition of the correlation function $g(r)$. We then average this over many lattices.

To estimate the correlation length ξ, we use the fact that

$$g(r) \propto r^{-\nu} e^{-\frac{r}{\xi}}$$

Which covers both the case in which $p = p_c$, and the case in which $|p - p_c| \ll 1$. It is somewhat surprising that this also provides a good approximation for the correlation function when even $|p - p_c| > 1$.

The results for the correlation function can be seen in Figure 11, along with the best fit using the ansatz above. Notice that this plot uses a logarithmic scale in the vertical direction. We give a higher weight to points with bigger r - the reason for this is that the approximation gets better the bigger r is (our goal is to approximate ξ, and its influence is only relevant for larger values of r). So even though the fit is not great for smaller values of r, this allows us to see the exponential decay, represented by the fact that the points are fitted to a line, fairly well.

One interesting fact to mention here is that we get much better results if we use the Manhattan distance, as opposed to the Euclidean distance. The reason for this is not completely clear, but we hypothesize that this is because the Manhattan distance better matches the geometry of the lattice - a "diagonal neighbour", for example, is really two time further away than a nearest neighbour, as opposed to $\sqrt{2}$ times further away, which is what the Euclidean distance would give us.

Figure 12 shows the correlation lengths obtained for multiple lattice sizes and occupation probabilities.

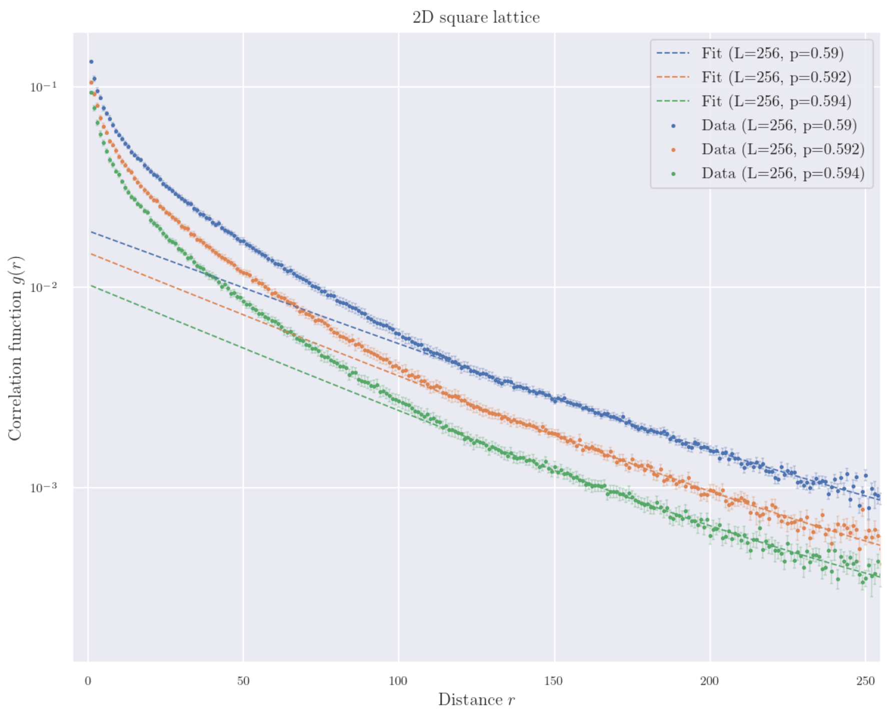

Figure 11: Correlation function for various sizes and occupation probabilities logarithmic scale in the vertical direction)

There's a couple of interesting things to notice in Figure 12. The first is that errors are greatly increased to the right of the peaks. This happens because the errors in the correlation function increase when the occupation probability is in that range - and that, in turn, happens because as we increase the occupation probability p past the p_c, many smaller merge clusters merge into the infinite cluster. So it gets much harder to find two pairs of nodes that are not part of the infinite cluster - essentially causing to correlation function to become closer and closer to being identically zero.

More interestingly, though, is to compare the numerical values for correlation length with the square root of the mean cluster size. The square root is necessary since the mean cluster size is closer to an area, since it gives us the average number of nodes in a cluster. By taking its square root, we obtain some kind of "average linear size". By comparing this with the correlation length, we can see that the difference is bigger closer to the peaks, indicating finite size artifacts start to appear. Farther from the peak (which is also roughly the critical occupation probability), these two values are close to each other, indicating that the size of the lattice is sufficiently large so as to be representative of an infinite lattice.

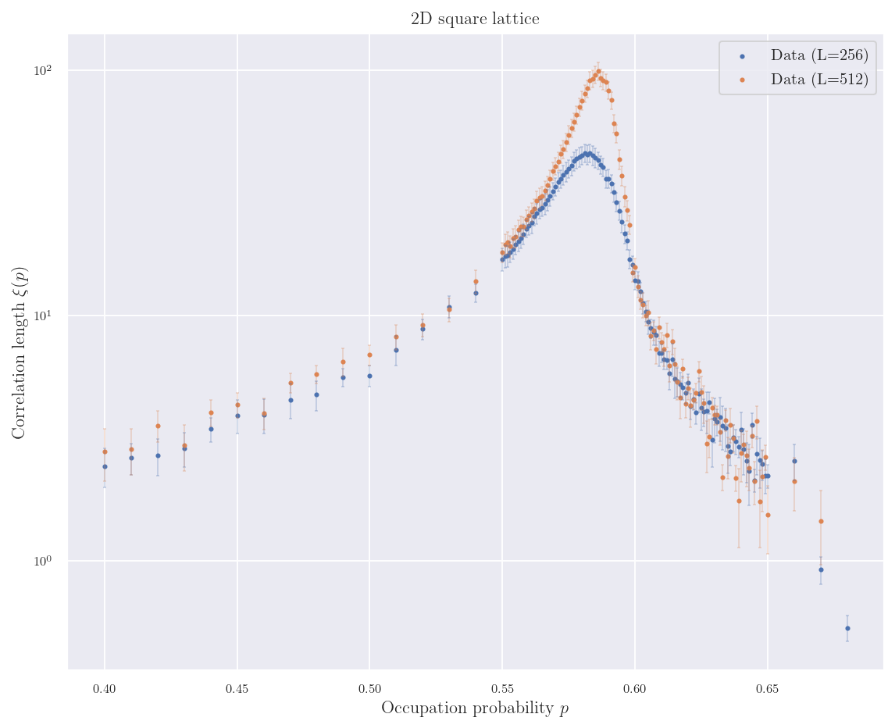

Figure 12: Correlation length for multiple lattice sizes (logarithmic scale in the vertical direction)

With this, we finish our analysis of the classical percolation model. Next, we'll look at the so called Mandelbrot percolation model.

Part II

MANDELBROT PERCOLATION

3 MANDELBROT PERCOLATION - METHODS AND RESULTS

3.1 INTRODUCTION

Now we turn our attention to the so called Mandelbrot percolation, which is also known in the literature as Fractal percolation. The idea is, again, to study the clusters structure and distribution, but the coloring of the lattice happens in a different way: instead of doing a single pass over the lattice and coloring the nodes at random, we do this process repeatedly - at each step, we subdivide each node in the lattice into a number of "sub-nodes" and color the remaining "alive" (not yet colored) nodes with probability p. An illustration of this process can be seen in Figure 13. Each iteration generates a lattice A_n (which is, of course, highly correlated with the previous lattice A_{n-1}). Mandelbrot percolation is the study of the limit of this process the number of steps increases arbitrarily, that, is $\lim_{n\to\infty} A_n$.

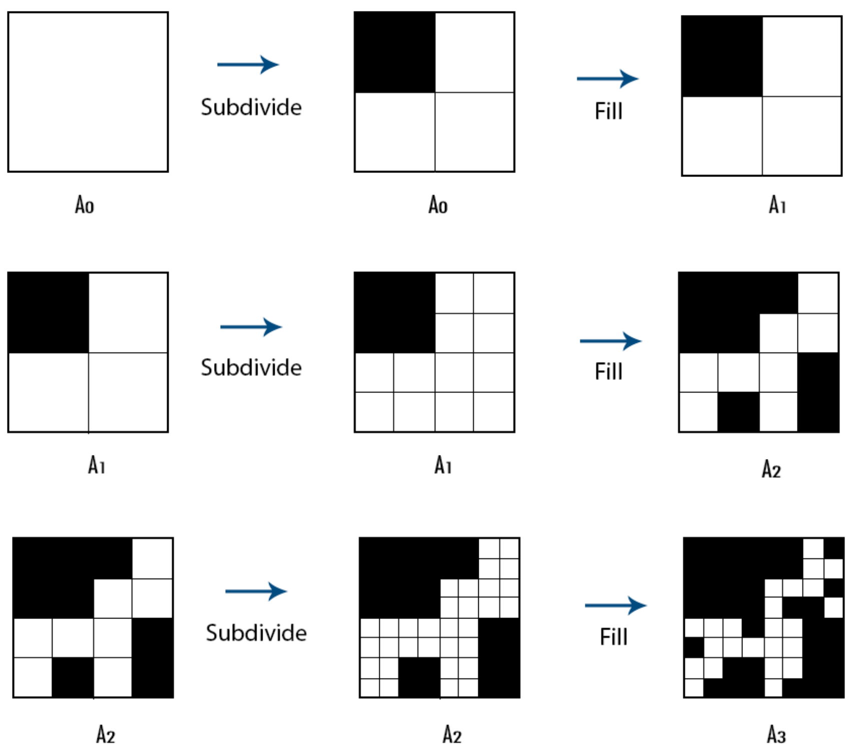

Figure 13: Illustration of the Mandelbrot percolation iteration process, showing three iterations

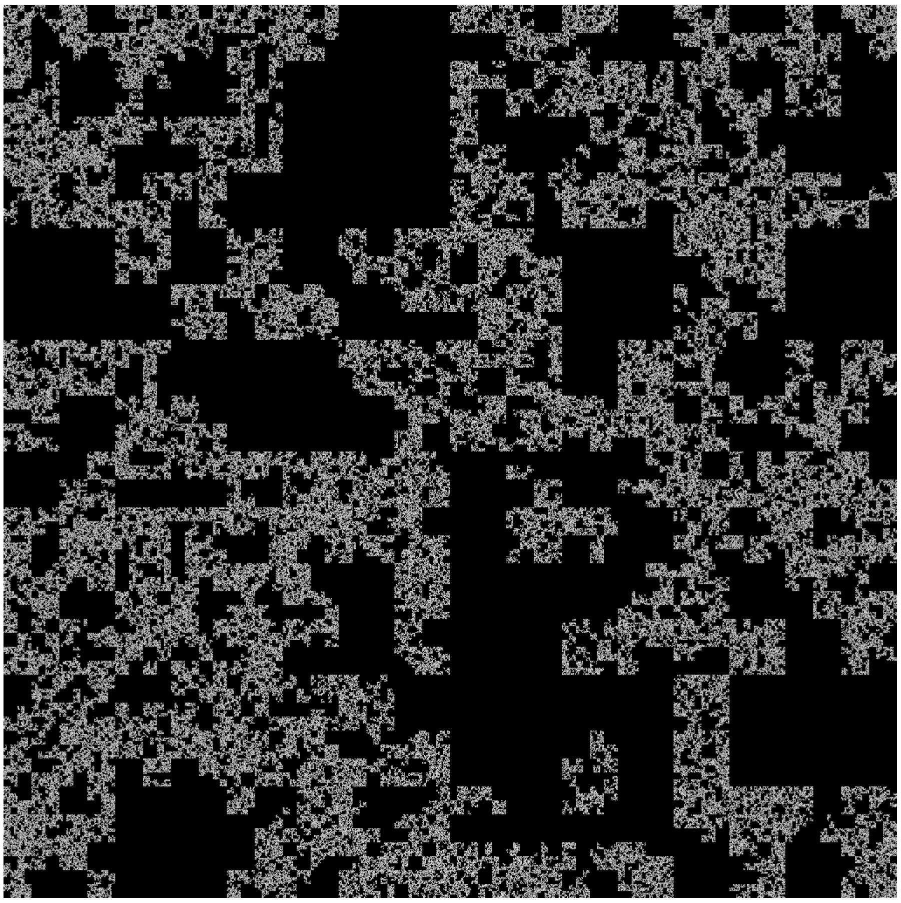

Figure 14: A Mandelbrot percolation lattice after 11 iterations

3.2 SIMULATION METHODS

The simulations were run, again, on a 2D periodic square lattice. We studied three properties: the percolation probability, mean cluster size and percolating cluster strength. We explicitly used exactly the same definitions, methods and algorithms as for the 2D case - the reason for this is that this way, a direct comparison is possible. The precise description of these methods is available in Chapter 2.

3.3 PERCOLATION PROBABILITY

In Figure 15, the first thing we notice is that the the critical probability p_c is much lower than in the classical percolation model - at first sight, it's clear that $p_c < 0.3$. The reason for this is somewhat obvious. For a given (p, L) pair, in classical percolation, we do a single pass over the lattice, whereas in the Mandelbrot model, we have done several passes previously, when the lattice was smaller. So on top of making multiple passes, the previous passes happened when each node represented a bigger part fraction of the lattice.

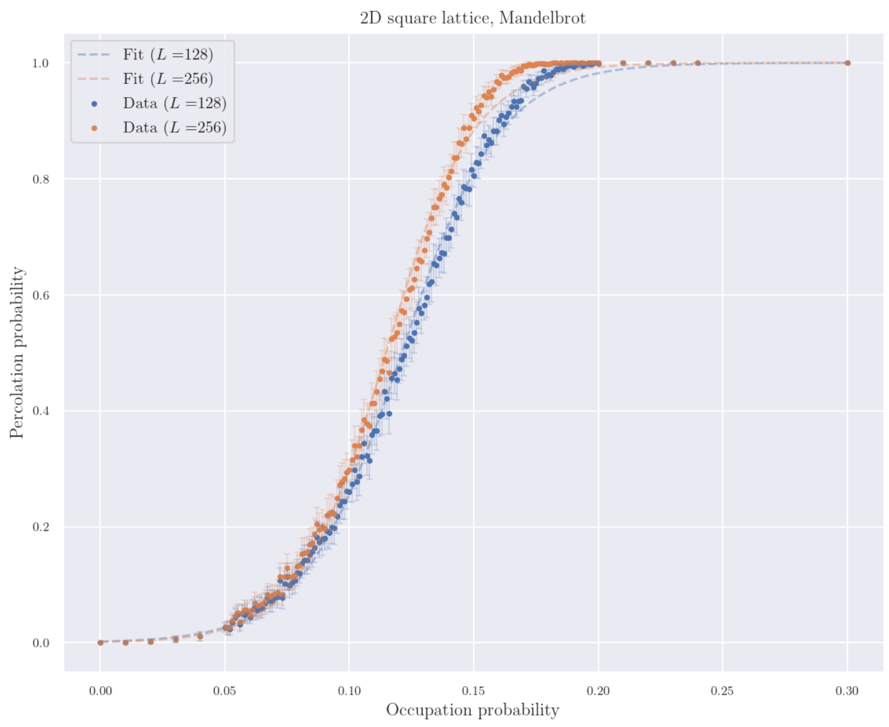

Figure 15: Percolation probability in the Mandelbrot percolation model

Just like in classical percolation, we fitted a hyperbolic tangent function to the data, and for $L = 256$, which is the largest size we simulated for the Mandelbrot model, we find that the the percolation probability reaches the 0.5 mark when $p =$ **0.1273**. We point out that the fit is noticeably worse for the Mandelbrot model than in the case of classical percolation, which suggests an underlying fundamental difference between the models. Additionally, we notice that plots obtained for $L = 128$ and $L = 256$ are in relation to each other in a different way than the equivalent curves in the classical percolation case. In comparison with Figure 5, for example, one notices that whereas there, the curves have a sharp increase at very different occupation probabilities, here they are much closer to each other for low values of p, and remain much closer. We hypothesize that this happens, again, because the lattices are much more closely related here - the additional pass used in the $L = 256$ happens at a smaller length scale and therefore has a less pronounced effect on the curve. The errors were computed using the same method as described in Section 2.2.1.

3.4 PERCOLATING CLUSTER STRENGTH

The percolation cluster strength results for the Mandelbrot percolation simulations can be seen in Figure 16. The curve looks qualitatively pretty similar to the one we had for classical percolation (see Figure 7).

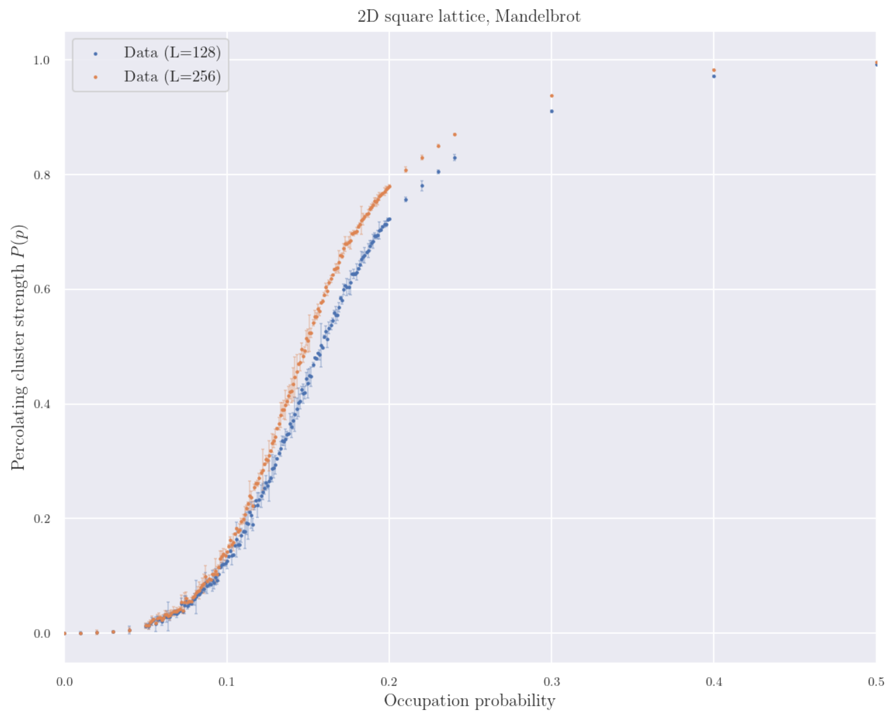

Figure 16: Percolating cluster strength for the Mandelbrot percolation model

Other than the lower values for the critical occupation probability discussed above, another interesting fact worth pointing out is that although for both models the curve becomes essentially a linear function towards higher values of the occupation probability p, the slope of the curve is noticeably smaller for the Mandelbrot case. The reason for this is not immediately obvious and warrants further investigation. The errors were computed using the same method as described in Section 2.3.1.

3.5 MEAN CLUSTER SIZE

The results for the mean cluster size can be seen in Figure 17.

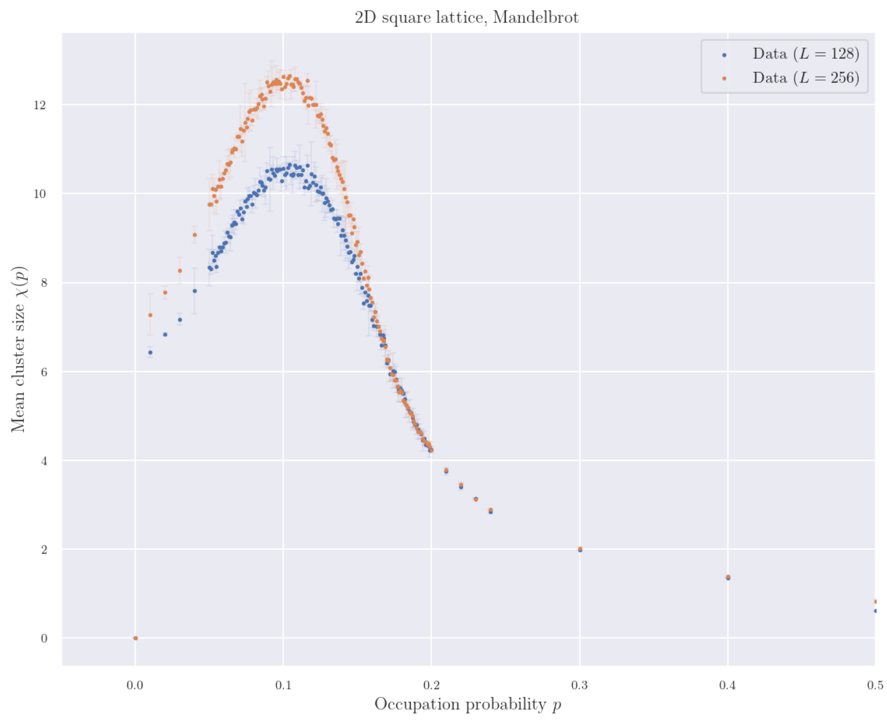

Figure 17: Mean cluster size in the Mandelbrot percolation model

In comparison with the classical percolation plots (see Figure 8), it is curious that the peak heights are smaller for the Mandelbrot percolation. The reason for this is, again not immediately obvious - the reader is invited to investigate this further. The errors were computed using the same method as described in Section 2.4.1.

3.6 CONCLUSION

Even in the case of classical percolation, which has been widely studied by many people in the last 50 years, we hope that this work provides a solid numerical and computation basis upon which further investigation can be made. We provide both the data used in these analysis and the code used to generate it in hopes of facilitating collaboration and reuse, as well as inspiring a more modern approach to computational research in general.

For the case of Mandelbrot percolation, this work is intended as a first step towards computational analysis of the model, in particular emphasizing differences and similarities as compared to the classical percolation models. While we didn't intend to provide a through and careful theoretical justification of the results obtained, this is encouraged for the curious reader.

The author is happy to answer any questions - feel free to email `alansammarone@gmail.com`.

Part III

APPENDIX

A CODE

A.1 OVERVIEW

Musk is a Python library developed as part of this work to help us run and store simulations and statistics in a distributed, highly parallel fashion. The code is available in Github[1].

To achieve an arbitrary amount of parallelism, Musk makes use of an independent queue service. The current version uses Amazon SQS[2], but the code is easily extensible to make use of other popular queue services, such as RabbitMQ or Kafka. The general idea is that there a script (referred to as the enqueuer) which sends messages to a specific queue, and each message represents a particular computation one wishes to perform. This is what a message looks like (in terms of data structures, it is just a Python dictionary):

```
{
    "parameters": {
        "probability": 0.553,
        "size":256,
        "repeat":128
    }
}
```

This particular message encapsulates the parameters of the simulation we want to run - the occupation probability, the lattice size, and how many time we want to repeat the process. In general, one generates thousands of messages like this and sends them to a queue (which is particular to a specific type of model - 2D classical percolation and Mandelbrot percolation, for example, have different queues), where they sit and wait further processing. We also have messages that represent instructions to compute statistics for a particular simulation, such as mean cluster size and percolation probability, but they follow the same general structure.

We also have a script called the dequeuer. It can (and should, in general) run on a number of separate machine and it's job is to read from messages from a queue and process them.

1 https://github.com/alansammarone/musk
2 https://aws.amazon.com/sqs/

Since it is completely separate from the enqueuer, we can horizontally scale the system as much as like - during our simulations, we used up to 30 virtual machines running 2 instances of dequeuer each, for a total of 60 dequeuer processes constantly (once every 2 seconds) reading the queues and in case a message is found, processing it and then deleting it.

To store data, we use the popular open source database called MySQL[3]. Each simulation is stored as a row, containing the parameters as well as clusters on the lattice after the simulation is run. The statistics (mean cluster size, percolating cluster strength, etc) are stored as rows on a different table.

In general, the process to run the simulations and statistical analysis is the following:

1. Run the simulation enqueuer with the appropriate parameter range configuration to generate messages representing the simulations one wishes to run

2. Run the simulation dequeuer (potentially on multiple machines) and wait for all the computation to finish (depending on the number of messages, this could take weeks). This is generate many rows in the MySQL database.

3. Run the statistics enqueuer, which will read the rows written in the previous step and generate one message per row

4. Run the statistics dequeuer, which will read the messages sent to the queue in the previous step, compute the desired statistics, and write the results in a separate table.

A.2 CLASSES AND SCRIPTS

The library is made up of several classes, each having a particular responsibility and implementation. We hope this section can serve as inspiration for anyone interested in using this library and/or coding their own distributed processing framework.

- **Lattice classes** (`Lattice`, `Square2DLattice`, `Square2DPeriodicLattice` and

3 https://www.mysql.com/

Square2DFiniteLattice)[4]: These represent the lattices, in increasing order of specialization (via inheritance). They contain operations such as neighbour resolution, filling the lattice randomly, subdividing, and cluster computations.

- **Model classes** (PercolationModel, PercolationStatsModel, P2MModel and P2MStatsModel)[5]: Models contain the interface to the database - each model class represents a single database table, and contains one attribute for each column in the corresponding table. The class also contain SQL queries for reading/writing. An instance of this class represents a row.

- **Simulation classes** (Simulation, PercolationSimulation and P2MSimulation)[6]: Simulation classes use the lattice classes to actually execute the simulations, and then use the model classes to store and/or retrieve the results to/from the database.

- **Processor classes** (Processor, PercolationProcessor and PercolationStatsProcessor, PercolationStatsProcessor, P2MProcessor)[7]: Processors are responsible for processing a queue message (and then, if the processing was successful, deleting the message). In the case of simulations, they delegate the bulk of the work to the Simulation classes described above. In the case of statistics, they contain the statistics computations directly.

- **Queue-related classes** (Queue, Message, SQSQueue, SQSMessage)[8]: These are responsible for various aspects of the queue system - creating and updating

4 https://github.com/alansammarone/musk/tree/master/musk/lattices
5 https://github.com/alansammarone/musk/tree/master/musk/percolation
6 https://github.com/alansammarone/musk/tree/master/musk/percolation
7 https://github.com/alansammarone/musk/tree/master/musk/percolation
8 https://github.com/alansammarone/musk/tree/master/musk/core

queues, as well as sending, reading and deleting messages from them.

- **Dequeuer class** (`Dequeuer`)[9]: This class receives an instance of the `Queue` class and an instance of a `Processor` class. Is is then responsible for reading messages from the queue and appropriately routing them to the processor instance.
- **Dequeuer runner class and script** (`DequeuerRunner`)[10]: The `DequeuerRunner` class collects a set of dequeuer instances (one for each queue-processor pair) and is responsible for scheduling the execution of each. This includes spawning new processes to handle messages after a given threshold is reached (this is necessary since Python contains inherent memory leaks that are very hard to deal with - it's simpler to just rerun the interpreter in a new process). There is also a `run_dequeuer.sh` script which is responsible for actually running the main Python file with correct environment variables and permissions.
- **Enqueuer script** [11]: This script is responsible for generating and sending messages to their appropriate queue for actual processing (which is done by the dequeuer classes described above).

A.3 DEPLOYMENT

Musk can be deployed quickly to any number of worker nodes (the deploy happens in parallel, so deploying to 100 workers takes around the same time as deploying to a single worker), each of which will process messages and write to the database independently of the others. To accomplish this, we use Ansible[12], an open source automation tool supported by the Red

9 `https://github.com/alansammarone/musk/blob/master/musk/core/dequeuer.py`
10 `https://github.com/alansammarone/musk/blob/master/scripts/dequeuer_runner.py`
11 `https://github.com/alansammarone/musk/blob/master/scripts/enqueuer.py`
12 `https://www.ansible.com`

Hat foundation. After configuring the inventory file, one simply needs to run the playbook filefile[13] (using `ansible-playbook`), which will then, in all worker nodes, install the proper version of Python, all the required dependencies, set up configuration files, clone the Github repository, and set up a cron job to watch the queues (which is run every minute, but this can easily be changed in the configurations).

13 https://github.com/alansammarone/musk/blob/master/deploy/worker.yaml

B DATA

We provide the simulation data used in this work as a `.sql` file[1], which is a MySQL database dump. If this file is not available at the time of reading, simply contact the author at the email address provided at the beginning of the thesis.

Below, for every table, we describe the fields and their data types. Each row corresponds to a particular simulation sample. The tables contains several dozen million rows each.

- Table **p2s**: This table contains the simulation data for the classical 2D square percolation model
  - **id** int: A primary key which uniquely identifies each row
  - **probability** float: The occupation probability used in this simulation
  - **size** smallint: The size of the lattice - note that a 2D square lattice with a given size has size^2 nodes.
  - **observables** mediumblob: Contains a bz2-compressed list of clusters present in the lattice
  - **took** float: How long the simulation took
  - **created** timestamp: The time at which this simulation was created.
- Table **p2s_stats**: This table contains the statistics for every simulation in the **p2s** table. There is a one-to-one relation between every row in both tables (this is encoded in the **simulation_id** column described below).
  - **id** int: A primary key which uniquely identifies each row
  - **simulation_id** int: A foreign key pointing to the equivalent row in the **p2s** table from which these statistics were computed.
  - **probability** float: The occupation probability of the simulation associated with this row

1 `http://tiny.cc/muskdatadump`

  - **size** smallint: The size of the lattice used in the simulation associated with this row
  - **has_percolate d** tinyint: Whether the associated lattice has percolated
  - **cluster_size_histogram** text: A histogram of the cluster sizes in the associated lattice
  - **mean_cluster_size** float: Average size of the clusters in the associated lattice
  - **correlation_function** longtext: Correlation function approximation in the associated lattice
  - **percolating_cluster_strength** float: Percolating cluster strength in the associated lattice (0 if the lattice has not percolated).

- Table **p2m**: Exactly the same structure as the **p2s** table, but each row contains simulation data for a 2D Mandelbrot simulation.
- Table **p2m_stats**: Exactly the same structure as the **p2s_stats** table, but contains the statistics for the Mandelbrot simulations stored in the **p2m** table.

C TABLES

Size	α	β
16	17.316	0.502909
24	22.9593	0.523217
32	28.7445	0.535866
48	37.8221	0.549603
64	46.2392	0.557582
96	61.2356	0.566495
128	75.4226	0.571311
192	102.451	0.576883
256	126.736	0.579948
294	139.832	0.581133

Table 1: Hyperbolic tangent least squares fit parameters

Parameter	value	standard deviation
a	1.56	0.08
b	4.3	0.83
n	0.786	0.0088

Table 2: Percolation probability, α finite-size scaling parameters

Parameter	value	standard deviation
a	38.2	44.12
b	0.22	0.053
c	0.590	0.00074
n	0.10	0.02

Table 3: Percolation probability, β finite-size scaling parameters

Size	Peak height
16	5.31 ± 0.162
24	6.75 ± 0.135
32	7.93 ± 0.203
48	9.5 ± 0.197
64	10.76 ± 0.249
96	12.48 ± 0.283
128	13.51 ± 0.272
192	14.96 ± 0.294
256	15.86 ± 0.298
294	16.27 ± 0.302
512	17.78 ± 0.392

Table 4: Mean cluster size, peak height

Size	Peak height
24	0.52 ± 0.032
32	0.53 ± 0.053
48	0.54 ± 0.01
64	0.552 ± 0.051
96	0.557 ± 0.044
128	0.562 ± 0.025
192	0.569 ± 0.03
256	0.575 ± 0.049
294	0.576 ± 0.024
512	0.581 ± 0.02

Table 5: Mean cluster size, peak positions

Parameter	value	standard deviation
a	2.02558273e+03	3.19001498e+06
b	1.58092558e+02	9.87273040e+03
c	8.59573639e-03	6.62713676e+00
n	-4.09236591e+03	3.19134574e+06

Table 6: Mean cluster size, peak position finite-size scaling parameters

Parameter	value	standard deviation
a	275.7	31.85
b	0.146	0.93
c	0.5986	0.0040
n	0.5566	0.05

Table 7: Mean cluster size, peak position finite-size scaling parameters

[1] Y. S. Cho and B. Kahng. “Discontinuous percolation transitions in real physical systems.” In: *Phys. Rev. E* 84.050102 (2011).

[2] P. Erdős and A. Rényi. “On random graphs, I.” In: *Publ. Math. Debrecen* 6 (1959), pp. 290–297.

[3] P. Erdős and A. Rényi. “On the evolution of random graphs.” In: *Publ. Math. Inst. Hungar. Acad. Sci.* 5 (1960), pp. 17–61.

[4] E. J. Friedman and A. S. Landsberg. “Construction and analysis of random networks with Explosive Percolation.” In: *Phys. rev. Lett.* 103.255701 (2009).

[5] F. Radicchi and S. Fortunato. “Explosive Percolation in scale-free networks.” In: *Phys. Rev. Lett.* 103.168701 (2009).

[6] O. Riordan and L. Warnke. “Achlioptas Process phase transitons are continuous.” In: *The Annals of Applied Probability* 22 (2012), pp. 1450–1464.

[7] M. Sahimi. *Applications of Percolation Theory*. 1st ed. Philadelphia, Pennsylvania: Taylor and Francis, 1994.

[8] D. Stauffer and A. Aharony. *Introduction to Percolation Theory*. 2nd ed. Philadelphia, Pennsylvania: Taylor and Francis, 2003.

[9] Yong Zhu and Xiaosong Chen. “Finite size scaling theory for percolation phase transition.” In: *arXiv* 1710.02957 (2017).