CS361 Programming Project: Percolation

Cristopher Moore

November 20, 2006

Suppose we fill in, randomly, a fraction p of the sites in an $L \times L$ lattice. The probability that there is a path from the left side of the lattice to the right side consisting entirely of occupied sites, where each step of the path goes up, down, left or right, jumps rapidly from 0 to 1 when p passes a critical threshold p_c. At the same point, the system goes from small isolated islands of occupied sites to a "giant component" that stretches across the entire system (with isolated islands outside the giant component).

This phenomenon is called a *phase transition*; it is mathematically similar to many other phase transitions, such as the freezing of water or when a disease goes from isolated outbreaks to an epidemic. Your project is to probe this transition experimentally, using the Union-Find data structure.

Your program should add each of the L^2 sites in random order. Each time a site is added, you should use the Union-Find structure to merge the new site with each of its four neighbors. I found it convenient to represent the left and right edges of the lattices with two additional sites.

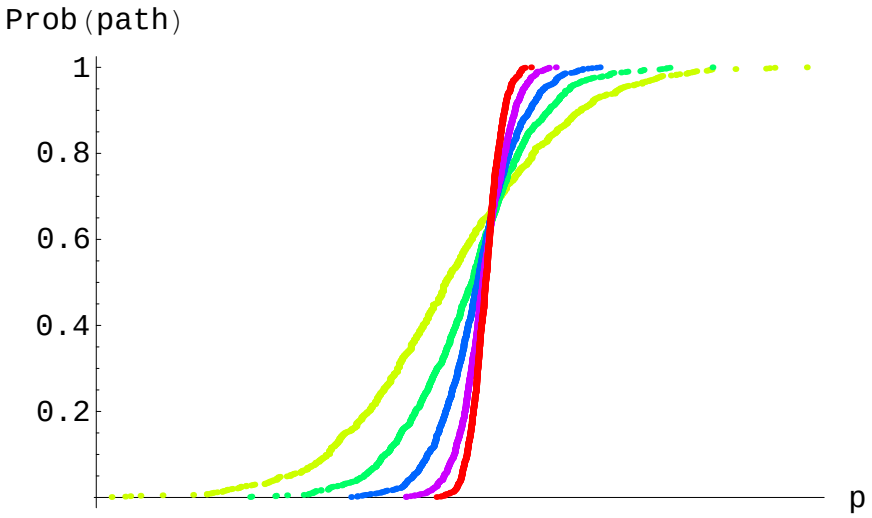

Figure 1: The probability a path exists as a function of p. The different colors represent $L \times L$ lattices with $L = 50, 100, 200, 400,$ and 800. I performed 1000 trials for each value of L. (The p axis has the ticks removed, so as not to spoil the fun of finding the threshold p_c.)

In experiment #1, you should stop adding sites as soon as a path from left to right exists. Then, by doing many independent trials, produce a plot like that in Figure 1, where Prob(path) is the fraction of those trials in which a path appeared when p or less of the sites have been occupied. (One nice way to produce these curves is to create a list of values of p at which a path appears, one value for each trial, sort this list, and then plot the list against the rank in sorted order.) Then, by looking at where the curves for different values of L intersect, estimate both the threshold p_c and the probability that a path exists at $p = p_c$.

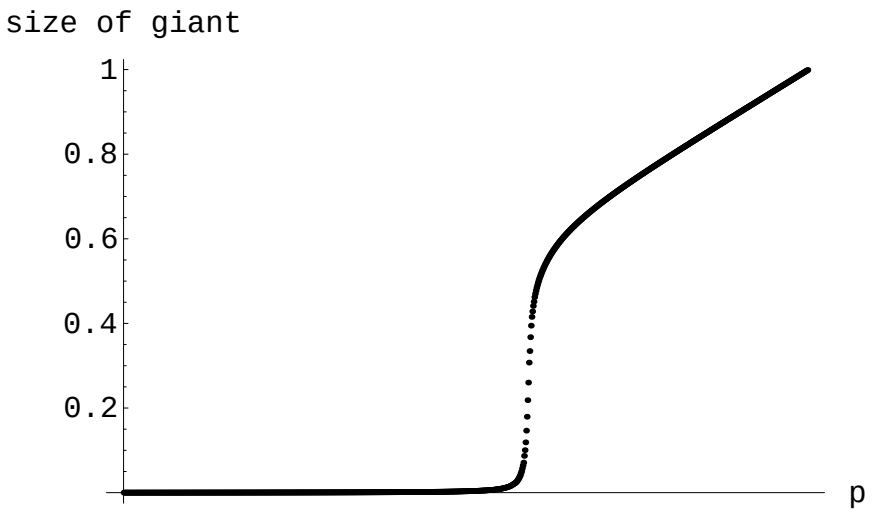

Figure 2: The size of the largest component, as a fraction of the size L^2 of the entire lattice. Here $L = 1000$, and the plot shows the average of 100 independent trials.

In experiment #2, you should continue adding sites until the lattice is full. This time, your goal is to find the size of the largest component as a function of p, and to produce a plot like that in Figure 2. Below p_c, the largest cluster is very small compared to the entire lattice; at p_c, it is a small but rapidly growing fraction of the lattice (see the next experiment); and above p_c, it includes almost all the occupied sites.

In experiment #3, the question is how large the giant component is when it first appears. Although it includes many sites, it actually takes up a smaller and smaller fraction of the lattice as L increases. This is because its size scales as $s \sim L^\beta$ where β is its *fractal dimension*. It turns out that $\beta < 2$. Estimate β by finding, for each L, the average size of the giant component when a path first comes into existence. Then, do a log-log plot of s vs. L, and find the slope β by fitting a straight line to the points.

In experiment #4, your job is to find the average size of a cluster as a function of p. In this case, you should turn off the sites representing the left and right edges, and use cyclic boundary conditions, where every site has 4 neighbors and we go "around the world" if we fall off the edge. Note that we want the average size of *the cluster to which a random site belongs*, which is the average cluster size where each cluster is weighted proportional to its size, which is $\sum_i s_i^2/n$ where s_i is the size of the ith cluster and n is the total number of occupied sites. (Hint: rather than calculating the sum $\sum_i s_i^2$ at each step, you can update it dynamically each time you add a site or merge two clusters. How?) This average starts out at 1 when p is very small, and diverges at p_c. Observe this divergence, and obtain a plot similar to Figure 3.

Experiment #5 is to measure the running time of your algorithm, e.g. of experiment #1. How long does it take if your Union-Find data structure just uses the simple rule that, when merging two components, the larger one should become the parent of the smaller one? How much difference does path compression make? Test your code on large enough lattices so that you can convincingly tell whether the running time is $O(L^2)$, or $O(L^2 \log L)$, or whatever. Comment on whether the limiting factor on the lattice size you can handle is running time or memory.

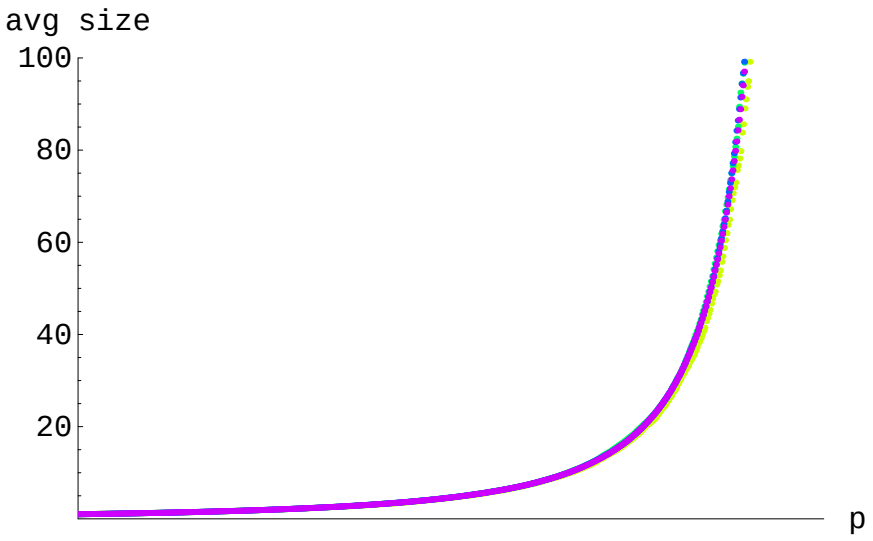

Figure 3: The average size of the component to which a random occupied site belongs. It diverges as p approaches p_c from below. The colors represent $L \times L$ lattices where $L = 50, 100, 200$, and 400. The plot shows the average of 100 independent trials for each value of L.

Extra Credit: Here are some opportunities for extra credit.

1. Repeat experiment #4 in the case where we look for a path from left to right, but calculate the average component size not including the giant component.

2. Rather than just the average, look at the distribution of component sizes, both below p_c and at p_c. You can do this by sorting all the component sizes from smallest to largest, and plotting this sorted list where the x component is the size and the y component is the rank in sorted order. By doing semi-log and log-log plots, see if you can confirm the claim that the size distribution drops of exponentially for $p < p_c$, but follows a power law when $p = p_c$.

3. Do several of these experiments for another version of percolation, where we add edges instead of sites. What is p_c in this case, and why?

How to do it, and what to turn in: You may use any programming language of your choice, and any graphics package of your choice to produce your graphs. (I used Mathematica, but Gnuplot and Matlab are also very good.) You may use external code for data structures such as Union-Find, as long as you give credit to wherever you found it. You are encouraged to discuss strategy and results with each other; as always, the overall code (besides using external packages if you want) must be your own. Please send your report in .pdf or .ps format, and please also send an electronic copy of your code, as well as what language and platform you used, so that we can compile it and run it if we need to.

Your report should have clearly labeled graphs, and should give enough information about lattice sizes, number of trials, and so on, so that anyone who reads your report could reproduce your results. If your results look strange, you should report this honestly rather than fudging the data. You do not need to include a lengthy introduction repeating information about percolation and so on: just explain how you did your experiments, what results you obtained, and any comments you want to make on these results.