# A percolation thresholds demo in NetLogo

## Course project for Complex Systems

Chun Tian

Scuola di Scienze, Università di Bologna
chun.tian@studio.unibo.it
Numero di matricola: 0000735539

**Abstract.** A NetLogo demo model has been made for computing site/bond percolation thresholds for any input graph, based on the Newman-Ziff algorithm. Various types of random graphs (ER, preferential attachments, small world network) and 11 Archimedean lattices were tested on the model, and the results are very close to the theory value or those previous computed high precision values by other researchers.

## 1 Motivation on the project choice

If the purpose of the whole Complex Systems course is to show a whole picture on all the related topics in Complex Networks, the only missing part which is not taught on class should be the Percolation Theory. The prove can be found in the paper *Statistical mechanics of complex networks* by Albert and Barabási [1], all contents except Section IV "Percolation Theory" were taught on the class. So, to complete the whole study of Complex Network, the project proposal 1 (*percolation threshold*) is an obvious choice.

## 2 Introduction

Percolation [2] is one of the best studied problems in statistical physics. It describes the behavior of connected clusters in a graph. There are two kinds of percolation models: *site percolation* and *bond percolation*. In site percolation, a site (a vertex of the graph) is "occupied" with probability $p$ or "empty" with probability $(1-p)$; it is possible to go from an empty site to another neighbor open site. In bond percolation, sites are always empty, but their bonds (graph edges) are open with probability $p$ , or closed with probability $(1-p)$; in this case, it is possible to go from a site to a neighbor when their related bond (link) is open.

One of the most interesting findings of random graph theory is the existence of a critical probability at which a giant cluster forms. Translated into network language, it indicates the existence of a critical probability $p_c$ such that below $p_c$ the network is composed of isolated clusters but above $p_c$ a giant cluster spans the entire network. For low-dimensional graphs, especially 2D or 3D regular lattices, the same critical probability exist. However, instead of detecting giant

clusters, the "percolates" of graph is usually identified by a connection between the "top" and "bottom" nodes. This says the same thing with "giant clusters", because they both indicate the "phase transition" of the whole graph.

## 3 Algorithm choice for percolation thresholds

In this project, we want to use NetLogo to build a handy and useful demo to compute the percolation threshold on any given finite graph. We're limited to NetLogo and PeerSim, but latter is not a good choice, unless we want to get very high precision of results with long-time running and no visualizations. Since the visualization is a key consideration, it's important to choose an algorithm which has some meaningful visualizations.

The trivial way to find percolation thresholds is to check the percolates on the given graph on different percolate probabilities according to the percolation models. For each of the percolate probability, there should be repeated test runs to find out the "probability of percolates". Since we know for any given test probability $p$, when $p < p_c$, the probability of percolates should be 0, and for $p > p_c$, the probability of percolates monotonously increases, it's straightforward to get the value of $p_c$. However, such a algorithm should be slow, and the choice of step values of $p$ cannot be too much. The most important thing is, there's nothing to demonstrate in NetLogo while doing the computation, except for the final curve and the value of $p_c$.

On the other side, there exists huge amount of literatures for percolation thresholds computation, most of them are Monte Carlo methods. In a master degree thesis [3] written by Michael James Lee, we've found a huge list of the historical work on the square site percolation threshold (see Fig. 1). Among of all these methods, the most beautiful one should be the *Hull-Gradient method* [4], which combined the two previous methods proposed by Robert M. Ziff and B. Sapoval. In another most recent paper [5], Paul N. Suding and Rober M. Ziff used the Hull-Gradient method to get high precision value for site percolation thresholds for eight Archimedean (uniform) lattices. The random walk in this method is perfect for NetLogo demonstration, and the final $p_c$ will be kept updated during the whole process with higher and higher precision. However, such methods only works on 2D lattices (planar graphs with small node degrees). For various kind of random graphs, there's no way to apply the Hull-Gradient method.

So our goal is to find out a method from the above list, which can support all kinds of graphs, no matter random graph or 2D lattices, and finally we've found the *Newman-Ziff method* [6], proposed by M. E. J. Newman and R. M. Ziff in 2000. In their later paper [7], they even gave some working C code for the algorithm, which is very clear to understand. And in the paper, there're discussions on applying the method to random graphs. The algorithm is quite advanced and belongs to the "union/find" algorithm family, a perfect case to show our NetLogo programming skills and the expressive power of NetLogo programming language, and has a natural visualization if we run the algorithm

| Year | Ref. | Author(s) | Method | PRNG(s) | Result |
|---|---|---|---|---|---|
| 1960 | [59] | Elliot *et al.* | Fifth-order series | | 0.48 |
| 1961 | [55] | Domb and Sykes | Ninth-order series | | 0.55 |
| 1961 | [76] | Frisch *et al.* | MC, 2000 sites | | 0.581(15) |
| 1963 | [45] | Dean | MC, $78^2$ sites | | 0.580(18) |
| 1964 | [252] | Sykes and Essam | Tenth-order series | | 0.59(1) |
| 1976 | [254] | Sykes *et al.* | 19th-order series | | 0.593(2) |
| 1976 | [219] | Rousenq *et al.* | MC, $1000^2$ sites | | 0.595 |
| 1976 | [243] | Stauffer | Series analysis | | 0.591(1) |
| 1976 | [115] | Hoshen *et al.* | MC, $4000^2$ sites | | 0.5927(3) |
| 1980 | [212] | Reynolds *et al.* | MC, $500^2$ sites | | 0.5931(6) |
| 1982 | [48] | Derrida and de Seze | Transfer matrix | | 0.5927(2) |
| 1982 | [53] | Djordjevic *et al.* | Series analysis | | 0.5923(7) |
| 1984 | [82] | Gebele | MC, $50000^2$ sites | | 0.59277(5) |
| 1985 | [208] | Rapaport | MC, $160000^2$ sites | | 0.5927(1) |
| 1985 | [49] | Derrida and Stauffer | Transfer matrix | | 0.59274(10) |
| 1985 | [218] | Rosso *et al.* | Gradient frontier | | 0.59280(1) |
| 1986 | [286] | Ziff | Perimeter walks | | 0.59275(3) |
| 1986 | [130] | Kertész | Transfer matrix | | 0.59273(6) |
| 1986 | [297] | Ziff and Sapoval | Hull-gradient | T | 0.592745(2) |
| 1988 | [299, 296] | Ziff and Stell | Hull-gradient | QTA | 0.5927460(5) |
| 1989 | [282] | Yonezawa *et al.* | Planar crossing | | 0.5930(1) |
| 1992 | [287] | Ziff | Hull-crossing | QTA | 0.5927460(5) |
| 1994 | [119, 117] | Hu | Histogram MC | F | 0.592(8) |
| 1995 | [120, 117] | Hu | Histogram MC | F | 0.5928(1) |
| 1996 | [121, 117] | Hu *et al.* | Histogram MC | F | 0.59278(2) |
| 1996 | [121, 117] | Hu *et al.* | Histogram MC | F | 0.59283(4) |
| 1996 | [121, 117] | Hu *et al.* | Histogram MC | F | 0.59267(6) |
| 1996 | [121, 117] | Hu *et al.* | Histogram MC | F | 0.5814(30) |
| 1996 | [121, 117] | Hu *et al.* | Histogram MC | F | 0.6041(30) |
| 2000 | [190, 191] | Newman and Ziff | Toroidal wrapping | TT, QTB | 0.59274621(13) |
| 2000 | [190, 191] | Newman and Ziff | Toroidal wrapping | TT, QTB | 0.59274636(14) |
| 2000 | [190, 191] | Newman and Ziff | Toroidal wrapping | TT, QTB | 0.59274606(15) |
| 2000 | [190, 191] | Newman and Ziff | Toroidal wrapping | TT, QTB | 0.59274629(20) |
| 2000 | [190, 285] | Ziff | Hull-gradient | QTB | 0.5927465(2) |
| 2002 | [296] | Ziff and Newman | Planar crossing | QTB | 0.5927464(5) |
| 2003 | [175, 174] | Martins and Plascak | Toroidal wrapping | C | 0.5927(1) |
| 2003 | [175, 174] | Martins and Plascak | Toroidal wrapping | C | 0.5929(3) |
| 2003 | [196, 195] | Oliveira *et al.* | Toroidal spanning | R | 0.59274675(88) |
| 2003 | [196, 195] | Oliveira *et al.* | Toroidal spanning | R | 0.59274621(33) |
| 2005 | [46, 21] | Deng and Blöte | Cylindrical correlation | TTT | 0.5927465(4) |
| 2005 | [46, 21] | Deng and Blöte | Cylindrical correlation | TTT | 0.5927466(6) |
| 2005 | [46, 21] | Deng and Blöte | Cylindrical correlation | TTT | 0.5927466(8) |
| 2005 | [46, 21] | Deng and Blöte | Cylindrical correlation | TTT | 0.5927468(10) |
| 2005 | [10] | Balister *et al.* | Semi-rigorous | MT | 0.5927(8) |
| 2007 | [214] | Riordan and Walters | Semi-rigorous | MT | 0.59275(25) |

**Fig. 1.** Previously published estimates of the square site percolation threshold

step by step. Besides, we also noticed that M. E. J. Newman works for Santa Fe Institute, which is one of the papers site for student presentation, and the two papers on Newman-Ziff method can actually be found by going the web site of Santa Fe Institute and search "percolation". All these things has made us strongly feel that professor may want us to find out these papers and the *Newman-Ziff method* and finally use it in the project.

## 4 The Newman-Ziff algorithm

The Newman-Ziff algorithm is based on a very simple idea. In the standard algorithms for percolation, one must create an entire new state of the lattice for every different value of $n$ one wants to investigate, and construct the clusters for that state. However, if we want to generate states for each value of $n$ from zero up to some maximum value, then we can save ourselves some effort by noticing that a correct sample state with $n+1$ occupied sites or bonds is given by adding *one extra randomly chosen* site or bond to a correct sample state with $n$ sites or bonds. In other words, we can create an entire set of correct percolation states by adding sites or bonds one by one to the lattice, starting with an empty lattice. The randomness of each algorithm runs, comes from the random order of chosen sites or bounds. In each step, we do the "union/find" algorithm to detect the current giant component size in the graph, and if two clusters amalgamated together by adding this new site/bond, we know how many nodes in the smaller cluster has been relabeled into the larger cluster. For the "union/find" algorithm itself, the paper gives two variants, and we've chosen the advanced one, which is a tree-based "union/find" algorithm.

In the tree-based "union/find" algorithm, each cluster has a single "root" site, which is the root of the corresponding tree, and all other sites possess pointers either to that root site or to another site in the cluster, such that by following a succession of such pointers we can get from any site to the root. By traversing trees in this way, it is simple to ascertain whether two sites are members of the same cluster: if their pointers lead to the same root site then they are, otherwise they are not. This scheme is illustrated for the case of bond percolation on the square lattice in Fig. 2.

For the performance of this algorithm, It's proven that the whole process takes only $O(M+N)$ time complexity for site percolation, and $O(M)$ for bond percolation, with $M$, $N$ as the edges and nodes of the given graph. This means the time spent on each "union/find" step is a constant irrelevant to the size of graphs, this is quite amazing.

However, we still feel difficulties to decide the $p_c$ even with above algorithm. Recall in each step, we know the current giant component size (the size of largest component in the graph) and the number of relabelings when two clusters (or components) are amalgamated, which equals to the size of the smaller cluster. We can certainly store all these step values across the whole algorithm process. But how to decide the percolation threshold from these values? The rest of Newman-Ziff papers didn't give us a clear answer, because it's again focused on

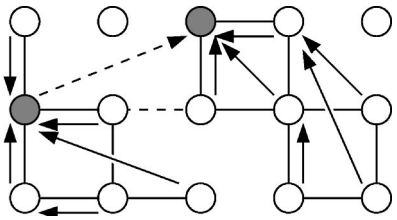

**Fig. 2.** An example of the tree structure described in the text, here representing two clusters. The shaded sites are the root sites and the arrows represent pointers. When a bond is added (dotted line in center) that joins sites that belong to different clusters (i.e., whose pointers lead to different root sites), the two clusters must be amalgamated by making one (left) a subtree of the other (right). This is achieved by adding a new pointer from the root of one tree to the root of the other.

regular lattices which has a clear "top" and "bottom" but we keep trying to avoid using them. So we have to invent our own ways, and the whole inaccuracy in our final work seems coming from this part, plus the facts that we cannot run the algorithm for too many times (i.e. 10,000 times) in NetLogo. As the user will see in our NetLogo GUI, we only give options to run the percolation algorithms for at most 100 times.

## 5 The estimates of percolation thresholds from giant component size and number of relabeling

Here the question is: suppose we're given an arbitrary finite graph with no concept of "top" and "bottom" nodes, and after randomly removing some sites or bonds, how do we tell if the rest of the graph is percolated or not? For random graphs by ER model, according to [1], Section IV, Part G, in a random graph of N nodes, the "percolation threshold", denoting the connection probability at which a giant cluster appears, should be $p_c \simeq 1/N$. And when $p = p_c = 1/N$, random graph theory has following predictions:

- A giant cluster, respectively an infinite cluster appears.
- The size of the giant cluster is $N^{2/3}$.

Above predictions seems indicates that if we keep tracking on the giant component size of a graph, while randomly adding new occupied sites/bonds to it, whenever the giant component size reaches $N^{2/3}$, we can almost say: the current fraction of occupied will be the percolation threshold $p_c$.

On the other side, in the Newman-Ziff paper, there's an observation (see Fig. 3) on the number of relabeling sites, that the maximal value of relabelings happens on the phase transition point, at which the fraction of occupied will equal to the percolation threshold $p_c$. Detection of maximal value of relabelings also seems reasonable intuitively: on an infinite 2D regular lattice, if by adding

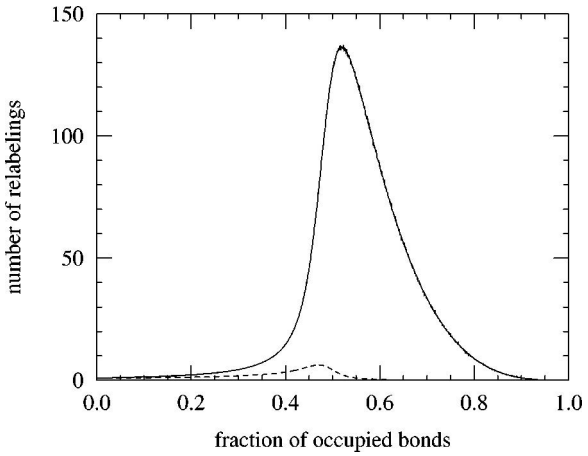


**Fig. 3.** Number of relabelings of sites performed, per bond added, as a function of fraction of occupied bonds

one more site/bond, the graph suddenly become percolated, which means there is a infinite giant cluster appearing in the whole graph, then the number of relabelings at this step must be infinite, which is the maximal value of the whole process.

Based on above analysis, we've decided to compute the percolation threshold by two methods, each focusing on one parameter that we produced during the "union/find" process in the the Newman-Ziff algorithm. And we think the maximum relabeling approach should be universal applicable to almost all graphs, while the test on giant component size $N^{2/3}$ is only for random graphs (not only ER model).

And we should mention that, not every given graph has a percolation threshold $p_c$. For example, if we have a ER random graph with N nodes, and the parameter $p$ (probability that each two nodes have a edge) is smaller than $1/N$, then according to the original percolate mode, even without removing any bond, there's already no way to have a giant component in the graph. So, for such a graph, the giant component size should never reaches $N^{2/3}$, and our estimates of bond percolation thresholds based on giant component size should give no results, while the "maximal relabeling" approach could in any way give a result, because any finite number sequence must have a maximum value.

## 6 Generation of various types of graphs

The graph generations for random graphs in this project is not written manually. Instead, it's based on the exist functionalities provided by NetLogo NW

extension[1]. This includes the random graph (ER model), preferential attachment (BA model) and small world network (WS) model. We didn't do this manually mostly because it's in general quite trivial to generate them according to their definitions. Also, using the NW extension we have following extra benefits:

- The ability of saving the created graph in various standard formats (i.e. GraphML) and loading them into NetLogo. In this way, our percolation model could run on any given graph as long as it's saved into these standard formats.
- The ability of computing various graph properties like degree distribution, average path length.

On the other side, the graph generation of 2D lattices were completely written manually. In the NW extension, there's also a function for generating 2D lattice, but unfortunately the only type of lattice supported is a grid ($4^4$ in Archimedean lattices terminology). In this project, we decided to give NetLogo the ability to generate all eleven Archimedean lattices (see Fig. 4), because the site and bond percolation thresholds for these lattices are all known as high precision values. Inspiring from a related paper [5], we managed to build them all. A "graph" is just a group of nodes with connections, to actually 'see' the lattices, we depend on the layout facility provided by NetLogo itself. The "spring" layout functionality provided by NetLogo is most useful for displaying 2d lattices, and the underlying algorithm is quite complicated.

Additionally, we also provided the ability of generating Cayley trees (see Fig. 5) with any given coordination number. However, the real Cayley tree is a infinite graph, which is not possible to create in NetLogo. And a finite Cayley tree actually cannot give the same percolation thresholds as the infinite Cayley tree which has a precise theory value as $\frac{1}{z-1}$. But any way, it's not hard to generate this kind of recursive structures in NetLogo, so we chose to keep it.

Therefore, we have all needed graphs for testing our percolation computing algorithm. For a screenshot of all these graphs we have drawn in NetLogo, see Fig. 6.

## 7 User Interface

The front-end of our NetLogo model is based on the "Network Extension General Demo" from the NW extension (see Fig. 7). We've added several new buttons and value displays into the original demo model for percolation related facility, and we've provided a chooser called "`lattice-2d-type?`" for the choice of any one of the 11 Archimedean lattices. (see Fig. 8).

We have the chooser "`percolation-type?`" with two possible values: "bond" and "site", as a switch of percolation types. Beside this, we have following percolation thresholds related buttons in the model:

[1] `https://github.com/NetLogo/NW-Extension`

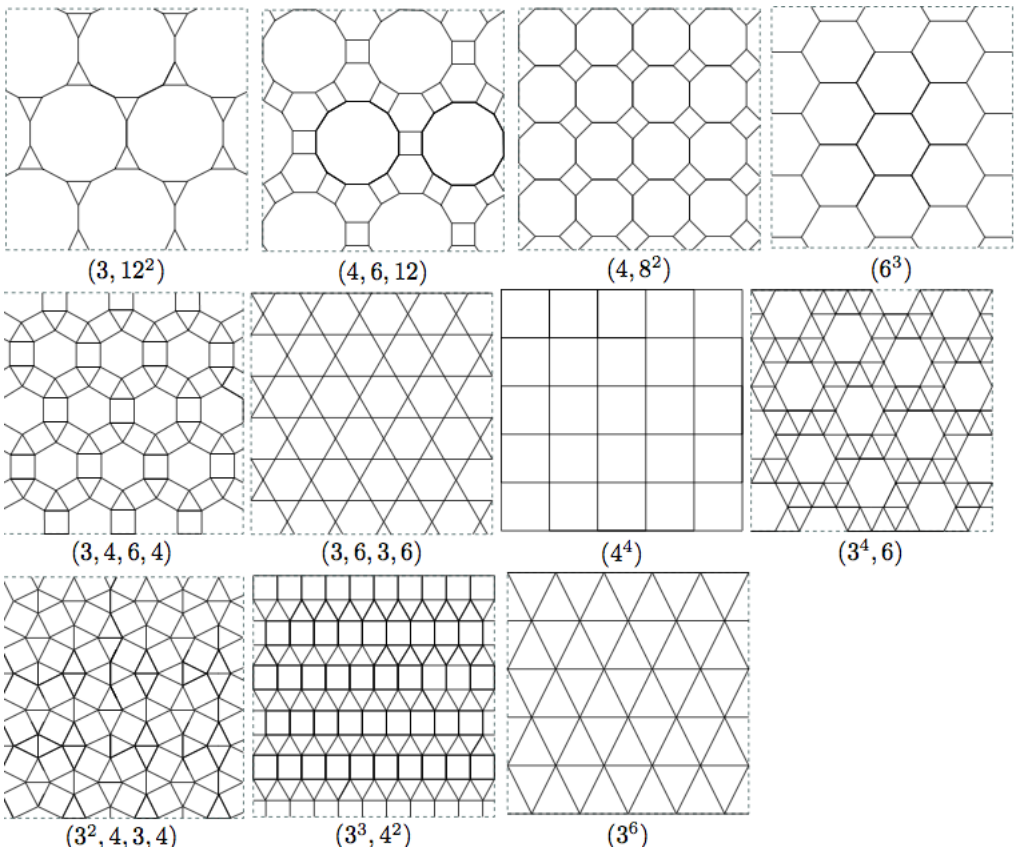


**Fig. 4.** 11 Archimedean Lattices or uniform tilings, in which all polygons are regular and each vertex is surrounded by the same sequence of polygons. The notation $(3^4, 6)$ for example means that every vertex is surrounded by four triangles and one hexagon

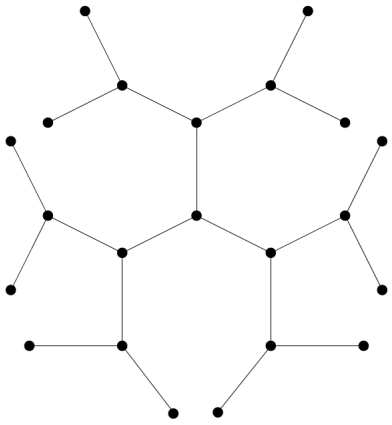

**Fig. 5.** Example of a Cayley tree with coordination number $z = 3$. All of the nodes have 3 edges, with the exception of those on the surface, which have only one edge.

- "`percolation-full`", the full version of percolation thresholds computing, it will run the single version multiple times and using all collected data to get the final $p_c$. The times of percolation runs are controllable through the slider "`percolation-runs`". At the end, the smooth curve for "giant component size" and "relabelings" will be drawn in the Plot area, and two $p_c$ values will be displayed as `pc1` and `pc2`, former is based on giant component size, latter is based on maximal relabelings during the "union/find" process.
- "`percolation`", this is the one-time run of percolation thresholds. Initially the graph is marked as all empty, then each site/bond gets occupied until all of them are occupied. At the end, the curve for "giant component size" and "relabelings" will be drawn in the Plot area.
- "`percolation-setup`", this is the setup stage of the single percolation run, it reset the occupied status of all nodes in the graph, and user can then click the `percolation-step` button for step demonstration of the "union/find" algorithm by Newman-Ziff.
- "`percolation-step`", this is the step run for percolation threshold, for each of the step, a new site/bond gets occupied in the graph and the numbers shown in the NetLogo world is the cluster size stored in each root node. For every newly added site/bond, we also changed their colors.

## 8 Experiments on the model

In this section, we present a series of experiments results on various type of graphs. For those graphs which has known percolation thresholds, we compare the theory value with the results given by our percolation models.

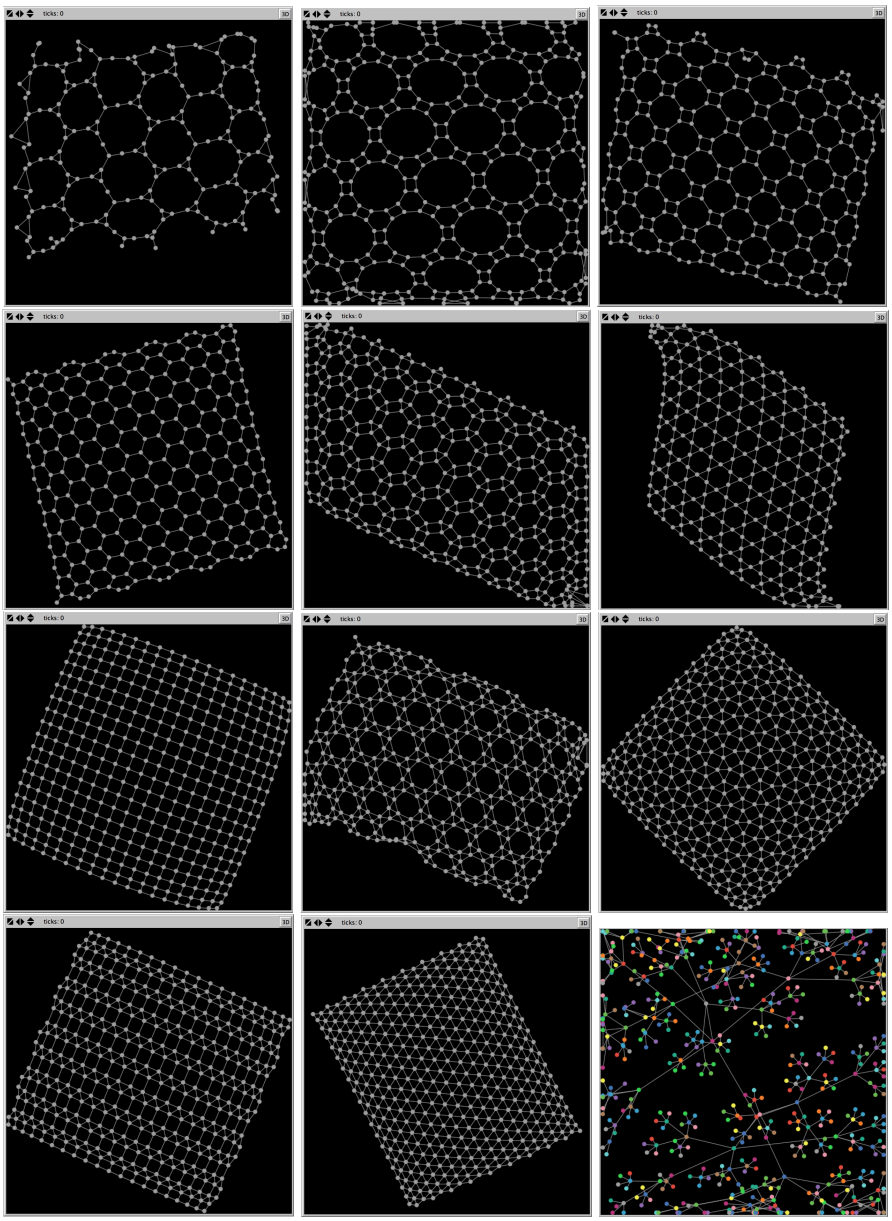

**Fig. 6.** Archimedean Lattices and Cayley tree in NetLogo, layout was done by NetLogo's `spring` function

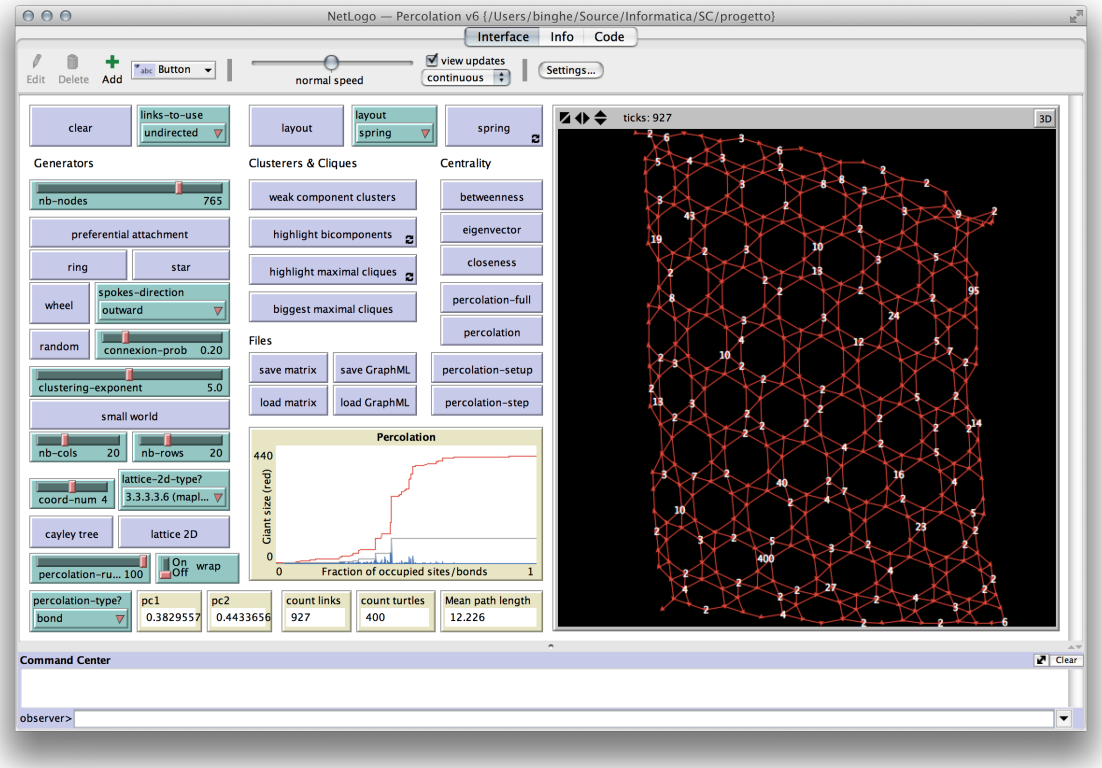


**Fig. 7.** The GUI frontend

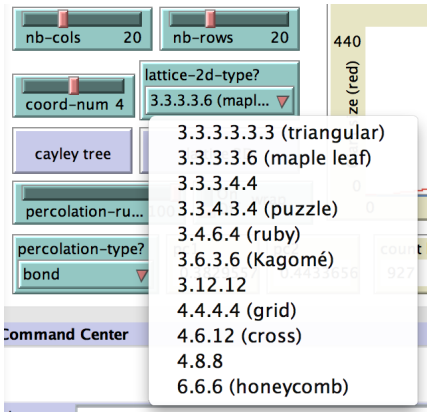


**Fig. 8.** The chooser of 2D lattice types

### 8.1 Bond percolation on ER random graph

For ER random graphs, the theory value of bond percolation is most clear. So we start with this experiment. First of all, we already know the giant component appears in random graphs when the probability parameter $p = 1/N$, $N$ is the number of nodes.Noticed that, bond percolation process actually changes the probability parameter $p$ when building the ER graph, so in theory, the bond percolation thresholds for ER random graph with probability parameter 1 (which means all node pairs are connected, in another word the graph is a complete graph) should be exactly $1/N$. Indeed, our experiments on such complete graphs (ER random graph with $p = 1$) with different number of nodes shows exactly the same thing (see Table 1).

**Table 1.** Bond percolation on complete graph (ER random graph with $p = 1$)

| **Nodes** | **Bond percolation** | **Theory value** $(1/N)$ |
|---|---|---|
| 25 | 0.04 | 0.04 |
| 50 | 0.02 | 0.02 |
| 100 | 0.0107 | 0.01 |
| 200 | 0.0051 | 0.005 |

Notice that we took the percolation thresholds results by testing on the giant component size $N^{2/3}$, which is the value of `pc1` in our model interface. For ER random graphs with $1/N < p < 1$, it's trivial to get the bond percolation thresholds as

$$p_c^{bond} = \frac{1}{Np} = \frac{1}{\langle d \rangle} \tag{1}$$

where $\langle d \rangle$ is the mean node degree. And our experiments shows the same results (see Table 2).

**Table 2.** Bond percolation on ER random graph with 100 nodes

| **Probability parameter** | **Bond percolation** | **Theory value** $(\frac{1}{Np}, N = 100)$ |
|---|---|---|
| 1.0 | 0.0107 | 0.01 |
| 0.5 | 0.0224 | 0.02 |
| 0.2 | 0.049 | 0.05 |
| 0.1 | 0.1 | 0.1 |
| 0.05 | 0.1977 | 0.2 |

### 8.2 Site percolation on ER random graph

For site percolation thresholds on ER random graph, it's more complicated than bond percolation. We don't have a precise formula as the bond percolation case, but our experiments shows that the percolation thresholds is irrelevant to the probability parameter $p$, instead it's only decided by the number of nodes in the given graph. And the relationship between N and $p_c$ is quite like a *Power Law*. We have some experiment results without correspond theory values. (see Table 3 and Fig. 9).

**Table 3.** Site percolation on ER random graph

| **Nodes** $N$ | **Site percolation** |
|---|---|
| 50 | 0.28 |
| 100 | 0.22 |
| 200 | 0.18 |
| 300 | 0.15 |
| 500 | 0.126 |

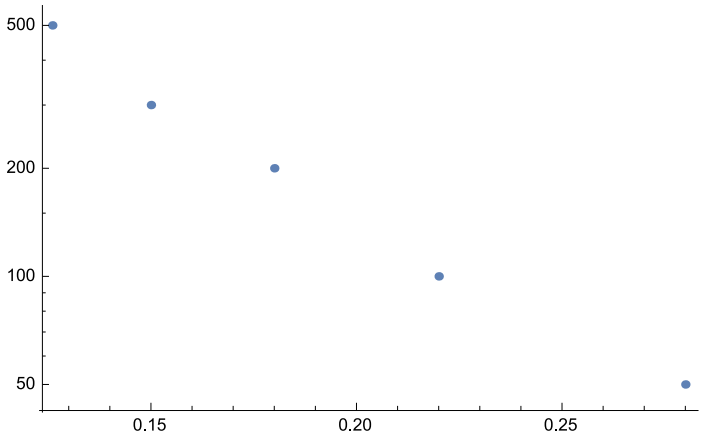


**Fig. 9.** Site percolation on ER random graph, the relationship between $p_c$ and N seems follow a Power Law

### 8.3 Percolation thresholds on other random graphs

Other random graphs are less interesting. For percolation thresholds on BA model with preferential attachment (which forms a Power Law node degree

distribution), we have reason to believe that the site or bond percolation are constant value not related to the number of nodes at all. For site percolation, $p_c^{site} \simeq 0.56$, and for bond percolation, $p_c^{bond} \simeq 0.5$. We've made a table based on different number of nodes in BA graph, but the results are not reliable at all, because each time we get slightly different results, but the overall trends shows the value of $p_c$ is not related to the number of nodes in the graph. (see Table 4)

**Table 4.** Site/bond percolation on BA random graph

| **Nodes** $N$ | **Site percolation** | **Bond percolation** |
|---|---|---|
| 50 | 0.56 | 0.5306 |
| 100 | 0.56 | 0.4949 |
| 150 | 0.6 | 0.5906 |
| 200 | 0.65 | 0.4673 |
| 250 | 0.56 | 0.4900 |
| 300 | 0.5833 | 0.4615 |
| 500 | 0.616 | 0.549 |

The small word model used in NetLogo's NW extension is not following WS model but the *Kleinberg Model* with three parameters: number of rows, number of columns and the clustering coefficient (up to 10.0). Our experiments found that the number of nodes in the small world network has no affects on the percolation thresholds, in another word, it's scale-free as expected. And the value of percolation thresholds changes slightly with the clustering coefficient. We can't see any explicit relationship between them, just the value of $p_c$ will slightly increase with the clustering coefficient $q$. (see Table 5)

**Table 5.** Site/bond percolation on Small World networks

| **Clustering coefficient** | **Site percolation** | **Bond percolation** |
|---|---|---|
| 10.0 | 0.4 | 0.2555 |
| 5.0 | 0.39 | 0.2552 |
| 2.0 | 0.295 | 0.2098 |
| 1.0 | 0.285 | 0.206 |
| 0.5 | 0.275 | 0.196 |

### 8.4 Percolation thresholds on 2D lattices

Finally we've tested our Percolation thresholds algorithms on all 11 Archimedean lattices. Since we doesn't define "top" and "bottom" nodes in the graph, and the number of runs are about 100 times, the results cannot be accurate but should

be closed to the real values. And this time we took the values from `pc2`, which is computed by checking maximal relabeling in the "union/find" algorithm. We compared our results with the precise values given on Wikipedia, it seems that they're very close. For all these tests, we have built lattice graphs with about 1000 nodes. That's big enough, and since our algorithm runs extremely fast, these tests didn't cost too much time, just a few minutes. For the results we have got, see Table 6.

**Table 6.** Site/bond percolation on Archimedean lattices

| **Lattice** | $p_c^{site}$ | $p_c^{site}$ in theory | $p_c^{bond}$ | $p_c^{bond}$ in theory |
|---|---|---|---|---|
| $(3, 12^2)$ | 0.8317 | 0.807900764 | 0.7870 | 0.740420800 |
| $(4, 6, 12)$ | 0.7444 | 0.7478008 | 0.6950 | 0.6937314 |
| $(4, 8^2)$ | 0.7367 | 0.7297232 | 0.6876 | 0.6768031269 |
| honeycomb $(6^3)$ | 0.6867 | 0.697040230 | 0.6766 | 0.652703645 |
| kagome $(3, 6, 3, 6)$ | 0.6578 | 0.652703645 | 0.5325 | 0.524404978 |
| $(3, 4, 6, 4)$ | 0.6567 | 0.62181207 | 0.5387 | 0.5248311 |
| square $(4^4)$ | 0.5856 | 0.592746010 | 0.5333 | 1/2 |
| $(3^4, 6)$ | 0.6144 | 0.579498 | 0.4577 | 0.43432764 |
| $(3^2, 4, 3, 4)$ | 0.5389 | 0.550806 | 0.4359 | 0.4141378476 |
| $(3^3, 4^2)$ | 0.5789 | 0.550213 | 0.4254 | 0.41964044 |
| triangular $(3^6)$ | 0.5144 | 1/2 | 0.3506 | 0.347296355 |

## 9 Conclusions

In this project, we've created a NetLogo demo model for computing the site/bond percolation of any given graph. By "any given graph" we mean two things: 1) the algorithms we chose can be applied to any kind of graph and get reasonable results. 2) our NetLogo demo has ability to load external graph files in GraphML format, this gives us the possibility to test the algorithm on graphs generated by other people and software.

We did extensive tests on various types of random graphs (ER random graph, BA model with preferential attachment, Small world networks) and all 11 Archimedean lattices. For those graphs that we already know the theory value of percolation thresholds, our model has shown high qualities and reasonable results under limited computing time. The highlight of our work is the use of Newman-Ziff algorithm with the "union/find" approach, and this gives us great performance. And our NetLogo code for drawing those Archimedean lattices is amazing, they should be committed to NetLogo official as the enhancements to the NW extension, so that other users can draw these graphs easily in the future.

And the most important thing we got from this project, is the extensive reading on more than 40 related literatures on Percolation Theory and the related computing techniques. These knowledge will be useful for future study. It seems

that there're still many open questions in this area, for example, the close-form value of bond percolation threshold of square lattice, and the close-form value of percolation thresholds of any 3D lattices (which are more important in Physics). However, due to the limited Math knowledge, there's no way for the author(s) of this report to make related contributions easily and quickly. But we'll remember these open questions and keep thinking on them in our rest of school careers.

## References


1. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Reviews of Modern Physics **74**(1) (2002) 47–97
2. Stauffer, D., Aharony, A.: Introduction to percolation theory. CRC press (1994)
3. Lee, M.J.: Methods in percolation. (2008)
4. Ziff, R.M., Sapoval, B.: The efficient determination of the percolation threshold by a frontier-generating walk in a gradient. Journal of Physics A: Mathematical and General **19**(18) (1986) L1169
5. Suding, P.N., Ziff, R.M.: Site percolation thresholds for archimedean lattices. Physical Review E **60**(1) (1999) 275
6. Newman, M., Ziff, R.: Efficient monte carlo algorithm and high-precision results for percolation. Physical Review Letters **85**(19) (2000) 4104
7. Newman, M.E., Ziff, R.M.: Fast monte carlo algorithm for site or bond percolation. Physical Review E **64**(1) (2001) 016706