Analysis of percolation thresholds in a square lattice

Toby Nuttall

April 2024

Abstract

Percolation theory is a relatively new field of computational physics, with research revealing a veritable wealth of physics and drawing parallels to phase transitions and critical phenomena. This project aims to find the onset of critical behaviour in a square lattice system, following closely the work carried out by Newman and Ziff in 2000[1]. A percolation threshold probability of 0.593 ± 0.001 was obtained, in good agreement with previous estimates from Newman-Ziff[1].

Research into the largest cluster sizes was also performed for different sized lattices and used to verify the percolation threshold result obtained. Python was used exclusively for this project and performance was a critical obstacle to obtaining results. Consequently, NumPy vectorisation was integral to the success of the project.

Word count: 3000

1 Introduction

Percolation theory is the study of cluster or pore growth in a medium, where the cluster or pore is propagated randomly throughout a lattice of points.

There are two main fields of study in percolation theory, which both revolve around an occupation probability P. In site percolation, P is the probability that a point in the lattice is populated, and in bond percolation it is the probability that a bond between points in the lattice is populated. Phrasing the problem in terms of occupation probability P is known as the canonical ensemble, in which statistics are measured as a function of probability. In this project, the microcanonical ensemble was also used extensively, in which statistics are measured in terms of an occupation number n (the number of sites or bonds occupied in the lattice).

A particular focus of this project was finding the percolation threshold, defined to be the occupation probability at which infinite clusters begin to form. This and the largest cluster size were the two main focuses of this project. A weighted union find algorithm was employed in tandem with the Newman-Ziff[1] algorithm to analyse percolation in lattices of different sizes.

2 Analysis of the computational physics

2.1 The Newman-Ziff[1] algorithm

The Newman-Ziff[1] algorithm was the first algorithm to apply a Monte-Carlo approach to percolation with great effect, achieving a scaling in time and computational weight of $O(N)$ (N is the total number of lattice sites). This is a large improvement to other more common methods in the field up to this point. For instance, a depth-first search algorithm scales with $O(N^2)$ to obtain a canonical statistic of interest, and a binary search scales as $O(N \ln N)$. Both are less computationally efficient than the Newman-Ziff[1] algorithm.

The Newman-Ziff[1] algorithm revolves around populating an empty lattice randomly with sites until it is full. The benefit of this compared to more conventional methods (where one considers whether or not each point is populated according to some random variable P) is two-fold. Firstly and most importantly, each site addition effectively generates a new lattice for which a statistic can be found and hence the Newman-Ziff[1] algorithm can obtain N statistics in a single run of the algorithm (as opposed to only a single statistic for conventional methods). This is in essence where the factor of N goes in the scaling and how the Newman-Ziff[1] algorithm can scale with $O(N)$ as opposed to $O(N^2)$. Secondly, any statistic which requires continual updating as the lattice progresses is much easier to obtain in the microcanonical ensemble (adding sites randomly). For instance, the percolation point requires the updating of pointer vectors (as described in section 3) for each site addition and hence the Newman-Ziff[1] algorithm can also find the percolation point in $O(N)$ steps. Therefore, the percolation point is used directly to calculate the percolation threshold.

The only "drawback" to the Newman-Ziff[1] algorithm is that it obtains results in the wrong ensemble, and so its results must be convoluted with a binomial distribution in order to be interpreted and used. However, even for the smallest lattice used here, the time taken to gather the microcanonical results is much larger than the time taken to convolve them (becoming more and more negligible as the lattice size increases), scaling roughly as $O(N^1/2)$. Hence, in practice the binomial convolution did not present any meaningful detriment to the overall performance of the project.

2.2 Mertens'[2] algorithm

An alternative and more recent approach comes from Mertens[2], documenting a transfer matrix methodology for percolation. Transfer matrices have been a staple of statistical physics since the Second World War after Lars Onsager[5] used a transfer matrix method to analytically solve the infinite 2D ising model. However, they had not been applied to percolation in any great depth until Mertens[2].

2.2.1 The transfer matrix method

The general transfer matrix method is as follows. A lattice of points is set up in matrix form represented by $A_{nm}(k)$ for k occupied sites. The goal of the method is to calculate the total number of lattices $A_{nm}(k)$ for which percolation has occurred. These percolating lattices can then be organised into generator functions as below:

$$F_{n,m}(z) = \sum_{k=0}^{nm} A_{nm}(k) z^k$$

Then the generator matrix can be calculated recursively according to

$$F^{\sigma}_{n,m}(z) = \sum_{\sigma'} T_{\sigma,\sigma'}(z) F^{\sigma'}_{n,m-1}(z)$$

with T being the transfer matrix, and the indices i, j referring to the signatures of the new row (all the possible configurations of sites that the new row might contain).

Consequently, the algorithm allows a reduction in complexity from $O(2^{mn})$ to $O(\lambda^n)$ (for $\lambda = 2.6$), since the 2D problem is reduced to a series of m 1D problems. That being said, the calculation and incorporation of these signatures is the key difficulty and challenge of the Mertens[2] algorithm and is the content of much of the paper.

The number of signatures to be computed is the main drawback of the Mertens[2] method. With the number of signatures S_n growing as

$$S_n = \sum_{r=0}^{\frac{(n-1)}{2}} \binom{n+1}{2r+2}\binom{2r+2}{2}\frac{3}{r+3}$$

for $m \geq n/2$. The exponential growth of the signature number limits the size of the lattice drastically: a quote from the paper states that "computing $F_{25,25}(z)$ would require a computer with a 4 TB memory and a couple of months CPU time"[2].

Consequently, even though the transfer matrix method allows for the exact computation of the percolation threshold of small lattices, it cannot be scaled up to any meaningful size as of yet. The computational speed and efficiency of the Newman-Ziff[1] algorithm, as of right now, more than makes up for what little it lacks in precision. However, in the future with better computers and/or faster algorithms, transfer matrix methods will come to replace Monte-Carlo methods as they allow for exact computation of percolation statistics.

3 Implementation and performance of the algorithm

For bond percolation, the largest cluster size was recorded and plotted against occupation probability. For site percolation, the percolation threshold probability was obtained. In this section of the report, the implementation and performance of the code will be discussed; in both cases this section acts to document and describe what was done, as well as to give some potential improvements.

3.1 Bond percolation

Bond percolation was analysed with a weighted union find algorithm adapted from Lee[3] in order to keep track of the tree structure of the clusters. The pointer method of Newman-Ziff[1] was employed to find percolation points.

3.1.1 Tree structure and merging

Initially, all the points are their own roots with a (0,0) pointer vector. Bonds were added randomly and the points on either side of the bond were merged.

A merging event occurs when a bond is added between two points belonging to separate clusters. Merging was the slowest step in the algorithm, and limited the entire project.

The first step of merging is to identify the roots of the points joined by the new bond. This was done using a standard find algorithm[3]. Each root saves the cluster size, so this is now known as well.

The smaller cluster is then merged into the larger, and the root of the smaller cluster is updated to match that of the larger cluster. This was done as documented in Newman-Ziff[1], by assigning the root point of the smaller cluster a new root - that of the larger cluster. Hence all of the smaller cluster's nodes are now re-routed to the larger cluster. This step requires one variable update and is very fast.

The next step in merging is to update the pointers of the smaller cluster. This was done by first evaluating the root displacement vector (the vector joining the root of the smaller cluster to that of the larger cluster), which is found by calling the pointers of the 2 points joined by the new bond and the unit-vector between them (requiring two calculations in total):

$$\overrightarrow{D}(R_2 - R_1) = \overrightarrow{D}(P_2) - \overrightarrow{D}(P_1) - \overrightarrow{r_{12}}$$

where $\overrightarrow{D}(R_2 - R_1)$ is the root displacement vector, $\overrightarrow{D}(P_2)$ and $\overrightarrow{D}(P_1)$ are the pointer vectors of the points joined by the new bond. $\overrightarrow{r_{12}}$ is the unit vector along the new bond.

Lastly, each point in the second cluster is updated so that the pointer of each is equal to $\overrightarrow{D}(P_i) + \overrightarrow{D}(R_2 - R_1)$. This requires $2m$ variable updates, for m points in a merging cluster. Hence, this pointer update step strongly limited performance when merging.

3.1.2 Percolation checking condition

When a bond is added between 2 points in the same cluster, the difference between their 2 pointers must be one lattice unit unless percolation has occurred. In the case of percolation, this difference will be equal to one minus the lattice size L (where $L^2 = N$) for the toroidal boundary conditions used in this project.

3.1.3 Largest cluster size

Finally, the largest cluster size was found as a function of occupation probability P. The size of each cluster was recorded when a bond was added to it. For M runs with each run requiring $2N$ bond additions (where N is the total number of sites), $N * M$ data points were obtained in the microcanonical ensemble. This data was then looped over to obtain the largest cluster size for each occupation number. Consequently, this was a slow performance-limiting step in the largest cluster size algorithm. The data was then convoluted with the binomial distribution (as discussed in section 3.3).

3.2 Site percolation

In site percolation, one considers beginning with an empty lattice and populating it with sites. Each site begins as its own root with a (0,0) pointer vector. Most of the methodology and implementation can be carried over from bond percolation. However, the merging condition is intrinsically more complicated.

3.2.1 Merging events

Merging events now occur whenever a site is selected with one or more occupied neighbors. Thus, one might need to apply the merging algorithm to multiple clusters for one site addition. The main added complexity comes from figuring out which of the site's neighbours need to be merged.

Firstly, one must find which of the given site's neighbours are occupied, which can be achieved by keeping track of the unoccupied sites with the union find algorithm. Likewise, the unit vector $\vec{r_{ij}}$ can also be ascribed to each neighbour, depending on where it is situated relative to the site. Next, the root and size of each of the neighbouring clusters is obtained with the modified union find algorithm. All the neighboring clusters and the new site are then merged into the largest cluster using a similar algorithm to the bond case, the only difference being that now the root displacement vector must be calculated with an extra unit vector:

$$\overrightarrow{D}(R_2 - R_1) = \overrightarrow{D}(P_2) - \overrightarrow{D}(P_1) + \overrightarrow{r_{n2}} - \overrightarrow{r_{n1}}$$

With $\overrightarrow{r_{n2}}$ and $\overrightarrow{r_{n1}}$ denoting the unit vectors between the site and the two merging neighbors respectively. Much like the bond case, this was the performance limiting step in the percolation threshold algorithm.

3.2.2 Percolation onset condition

The system can percolate only when the same cluster borders a new site on more than one side. This is a strong constrain, hence it is not checked for frequently. Aside from this, the percolation condition is the same as the bond case.

3.3 The convolution method

The convolver function was defined specifically to convert between the microcanonical and canonical ensembles. Given a set of data Q_n, Q(p) requires a direct sum over n according to:

$$\sum_n \binom{N}{n} p^n (1-p)^{N-n} Q_n$$

As such, the matrix Q_{pn} was defined, where each value in the occupation probability range (of arbitrary size 1000) took one index p and each value of n took the other index:

$$Q_{pn} = \binom{N}{n} p^n (1-p)^{N-n} Q_n$$

The columns of Q_{pn} were evaluated with a loop requiring n calculations. The index n was then summed over, using a standard matrix multiplication with a size N vector of ones.

$$Q(P) = \begin{bmatrix} \cdots & Q_{0n} & \cdots \\ \cdots & Q_{1n} & \cdots \\ & \vdots & \\ \cdots & Q_{p-1n} & \cdots \\ \cdots & Q_{pn} & \cdots \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix}$$

This produced the desired convolution in $n+1$ calculations and scaled in time with $O(L^{2.05\pm0.03})$. As illustrated in Table 1 below, the run time of the convolver function was suitably fast and not performance limiting.

3.4 Time scaling relations

A table of run times for each section is presented in Table 1 below and the scaling of run time against lattice size L was evaluated in each case. The run times given are from one run of the code and each is subject to some statistical error which is unaccounted for. It is worth noting that the fourth run time for $L = 256$ in the bond percolation cluster size is subject to a large error due to temporary hardware issues stalling the code runtime (computations were performed using a Dell Latitude 5430 with an intel i5 processor). As such it is left in the table, but was not used to evaluate the scaling.

Table 1: **Table of run times in seconds for each section of the project**

Lattice size	Bond Percolation Cluster Size time(s)	Site Percolation Threshold time(s)	Convolution Function time(s)
L = 32	44	23	0.00527
L = 64	230	101	0.202
L = 128	2225	584	0.857
L = 256	88117	7383	3.728

3.4.1 Scaling of run time with size

The largest cluster size scaled with L^α with $\alpha = 2.83 \pm 0.26$. The site percolation threshold scaled with L^γ with $\gamma = 2.75 \pm 0.25$. The convolution function scaled with L^δ with $\delta = 2.05 \pm 0.03$.

The scaling of both the site and bond sections of the code does not agree with the quoted scaling of Newman-Ziff[1] of L^2. This may be a consequence of the slow merging events described in section 3.2.1 in the site case, and the slow largest cluster algorithm described in section 3.1.3. The convolution function scaling was also too high, relative to expectation, but this is a small enough deviation that it is attributed to statistical fluctuation of the error.

3.5 Potential improvements to the code

One might only record the size of the largest cluster when it changes and by interpolation obtain the same results. This should boost performance, as less data would be collected and that data would not need to be looped over.

Also, one might not compress the pointers to improve performance so that in each merging event the cluster of pointers would not need to be updated. This would then require only 3 calculations as opposed to $2m + 1$ for each merging event. A pointer find function would now be required, to compress a node's pointer to check for percolation. Given percolation checking occurs far less frequently than merging events, this should result in a large performance boost.

4 Results

The results section of this report is split into two main parts: site percolation threshold probability and bond percolation largest cluster size. The results from each section are presented in turn, each preceding a discussion of these results in relation to theory.

4.1 Site percolation thresholds

Figures 1 and 2 below show the wrapping probability horizontally or vertically against occupation probability P. Newman-Ziff[1] compares this result to an infinite lattice in which one should obtain a step function at $P_c = 0.59274606(15)$, with $R_L(P_c) = 0.690473725$. Upon increasing the lattice size (which is $L * L$), the wrapping probability functions $R_L(P)$ take the form of sigmoidal curves [see figure 1], which get steeper for larger lattices. Moreover these curves should cross at $(R_L(P_c), P_c)$.

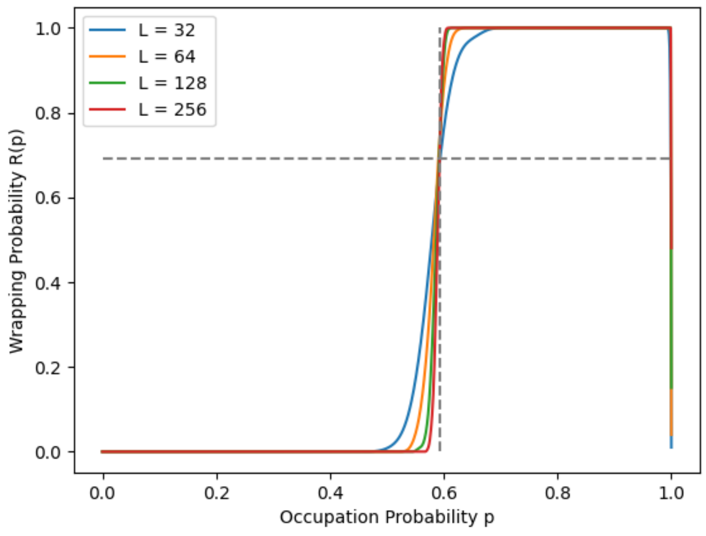

Figure 1: Plot of wrapping probability $R_L(P)$ against occupation probability P for site percolation

Figure 2, given below, shows well the how close the curves are to theory. It is difficult to give a numerical value for the precision here as the curves do not all cross at one point due to statistical fluctuation. One hundred trial lattices were used for each value of L, yet the shapes of the curves varied to a significant degree from run to run and so in this report any inaccuracy in these results is attributed to that. Using more trial lattices would boost the accuracy and precision of these results. However, this project was limited by run time. Each of these figures took about 2.5 hours to run; more trial lattices were not practical.

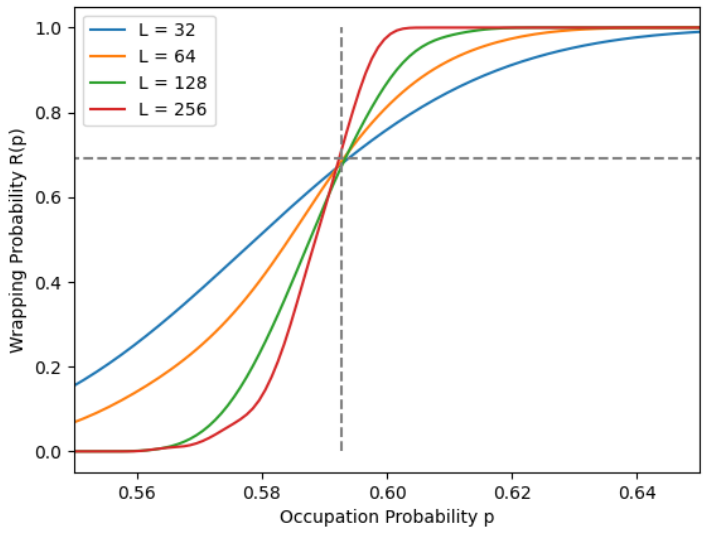

Figure 2: The same plot as Figure 1, only showing the range $0.55 < P < 0.65$ for clarity

As a final point about this section of the project, it is worth noting that both figures show good agreement with accepted theory, resulting in a final $P_c = 0.593 \pm 0.001$, in line with the accepted value. This could be improved only by running the code for longer with more trials, or speeding the code up so as to improve the results in the same run time.

4.2 Bond percolation cluster size

In this section, Figure 3 is shows the largest cluster size $Q_L(P)$ as a function of occupation probability P for a number of differently sized lattices $L = (32, 64, 128, 256)$. This plot is not particularly instructive to the percolation threshold, and its purpose in this report is two-fold. It illustrates an interesting point about cluster sizes: that a cluster of order $O(L)$ forms at the percolation threshold, and that L becomes increasingly dwarfed by L^2 as the lattice size increases. Consequently, Figure 3 provides a qualitative check of the percolation threshold.

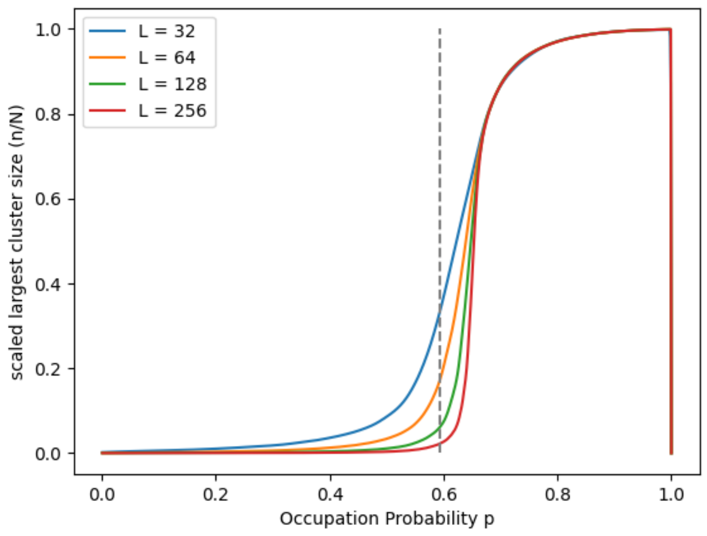

Figure 3: Plot of scaled cluster size n/L^2 against occupation probability P for bond percolation. The dashed line shows the location of the percolation threshold $P_c = 0.59274621$ [1]

In Figure 3, one can clearly see that the plots have a sigmoidal shape, which begins to increase roughly at the percolation threshold (dashed line). As the lattice size increases, $Q_L(P_c)$ decreases, taking values $(0.064, 0.144, 0.340)$ for each lattice. These values of Q_L decrease with $1/L$, as expected for a cluster of size L, scaled by L^2. A log/log plot of these values against L shows a gradient of 1.20 ± 0.3,which is in agreement with a gradient of 1. The quoted error comes from calculating the gradient with a least squares regression and also from statistical fluctuation in the curves. This gives some numerical grounding that at the percolation threshold there exists a cluster of rough size L. This is precisely the definition of the percolation threshold, and provides a soft check for the hard result given in the previous section.

Finally, it is worth noting that extensions to this section on cluster sizes could include an evaluation of the average cluster size $S(p, L)$, the cluster size distribution $n_s(p, L)$ and the critical exponent $sigma$ [3]. The cluster size distribution (defined as $N_s(p, L)$ the total number of clusters of size s in the lattice divided by L) is used to calculate an expectation value of the average cluster size [3], with $P(p, L)$ being the probability that any given site is part of a finite cluster. At the critical point, $p \to p_c$, the average cluster size diverges as $|p - p_c|^{-\gamma}$, with γ being the critical exponent. Analogously to phase transitions, γ characterises the percolation transition (from small clusters to infinite clusters), resulting in an interesting parallel between classical statistical mechanics and percolation theory. As the system approaches the critical point $p \rightarrow p_c$, one notes the cluster size distribution $n_s(p)$ also diverges, tending to $s^{-\tau}$, where τ is the Fisher exponent and s is some quantity greater than 1. Full research into scaling laws would also have been a fascinating direction to take this project in.

5 Conclusion

In summary, good agreement to theory was obtained for the percolation threshold in the site percolation regime, with a value of $P_c = 0.593 \pm 0.001$ obtained (with the error attributed to statistical fluctuation). A discussion of the largest cluster size was given for the bond case, and a quantitative check was performed on the data to verify that the largest cluster size at the percolation threshold scaled with $O(\beta)$ with $\beta = 1.20 \pm 0.3$ (implying that the largest cluster was of the order of the system size L at the percolation threshold). The performance of the project was evaluated in terms of run time. This was found to scale, for site and bond percolation respectively, as $O(\alpha)$ with $\alpha = 2.83 \pm 0.26$ and $O(\gamma)$ with $\gamma = 2.75 \pm 0.25$, both of which are large when compared to theory. This was attributed to superfluous calculation in the merging events and the largest cluster algorithm respectively. Finally, it was found that the time to convolute results between the microcanonical and canonical ensemble scaled with $O(\delta)$ with $\delta = 2.05 \pm 0.03$. which is slightly larger than expected (attributed to statistical fluctuation).

In conclusion, percolation theory is a vast field of study with numerous interesting avenues to explore, many of which opened up throughout the project and left one wishing for more time to explore them. At numerous points in this report detail has been given to how the project could have been improved or expanded upon, which is a testament to the wealth of physics in percolation theory.

References

1 - M. E. J. Newman and R. M. Ziff, 2000.*Efficient Monte Carlo Algorithm and High-Precision Results for Percolation:*
Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, New Mexico 87501,
Department of Chemical Engineering, University of Michigan, Ann Arbor, Michigan 48109-2136

2 - S. Mertens, 2022. *Exact site-percolation probability on the square lattice:*
J. Phys. A: Math. Theor

3 - Medium, accessed 28/04/2024. *Union Find Algorithm:* Claire Lee
https://yuminlee2.medium.com/union-find-algorithm-ffa9cd7d2dba

4 - pypercolate documentation, accessed 28/04/22. *Percolation Theory:* Andreas Sorge
https://pypercolate.readthedocs.io/en/stable/percolation-theory.html

5 - L.Onsager, 1944. *Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition:*
Phys. Rev. 65, 117.

Appendix A: Source Code Structure

Bond percolation folder

cluster size plot: a file which plots the largest cluster size against probability for 4 different lattice sizes. It generates Figure 3 in the main document.

percolation clustersize: contains the RUN, convolver and reset functions called repeatedly in **cluster size plot**. The RUN function runs a lattice until it is fully populated with bonds and returns the largest cluster size as a function of bond number. The reset function resets the lattice so that RUN can be called again on a fresh lattice. Convolver, convolutes the microcanonical results into the canonical ensemble. Used in **cluster size plots**

nearest neighbors: represents the geometry with functions: position, nearest, nearest2, nearest bonds and lchoose. All except lchoose are called in all other files in this section, and are used to find nearest neighbors, or convert site nodes (given as numbers) to (i,j) vectors. lchoose is used only in the convolver function in order to return the natural log of nCr. Used in **all of the other files**.

union find 2: The specific union find algorithm used for the cluster sizes, containing the find and union algorithms, as well as storing the un-selected bonds to be called. This version of union find is used in **cluster size plot and percolation clustersize**

percolation point: contains code to find the percolation point in bond percolation for a 100 by 100 lattice. It calls random bonds to form and as a consequence can call the same bond twice.

union find: the original union find algorithm used for this project, with a standard find function and an updated union function. The union function updates the pointers of the 2 clusters merged by this function and keeps track of the pointer vectors of all the points in the lattice. used in **Percolation point**

Site Percolation folder

plot: the file used to plot the wrapping probability against site occupation probability for 4 different lattice. It also contains the data collection function, which loops over 100 trial lattices and obtains the wrapping probability for a given lattice size.

main: a file containing the main functionality of the site percolation section. It has the functions:update, RUN, reset, convolver defined within it. Update adds a site to the lattice and merges any clusters bordering it, RUN loops over the update function until percolation has occured and returns the number of steps required. Reset and convolver have the same functionality as they do in the bond case. Used in **plot**

nearest neighbors: the nearest neighbors function in the site case, used to reflect the geometry of the system. Containing the functions: nearest,nearest2, position,neighbor and lchoose. These are almost identical to those functions in the bond case. All of which except for lchoose are called in all other files in this section, and are used to find nearest neighbors, or convert site nodes (given as a number), to (i,j) vectors. lchoose is used only in the convolver function, to return the natural log of nCr. Used in **all of the other files**.

union find: the version of union find used in site percolation in this project. It contains the standard union and find algorithm and keeps track of the pointers of all the points in the lattice as well as the set of un-selected nodes.Used in **plot, main**.