

Chapter 12

Percolation

©2005 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
5 April 2005

We introduce several geometrical concepts associated with percolation, including the percolation threshold, clusters, and cluster finding algorithms. We also introduce the ideas of critical phenomena in the context of the percolation transition, including critical exponents, scaling relations, and the renormalization group.

12.1 Introduction

If a container is filled with small glass beads and a battery is connected to the ends of the container, no current would pass and the system would be an insulator. Suppose that we choose a glass bead at random and replace it by a small steel ball. Clearly, the system would still be an insulator. If we continue randomly replacing glass beads with steel balls, eventually a current would pass. What percentage of steel balls would be needed to make the container into a conductor? The change from the insulating to the conducting state that occurs as the percentage of steel balls is increased is an example of a *percolation phase transition*.

Although the term *percolation* might be familiar to you in the context of brewing coffee, our use of the term has little to do with it. Instead, we consider another example from the kitchen and imagine a large metal sheet on which we randomly place drops of cookie dough. Assume that each drop of cookie dough will spread while the cookies are baking in an oven. If two cookies touch, they coalesce to form one cookie. If we are not careful, we might find a very large cookie that spans from one edge of the sheet to the opposite edge (see Figure 12.1). If such a spanning cookie exists, we say that there has been a percolation transition. As we will discuss in more detail, percolation has to do with *connectivity*. There is no percolation transition for ground coffee, because the water dissolves some of the ground coffee beans and flows, regardless of the density of the ground coffee.

Our discussion of percolation will require little background in physics, for example, no classical or quantum mechanics, and little statistical physics. All that is required is some understanding of geometry and probability. Much of the appeal of percolation is its game-like aspects and intuitive

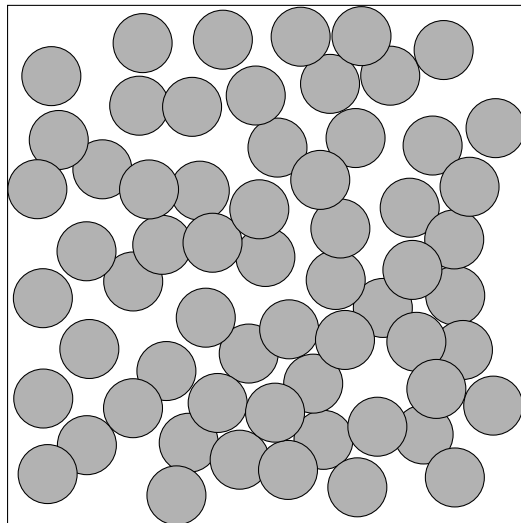


Figure 12.1: Cookies (circles) placed at random on a large sheet. Note that in this case there is a path of overlapping circles that connects the bottom and top edges of the cookie sheet. If such a path exists, we say that the cookies percolate, and there is a spanning path. See Problem 12.4e for a discussion of the algorithm used to generate this configuration.

simplicity. Of course, if you have a background in physics, this chapter will be more meaningful, and can serve as an introduction to phase transitions and to important ideas such as scaling relations, critical exponents, and the renormalization group.

Let us model the cookie example in a simple way to make the concept of percolation more explicit. We represent the cookie sheet by a lattice where each site can be in one of two states, occupied or empty. Each site is occupied independently of its neighbors with probability p . This model of percolation is called *site percolation*. The occupied sites form *clusters*, which are groups of occupied nearest neighbor lattice sites (see Figure 12.2).

An easy way to study site percolation is to generate a random number r in the unit interval $0 < r \leq 1$ for each site in the lattice. A site is occupied if its random number satisfies the condition $r \leq p$. If p is small, we expect that only small isolated clusters will be present (see Figure 12.3a). If p is near unity, we expect that most of the lattice will be occupied, and the occupied sites will form a large cluster that extends from one end of the lattice to the other (see Figure 12.3c). Such a cluster is said to be a *spanning cluster*. Because there is no spanning cluster for small p and there is a spanning cluster for p near unity, there must be an intermediate value of p at which a spanning cluster first exists (see Figure 12.3b). We shall see that in the limit of an infinite lattice, there exists a well defined threshold probability p_c such that:

For $p < p_c$, no spanning cluster exists and all clusters are finite.

For $p \geq p_c$, a spanning cluster exists.

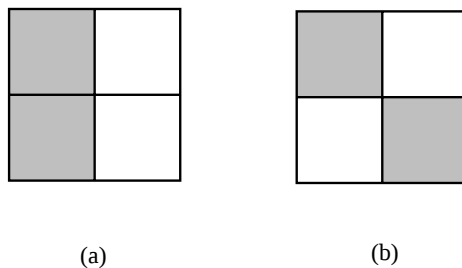


Figure 12.2: Example of a site percolation cluster on a square lattice of linear dimension $L = 2$. The two nearest neighbor occupied sites (shaded) in (a) are part of the same cluster of size two; the two occupied sites in (b) are not nearest neighbor sites and do not belong to the same cluster; each occupied site is a cluster of size one.

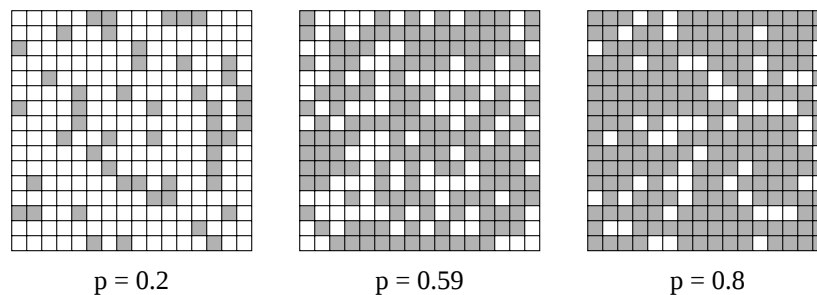


Figure 12.3: Examples of site percolation clusters on a square lattice of linear dimension $L = 16$ for $p = 0.2$, 0.59 , and 0.8 . On average, the fraction of occupied sites (shaded squares) is equal to p . Note that in this example, there exists a cluster that spans the lattice horizontally and vertically for $p = 0.59$.

We emphasize that the defining characteristic of percolation is *connectedness*. Because the connectedness exhibits a qualitative change at a well defined value of a continuous parameter, we shall see that the transition from a state with no spanning cluster to a state with one spanning cluster is an example of a *phase transition*.

An example of percolation that can easily be observed in the laboratory has been done with a wire mesh. Watson and Leath measured the electrical conductivity of a uniform metallic screen as a function of the fraction of the nodes. The coordinates of the nodes to be removed were determined by a random number generator. The measured electrical conductivity is a rapidly decreasing function of the fraction of nodes p that are still present and vanishes below a critical threshold. A related conductivity measurement on a sheet of conducting paper with random holes has been performed (see Mehr et al.).

The applications of percolation phenomena go beyond metal-insulator transitions and the conductivity of wire mesh, and include the spread of disease in a population, the behavior of

magnets diluted by nonmagnetic impurities, the flow of oil through porous rock, the microstructure of fiber-reinforced concrete, and the characterization of gels. Percolation ideas also have been used to understand clusters in such diverse systems as granular matter and social networks. We concentrate on understanding several simple models of percolation that have an intuitive appeal of their own. Some of the applications of percolation phenomena are discussed in the references.

12.2 The Percolation Threshold

Consider a square lattice of linear dimension L and unit lattice spacing, and associate a random number between zero and one with each site in the lattice. A site is occupied if its random number is less than p . Class `PercolationApp` generates site percolation configurations and shows the occupied sites as filled squares of unit area. The array `siteValues` stores the random number associated with each lattice site. The state of each site is stored in `LatticeFrame`, which contains the data for the frame as well as methods to display the lattice. The method `LatticeFrame.setAll` is used to initialize the lattice to the desired size, and `LatticeFrame.setValue(i,j,value)` sets the value at lattice site (i,j) . An unoccupied site has the value 0, and an occupied but uncolored site has the value 1. The values 2–8 are used to color the clusters. The class uses the `InteractiveMouseHandler` interface to allow the user to click on an occupied site. Then the `colorCluster` method recursively colors all the sites in this cluster. To create a new configuration, press the `new lattice` button. The lattice is displayed whenever the calculate button is pressed.

Listing 12.1: The `PercolationApp` class.

```
package org.opensourcephysics.sip.ch12;
import java.awt.*;
import java.awt.event.*;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.frames.*;

/**
 * PercolationApp displays site percolation clusters .
 *
 * Click on a cluster to select and change its color .
 *
 * @author Jan Tobochnik, Wolfgang Christian, Harvey Gould
 * @version 1.0 revised 04/04/05
 */
public class PercolationApp extends AbstractCalculation implements InteractiveMouseHandler {
    long seed = 0;
    java.util.Random random = new java.util.Random(seed);
    LatticeFrame frame = new LatticeFrame("Percolation");
    int L;
    byte clusterNumber; // used to color clusters

    /**
     * Creates the PercolationApp and sets the colors for lattice .
     */
}
```

```

    */
    public PercolationApp() {
        frame.setInteractiveMouseHandler(this);
        frame.setIndexedColor(0, Color.BLACK); // unoccupied sites
        frame.setIndexedColor(1, Color.RED); // occupied sites that are not cluster colored
        frame.setIndexedColor(2, Color.GREEN);
        frame.setIndexedColor(3, Color.BLUE);
        frame.setIndexedColor(4, Color.YELLOW);
        frame.setIndexedColor(5, Color.MAGENTA);
        frame.setIndexedColor(6, Color.CYAN);
        frame.setIndexedColor(7, Color.ORANGE);
        frame.setIndexedColor(8, Color.PINK);
    }

    /**
     * Sets a new random seed.
     */
    public void newSeed() {
        seed = random.nextLong();
        calculate ();
        frame.repaint ();
    }

    /**
     * Hadles mouse actions by coloring clusters .
     * @param panel InteractivePanel
     * @param evt MouseEvent
     */
    public void handleMouseAction(InteractivePanel panel, MouseEvent evt) {
        panel.handleMouseAction(panel, evt);
        switch(panel.getMouseAction()) {
            case InteractivePanel.MOUSE_PRESSED :
                int index = frame.indexFromPoint(panel.getMouseX(), panel.getMouseY());
                if(index<0) {
                    return;
                }
                if(frame.getAtIndex(index)==1) {
                    clusterNumber++;
                    if(clusterNumber>8) { // use up to 7 different colors for clusters
                        clusterNumber = 2;
                    }
                    // convert index from row-major order to ix and iy
                    int ix = index%L;
                    int iy = index/L;
                    colorCluster(ix, iy);
                    frame.repaint ();
                }
                break;
        }
    }
}

```

```

/**
 * Calculates the percolation cluster.
 */
public void calculate() {
    L = control.getInt("Lattice size");
    frame.setAll(new byte[L][L], 0, L, 0, L); // zeros the lattice
    clusterNumber = 1;
    random.setSeed(seed); // resetting the seed will generate the same set of random numbers
    double p = control.getDouble("p");
    for(int i = 0; i < L; i++) {
        for(int j = 0; j < L; j++) {
            if(random.nextDouble() < p) {
                frame.setValue(i, j, 1);
            }
        }
    }
}

void colorCluster(int i, int j) {
    frame.setValue(i, j, clusterNumber);
    int sitesToTest = 1;
    int x[] = new int[L*L];
    int y[] = new int[L*L];
    x[0] = i;
    y[0] = j;
    // find neighbors of site in cluster to see if occupied
    while(sitesToTest > 0) {
        sitesToTest--;
        i = x[sitesToTest];
        j = y[sitesToTest];
        int ip = i+1;
        int im = i-1;
        int jp = j+1;
        int jm = j-1;
        if((ip < L) && (frame.getValue(ip, j) == 1)) {
            sitesToTest = addList(x, y, sitesToTest, ip, j);
        }
        if((im > -1) && (frame.getValue(im, j) == 1)) {
            sitesToTest = addList(x, y, sitesToTest, im, j);
        }
        if((jp < L) && (frame.getValue(i, jp) == 1)) {
            sitesToTest = addList(x, y, sitesToTest, i, jp);
        }
        if((jm > -1) && (frame.getValue(i, jm) == 1)) {
            sitesToTest = addList(x, y, sitesToTest, i, jm);
        }
    }
}

```

```

int addList(int x [], int y [], int sitesToTest, int i, int j) {
    frame.setValue(i, j, clusterNumber); // color site
    x[sitesToTest] = i; // add to list to test neighbors
    y[sitesToTest] = j;
    sitesToTest++;
    return sitesToTest;
}

public void reset() {
    control.setValue("Lattice size", 32);
    control.setValue("p", 0.2);
    seed = 0; // always start with the same random numbers
    random.setSeed(seed);
    calculate ();
}

public static void main(String args[]) {
    CalculationControl control = CalculationControl.createApp(new PercolationApp());
    control.addButton("newSeed", "New Seed");
}
}

```

The percolation threshold p_c is defined as the probability p at which a spanning cluster first appears in an infinite lattice. However, for the finite lattices of linear dimension L that we can simulate on a computer, there is a nonzero probability of a spanning cluster connecting one side of the lattice to the opposite side for any value of $p > 0$. For small p , this probability is order p^L (see Figure 12.4), and the probability of spanning goes to zero as L becomes large, and hence for small p and sufficiently large L , only finite clusters exist.

For a finite lattice, the definition of spanning is arbitrary. For example, we can define a spanning cluster as one that (i) spans the lattice either horizontally or vertically; (ii) spans the lattice in a fixed direction, for example, vertically; or (iii) spans the lattice both horizontally and vertically. In addition, the criteria for defining $p_c(L)$ for a finite lattice are somewhat arbitrary. One possibility is to define $p_c(L)$ as the average value of p at which a spanning cluster first appears. Another possibility is to define $p_c(L)$ as the value of p for which half of the configurations generated at random span the lattice. These criteria will lead to the same extrapolated value for p_c in the limit $L \rightarrow \infty$. These spanning rules are based on open boundary conditions, which we will use because the resulting clusters are easier to visualize. In Problem 12.1 we will find an estimated value for $p_c(L)$ that is accurate to about 10%. A more sophisticated analysis discussed in Project 12.14 allows us to extrapolate our results for $p_c(L)$ to $L \rightarrow \infty$. In Project 12.17 we will discuss the use of periodic boundary conditions to define the clusters.

Problem 12.1. Site percolation on the square lattice

- a. Use `PercolationApp` to generate random site configurations on a square lattice. Estimate $p_c(L)$ by finding the average value of p at which a spanning cluster is first attained. Choose a spanning rule, take $L = 4$, and begin at a value of p for which a spanning cluster is unlikely to be present. Then increase p in increments of 0.01 and record the value of p at which spanning first occurs.

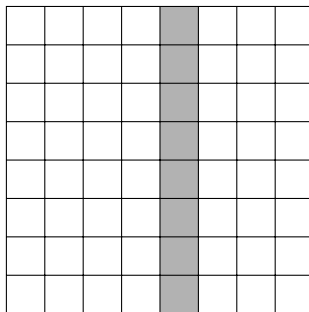


Figure 12.4: An example of a spanning cluster of probability p^L on a $L = 8$ lattice. In comparison, how many other ways are there of realizing a spanning cluster of L sites?

Repeat this process for a total of ten configurations and find the average value of $p_c(L)$. (Each configuration corresponds to a different set of random numbers.

- b. Repeat part (a) for $L = 16$ and 32 . Is $p_c(L)$ better defined for larger L , that is, are the values of $p_c(L)$ spread over a smaller range of values? How quickly can you visually determine the existence of a spanning cluster? Describe your visual “algorithm” for determining if a spanning cluster exists.

The value of p_c depends on the symmetry of the lattice and on its dimension. In addition to the square lattice, the most common two-dimensional lattice is the triangular lattice. As discussed in Chapter 8, the essential difference between the square and triangular lattices is the number of nearest neighbors.

***Problem 12.2.** Site percolation on the triangular lattice

Modify the `PercolationApp` class to simulate random site percolation on a triangular lattice. Assume that a connected path connects the top and bottom sides of the lattice (see Figure 12.5). Do you expect p_c for the triangular lattice to be smaller or larger than the value of p_c for the square lattice? Estimate $p_c(L)$ for $L = 4, 16$, and 32 . Are your results for p_c consistent with your expectations?

In *bond* percolation each lattice site is occupied, but only a fraction of the sites have connections or bonds between them and their nearest neighbor sites (see Figure 12.6). Each bond either is occupied with probability p or not occupied with probability $1 - p$. A cluster is a group of sites connected by occupied bonds. The wire mesh described in Section 12.1 is an example of bond percolation if we imagine cutting the bonds between the nodes rather than removing the nodes themselves. An application of bond percolation to the description of gelation is discussed in Problem 12.3.

***Problem 12.3.** Bond percolation on a square lattice

Suppose that all the lattice sites of a square lattice are occupied by monomers, each with functionality four, that is, each monomer can react to form a maximum of four bonds. This model is

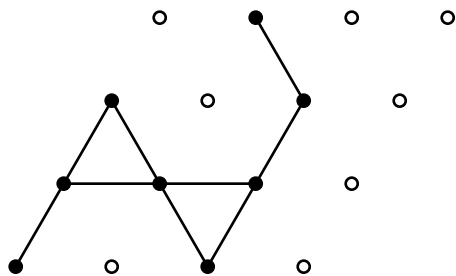


Figure 12.5: Example of a spanning cluster on a $L = 4$ triangular lattice. The bonds between the occupied sites (filled circles) are drawn to clarify the symmetry of the lattice.



Figure 12.6: Two examples of bond clusters. The occupied bonds are shown as bold lines.

equivalent to bond percolation on a square lattice. Also assume that the presence or absence of a bond between a given pair of monomers is random and is characterized by the probability p . For small p , the system consists of only finite polymers (groups of monomers) and the system is in the *sol* phase. For some threshold value p_c , there will be a single polymer that spans the lattice. We say that for $p \geq p_c$, the system is in the *gel* phase. How does a bowl of jello, an example of a gel, differ from a bowl of broth? Write a class to simulate bond percolation on a square lattice and determine the bond percolation threshold. Are your results consistent with the exact result, $p_c = 1/2$?

We also can consider *continuum* percolation models. For example, we can place disks at random into a two-dimensional box. Two disks are in the same cluster if they touch or overlap. A typical continuum (off-lattice) percolation configuration is depicted in Figure 12.7. One quantity of interest is the quantity ϕ , the fraction of the area (volume in three dimensions) in the system that is covered by disks. In the limit of an infinite size box, it can be shown that

$$\phi = 1 - e^{-\rho\pi r^2}, \quad (12.1)$$

where ρ is the number of disks per unit area, and r is the radius of a disk (see Xia and Thorpe). Equation (12.1) is not accurate for small boxes because disks located near the edge of the box might have a significant fraction of their area located outside of the box.

Problem 12.4. Continuum percolation

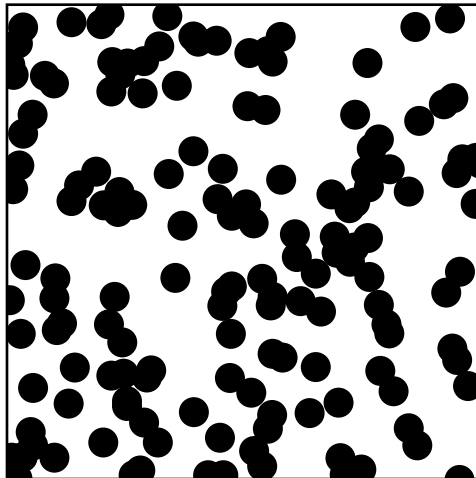


Figure 12.7: A model of continuum (off-lattice) percolation realized by placing disks of unit diameter at random into a square box of linear dimension L . If we concentrate on the voids between the disks rather than the disks, then this model of continuum percolation is known as the Swiss cheese model.

- a. Suppose that disks of unit diameter are placed at random on the sites of a square lattice with unit lattice spacing. Define ϕ as the area fraction covered by the disks. Convince yourself that $\phi_c = \pi p_c/4$.
- b. Modify `PercolationApp` to simulate continuum percolation. Instead of placing the disks on regular lattice sites, place their centers at random in a square box of area L^2 . The relevant parameter is the density ρ , the number of disks per unit area, instead of the probability p . We can no longer use the `LatticeFrame` class. Instead two arrays are needed to store the x and y locations of the disks. When the mouse is clicked on a disk, your program will need to determine which disk is at the location of the mouse, and then check all the other disks to see if they overlap with the disk you have chosen. This check needs to be recursively continued for all overlapping disks. To color the disks as in `PercolationApp`, it is useful to have an array that keeps track of the `clusterNumber` for each disk. Then only disks that have not been assigned a cluster number need to be checked for overlaps.
- c. Estimate the value of the percolation threshold ρ_c . Given this value of ρ_c , use a Monte Carlo method to estimate the corresponding area fraction ϕ_c (see Section 11.3). The procedure is to choose points at random in the box and compute the fraction of points that are within any disk.) Compare the value of ϕ_c for site and continuum percolation. Explain why ϕ_c is larger for continuum percolation than for site percolation. Compare your direct Monte Carlo estimate

of ϕ_c with the indirect value of ϕ_c obtained from (12.1) using the value of ρ_c . Explain any discrepancy.

- d. Consider the simple model of the cookie problem discussed in Section 12.1. Write a class that places disks at random into a square box and chooses their diameter randomly between 0 and 1. Estimate the value of ρ_c at which a spanning cluster first appears. How is the value of ρ_c changed from your estimate found in part (c)? Is your value for ϕ_c more or less than what was found in part (c)?
- e. A more realistic model of the cookie problem is to place disks with unit diameter at random in a square box with the constraint that the disks do not overlap. Continue to add disks until the probability of placing an additional disk becomes less than 1%, that is, when one hundred successive attempts at adding a disk are not successful. Then increase the diameters of all the disks at a constant rate (in analogy to the baking of the cookies) until a spanning cluster is attained. How does ϕ_c for this model compare with ϕ_c found in part (d)?

A continuum model that is applicable to random porous media is known as the *Swiss cheese* model. In this model the relevant quantity (the cheese) is the space between the disks. For the Swiss cheese model in two dimensions, the cheese area fraction at the percolation threshold, ψ_c , is given by $\psi_c = 1 - \phi_c$, where ϕ_c is the disk area fraction at the percolation threshold of the disks. Does such a relation hold in three dimensions (see Project 12.15)?

Our discussion of percolation has emphasized the existence of the percolation threshold p_c and the appearance of a spanning cluster or path for $p \geq p_c$. Another quantity that characterizes percolation is $P_\infty(p)$, the probability that an occupied site belongs to the spanning cluster. The probability P_∞ is defined as

$$P_\infty = \frac{\text{the number of sites in the spanning cluster}}{\text{the total number of occupied sites}}. \quad (12.2)$$

As an example, $P_\infty(p = 0.59) = 140/154$ for the single configuration shown in Figure 12.3b. A realistic calculation of P_∞ involves an average over many configurations for a given value of p . For an infinite lattice, $P_\infty(p) = 0$ for $p < p_c$ and $P_\infty(p) = 1$ for $p = 1$. Between p_c and 1, $P_\infty(p)$ increases monotonically.

More information can be obtained from the *mean cluster size distribution* $n_s(p)$ defined as

$$n_s(p) = \frac{\text{the average number of clusters of size } s}{\text{the total number of lattice sites}}. \quad (12.3)$$

For $p \geq p_c$, the spanning cluster is excluded from n_s . (For historical reasons, the *size* of a cluster refers to the *number* of sites in the cluster rather than to its spatial extent.) As an example, we see from Figure 12.3a that $n_s(1) = 20$, $n_s(2) = 4$, $n_s(3) = 5$, and $n_s(7) = 1$ for $p = 0.2$ and is zero otherwise.

Because $N \sum_s s n_s$ is the total number of occupied sites (N is the total number of lattice sites), and $N n_s$ is the number of occupied sites in clusters of size s , the quantity

$$w_s = \frac{s n_s}{\sum_s s n_s} \quad (12.4)$$

is the probability that an occupied site chosen at random is part of an s -site cluster. Hence, the mean cluster size S is given by

$$S = \sum_s s w_s = \frac{\sum_s s^2 n_s}{\sum_s s n_s}. \quad (12.5)$$

The sum in (12.5) is over the finite clusters only. As an example, the weights corresponding to the clusters in Figure 12.3a are $w_s(1) = 20/50$, $w_s(2) = 8/50$, $w_s(3) = 15/50$, and $w_s(7) = 7/50$, and hence $S = 130/50$.

Problem 12.5. Qualitative behavior of $n_s(p)$, $S(p)$, and $P_\infty(p)$

- Visually determine the cluster size distribution $n_s(p)$ for a square lattice with $L = 16$ and $p = 0.4$, $p = p_c$, and $p = 0.8$. Take $p_c = 0.5927$. Consider at least five configurations for each value of p and average $n_s(p)$ over the configurations. Because the lattice is finite, more consistent results can be obtained by discarding those configurations that have a spanning cluster for $p < p_c$ and those that do not have a spanning cluster for $p \geq p_c$. For each value of p , plot n_s as a function of s and describe the observed s -dependence. Does n_s decrease more rapidly with s for $p = p_c$ or for $p \neq p_c$? Better results for n_s can be found if periodic boundary conditions are used (see Project 12.17.)
- Use the same configurations considered in part (a) to compute the mean cluster size S as a function of p . Remember that for $p > p_c$, the spanning cluster is excluded.
- Similarly, compute $P_\infty(p)$ for $L = 16$, and for various values of $p \geq p_c$. Plot $P(p)$ as a function of p and discuss its qualitative behavior.
- Verify that $\sum_s s n_s(p) = p$ for $p < p_c$ and explain this relation. How is this relation modified for $p \geq p_c$?

12.3 Cluster Labeling

Your visual algorithm for determining the existence of a connected path is probably very sophisticated. Although `PercolationApp` helps us to estimate various quantities for a single configuration, we need to average over many configurations to obtain quantitative results. Hence, we need to develop an algorithm that finds the clusters quickly. In the following, we will find that this task is not straightforward because the assignment of a site to a cluster is a *global* rather than a *local* property of the site.

We first consider the cluster labeling algorithm of Hoshen and Kopelman. The algorithm can best be described by an example. Consider the configuration shown in Figure 12.8. The array `site` stores the occupancy of the sites; an occupied site initially is assigned the value -1 and an unoccupied site is assigned the value -2 . The array `site` is a one-dimensional array even though the lattice is two dimensional so that we can refer to each site by a single index to make the coding easier. The site index at (x, y) is $x + L*y$.

We assign cluster labels to sites beginning at the lower left corner and continue from left to right. Because the lower left hand corner site in Figure 12.8 is occupied, we assign to it cluster

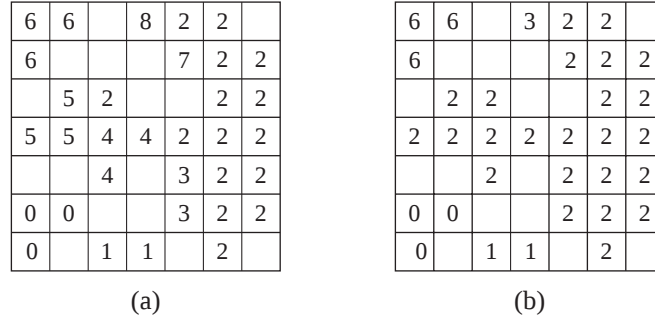


Figure 12.8: A percolation configuration on a square lattice with $L = 7$. Site coordinates are measured from the origin at the lower left corner $(0, 0)$. Part (a) shows the improper cluster labels initially assigned to the sites using the Hoshen-Kopelman algorithm. Part (b) shows the proper cluster labels.

label 0, the first cluster label. The next site is empty, and hence is not labeled. The next occupied site in the first row is at $x = 2$. Because its left neighbor is unoccupied, we assign to it the next available cluster label, label 1. The assignment of cluster labels to the sites in the remainder of the row is straightforward, and we proceed to the first site of the second row. Because this site is occupied and its nearest neighbor in the preceding row is labeled 0, we assign label 0 to `site[7]`. We continue from left to right along the second row checking the occupancy of each site. If a site is occupied, we check the occupancy of its nearest neighbors in the previous row and column. If neither neighbor is occupied, we assign the next available cluster label. If only one nearest neighbor site is occupied, the site is assigned the label of its occupied neighbor. For example, `site[8]` at $(1, 1)$ is assigned label 0 because its occupied neighbor, `site[7]` has label 0.

The initial difficulty arises when we come to an occupied site at which two clusters coalesce and cluster labels need to be reassigned. This case first occurs at `site[12]` at $(x, y) = (5, 1)$; its two neighbors in the previous row and column have labels 2 and 3, respectively. We define the *proper* cluster label assignment at `site[12]` to be the smaller of labels 2 and 3. Hence `site[12]` is assigned cluster label 2 and label 3 should be reassigned to label 2. It is inefficient to continually relabel the clusters, because there likely will be further reassignments. Hence, we delay the reassignment of cluster labels until the entire lattice is surveyed, and instead, keep track of the connections of the labels through a *label tree*. We introduce the array `properLabel` to distinguish proper and improper labels and provide their connections. Again, it is easier to explain the use of this array by considering the configuration shown in Figure 12.8. Before we came to `site[12]`, labels 0 through 3 were proper labels, and we set

$$\text{properLabel}[0] = 0, \text{properLabel}[1] = 1, \text{properLabel}[2] = 2, \text{properLabel}[3] = 3. \quad (12.6)$$

At `site[12]` where labels 2 and 3 are linked, we set `properLabel[3] = 2`. This reassignment of `properLabel[3]` tells us that label 3 is improper, and the value of `properLabel[3]` tells us that label 3 is linked to label 2. Note that the argument `i` of `properLabel[i]` is always greater than or equal to the value of `properLabel[i]`.

This procedure is still not complete. What should we do when we come to a site with two

previously labeled neighbors one or both of which are improper? For example, consider `site[25]` at (4,3) which has two occupied neighbors with labels 4 and 3. We might be tempted to assign `site[25]` the label 3 and set `properLabel[4] = 3`. However, instead of assigning to a site the minimum label of its two neighbors, we should assign to it the minimum of the *proper* labels of the two neighboring sites. In addition, if the two neighboring sites have different proper labels, then we should set `properLabel` of the maximum proper label equal to the minimum proper label. In this example, we have `properLabel[5] = 2`.

This version of the Hoshen-Kopelman cluster algorithm is implemented in class `Clusters`. There are five important arrays. Each element of `site` is initially either `-1` or `-2` indicating whether the site is occupied or not. Later `site[i]` will hold the cluster label for site `i`. Given the site `i` we can calculate `x` and `y` by `x = i % L` and `y = i / L`. The array elements `nn[i][0]` and `nn[i][1]` give the left and down neighbors of site `i`, respectively. Because there are no sites to the left of the first column and below the first row, these neighbors are assigned the value `N` which is one greater than the largest value of `i`, and `site[N]` is assigned the value `-2` which indicates an unoccupied site. The array element `next[i]` is the next site after `i` in the cluster containing `i`. If `next[i] == i`, then there are no more sites in this cluster. The ordering of the sites in a cluster is determined by the order used in labeling the sites. The array `first` is indexed by cluster labels and gives the first site in a cluster. The array `properLabel` gives the proper label for any label.

Listing 12.2: The `Clusters` class which implements the Hoshen-Kopelman algorithm.

```
/* Uses Hoshen-Kopelman algorithm for cluster labeling */
package org.opensourcephysics.sip.ch12;
import org.opensourcephysics.controls.Control;
import org.opensourcephysics.frames.PlotFrame;

public class Clusters {
    int[] site, next;
    int[][] nn;
    int L, N;
    int[] properLabel, first;
    int clusterNumber = 0;
    double p;
    double pInfinity = 0;
    double connectednessLength = 0;
    double spanProbability = 0;
    double meanSize = 0;
    double clusterDistribution[];
    int numberOccupied = 0;
    int numberOfTrials = 0;
    public void initialize() {
        N = L*L;
        setNeighborTable();
        reset();
    }

    public void setNeighborTable() {
        nn = new int[N][2];
        for(int i = 0; i < N; i++) {
```

```

        nn[i][0] = i-1; // left neighbor
        nn[i][1] = i-L; // down neighbor
    }
    for(int j = 0; j < L; j++) { // correct for fixed B.C.
        nn[j*L][0] = N; // N indicates a site outside lattice
        nn[j][1] = N;
    }
}

public void newLattice() {
    numberOfTrials++;
    clusterNumber = 0;
    numberOccupied = 0;
    site = new int[N+1];
    next = new int[N]; // index of next is a site number
    site[N] = -2; // indicates a site not in the lattice
    properLabel = new int[N];
    first = new int[N]; // index of first is a cluster number
    for(int i = 0; i < N; i++) {
        if(Math.random() < p) {
            site[i] = -1;
            numberOccupied++;
        } else {
            site[i] = -2; // not occupied
        }
        first[i] = -1;
    }
    assignLabels();
    accumulateAverages();
}

public void assignLabels() {
    for(int i = 0; i < N; i++) {
        if(site[i] == -1) {
            if(site[nn[i][0]] + site[nn[i][1]] == -4) { // new cluster, left and down site not occupied
                site[i] = clusterNumber;
                properLabel[clusterNumber] = clusterNumber;
                clusterNumber++;
            } else {
                merge(i);
            }
        }
    }
}

for(int i = 0; i < N; i++) {
    if(site[i] >= 0) {
        int properLabel = findProperLabel(site[i]); // assign proper labels
        site[i] = properLabel;
        if(properLabel == -1) { // first site in this cluster
            first[properLabel] = i;
            next[i] = i; // no next site
        }
    }
}

```

```

        } else {
            int temp = first[properLabel];
            first [properLabel] = i;
            next[i] = temp;
        }
    }
}

public void merge(int i) {
    if(( site [nn[i][0]]>=0)&&(site[nn[i][1]]>=0)) { // both left  and lower neighbors occupied and labeled
        minimumLabel(i);
    } else if( site [nn[i][0]]>=0) {
        site [i] = site [nn[i ][0]];
    } else {
        site [i] = site [nn[i ][1]];
    }
}

public void minimumLabel(int i) {                // both neighbors occupied
    if( site [nn[i][0]]==site [nn[i ][1]]) {
        site [i] = site [nn[i ][1]];
    } else {
        // find minimum cluster label
        int leftProper = findProperLabel(site[nn[i ][0]]);
        int downProper = findProperLabel(site[nn[i][1]]);
        int maxLabel = Math.max(leftProper, downProper);
        int minLabel = Math.min(leftProper, downProper);
        site [i] = minLabel;
        if(minLabel!=maxLabel) {
            properLabel[maxLabel] = minLabel;
        }
    }
}

public int findProperLabel(int label) { // recursive method
    if(properLabel[label]==label) {
        return label;
    } else {
        return findProperLabel(properLabel[label]);
    }
}

public void accumulateAverages() {
    double s2sum = 0;
    double ssum = 0;
    double xi2 = 0;
    boolean span = false;
    for(int label = 0;label<clusterNumber;label++) {
        if( first [label]>=0) { // cluster exists with this label
            boolean left = false;

```

```

    boolean right = false;
    double xbar = 0;
    double ybar = 0;
    int size = 0;
    int j = first [label];
    int i;
    do {
        i = j;
        int x = i%L;
        int y = i/L;
        if(x==0) {
            left = true;
        } else if(x==L-1) {
            right = true;
        }
        xbar += x;    // find center of mass
        ybar += y;
        size++;
        j = next[i];
    } while(i!=j);
    if( left && right) { //cluster spans horizontally
        if(!span) {
            spanProbability++;
            span = true;
        }
        pInfinity += (1.0*size)/numberOccupied;
    } else {
        clusterDistribution [ size]++;
        s2sum += 1.0*(size*size);
        ssum += 1.0*size;
        xbar = xbar/size;
        ybar = ybar/size;
        double r2sum = 0;
        j = first [label];
        do {
            i = j;
            r2sum += Math.pow(i%L-xbar, 2)+Math.pow(i/L-ybar, 2);
            j = next[i];
        } while(i!=j);
        xi2 += r2sum*size;
    }
} // end sum over clusters
}
connectednessLength += Math.sqrt(xi2/s2sum);
meanSize += s2sum/ssum;
}

public void reset() {
    pInfinity = 0;
    connectednessLength = 0;
}

```

```

    spanProbability = 0;
    meanSize = 0;
    numberOccupied = 0;
    numberOfTrials = 0;
    clusterDistribution = new double[N];
}

public void computeAverages(Control control) {
    control.println("\n" + "p = " + p + " L = " + L);
    control.println("Number of trials = " + numberOfTrials);
    control.println("Spanning probability = " + spanProbability/numberOfTrials);
    control.println("Probability of being in spanning cluster = " + pInfinity/numberOfTrials);
    control.println("Mean size = " + meanSize/numberOfTrials);
    control.println("Connecteness Length = " + connectednessLength/numberOfTrials);
}

public void distribution(PlotFrame plotFrame) {
    plotFrame.clearData();
    for(int size = 1; size < L*L; size++) {
        if( clusterDistribution [size] > 0) {
            plotFrame.append(0, Math.log(size), Math.log(clusterDistribution[size]/(numberOfTrials*L*L)));
        }
    }
    plotFrame.setVisible(true);
}
}

```

A more efficient version of the Hoshen-Kopelman algorithm written has been given by Stauffer and Aharony.

Problem 12.6. Test of the cluster labeling algorithm

Write a target class for **Clusters** and add a method to **Clusters** to display the site labels for a 8×8 lattice. Call this method before the labels are assigned; in method **assignLabels** after the labels have been assigned, but before each site is assigned the proper label; and after each site is assigned a proper label. Run your application and check that it is working correctly and you understand the Hoshen-Kopelman algorithm.

After the clusters are labeled, we can obtain a number of geometrical quantities of interest. The **Clusters** class includes a method to calculate the spanning probability P_∞ , the mean cluster size S , the mean connectedness length ξ , and the cluster size distribution, n_s . The mean cluster size S is computed from the relation

$$S = \frac{\sum_i m^2(i)}{\sum_i m(i)}. \quad (12.7)$$

where $m(i)$ is the number of sites in cluster i . Convince yourself that (12.7) is equivalent to the definition (12.5) of S . Note that the spanning cluster is not included in the sums in (12.7).

It is convenient to associate a characteristic linear dimension or *connectedness length* $\xi(p)$ with the clusters. One way to do so is to define the radius of gyration R_s of a single cluster of s particles

as

$$R_s^2 = \frac{1}{s} \sum_{i=1}^s (\mathbf{r}_i - \bar{\mathbf{r}})^2, \quad (12.8)$$

where

$$\bar{\mathbf{r}} = \frac{1}{s} \sum_{i=1}^s \mathbf{r}_i, \quad (12.9)$$

and $\mathbf{r}_i$ is the position of the i th site in the same cluster. The quantity $\bar{\mathbf{r}}$ is the familiar definition of the center of mass of the cluster. From (12.8), we see that R_s is the root mean square radius of the cluster measured from its center of mass.

The connectedness length ξ can be defined as an average over the radii of gyration of all the finite clusters. To find the appropriate average for ξ , consider a site on a cluster of s sites. The site is connected to $s - 1$ other sites and the mean square distance to these sites is R_s^2 . The probability that a site belongs to a cluster of size s is $w_s = sn_s$. These considerations suggest that a reasonable definition of ξ is

$$\xi^2 = \frac{\sum_s (s-1) w_s R_s^2}{\sum_s (s-1) w_s}. \quad (12.10)$$

To simplify the expression for ξ , we write s instead of $s - 1$, and let $w_s = sn_s$:

$$\xi^2 = \frac{\sum_s s^2 n_s R_s^2}{\sum_s s^2 n_s}. \quad (12.11)$$

As before, the sum in (12.11) is over only nonspanning clusters.

In Problem 12.7 we apply the Hoshen-Kopelman cluster algorithm to a more systematic study of site percolation.

Problem 12.7. Applications of the cluster labeling algorithm

- Modify your target class from Problem 12.6 to display the results from the `Clusters` class. Add code to your classes to determine if a spanning cluster exists. Assume that a spanning cluster spans the lattice both horizontally and vertically. Collect data for P_∞ , S , and ξ for $p = 0.55$ to $p = 0.65$ for $L = 8, 32$, and 128 . Also compute $P_{\text{span}}(p)$, the probability that a configuration spans. Average over at least 100 configurations.
- How does the qualitative behavior of $P_{\text{span}}(p)$ change with increasing L ? At what value of p is $P_{\text{span}} \approx 0.5$ for each value of L ? Call this value $p_c(L)$. How strongly does $p_c(L)$ depend on L ?
- Compute P_∞ for $p = p_c$. Use either the estimated value of $p_c(L)$ determined in part (b) or the known value $p_c \approx 0.5927$. What is the qualitative dependence of $P_\infty(p)$? Is $P_\infty(p = p_c)$ an increasing or decreasing function of L ? (Discard those configurations that do not have a spanning cluster.)
- What is the qualitative p -dependence of $S(p)$? How does $S(p = p_c)$ depend on L ?

- e. Plot $\xi(p)$ and discuss its qualitative dependence on p . Is $\xi(p)$ a monotonically increasing or decreasing function of p for $p < p_c$ and $p > p_c$? Modify your code to also display the results for $\xi(p)$ when the largest nonspanning cluster is not included. Does the largest nonspanning cluster make the dominant contribution to the sum? Determine its contribution for a few values of p near p_c .
- f. Modify your calculation of the averages so that for $p < p_c$ your program discards those configurations that contain a spanning cluster, and for $p > p_c$ it discards those configurations that do not have a spanning cluster. Then repeat parts (d) and (e). Do any of your conclusions change?
- g. Consider the cluster distribution, $n_s(p)$. Consider $p = p_c$ and $p = p_c \pm 0.1$ for $L = 8, 32$, and 128 and average over at least 100 configurations. Why is n_s a decreasing function of s ? Does n_s decrease more quickly for $p = p_c$ or for $p \neq p_c$?

12.4 Algorithm of Newman and Ziff

We now discuss a modification of the Hoshen-Kopelman algorithm based on work by Newman and Ziff. In the Hoshen-Kopelman algorithm the clusters for a given configuration at a particular value of p are determined. At each value of p the analysis must be repeated. In the Newman-Ziff algorithm the clusters are labeled continuously as we randomly occupy a new site on the lattice. Because $p = n/L^2$, where n is the number of occupied sites, p increases by $1/L^2$ each time we occupy a new site. As each site is occupied, we determine whether it becomes a new cluster, a neighbor of an existing cluster, or whether it bridges two or more clusters, thus leading to new cluster labels. The procedure is similar to that of the Hoshen-Kopelman algorithm except that we must consider all four neighbors, and need to be prepared to merge more than two clusters. Because it would be inefficient to relabel all the sites by the proper labels after each site is added, we keep track of the clusters as we occupy each new site. To do so we update the `first` and `next` arrays continuously and introduce another array, `last` which returns the last site in a cluster. When two clusters merge, we set `next[last[minLabel]]` equal to `first[maxLabel]` and `[last[minLabel]]` equal to `last[maxLabel]`. We also set `textttfirst[maxLabel]` equal to `-1` to indicate that `maxLabel` is no longer a proper label.

As an example, suppose two clusters contain the following sites: $A = \{7, 3, 17, 22, 14\}$ and $B = \{30, 31, 8\}$ and the proper label for A is 20 and the proper label for B is 10. Thus, `minLabel` = 10 and `maxLabel` = 20, which leads to `last[10]` = 8, `[last[20]` = 14, and `[first[20]` = 7. We want the combined cluster to be $\{30, 31, 8, 7, 3, 17, 22, 14\}$ with a proper label of 10. Thus, we set `next[last[minLabel]]` = `next[last[10]]` = `next[8]` equal to 7; `[last[minLabel]` = `last[10]` equal to `last[maxLabel]` = `last[20]` = 14; and `first[maxLabel]` = `first[20]` equal to `-1`. If there is a third cluster connected by the new occupation of a site, then this procedure must be repeated with the just combined cluster and the third cluster. This procedure would be repeated again if a fourth cluster is involved.

Because we wish to compute averages at many values of p , the accumulated data should be stored in arrays. The variable `index` can be used to index the arrays so that the array element `valueOfP[index]` is the value of p corresponding to `index`. Also because we do not want to compute averages for the entire range of p , the program is written so that it computes averages from `pMin` to `pMax` at intervals of p separated by `dp`. Two classes, `FastPercolation` and

FastPercolationApp, which implement the Newman-Ziff algorithm can be downloaded from the text website.

Problem 12.8. Applications of the Newman-Ziff cluster labeling algorithm

- a. Determine the physical quantities discussed in Problem 12.7 using the Newman-Ziff algorithm. How does the speed of the Newman-Ziff algorithm compare to that of the Hoshen-Kopelman algorithm?
- b. Compute the standard error of the mean for each quantity computed and estimate how many configurations need to be used to obtain 5% accuracy. Do you need more configurations near p_c ?

12.5 Critical Exponents and Finite Size Scaling

We are familiar with different phases of matter from our everyday experience. The most familiar example is water which can exist as a gas, liquid, or solid. It is well known that water changes from one phase to another at a well defined temperature and pressure, for example, the transition from ice to liquid water occurs at 0°C at atmospheric pressure. Such a change of phase is an example of a *thermodynamic phase transition*. Most substances also exhibit a *critical point*. For example, beyond a particular temperature and pressure, it is not possible to distinguish between the liquid and gaseous phases and the phase boundary terminates.

Another example of a critical point occurs in magnetic systems at the Curie temperature T_c and zero magnetic field. We know that at low temperatures some substances such as iron exhibit ferromagnetism, a spontaneous magnetization in the absence of an external magnetic field. If we raise the temperature of a ferromagnet, the spontaneous magnetization decreases and vanishes *continuously* at a critical temperature T_c . For $T > T_c$, the system is a paramagnet. In Chapter 15 we will use Monte Carlo methods to investigate the behavior of a magnetic system near the magnetic critical point.

In the following, we will find that the properties of the *geometrical* phase transition in percolation are qualitatively similar to the properties of the critical point in thermodynamic transitions. We will see that in the vicinity of a critical point, the qualitative behavior of the system is governed by the occurrence of long-range correlations.

We have found that the essential physics near the percolation threshold is associated with the existence of large but finite clusters. For example, for $p \neq p_c$, we found in Problem 12.7d that n_s decays rapidly with s . However for $p = p_c$, the s -dependence of n_s is qualitatively different, and n_s decreases much more slowly. This different behavior of n_s at $p = p_c$ is due to the presence of clusters of all length scales, for example, the “infinite” spanning cluster and the finite clusters of all sizes. We also found (see Problem 12.7e) that the mean connectedness length $\xi(p)$ is finite, and an increasing function of p for $p < p_c$ and a decreasing function of p for $p > p_c$ (see Figure 12.9). Moreover, we know that $\xi(p = p_c)$ is approximately equal to L and hence diverges as $L \rightarrow \infty$. These qualitative considerations lead us to conjecture that in the limit $L \rightarrow \infty$, $\xi(p)$ grows rapidly

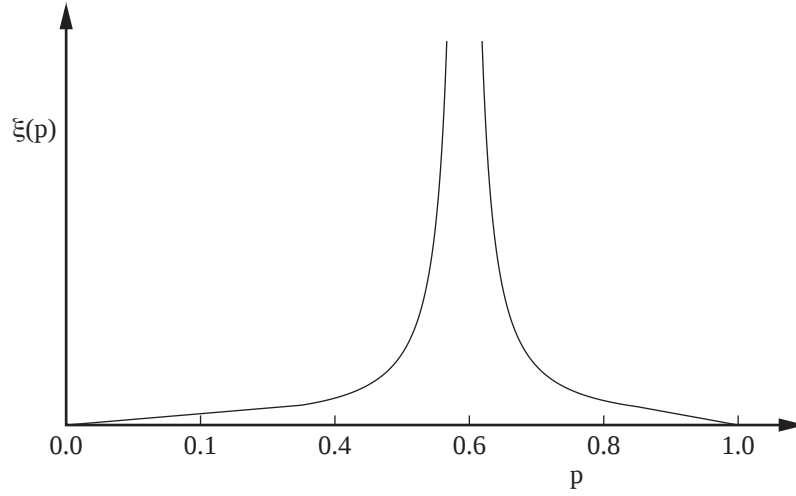


Figure 12.9: Qualitative p -dependence of the connectedness length $\xi(p)$. The divergent behavior of $\xi(p)$ in the critical region is characterized by the exponent ν (see (12.12)).

in the *critical region*, $|p - p_c| \ll 1$. We can describe the quantitative behavior of $\xi(p)$ for p near p_c by introducing the *critical exponent* ν , defined by the relation

$$\xi(p) \sim |p - p_c|^{-\nu}. \quad (12.12)$$

Of course, there is no *a priori* reason why the divergence of $\xi(p)$ can be characterized by a simple power law. Note that the exponent ν is assumed to be the same above and below p_c .

How do the other quantities that we have considered behave in the critical region in the limit $L \rightarrow \infty$? According to the definition (12.2) of P_∞ , $P_\infty = 0$ for $p < p_c$ and is an increasing function of p for $p > p_c$. We conjecture that in the critical region, the increase of P_∞ with increasing p is characterized by the exponent β defined by the relation

$$P_\infty(p) \sim (p - p_c)^\beta. \quad (12.13)$$

Note that P_∞ is assumed to approach zero continuously as p approaches p_c from above, that is, the percolation transition is an example of a *continuous* phase transition. In the language of critical phenomena, P_∞ is an example of an *order parameter*. We see that P_∞ is nonzero in the ordered phase, $p \geq p_c$ and zero in the disordered phase, $p < p_c$.

The mean number of sites in the finite clusters, $S(p)$, also diverges in the critical region. Its critical behavior is written as

$$S(p) \sim |p - p_c|^{-\gamma}, \quad (12.14)$$

which defines the critical exponent γ . The common critical exponents for percolation are summarized in Table 12.1. The analogous critical exponents of a magnetic critical point also are shown.

Quantity	Functional form	Exponent	$d = 2$	$d = 3$
Percolation				
order parameter	$P_\infty \sim (p - p_c)^\beta$	β	5/36	0.4
mean size of finite clusters	$S(p) \sim p - p_c ^{-\gamma}$	γ	43/18	1.8
connectedness length	$\xi(p) \sim p - p_c ^{-\nu}$	ν	4/3	0.9
cluster numbers	$n_s \sim s^{-\tau} (p = p_c)$	τ	187/91	2.2
Ising model				
order parameter	$M(T) \sim (T_c - T)^\beta$	β	1/8	0.32
susceptibility	$\chi(T) \sim T - T_c ^{-\gamma}$	γ	7/4	1.24
correlation length	$\xi(T) \sim T - T_c ^{-\nu}$	ν	1	0.63

Table 12.1: Several of the critical exponents for the percolation and magnetism phase transitions in $d = 2$ and $d = 3$ dimensions. Ratios of integers correspond to known exact results. The critical exponents for the Ising model are discussed in Chapter 15.

Because we can simulate only finite lattices, a direct fit of the measured quantities ξ , P_∞ , and $S(p)$ to their assumed critical behavior for an infinite lattice would not yield good estimates for the corresponding exponents ν , β , and γ (see Problem 12.9a). The problem is that if p is close to p_c , the connectedness length of the largest cluster becomes comparable to L , and the nature of the clusters is affected by the finite size of the system. In contrast, for p far from p_c , $\xi(p)$ is small in comparison to L , and the measured values of ξ , and hence the values of other physical quantities, are not appreciably affected by the finite size of the lattice. Hence for $p \ll p_c$ and $p \gg p_c$, the properties of the system are indistinguishable from the corresponding properties of a truly macroscopic system ($L \rightarrow \infty$). However, if p is close to p_c , $\xi(p)$ is comparable to L and the nature of the system differs from that of an infinite system. In particular, a finite lattice cannot exhibit a true phase transition characterized by divergent physical quantities. Instead, ξ reaches a finite maximum at $p = p_c(L)$.

The effects of the finite system size can be made more quantitative by the following argument. Consider for example, the critical behavior (12.13) of P_∞ . If $\xi \gg 1$, but is much less than L , the power law behavior given by (12.13) is expected to hold. However, if ξ is comparable to L , ξ cannot change appreciably and (12.13) is no longer applicable. This qualitative change in the behavior of P_∞ and other physical quantities occurs for

$$\xi(p) \sim L \sim |p - p_c|^{-\nu}. \quad (12.15)$$

We invert (12.15) and write

$$|p - p_c| \sim L^{-1/\nu}. \quad (12.16)$$

The difference $|p - p_c|$ in (12.16) is the “distance” from the percolation threshold point at which finite size effects occur. Hence, if ξ and L are approximately the same size, we can replace (12.13) by the relation

$$P_\infty(p = p_c) \sim L^{-\beta/\nu}. \quad (L \rightarrow \infty) \quad (12.17)$$

The relation (12.17) between P_∞ and L at $p = p_c$ is consistent with the fact that a phase transition is defined only for infinite systems.

One implication of (12.17) is that we can use it to determine the critical exponents. This method of analysis is known as *finite size scaling*. Suppose that we generate percolation configurations at $p = p_c$ for different values of L and analyze P_∞ as a function of L . If our values of L are sufficiently large, we can use the asymptotic relation (12.17) to estimate the ratio β/ν . A similar analysis can be used for $S(p)$ and other quantities of interest. We use this method in Problem 12.9.

Problem 12.9. Finite size scaling analysis of critical exponents

- Compute P_∞ at $p = p_c$ for at least 100 configurations for $L = 10, 20, 40$, and 80 . Include in your average only those configurations that have a spanning cluster. Best results are obtained using the value of p_c for the infinite square lattice, $p_c \approx 0.5927$. Plot $\ln P_\infty$ versus $\ln L$ and estimate the ratio β/ν .
- Use finite size scaling arguments to determine the dependence of the mean cluster size S on L at $p = p_c$. Average S over the same configurations as considered in part (a). Remember that S is the mean number of sites in the nonspanning clusters.
- Find the mass (number of particles) M in the spanning cluster at $p = p_c$ as a function of L . Use the same configurations as in part (a). Determine an exponent from a plot of $\ln M$ versus $\ln L$. This exponent is called the fractal dimension of the cluster and is discussed in Chapter 13.

We found in Section 12.2 that the numerical value of the percolation threshold p_c depends on the symmetry and dimension of the lattice, for example, $p_c \approx 0.5927$ for the square lattice and $p_c = 1/2$ for the triangular lattice. A remarkable feature of the power law dependencies summarized in Table 12.1 is that the values of the critical exponents do not depend on the symmetry of the lattice and are independent of the existence of the lattice itself, for example, they are identical for the continuum percolation model discussed in Problem 12.4. Moreover, it is not necessary to distinguish between the exponents for site and bond percolation. In the vocabulary of critical phenomena, we say that site, bond, and continuum percolation all belong to the same *universality class* and that their critical exponents are identical for the same spatial dimension.

Another important idea in critical phenomena is the existence of relations between the critical exponents. An example of such a *scaling law* is

$$2\beta + \gamma = \nu d, \quad (12.18)$$

where d is the spatial dimension of the lattice. The scaling law (12.18) indicates that the universality class depends on the spatial dimension. A more detailed discussion of finite size scaling and the scaling laws can be found in Chapter 15 and in the references.

12.6 The Renormalization Group

In Section 12.5, we studied the properties of various quantities on different length scales to determine the values of the critical exponents. The idea of examining physical quantities near the critical point on different length scales can be extended beyond finite size scaling and is the basis of the *renormalization group* method, probably the most important new method developed in theoretical physics in the last several decades. Kenneth Wilson was honored with the Nobel prize in physics

in 1981 for his contributions to the development of the renormalization group method. Although the method was first applied to thermodynamic critical points, it is simpler to understand the method in the context of the percolation transition. We will find that the renormalization group method yields the critical exponents directly, and in combination with Monte Carlo methods, is more powerful than Monte Carlo methods alone.

To introduce the method, consider a photograph of a percolation configuration generated at $p = p_0 < p_c$. If we view the photograph from further and further distances, what would we see? Convince yourself that when you are far from the photograph, you would not be able to distinguish occupied sites that are adjacent to each other and would not be able to observe single site clusters. In addition, branches emanating from larger clusters and narrow bridges connecting large “blobs” of occupied sites would be lost in your distant view of the photograph. Hence for $p_0 < p_c$, the distant photograph looks like a percolation configuration generated at a value of $p = p_1$ less than p_0 . In addition, the connectedness length $\xi(p_1)$ of the remaining clusters is smaller than $\xi(p_0)$. If we took a snapshot of the distant photograph, enlarge it so that it is the same size as the original, and then view the new photograph from afar, it would look like a percolation configuration generated at a value of $p = p_2$ less than p_1 . If we were to continue this process, we eventually would be unable to distinguish any clusters, and the photograph would appear as if the configurations were generated at the trivial *fixed point* $p = 0$.

What would we observe as we move away from the photograph for $p_0 > p_c$? The same reasoning implies that we would begin to have difficulty seeing small regions of unoccupied sites as we move further away from the photograph. These regions of unoccupied sites become less discernible and the configuration would look as though a larger percentage of the lattice were occupied. Hence, the photograph would look like a configuration generated at a value of $p = p_1$ greater than p_0 with $\xi(p_1) < \xi(p_0)$. Similar reasoning suggests that if we continue this process, the configurations in the new photographs would eventually appear to be at the other trivial fixed point $p = 1$.

What would we observe at $p_0 = p_c$? At the percolation threshold, all length scales are present, and it does not matter which length scale we use to observe the system. Hence, the configurations would appear to be similar regardless of the distance at which we observe the photograph. In this sense, p_c is a special, *nontrivial* fixed point.

We now consider a way of using a computer to change the configurations in a way that is similar to the procedure that we have just described. Consider a square lattice that is partitioned into *cells* or *blocks* that cover the lattice (see Figure 12.10). If we view the lattice from the perspective in which the sites in a cell merge to become a new supersite or renormalized site, then the new lattice has the same symmetry as the original lattice. However, the replacement of cells by the new sites has changed the length scale — all distances are now smaller by a factor of b , where b is the linear dimension of the cell. Hence, the effect of a “renormalization” is to replace each group of sites by a single renormalized site and to rescale the connectedness length for the renormalized lattice by a factor of b .

How can we decide whether the renormalized site is occupied or not? Because we want to preserve the main features of the original lattice and hence its connectedness (and its symmetry), we assume that a renormalized site is occupied if the original group of sites spans the cell. For simplicity, we adopt the vertical spanning criterion. The effect of performing a renormalization transformation on typical percolation configurations for p above and below p_c is illustrated in

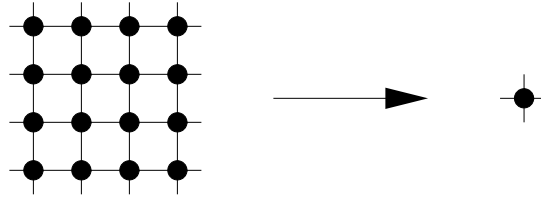


Figure 12.10: An example of a $b = 4$ cell used on the square lattice. The cell contains b^2 sites which are rescaled to a single supersite after a renormalization group transformation.

Figures 12.11 and 12.12 respectively. In both cases, the effect of the successive transformations is to move the system away from p_c . We see that for $p = 0.7$, the effect of the transformations is to drive the system toward $p = 1$. For $p = 0.5$, the trend is to drive the system toward $p = 0$. Because we began with a finite lattice, we cannot continue the renormalization transformation indefinitely.

The class `RGApp` implements a visual interpretation of the renormalization group. This class creates four windows with the original lattice in the first window and three renormalized lattices in the other three windows.

Listing 12.3: The visual renormalization class.

```
package org.opensourcephysics.sip.ch12;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.*;
import java.awt.Color;

public class RGApp extends AbstractCalculation {
    LatticeFrame originalLattice = new LatticeFrame("Original Lattice");
    LatticeFrame block1 = new LatticeFrame("First Blocked Lattice");
    LatticeFrame block2 = new LatticeFrame("Second Blocked Lattice");
    LatticeFrame block3 = new LatticeFrame("Third Blocked Lattice");
    public RGApp() {
        setLatticeColors( originalLattice );
        setLatticeColors(block1);
        setLatticeColors(block2);
        setLatticeColors(block3);
    }

    public void calculate() {
        int L = control.getInt("L");
        double p = control.getDouble("p");
        newLattice(L, p, originalLattice );
        block( originalLattice , block1, L/2); // block original lattice
        block(block1, block2, L/4);           // next blocking
        block(block2, block3, L/8);           // final blocking
        originalLattice .setVisible(true);
        block1.setVisible(true);
        block2.setVisible(true);
    }
}
```

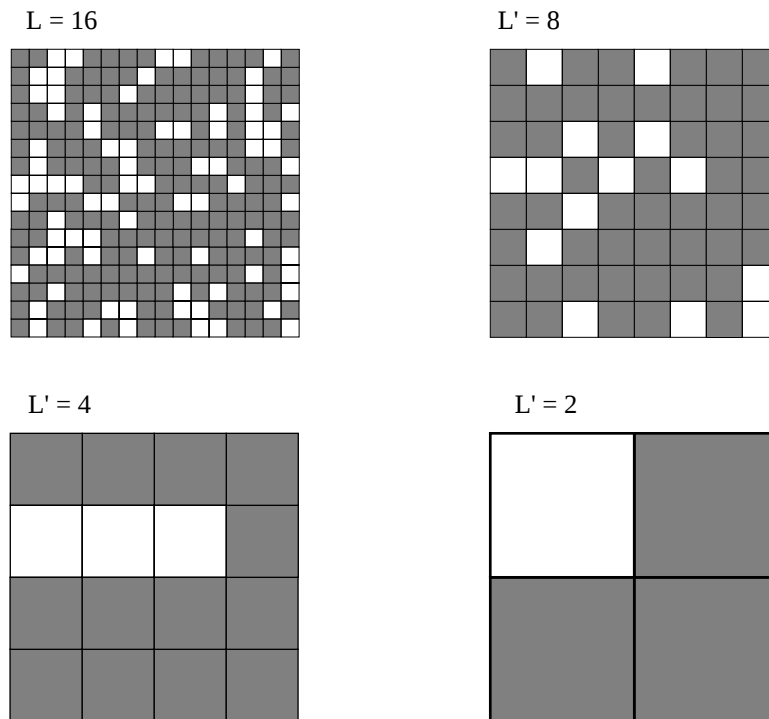


Figure 12.11: A percolation configuration generated at $p = 0.7$. The original configuration has been renormalized three times by transforming cells of four sites into one new supersite. What would be the effect of an additional transformation?

```

    block3.setVisible(true);
}

public void reset() {
    control.setValue("L", 64);
    control.setValue("p", 0.6);
}

public void newLattice(int L, double p, LatticeFrame lattice) { // new lattice
    lattice.resizeLattice(L, L);
    for(int i = 0; i < L; i++) {
        for(int j = 0; j < L; j++) {
            if(Math.random() < p) {
                lattice.setValue(i, j, 1);
            }
        }
    }
}
}

```

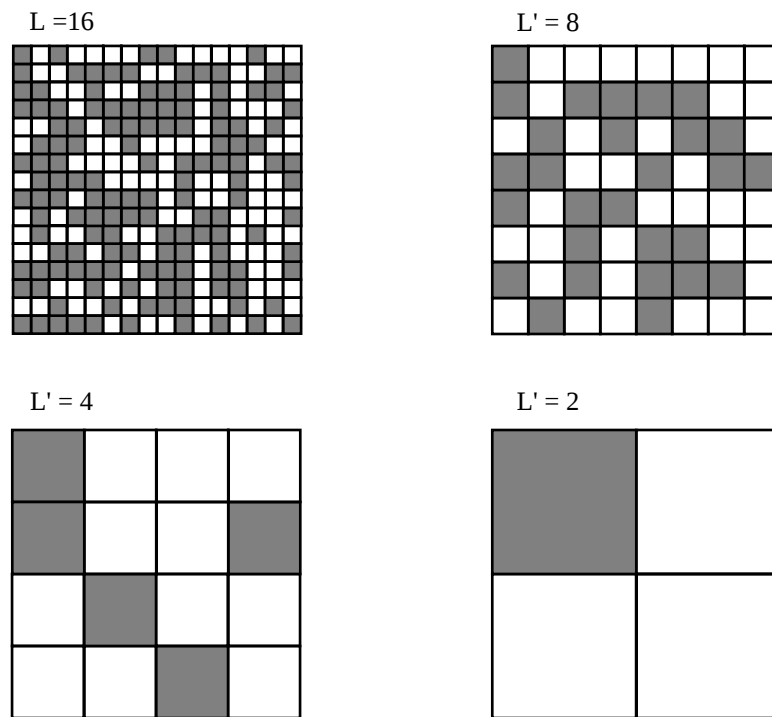


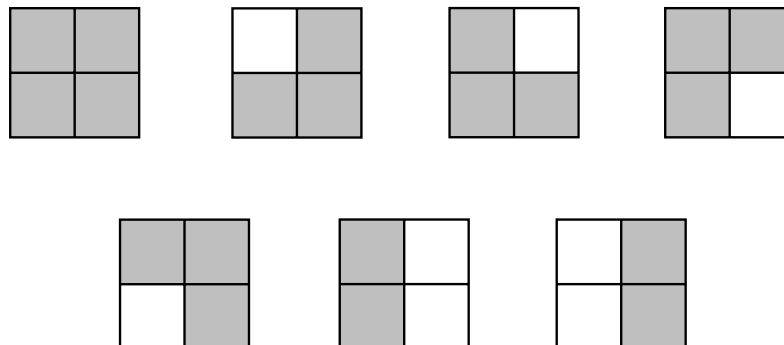
Figure 12.12: A percolation configuration generated at $p = 0.5$. The original configuration has been renormalized three times by transforming blocks of four sites into one new site. What would be the effect of an additional transformation?

```

public void block(LatticeFrame lattice, LatticeFrame blockedLattice, int Lb) {
    blockedLattice.resizeLattice(Lb, Lb);
    for(int ib = 0; ib < Lb; ib++) {
        for(int jb = 0; jb < Lb; jb++) {
            int leftCellsProduct = lattice.getValue(2*ib, 2*jb)*lattice.getValue(2*ib, 2*jb+1);
            int rightCellsProduct = lattice.getValue(2*ib+1, 2*jb)*lattice.getValue(2*ib+1, 2*jb+1);
            if(leftCellsProduct==1||rightCellsProduct==1) {
                blockedLattice.setValue(ib, jb, 1); // vertical spanning rule
            }
        }
    }
}

public void setLatticeColors(LatticeFrame lattice) {
    lattice.setIndexedColor(0, Color.WHITE);
    lattice.setIndexedColor(1, Color.BLUE);
}

```

Figure 12.13: The seven (vertically) spanning configurations on a $b = 2$ cell.

```

public static void main(String[] args) {
    CalculationControl.createApp(new RGApp());
}
}

```

Problem 12.10. Visual renormalization group

Use `RGApp` with $L = 64$ to estimate the value of the percolation threshold. For example, confirm that for small p , for example, $p = 0.4$, the renormalized lattice almost always renormalizes to a nonspanning cluster. What happens for $p = 0.8$? How can you use the properties of the renormalized lattices to estimate p_c ?

Although a visual implementation of the renormalization group allows us to estimate p_c , it does not allow us to estimate the critical exponents. In the following, we present a renormalization group method that allows us to obtain p_c and the critical exponent ν associated with the connectedness length. This analysis follows closely the method presented by Reynolds et al.

We adopt the same procedure as before, that is, we replace the b^d sites within a cell of linear dimension b by a single site that represents whether or not the original lattice sites span the cell. The second step is to determine the parameters that specify the new renormalized configuration. We make the simple approximation that each cell is independent of all the other cells and is characterized only by the probability p' that the cell is occupied. The relation between p and p' (the renormalization transformation) reflects the fact that the basic physics of percolation is connectedness, because we define a cell to be occupied only if it contains a set of sites that span the cell. If the sites are occupied with probability p , then the cells are occupied with probability p' , where p' is given by a renormalization transformation or a *recursion relation* of the form

$$p' = R(p). \quad (12.19)$$

The quantity $R(p)$ is the total probability that the sites form a spanning path.

An example will make the formal relation (12.19) more clear. In Figure 12.13, we show the seven vertically spanning site configurations for a $b = 2$ cell. The probability p' that the renormal-

ized site is occupied is given by the sum of the probabilities of all the spanning configurations:

$$p' = R(p) = p^4 + 4p^3(1-p) + 2p^2(1-p)^2. \quad (12.20)$$

In general, the probability p' of the occupied renormalized sites is different than the occupation probability p of the original sites. For example, suppose that we begin with $p = p_0 = 0.5$. After a single renormalization transformation, the value of p' from (12.20) is $p_1 = p' = R(p_0 = 0.5) = 0.44$. If we perform a second renormalization transformation, we have $p_2 = R(p_1) = 0.35$. It is easy to see that further transformations drive the system to the fixed point $p = 0$. Similarly, if we begin with $p = p_0 = 0.7$, we find that successive transformations drive the system to the fixed point $p = 1$. This behavior is qualitatively similar to what we observed in the visual renormalization group.

To find the nontrivial fixed point associated with the critical threshold p_c , we need to find the special value of p such that

$$p^* = R(p^*). \quad (12.21)$$

For the recursion relation (12.20), we find that the solution of the fourth degree equation for p^* yields the two trivial fixed points, $p^* = 0$ and $p^* = 1$, and the nontrivial fixed point $p^* = 0.61804$ which we associate with p_c . This calculated value of p^* for $b = 2$ should be compared with $p_c \approx 0.5927$.

To calculate the critical exponent ν , we recall that all lengths are reduced on the renormalized lattice by a factor of b in comparison to the lengths in the original system. Hence the connectedness length transforms as

$$\xi' = \xi/b. \quad (12.22)$$

Because $\xi(p) = A|p - p_c|^{-\nu}$ for p near p_c , where A is a constant, we have

$$|p' - p^*|^{-\nu} = b^{-1}|p - p^*|^{-\nu}, \quad (12.23)$$

where we have identified p_c with p^* . To find the relation between p' and p near p_c , we expand the renormalization transformation (12.19) in a Taylor series about p^* and obtain to first order in $(p - p^*)$:

$$p' - p^* = R(p) - R(p^*) \approx \lambda(p - p^*), \quad (12.24)$$

where

$$\lambda = \frac{dR(p = p^*)}{dp}. \quad (12.25)$$

We need to do a little algebra to obtain an explicit expression for ν . We first raise both sides of (12.24) to the ν th power and write

$$|p' - p^*|^\nu = \lambda^\nu(p - p^*)^\nu. \quad (12.26)$$

We then compare (12.26) and (12.23) and obtain

$$b = \lambda^\nu. \quad (12.27)$$

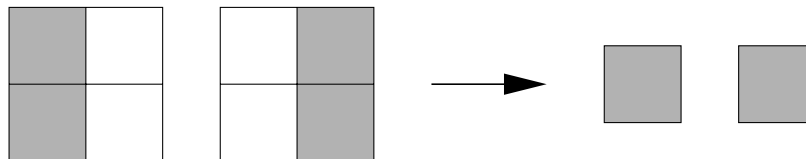


Figure 12.14: Example of the interface problem between cells. Two cells that are not connected at the original site level, but that are connected at the cell level.

Finally, we take the logarithm of both sides of (12.27) and obtain the desired relation for the critical exponent ν :

$$\nu = \frac{\log b}{\log \lambda}. \quad (12.28)$$

As an example, let us calculate λ for a square lattice with $b = 2$. We write (12.20) in the form $R(p) = -p^4 + 2p^2$. The derivative of $R(p)$ with respect to p yields $\lambda = 4p(1 - p^2) = 1.5279$ at $p = p^* = 0.61804$. We then use the relation (12.28) to obtain

$$\nu = \frac{\log 2}{\log 1.5279} \approx 1.635. \quad (12.29)$$

A comparison of (12.29) with the exact result $\nu = 4/3$ (see Table 12.1) in two dimensions shows remarkable agreement for such a simple calculation. (What would we be able to conclude if we were to measure $\xi(p)$ directly on a 2×2 lattice?) However, the accuracy of our calculation of ν is not known. What is the nature of our approximations? Our major assumption has been that the occupancy of each cell is independent of all other cells. This assumption is correct for the original sites, but after one renormalization, we lose some of the original connecting paths and gain connecting paths that are not present in the original lattice. An example of this interface problem is shown in Figure 12.14. Because this surface effect becomes less important with increasing cell size, one way to improve the renormalization group calculation is to consider larger cells. In Project 12.14 we combine the renormalization group method with a Monte Carlo approach to treat still larger cells. In Project 12.13 we consider a cell-to-cell method that does not require large cells and yields comparable accuracy.

Problem 12.11. Renormalization group method for small cells

- Enumerate the spanning configurations for a $b = 2$ cell assuming that a cell is occupied if a spanning path exists in either the vertical or the horizontal directions. Obtain the recursion relation and solve for the fixed point p^* . Use either a root finding algorithm or simple trial and error to find the value of $p = p^*$ such that $R(p) - p$ is zero. How do p^* and ν compare to their values using the vertical spanning criterion?
- Repeat the simple renormalization group calculation in part (a) using the criterion that a cell is occupied only if a spanning path exists in both directions.

- c.* The association of p_c with p^* is not the only possible one. Two alternatives involve the derivative $R'(p) = dR/dp$. For example, we could let $p_c = \int_0^1 p R'(p) dp$. Alternatively, we could choose $p_c = p_{\max}$, where $p_{\max}$ is the value of p at which $R'(p)$ has its maximum value. Compute p_c using these two alternative definitions and the various spanning criteria. In the limit of large cells, all three definitions should lead to the same values of p_c .
- d.* Enumerate the possible spanning configurations of a $b = 3$ cell, assuming that a cell is occupied if a cluster spans the cell vertically. Determine the probability of each configuration, and verify the renormalization transformation $R(p) = p^9 + 9p^8q + 36p^7q^2 + 67p^6q^3 + 59p^5q^4 + 22p^4q^5 + 3p^3q^6$. Solve the recursion relation (12.21) for p^* . Use this value of p^* to find the slope λ and the exponent ν . Then assume a cell is occupied if a cluster spans the cell both vertically and horizontally and obtain $R(p)$. Determine $p^*(b = 3)$ and $\nu(b = 3)$ for the two spanning criteria. Are your results for p^* and ν closer to their known values than for $b = 2$ for the same spanning criteria?

Problem 12.12. Renormalization group method for triangular lattice

- a. There are some difficulties with the renormalization group method we have discussed in the infinite cell limit, if a cell is said to span when there is a path in one fixed direction (see Ziff). This problem is absent for the triangular lattice. For this symmetry a cell can be formed by grouping three sites that form a triangle into one renormalized site. The only reasonable spanning criterion is that the cell spans if any two sites are occupied. Verify that $R(p) = p^3 + 3p^2(1 - p)$ and find $p_c = p^*$. How does p^* compare to the exact result $p_c = 1/2$?
- b. Calculate the critical exponent ν and compare its value with the exact result. Explain why b is given by $b^2 = 3$. Give a qualitative argument why the renormalization group argument might work better for small cells on a triangular lattice than on a square lattice.

It is possible to improve our renormalization group results for p_c and ν by enumerating the spanning clusters for larger b . However, because the 2^{b^2} possible configurations for a $b \times b$ cell increase rapidly with b , exact enumeration is not practical for $b > 7$, and we must use Monte Carlo methods if we wish to proceed further. Two Monte Carlo approaches are discussed in Project 12.14. The combination of Monte Carlo and renormalization group methods provides a powerful tool for obtaining information on phase transitions and other properties of materials.

As summarized in Table 12.1, the various critical exponents for percolation in two-dimensions are known exactly. For example, the exponent ν , corresponding to the divergence of the connectedness length, is $\nu = 4/3$. It is interesting that the argument for this result is algebraic, even though percolation is a geometrical phenomena. The percolation threshold is known exactly for the triangular lattice ($p_c = 1/2$). The most accurate estimate of p_c for the square lattice is $p_c = 0.59274621(13)$.

12.7 Projects

Most of the following projects require larger systems and more computer resources than the problems in this chapter, but they are not much more difficult conceptually. More ideas for projects can be obtained from the references.

Project 12.13. Cell-to-cell renormalization group method

In Section 12.6 we discussed the cell-to-site renormalization group transformation for a system of cells of linear dimension b . An alternative transformation is to go from cells of linear dimension b_1 to cells of linear dimension b_2 . For this cell-to-cell transformation, the rescaling length b_1/b_2 can be made close to unity. Many errors in a cell-to-cell renormalization group transformation cancel, resulting in a transformation that is more accurate in the limit in which the change in length scale is infinitesimal. We can use the fact that the connectedness lengths of the two systems are related by $\xi(p_2) = (b_1/b_2)^{-1}\xi(p_1)$ to derive the relation

$$\nu = \frac{\ln b_1/b_2}{\ln \lambda_1/\lambda_2}, \quad (12.30)$$

where $\lambda_i = dR(p^*, b_i)/dp$ is evaluated at the solution to the fixed point equation, $R(b_2, p^*) = R(b_1, p^*)$. Note that (12.30) reduces to (12.28) for $b_2 = 1$. Use the results you found in Problem 12.11d for one of the spanning criteria to estimate ν from a $b_1 = 3$ to $b_2 = 2$ transformation. Then consider larger values of b_2 and b_1 . Very accurate results for ν can be found for $b_1 = 5$ and $b_2 = 4$.

Project 12.14. Monte Carlo renormalization group

One way to estimate $R(p)$, the total probability of all the spanning clusters, can be understood by writing $R(p)$ in the form

$$R(p) = \sum_{n=1}^N S(n)P_N(n, p), \quad (12.31)$$

where

$$P_N(n, p) = \binom{N}{n} p^n q^{(N-n)}, \quad (12.32)$$

and $N = b^2$. The binomial coefficient $\binom{N}{n} = N! / [(N-n)!n!]$ represents the number of possible configurations of n occupied sites and $N-n$ empty sites; $P_N(n, p)$ is the probability that n sites out of N are occupied with probability p . The quantity $S(n)$ is the probability that a random configuration of n occupied sites spans the cell. A comparison of (12.20) and (12.31) shows that for $b = 2$ and the vertical spanning criterion, $S(1) = 0$, $S(2) = 2/6$, $S(3) = 1$, and $S(4) = 1$. What are the values of $S(n)$ for $b = 3$?

We can estimate the probability $S(n)$ by straightforward Monte Carlo methods. One way to sample $S(n)$ is to add a particle at random to an unoccupied site and check if a spanning cluster exists. If a spanning cluster does not exist, add another particle at random to a previously unoccupied site. If a spanning cluster exists after s particles are added, then let $S(n) = S(n) + 1$ for all $n \geq s$ and generate a new configuration. After a reasonable number of configurations, the results for $S(n)$ can be normalized. Of course, this procedure can be made more efficient by checking for a spanning cluster only after the total number of particles added is near $s \sim p^*N$.

- a. Write a Monte Carlo program to sample $S(n)$. Store the location of the unoccupied sites in a separate array. To check your program, first sample $S(n)$ for $b = 2$ and $b = 3$ and compare

your results to the exact results for $S(n)$. Consider larger values of b and determine $S(n)$ for $b = 5, 8, 16$, and 32 . Because the number of sites in the lattice can become very large, the direct evaluation of the binomial coefficients using factorials is not possible. One way to proceed is to approximate the probability of a configuration of n occupied sites by a Gaussian:

$$P_N(n, p) \approx \binom{N}{n} p^n q^{(N-n)} \approx (2\pi Npq)^{-\frac{1}{2}} e^{-(n-pN)^2/2Npq}. \quad (12.33)$$

Because $P_N(n)$ is sharply peaked for large b , it is necessary to sample $S(n)$ only near $n = p^*N$.

- b. As pointed out by Newman and Ziff, the Gaussian approximation for $P_N(n, p)$ is not sufficiently accurate for high precision studies. Instead, they used the following method. The binomial distribution is a maximum for a given N and p when $n = n_{\max} = pN$. Set this value to 1 for the moment. Then compute $P_N(n)$ iteratively for all other n using

$$P_N(n, p) = \begin{cases} P_N(N, n-1, p) \frac{N-n+1}{n} \frac{p}{1-p} & (n > n_{\max}) \\ P_N(N, n+1, p) \frac{n+1}{N-n} \frac{1-p}{p} & (n < n_{\max}) \end{cases} \quad (12.34)$$

Then calculate the normalization coefficient $C = \sum_n P_N(n, p)$ and divide all the $P_N(n, p)$ by C to normalize the probability distribution.

- c. It is possible to extrapolate the results for the successive estimates $p_c(b)$ and $\nu(b)$ to the limit $b \rightarrow \infty$. Finite size scaling arguments (cf. Stauffer and Aharony) suggest that

$$\nu(b) \approx \nu(\infty) - c_1/\ln b, \quad (12.35a)$$

and

$$p^*(b) \approx p_c(\infty) - a_1 b^{-1/\nu}, \quad (12.35b)$$

for b sufficiently large. The quantities a_1 and c_1 in (12.35) are fitting parameters. The relation (12.35a) suggests that the sequence $\nu(b)$ should be plotted as a function of $1/\ln b$ and the extrapolated result should be a straight line with an intercept of ν . The relation (12.35b) suggests that we should plot $p_c(b)$ versus $b^{-1/\nu}$ using the value of ν found from (12.35a). How sensitive are your result for p_c to on the assumed value of ν ? It is necessary to consider large cells and do a more sophisticated analysis of $\nu(b)$ and $p^*(b)$, to obtain extrapolated values that are consistent with the exact value $\nu = 4/3$ and the estimate $p_c = 0.5927$.

Project 12.15. Percolation in three dimensions

- a. The value of p_c for site percolation on the simple cubic lattice is approximately 0.311. Do a simulation to verify this value. Compute ϕ_c , the volume fraction occupied at p_c , if a sphere with a diameter equal to the lattice spacing is placed at each occupied site.
- b. Consider continuum percolation in three dimensions where spheres of unit diameter are placed at random in a cubical box of linear dimension L . Two spheres that overlap are in the same cluster. The volume fraction occupied by the spheres is given by

$$\phi = 1 - e^{-\rho 4\pi r^3/3}, \quad (12.36)$$

where ρ is the number density of the spheres, and r is their radius. Write a program to simulate continuum percolation in three dimensions and find the percolation threshold ρ_c . Use the Monte Carlo procedure discussed in Problem 12.4 to estimate ϕ_c and compare its value with the value determined from (12.36). How does ϕ_c for continuum percolation compare with the value of ϕ_c found for site percolation in part (a)? Which do you expect to be larger and why?

- c. In the Swiss cheese model in three dimensions, we are concerned with the percolation of the space between the spheres. This model is appropriate for porous rock with the spheres representing solid material and the space between the spheres representing the pores. Because we need to compute the connectivity properties of the space between the spheres, we superimpose a regular grid with lattice spacing equal to $0.1r$ on the system, where r is the radius of the spheres. If a point on the grid is not within any sphere, it is “occupied.” The use of the grid allows us to determine the connectivity between different regions of the pore space. Use your cluster labeling routine from part (a) to label the clusters, and determine $\tilde{\phi}_c$, the volume fraction occupied by the pores at threshold. You might be surprised to find that $\tilde{\phi}_c$ is relatively small. If time permits, use a finer grid and repeat the calculation to improve the accuracy of your results.
- d.* Use finite size scaling to estimate the critical percolation exponents for the three models presented in parts (a)–(c). Are they the same within the accuracy of your calculation?

Project 12.16. Fluctuations of the stock market

Although the fluctuations of the stock market are believed to be Gaussian for long time intervals, they are not Gaussian for short time intervals. The model of Cont and Bouchaud assumes that percolation clusters act as groups of traders who influence each other. The sites are occupied with probability p as usual. Each occupied site is a trader, and clusters are groups of traders (agents) who buy and sell together an amount proportional to the number s of traders in the cluster. At each time step each cluster is independently active with probability $2p_a$ and is inactive with probability $1 - 2p_a$. If a cluster is active, it buys with probability p_b and sells with probability $p_s = 1 - p_b$. In the simplest version of the model the change in the price of a stock is proportional to the difference between supply and demand, that is,

$$R = \sum_{\text{buy}} sn_s - \sum_{\text{sell}} sn_s, \quad (12.37)$$

where the constant of proportionality is taken to be one. If the probability p_a is small, at most one cluster trades at a time, and the distribution $P(R)$ of relative price changes or “returns” scales as $n_s(p)$. In contrast, for large p_a , the relative price variation is the sum of many clusters (not counting the spanning cluster), and the central limit theorem implies that $P(R)$ converges to a Gaussian for large systems (except at $p = p_c$). Confirm these statements and find the shape of $P(R)$ for $p = p_c$ and $p_a = 0.25$. Variations of the Cont-Bouchaud model can be found in the references. The application of methods of statistical physics and simulations to economics and finance is now an active area of research and is commonly known as *econophysics*.

Project 12.17. Spanning clusters and periodic boundary conditions

For simplicity, we have used open boundary conditions, partly for historical reasons, and partly because these boundary conditions make it easier to define a spanning cluster. An alternative is to use periodic boundary conditions and define a spanning cluster as one that wraps all the way

around the lattice. The use of periodic boundary conditions gives better results for the percolation threshold p_c and the cluster size distribution n_s for the same size lattice.

A clever method for detecting cluster wrapping has been employed by Machta et al. We describe the method for bond percolation, although it is easily applied to site percolation. We add to each site two integer variables giving the x and y displacements from that site to the site's parent in the appropriate tree. When we traverse the tree, we sum these displacements along the path traversed to find the total displacement to the root site. (We also update all displacements along the path when we carry out the path compression.) When an added bond connects together two sites that belong to the same cluster, we compare the total displacements to the root site for those two sites. If these displacements differ by just one lattice spacing, then cluster wrapping has not occurred. If they differ by any other amount, cluster wrapping has occurred. [xx this problem will be finish later xx]

Modify the Newman-Ziff algorithm for use with periodic boundary conditions to estimate p_c and n_s and determine if periodic boundary conditions give better results than open boundary conditions.

Project 12.18. Conductivity in a random resistor network

- a. An important critical exponent for percolation is the conductivity exponent t defined by

$$\sigma \sim (p - p_c)^t, \quad (12.38)$$

where σ is the conductance (or inverse resistance) per unit length in two dimensions. Consider bond percolation on a square lattice where each occupied bond between two neighboring sites is a resistor of unit resistance. Unoccupied bonds have infinite resistance. Because the total current into any node must equal zero by Kirchhoff's law, the voltage at any site (node) is equal to the average of the voltages of all nearest neighbor sites connected by resistors (occupied bonds). Because this relation for the voltage is the same as the algorithm for solving Laplace's equation on a lattice, the voltage at each site can be computed using the relaxation method discussed in Chapter 10. To compute the conductivity for a given $L \times L$ resistor network, we fix the voltage $V = 0$ at sites for which $x = 0$ and fix $V = 1$ at sites for which $x = L + 1$. In the y direction we use periodic boundary conditions. We then compute the voltage at all sites using the relaxation method. The current through each resistor connected to a site at $x = 0$ is $I = \Delta V/R = (V - 0)/1 = V$. The conductivity is the sum of the currents through all the resistors connected to $x = 0$ divided by L . In a similar way, the conductivity can be computed from the resistors attached to the $x = L + 1$ boundary. Write a program to implement the relaxation method for the conductivity of a random resistor network on a square lattice. An indirect, but easier way of computing the conductivity, is considered in Problem 13.8.

- b. The bond percolation threshold on a square lattice is $p_c = 0.5$. Use your program to compute the conductivity for a $L = 30$ square lattice. Average over at least ten spanning configurations for $p = 0.51, 0.52$, and 0.53 . Note that you can eliminate all bonds that are not part of the spanning cluster and all occupied bonds connected to only one other occupied bond. Why? If possible, consider more values of p . Estimate the critical exponent t defined in (12.38).
- c. Fix p at $p = p_c = 1/2$ and use finite size scaling to estimate the conductivity exponent t .

- d.* Use larger lattices and the multigrid method (see Project 10.26) to improve your results. If you have sufficient computing resources, compute t for a simple cubic lattice for which $p_c \approx 0.247$. (In two dimensions t is the same for lattice and continuum percolation. However, in three dimensions t can be different.)

References and Suggestions for Further Reading

- Joan Adler, “Series expansions,” *Computers in Physics* **8**, 287 (1994). The critical exponents and the value of p_c also can be determined by doing exact enumeration.
- I. Balberg, “Recent developments in continuum percolation,” *Phil. Mag.* **56**, 991 (1987). An earlier paper on continuum percolation is by Edward T. Gawlinski and H. Eugene Stanley “Continuum percolation in two dimensions: Monte Carlo tests of scaling and universality for non-interacting discs,” *J. Phys. A: Math. Gen.* **14**, L291 (1981). These workers divide the system into cells and use the Poisson distribution to place the appropriate number of disks in each cell.
- Armin Bunde and Shlomo Havlin, editors, *Fractals and Disordered Systems*, revised edition, Springer-Verlag (1996). Chapter 2 by the editors is on percolation.
- R. Cont and J.-P. Bouchaud, “Herd behavior and aggregate fluctuations in financial markets,” *cond-mat/9712318*.
- P. M. C. deOliveira, R. A. Nobrega, and D. Stauffer, “Are the tails of percolation thresholds Gaussians?,” *J. Phys. A* **37**, 3743–3748 (2004). The authors compute the probability that there is a spanning cluster at $p = p_c$.
- C. Domb, E. Stoll, and T. Schneider, “Percolation clusters,” *Contemp. Phys.* **21**, 577 (1980). This review paper discusses the nature of the percolation transition using illustrations from a film of a Monte Carlo simulation of a percolation process.
- J. W. Essam, “Percolation theory,” *Reports on Progress in Physics* **53**, 833 (1980). A mathematically oriented review paper.
- Jens Feder, *Fractals*, Plenum Press (1988). See Chapter 7 on percolation. We discuss the fractal properties of the spanning cluster at the percolation threshold in Chapter 13.
- J. P. Fitzpatrick, R. B. Malt, and F. Spaepen, “Percolation theory of the conductivity of random close-packed mixtures of hard spheres,” *Phys. Lett. A* **47**, 207 (1974). The authors describe a demonstration experiment done in a first year physics course at Harvard.
- J. Hoshen and R. Kopelman, “Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm,” *Phys. Rev. B* **14**, 3438 (1976). The original paper on an efficient cluster labeling algorithm.
- Chin-Kun Hu, Chi-Ning Chen, and F. Y. Wu, “Histogram Monte Carlo position-space renormalization group: applications to site percolation,” *J. Stat. Phys.* **82**, 1199–1206 (1996). The authors use a histogram Monte Carlo method that is similar to the method discussed in

- Project 12.14. A similar Monte Carlo method was used by M. Ahsan Khan, Harvey Gould, and J. Chalupa, “Monte Carlo renormalization group study of bootstrap percolation,” *J. Phys. C* **18**, L223 (1985).
- J. Machta, Y. S. Choi, A. Lucke, T. Schweizer, and L. M. Chayes, “Invaded cluster algorithm for Potts models,” *Phys. Rev. E* **54**, 1332–1345 (1996). The authors discuss the definition of a spanning cluster for periodic boundary conditions.
- P. H. L. Martins and J. A. Plascak, “Percolation on two- and three-dimensional lattices,” *Phys. Rev. E* **67**, 046119-1–6 (2003). The authors use the Newman-Ziff algorithm to compute various quantities.
- Ramit Mehr, Tal Grossman, N. Kristianpoller, and Yuval Gefen, “Simple percolation experiment in two dimensions,” *Am. J. Phys.* **54**, 271 (1986). A simple experiment for an undergraduate physics laboratory is proposed.
- M. E. J. Newman and R. M. Ziff, “Fast Monte Carlo algorithm for site or bond percolation,” *Phys. Rev. E* **64**, 016706 (2001).
- Peter J. Reynolds, H. Eugene Stanley, and W. Klein, “Large-cell Monte Carlo renormalization group for percolation,” *Phys. Rev. B* **21**, 1223 (1980). An especially clearly written research paper. Our discussion on the renormalization group in Section 12.6 is based upon this paper.
- Muhammad Sahimi, *Applications of Percolation Theory*, Taylor & Francis (1994). The emphasis is on modeling various phenomena in disordered media.
- Jean-Philippe Bouchaud and Marc Potters, *Theory of Financial Risk and Derivative Pricing: From Statistical Physics to Risk Management*, second edition, Cambridge University Press (2003); Rosario N. Mantegna and H. Eugene Stanley, *An Introduction to Econophysics: Correlations and Complexity in Finance*, Cambridge University Press (2000); Johannes Voit, *The Statistical Mechanics of Financial Markets*, second edition, Springer (2004). These texts introduce the general field of econophysics.
- Dietrich Stauffer, “Percolation models of financial market dynamics,” *Advances in Complex Systems* **4**, 19–27 (2001).
- D. Stauffer, “Percolation clusters as teaching aid for Monte Carlo simulation and critical exponents,” *Am. J. Phys.* **45**, 1001 (1977); D. Stauffer, “Scaling theory of percolation clusters,” *Physics Reports* **54**, 1 (1979).
- Dietrich Stauffer and Amnon Aharony, *Introduction to Percolation Theory*, second edition, Taylor & Francis (1994). A delightful book by two of the leading workers in the field. An efficient Fortran implementation of the Hoshen-Kopelman algorithm is given in Appendix A.3.
- B. P. Watson and P. L. Leath, “Conductivity in the two-dimensional-site percolation problem,” *Phys. Rev. B* **9**, 4893 (1974). A research paper on the conductivity of chicken wire.
- John C. Wierman and Dora Passen Naor, “Criteria for evaluation of universal formulas for percolation thresholds,” *Phys. Rev. E* **71**, 036143-1–7 (2005). Percolation theory was originally

conceived by mathematicians Broadbent and Hammersley in 1957 and continues to be of interest to mathematicians. Wierman and Naor evaluate several universal formulas that predict approximate values for p_c for various lattices.

Kenneth G. Wilson, “Problems in physics with many scales of length,” *Sci. Am.* **241**, 158 (1979).

An accessible article on the renormalization group method and its applications in particle and condensed matter physics. See also K. G. Wilson, “The renormalization group and critical phenomena,” *Rev. Mod. Phys.* **55**, 583 (1983). The latter article is the text of Wilson’s lecture on the occasion of the presentation of the 1982 Nobel Prize in Physics. In this lecture he claims that he “... found it very helpful to demand that a correctly formulated field theory be soluble by computer, the same way an ordinary differential equation can be solved on a computer ...”

W. Xia and M. F. Thorpe, “Percolation properties of random ellipses,” *Phys. Rev. A* **38**, 2650 (1988). The authors consider continuum percolation and show that the area fraction remaining after punching out holes at random is given by $\phi = e^{-A\rho}$, where A is the area of a hole, and ρ is the number density of the holes. This relation does not depend on the shape of the holes.

Richard Zallen, *The Physics of Amorphous Solids*, Wiley-Interscience (1983). Chapter 4 discusses many of the applications of percolation concepts to realistic systems.

Robert M. Ziff, “Spanning probability in 2D percolation,” *Phys. Rev. Lett.* **69**, 2670 (1992).