Exercises

2.13 **Building a percolation network.**[1] [2] (Complexity, Computation) ④

Figure 2.12 shows what a large sheet of paper, held at the edges, would look like if small holes were successively punched out at random locations. Here the ensemble averages over the different choices of random locations for the holes; this figure shows the sheet just before it fell apart. Certain choices of hole positions would cut the sheet in two far earlier (a straight line across the center) or somewhat later (checkerboard patterns), but for the vast majority of members of our ensemble the paper will have the same kinds of hole patterns seen here. Again, it is easier to analyze all the possible patterns of punches than to predict a particular pattern.

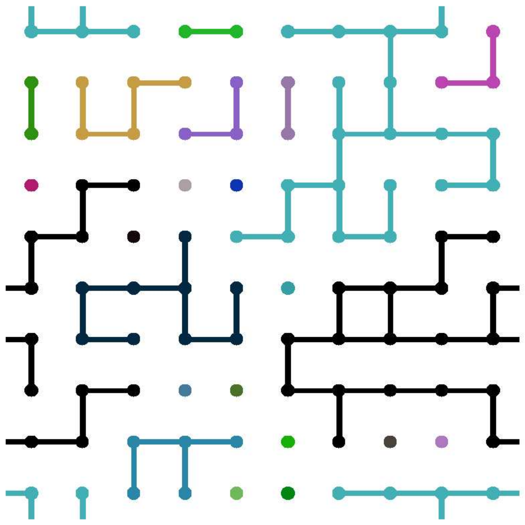

Fig. 2.12 Bond percolation network. Each bond on a 10×10 square lattice is present with probability $p = 0.4$. This is below the percolation threshold $p = 0.5$ for the infinite lattice, and indeed the network breaks up into individual clusters (each shaded separately). Note the periodic boundary conditions. Note there are many small clusters, and only a few large ones; here twelve clusters of size $S = 1$, three of size $S = 2$, and one cluster of size $S = 29$ (black). For a large lattice near the percolation threshold the probability distribution of cluster sizes $\rho(S)$ forms a power law (Exercise 12.12).

Percolation theory is the study of the qualitative change in connectivity of a large system as its components are randomly removed. Outside physics, it has become an archetype of criticality at continuous transitions, presumably because the problem is simple to state and the analysis does not demand a background in equilibrium statistical mechanics.[3] In this exercise, we will study bond percolation and site percolation (Figs 2.12 and 2.13) in two dimensions.

In the computer exercises portion of the web site for this book [129], you will find some hint files and graphic routines to facilitate the working of this exercise.

Bond percolation on a square lattice.

(a) *Define a 2D bond percolation network with periodic boundary conditions on the computer, for size $L \times L$ and bond probability p. For this exercise, the nodes will be represented by pairs of integers (i, j). You will need the method* `GetNeighbors(node)`*, which returns the neighbors of an existing node. Use the bond-drawing software provided to draw your bond percolation network for various p and L, and use it to check that you have implemented the periodic boundary conditions correctly.* (There

[1] From *Statistical Mechanics: Entropy, Order Parameters, and Complexity* by James P. Sethna, copyright Oxford University Press, 2007, page 33. A pdf of the text is available at pages.physics.cornell.edu/sethna/StatMech/ (select the picture of the text). Hyperlinks from this exercise into the text will work if the latter PDF is downloaded into the same directory/folder as this PDF.

[2] This exercise and the associated software were developed in collaboration with Christopher Myers.

[3] Percolation is, in a formal sense, an equilibrium phase transition. One can show that percolation is the $q \to 1$ limit of an equilibrium q-state Potts model—a model where each site has a spin which can take q different states (so $q = 2$ is the Ising model) [25, section 8.4]. But you do not need partition functions and the Boltzmann distribution to define the problem, or to study it.

Copyright Oxford University Press 2006 v1.0

are two basic approaches. You can start with an empty network and use `AddNode` and `AddEdge` in loops to generate the nodes, vertical bonds, and horizontal bonds (see Exercise 1.7). Alternatively, and more traditionally, you can set up a 2D array of vertical and horizontal bonds, and implement `GetNeighbors(node)` by constructing the list of neighbors from the bond networks when the site is visited.)

The percolation threshold and duality. In most continuous phase transitions, one of the challenges is to find the location of the transition. We chose bond percolation on the square lattice because one can argue, in the limit of large systems, that the percolation threshold $p_c = 1/2$. The argument makes use of the *dual lattice*.

The nodes of the dual lattice are the centers of the squares between nodes in the original lattice. The edges of the dual lattice are those which do not cross an edge of the original lattice. Since every potential dual edge crosses exactly one edge of the original lattice, the probability p^* of having bonds on the dual lattice is $1 - p$, where p is the probability of bonds for the original lattice. If we can show that the dual lattice percolates if and only if the original lattice does not, then $p_c = 1/2$. This is easiest to see graphically.

(b) *Generate and print a small lattice with $p = 0.4$, picking one where the largest cluster does not span across either the vertical or the horizontal direction (or print Fig. 2.12). Draw a path on the dual lattice spanning the system from top to bottom and from left to right. (You will be emulating a rat running through a maze.) Is it clear for large systems that the dual lattice will percolate if and only if the original lattice does not?*

Finding the clusters.

(c) *Write the following two functions that together find the clusters in the percolation network.*

(1) `FindClusterFromNode(graph, node, visited)`, *which returns the cluster in* `graph` *containing* `node`, *and marks the sites in the cluster as having been* `visited`. *The cluster is the union of* `node`, *the neighbors, the neighbors of the neighbors, etc. The trick is to use the set of* `visited` *sites to avoid going around in circles. The efficient algorithm is a* breadth-first traversal *of the graph, working outward from* `node` *in shells. There will be a* `currentShell` *of nodes whose neighbors have not yet been checked, and a* `nextShell` *which will be considered after the current one is finished (hence breadth first), as follows.*

```
– Initialize visited[node] = True,
  cluster = [node], and
  currentShell
          = graph.GetNeighbors(node).
– While there are nodes in the new
  currentShell:
   * start a new empty nextShell;
   * for each node in the current shell, if the
     node has not been visited,
      · add the node to the cluster,
      · mark the node as visited,
      · and add the neighbors of the node to the
        nextShell;
   * set the current shell to nextShell.
– Return the cluster.
```

(2) `FindAllClusters(graph)`, *which sets up the* `visited` *set to be* `False` *for all nodes, and calls* `FindClusterFromNode(graph, node, visited)` *on all nodes that have not been visited, collecting the resulting clusters. Optionally, you may want to order the clusters from largest to smallest, for convenience in the graphics (and in finding the largest cluster).*

Check your code by running it for small L and using the graphics software provided. Are the clusters, drawn in different colors, correct?

Site percolation on a triangular lattice. Universality states that the statistical behavior of the percolation clusters at long length scales should be independent of the microscopic detail. That is, removing bonds from a square lattice should leave the same fractal patterns of holes, near p_c, as punching out circular holes in a sheet just before it falls apart. Nothing about your algorithms from part (c) depended on their being four neighbors of a node, or their even being nodes at all sites. Let us implement site percolation on a triangular lattice (Fig. 2.13); nodes are occupied with probability p, with each node connected to any of its six neighbor sites that are also filled (punching out hexagons from a sheet of paper). The triangular site lattice also has a duality transformation, so again $p_c = 0.5$.

It is computationally convenient to label the site at (x, y) on a triangular lattice by $[i, j]$, where $x = i + j/2$ and $y = (\sqrt{3}/2)j$. If we again use periodic boundary conditions with $0 \leq i < L$ and

Copyright Oxford University Press 2006 v1.0

$0 \leq j < L$, we cover a region in the shape of a 60° rhombus.[4] Each site $[i, j]$ has six neighbors, at $[i, j] + e$ with $e = [1, 0], [0, 1], [-1, 1]$ upward and to the right, and minus the same three downward and to the left.

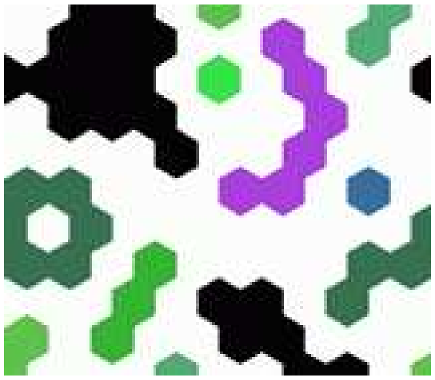

Fig. 2.13 Site percolation network. Each site on a 10×10 triangular lattice is present with probability $p = 0.5$, the percolation threshold for the infinite lattice. Note the periodic boundary conditions at the sides, and the shifted periodic boundaries at the top and bottom.

(d) *Generate a site percolation network on a triangular lattice. You can treat the sites one at a time, using* `AddNode` *with probability p, and check* `HasNode(neighbor)` *to bond to all existing neighbors. Alternatively, you can start by generating a whole matrix of random numbers in one sweep to determine which sites are occupied by nodes, add those nodes, and then fill in the bonds. Check your resulting network by running it for small L and using the graphics software provided. (Notice the shifted periodic boundary conditions at the top and bottom, see Fig. 2.13.) Use your routine from part (c) to generate the clusters, and check these (particularly at the periodic boundaries) using the graphics software.*

(e) *Generate a small square-lattice bond percolation cluster, perhaps* 30×30*, and compare with a small triangular-lattice site percolation cluster. They should look rather different in many ways. Now generate a large*[5] *cluster of each, perhaps* 1000×1000 *(or see Fig. 12.7). Stepping back and blurring your eyes, do the two look substantially similar?*

Chapter 12 and Exercise 12.12 will discuss percolation theory in more detail.

[4]The graphics software uses the periodic boundary conditions to shift this rhombus back into a rectangle.

[5]Your code, if written properly, should run in a time of order N, the number of nodes. If it seems to slow down more than a factor of 4 when you increase the length of the side by a factor of two, then check for inefficiencies.

Copyright Oxford University Press 2006 v1.0