Institute for Theoretical Physics
University of Cologne

Prof. Dr. S. Trebst
Jan Attig

Computational Many-Body Physics

Exercise Sheet 1

Summer Term 2018

Due date: Monday, **23rd April** 2018, 10 am

Website: www.thp.uni-koeln.de/trebst/Lectures/2018-CompManyBody.shtml

Exercise 4: Percolation

Percolation theory is concerned with how **clusters** form and behave[1]. Although this might look like a rather academic exercise at first glance, percolation theory has many real-world applications where the lattice and its occupation structure represent such diverse things as the lattice of a solid and its magnetic moments, a forest and burning trees spreading in a wildfire, or the internet and virus-affected computers in it. As one varies the amount of occupied sites, the clusters undergo a phase transition where a large system-spanning cluster appears instead of only smaller disconnected clusters. This so called **percolation transition** and is probably the simplest manifestations of a **continuous phase transition**.

In this exercise, we want to study this very phase transition on a square lattice with a random occupation. The central element is to find an efficient algorithm that allows the identification of clusters. In fact, this problem is also well known in the field of computer science, albeit in a different context. The so-called **union-find** algorithm can be used to compute equivalent classes of a set. The corresponding equivalence relation in the percolation problem is whether two occupied lattice sites are connected or not. In statistical physics, this algorithm goes by the name of **Hoshen-Kopelman** algorithm.
Let us now describe the algorithm in a bit more detail. We start with a square lattice and initially fill the squares with zeros. For each square of the lattice we pick a random number and mark it "1" with a probability p. Connected regions of 1s are then called a cluster.

The identification using the Hosh-Kopelman algorithm works as follows. We start in the upper left corner and scan the lattice row by row from left to right. To each site, we then associate an integer that serves the purpose of identifying the respective cluster. If the site to the left or on top is also occupied and therefore already carries a label, we associate the same label to the current site. If not, we increment the label by one and assign this new label to the site at hand. It might happen that we encounter a site where both neighbors, left and top, were already assigned some label but not the same. In that case, we choose the smaller one and save that the two seemingly clusters are actually one and the same. The possible processes are depicted in Fig. 1.

The overall goal of this exercise is to verify the phase transition of the model. To reach this goal, the exercise is split into four subparts which are to be solved one after the other. It is highly encouraged to use the julia formalism of using functions.

[1] The word cluster, in the context of this exercise, describes a connected region on a randomly occupied lattice.

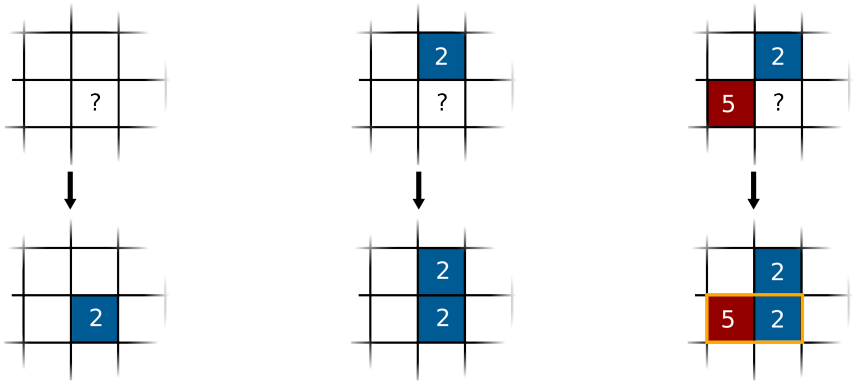

Figure 1: Rules for building clusters in the Hoshen-Kopelman algorithm: If an occupied site does not border any other occupied position yet, we assign a new cluster index (left). If only one site already carries a label, we simply assign the same one to the current site (middle). In the special case where two clusters border the same site, we use the smaller of the two labels and keep in mind that the clusters are actually connected (right).

a) As a necessary first step of any percolation problem, one needs to be able to generate a **randomly occupied (square) lattice**, i.e. in the present case a matrix of size $L \times L$ with entries "0" or "1", where a "1" has probability p to occur.

Your task is the following: First, implement a julia function which returns such a randomly occupied matrix when passing the value p of the probability. Then implement a second function to plot a given randomly occupied square lattice as an image. Verify both functions, i.e. check they give the expected result for a reasonable system size (e.g. $L = 40$).

b) In a second step, one has to **label all clusters** of a given randomly occupied square lattice.

Your task is to achieve this by implementing the Hoshen-Kopelman algorithm as a julia function that takes a randomly filled square lattice as the input and gives a cluster labeling as an output. More precisely: The idea is to feed this function the output of the function in part a) as an input and let it return another matrix in which every element is the cluster label of the respective site (or "-1" if the site is unoccupied).
Also implement a function to plot a given cluster labeling to visualy verify the correctness of your Hoshen-Kopelman implementation.

c) As a third step, one is now interested in **(quantitative) features** of a certain cluster labeling.

Your task is to implement functions which take a cluster labeling as an input and give

- the number of different clusters
- the size of the biggest cluster (number of connected sites)
- a boolean determining if there is a percolating cluster

Verify these functions with a square lattice of size $L = 10$ and check your results visually using the function of part b).

d) Up to this point, you have managed to implement various tools to generate and investigate a single lattice concerning its cluster occupation. However, as the last part of the exercise, one is interested in the **percolation probability** p_C of the system which can be determined numerically from the statistics of an ensemble of lattices.

Your task is to determine this probability by generating N randomly filled lattices and determining how many of them have a percolating cluster.

Plot the fraction of percolating lattices vs. the occupation probability and explain the resulting graph as well as how you can read off the percolation probability from this graph.

Further, plot the graph for various values of L into the coordinate system and explain the differences.

You may want to attempt a finite-size rescaling to see whether you can make the different finite-size data sets collapse onto one another.