UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Biophysics
A Computational Approach
Concepts, Models, Methods and Algorithms
Lecture 5: Pattern and Structure Formation

Dieter W. Heermann

Heidelberg University

August 31, 2016

Table of Contents

1. Introduction

2. Fractals
   - Basic Definitions
   - Iterated Function Systems
   - Percolation: Forrest Fires and the Like
   - Fractal Flow
3. Excercises
4. Bibliography
5. Index

Introduction I

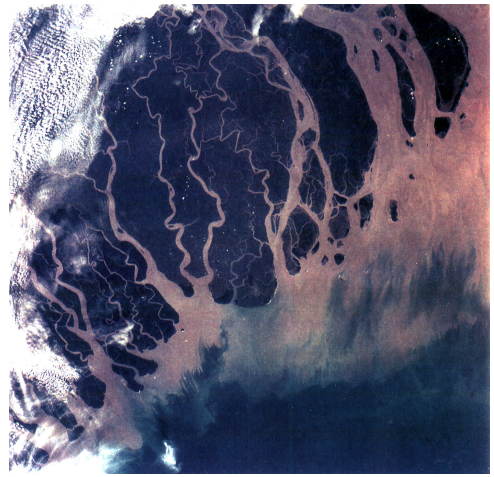
Pine cone

Source:http://jadecrompton.blogspot.de/2012/02/fibonacci-in-nature.html

Snowflake

Source: Wikipedia

https://en.wikipedia.org/wiki/Snowflake



River delta

Source: Wikipedia

https://en.wikipedia.org/wiki/River_delta

Fractals I

If every point in a set S has arbitrarily small neighbourhoods whose boundaries do not intersect S, then S has a topological dimension of 0.
The **topological dimension** of a subset of $S \subseteq \mathbb{R}^n$ is the least non-negative interger k such that each point of S has arbitrarily small neighbourhoods whose boundaries meet S in a set of dimension $k-1$.
A set S is **self-similar** (deterministically), if it can be divided into N congurent subsets, each of which when magnified by a constant factor M yields the entire set S.
The *fractal dimension* of a self-similar set S is

$$d_f = \frac{\log(N)}{\log(M)} \tag{1}$$

A **fractal** is a set whose fractal dimension exceeds it topological dimension. Clearly, there are many sets that are self-similar but not fractal. Since natural object are not the union of exact reduced copies of the entire set we need to extend the definition to include variations, i.e. **statistical self-similarity** .

Fractals: Koch Curve

The *Koch Curve* starts with a closed unit interval. At the first stage remove the middle third of the interval an replace it with two line segments of length 1/3 to form a virtual triangle. Repeat the procedure on every line segment.
The following function implements the Koch algorithm recursively.

```
def koch(length, depth):
    if length == 0:
        forward(depth)
    else:
        koch(t, length-1, depth/3)
        go_left(60)
        koch(length-1, depth/3)
        go_right(120)
        koch(length-1, depth/3)
        go_left(60)
        koch(length-1, depth/3)
```

Fractals: Koch Curve

Another possibility to implement the Koch algorithm is to use pattern matching and replacement of certain pattern in a resulting string. This is shown in the following listing

```
1 for i in xrange(steps):
      path = path.replace("F", "FLFRFLF")
```

Starting with a series of line segments that is closed we can generate constructs that have the appearance of snow flakes (c.f. Figure 1).

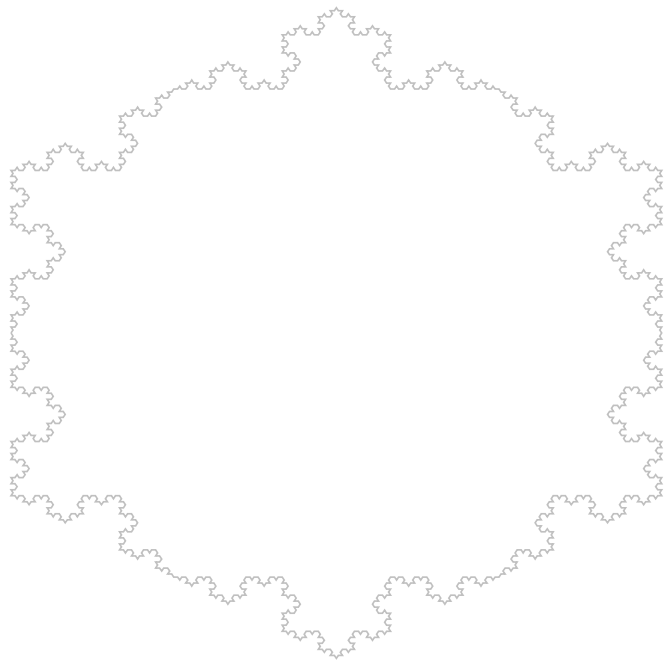


Figure: Koch curve generated by an iteration algorithm

Fractals (continued)

the relationship between scale and detail
Felix Hausdorff (1868-1942) and Abram Besicovitch(1891-1970)
The Hausdorff-Besicovitch Dimension
A natural idea to calculate the fractal dimension is to cover the object. Thus we cover an object with boxes and then calculate how many boxes N are needed to cover the object entirely. This depends on the linear dimension of the box b. Plot the results on a log-log plot and determine the slope. The resulting value of the slope is the box dimension

$$d_B \approx d_f \tag{2}$$

Algorithm 1 Box Counting Algorithm

Generate a non-overlapping regular lattice with lattice constant l
for boxes of the lattice **do**
  Compute the number of objects within the box
end for

Fractals: Box Counting Algorithm

In the box counting algorithm (box-covering method) for networks [1] we calculate the minimum number of boxes N_B of linear dimension l that is needed to cover the object

$$N_B(l) \sim l^{d_B} \tag{3}$$

where d_B is the fractal dimension.
In the cluster-growing method on counts the average number objects M_c that are within a range l

$$< M_c(l) > \sim l^{d_B} \tag{4}$$

if $N \sim N_B(l) < M_c(l) >$, then both calculated the same fractal dimension. In scale free networks we have

$$< M_c(l) > \sim e^{l/l_0} \tag{5}$$

with some constant l_0 [1].

Fractals: Box Counting Algorithm (continued)

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Algorithm 2 Random Sequential Box Covering Algorithm

while all not all vertices are burned and assigned to their respective boxes **do**
  Label all vertices as not burned *NB*
  Select a vertex randomly at each step; this vertex serves as a seed.
  Search the network by a distance *l* from the seed
  Burn all *NB* vertices that are within the distance *l* from the seed
  Assign newly burned vertices to the new box.
  if no newly burned vertex is found **then**
    the box is discarded
  end if
end while

Iterated Function Systems

Assume that we are given x_0 as a starting point and that we define two functions

$$F_0 = \frac{1}{3}x \tag{6}$$

$$F_1 = \frac{1}{3}(x-1)+1 \tag{7}$$

Consider the orbit of the initial value x_0 under the system of functions $\{F_0, F_1\}$ where at each step we choose to apply either F_0 or F_1 randomly with equal probability. The application of the system of functions is repeated for a fixed number of iterations. The end point of this iteration is then one point in a generated set of initial conditions. This basic algorithm is shown in Algorithm 3.

Algorithm 3 Iterated Function System

for *n_points* **do**
  Choose initial condition $x = x_0$
  for *n_cycles* **do**
    r iid from $\{0, 1, ...n-1\}$
    if $r == 0$ **then**
      $x = F_0(x)$
    else if $r == 1$ **then**
      $x = F_1(x)$
    else if $r == 2$ **then**
      ...
    else if $r == n-1$ **then**
      $x = F_{n-1}(x)$
    end if
  end for
  Add new fixed point x to set
end for

The above example can be easily generalized to higher dimensions.
As a further example we generate a fern. For this we we apply the following linear transformation

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} \tag{8}$$

following and using the parameters Bradley [2]:

$$F_0 \quad : \quad \begin{pmatrix} +0.00 & +0.00 \\ +0.00 & +0.16 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad 1\% \tag{9}$$

$$F_1 \quad : \quad \begin{pmatrix} +0.85 & +0.04 \\ -0.04 & +0.85 \end{pmatrix} \begin{pmatrix} 0 \\ 1.6 \end{pmatrix} \quad 85\% \tag{10}$$

$$F_2 \quad : \quad \begin{pmatrix} +0.20 & -0.26 \\ +0.23 & +0.22 \end{pmatrix} \begin{pmatrix} 0 \\ 1.6 \end{pmatrix} \quad 7\% \tag{11}$$

$$F_3 \quad : \quad \begin{pmatrix} -0.15 & +0.28 \\ +0.26 & +0.24 \end{pmatrix} \begin{pmatrix} 0 \\ 0.44 \end{pmatrix} \quad 7\% \tag{12}$$

Iterated Function Systems

Figure: Fern fractal produced by an iterated function system. The parameters are taken from *http://www.stsci.edu/ lbradley/seminar/ifs.html* with 500 iteration steps. Shown is the result for 100000.

Percolation I

Consider a lattice, which we take, for simplicity, as a two-dimensional square lattice. Each lattice site can be either occupied or unoccupied. A site is occupied with a probability $p \in [0, 1]$ and unoccupied with a probability $1 - p$. If p is small then only a tiny fraction of the sites is occupied and only small isolated patches of occupied sites that are close to each other exist. On the other hand, if p is large, then large patches of near lying occupied sites exist. We may at this point speculate that for p less than a certain probability p_c only finite clusters exist on the lattice. We define more precisely what we mean by patches or clusters by saying: A cluster is a collection of occupied sites connected by nearest-neighbour distances (see Figure 3). For p larger than or equal to p_c, there exists a large cluster (for an infinite lattice, i.e., in the thermodynamic limit) such that for an infinite lattice the fraction of sites belonging to the largest cluster is zero below p_c, and non zero above p_c. For $d = 1$ the situation is straight-forward: $p_c = 1$. For $d > 1$ the computation of p_c is non-trivial, and analytic results for the percolation threshold p_c are only available for certain dimensions or special lattices.

Percolation II

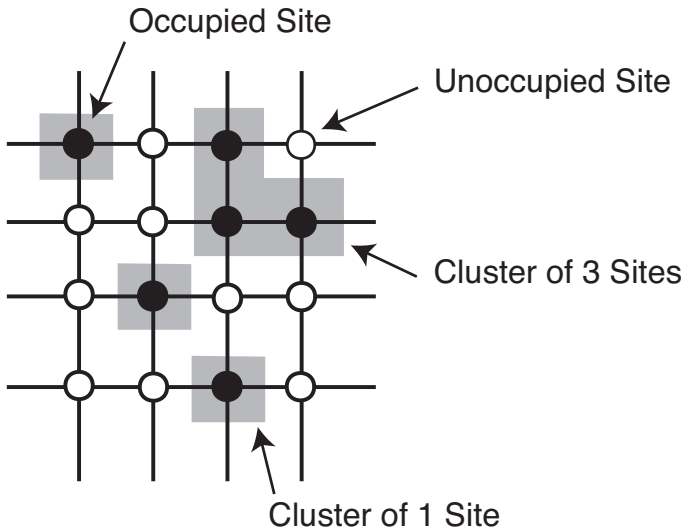
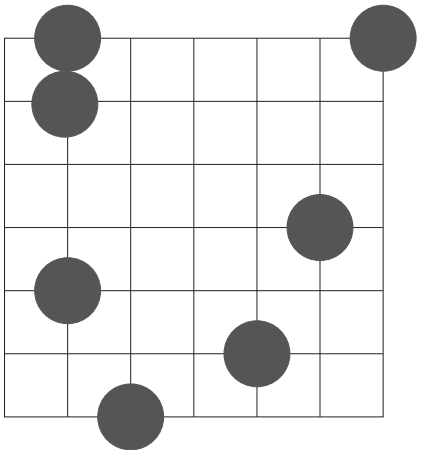
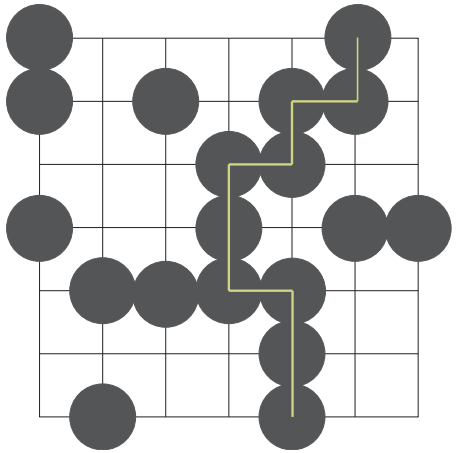


Figure: Definition of the percolation problem

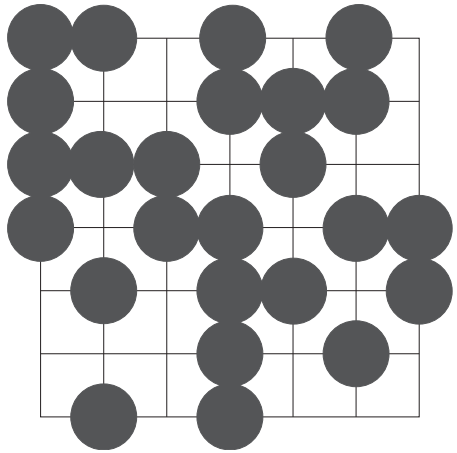
Percolation III



Occupation probability
$p = 0.1$



Occupation probability
$p = 0.59$



Occupation probability
$p > 0.59$

$$n_s(p, L) \tag{13}$$

$$p = \sum_{s<\infty} s n_s(p, L) \tag{14}$$

$$< s >= \sum_{s>0} s^2 n_s(p, L) / \sum_{s>0} s n_s(p, L) \tag{15}$$

Percolation IV

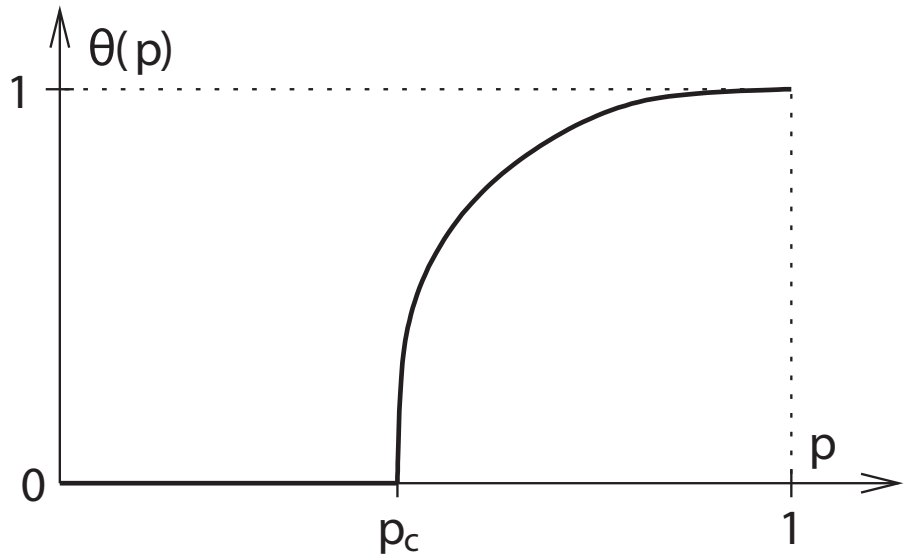


Figure: percolation probability

Percolation V

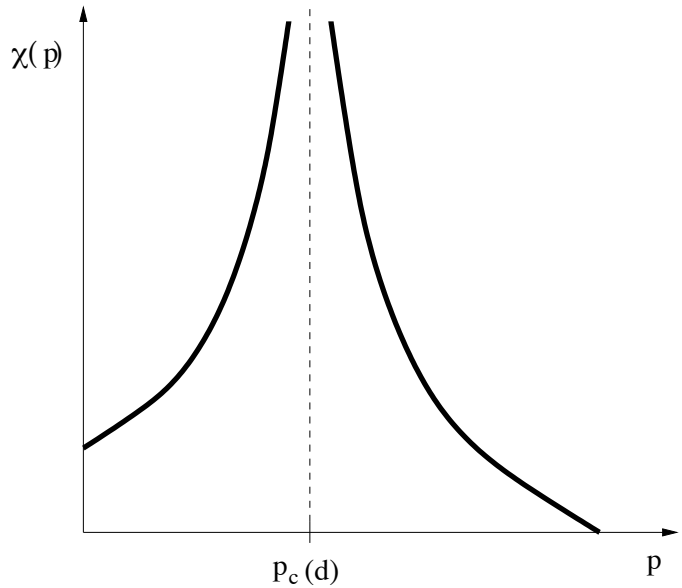


Figure: percolation susceptibility

Percolation VI

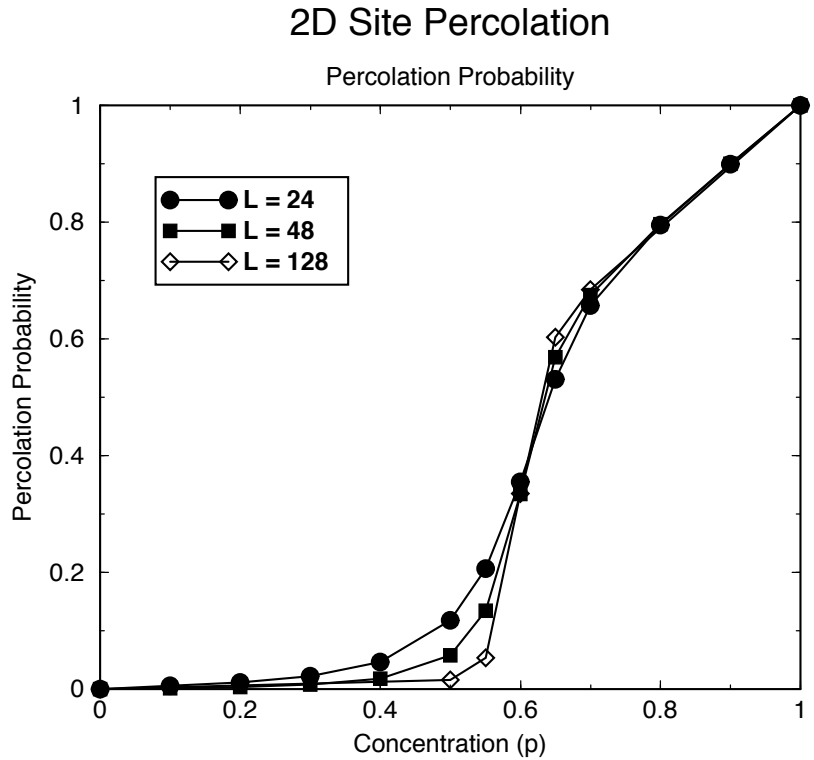


Figure: percolation probability

Percolation VII

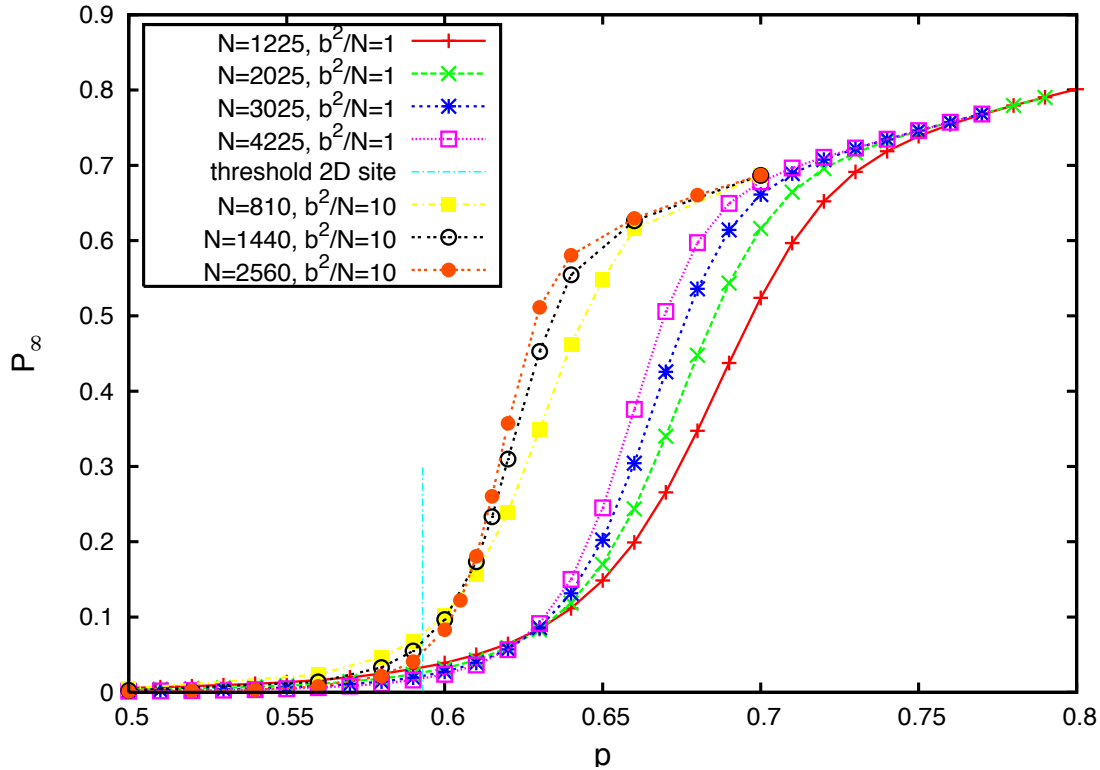


Figure: spanning probability

Percolation VIII

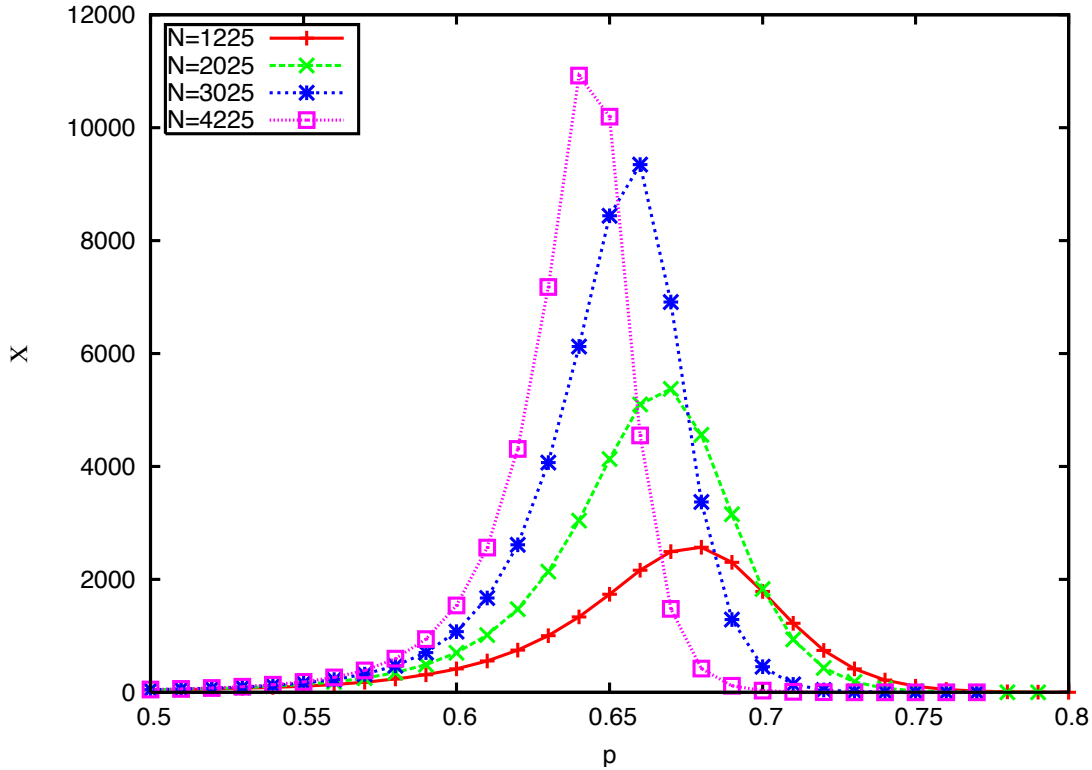


Figure: spanning susceptibility

Percolation IX

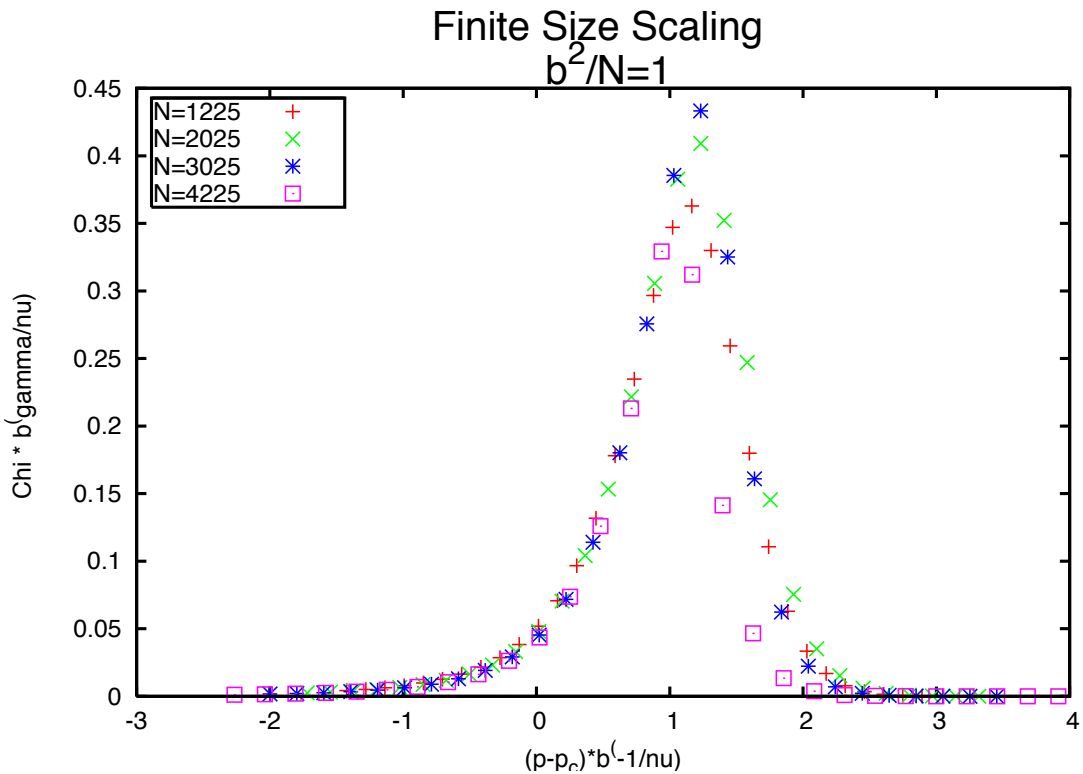


Figure: Scaling

Percolation I

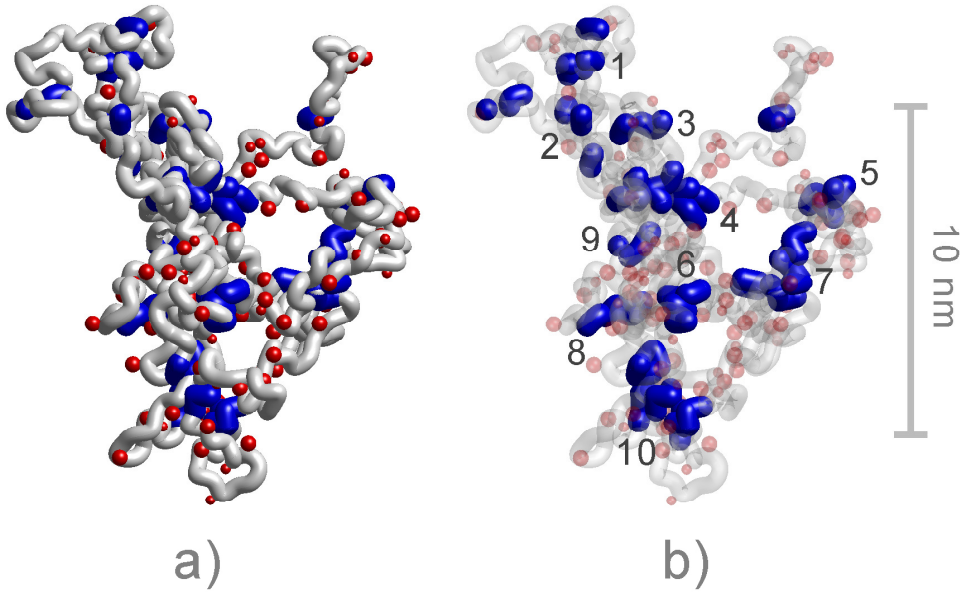


Figure: NPC cluster definition

Percolation II

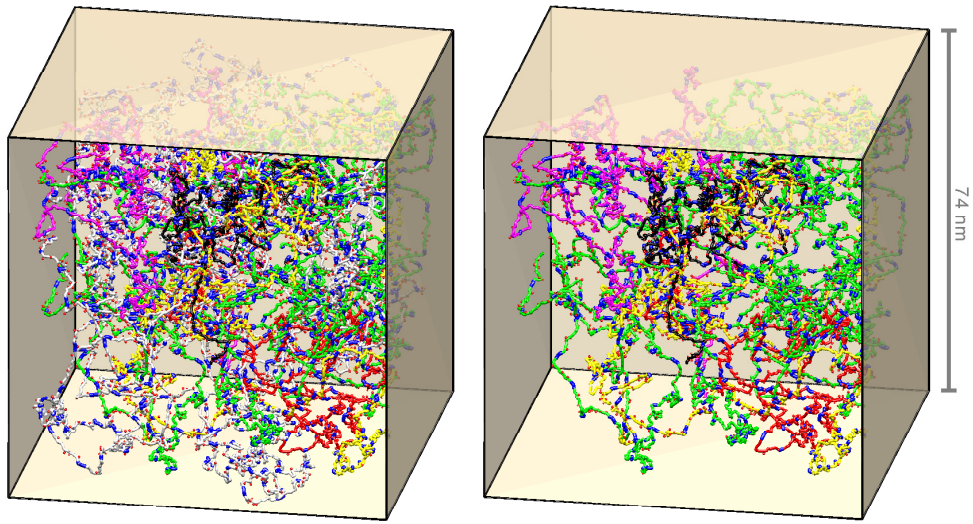


Figure: NPC cluster

Union Find Algorithm

```
def union(x,y,labels):
    if (x == 0):
        labels[y] += 1
        return y
    if (y == 0):
        labels[x] += 1
        return x
    if (x < y):
        count = labels[y] + 1
        labels[y] = -x
        labels[x] += count
        return x
    elif (x == y):
        labels[x] += 1
        return x
    else:
        count = labels[x] + 1
        labels[x] = -y
        labels[y] += count
        return y

    return x
```

Code 1: union.py

```
def find(x,labels):
   pointer = labels[x]
   if (pointer < 0):
      while( pointer < 0 ):
         la = -pointer
         pointer = labels[la]
      labels[x] = -la
      x         = la
   return x
```

Code 2: union.py

```
    # Find the first non-zero site
    i = 0
    found = False
    while (i<L and not found):
        j = 0
        while (j<L and not found):
            if (lattice[i][j]==1):
                index = (i,j)
                found = True
            j += 1
        i += 1

    if (not found):
        # All sites are empty: case p=0 or p very small
        return
    # create the first label
    i = index[0]
    j = index[1]
    lattice[i][j] = create(labels)
```

Code 3: union.py

```
def find_clusters(lattice,L,labels):

   #continue from there on, but first row and column are specific
   while (i<L):
      while (j<L):
         left = j-1
         if (left < 0):
            left = L-1
         top = i-1
         if (top < 0):
            top = L-1
         if (lattice[i][j] == 1):
            if (i == 0):
               top_label = 0
            else:
               top_label  = find(lattice[top][j],labels)
            if (j == 0):
               left_label = 0
            else:
               left_label  = find(lattice[i][left],labels)
            if ( top_label == 0 and left_label == 0):
               # isolated site
               lattice[i][j] = create(labels)
            else:
               lattice[i][j] = union(left_label,top_label,labels)
         j += 1
      j = 0
      i += 1

   return
```

Code 4: union.py

Percolation I

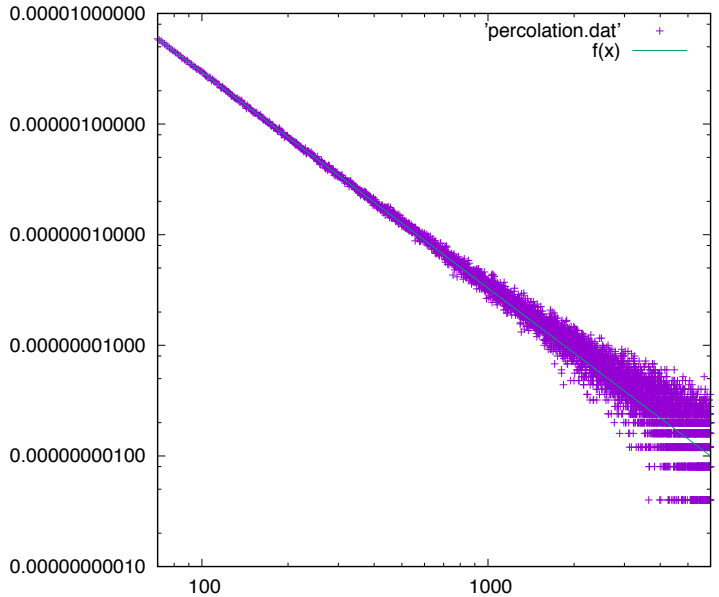


Figure: Number of clusters percolation: seed = 47115 max_steps = 10000 L = 500 p = 0.5927460

Fractal Flow I

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

The global objective of project B-2 arises from the question whether, and if yes what, statistical physics can contribute to a better understanding of porous materials. It is well known that important paradigms of statistical physics are closely related to phenomena arising during hydrodynamic flow in porous materials. When displacing a high viscosity fluid (such as oil) by a low viscosity fluid (such as water) in a porous medium one observes percolation clusters (more precisely invasion percolation) when capillary forces dominate. One observes diffusion limited aggregation clusters when viscous forces are dominant (see e.g. R. Lenormand and G. Daccord in "Random fluctuations and pattern growth" H.E. Stanley and N. Ostrowsky (eds.) NATO ASI Series E vol 157, Kluwer, Dordrecht 1988). A common feature of these observations is selfsimilarity. Self- similar or fractal phenomena have recently attracted much attention from statistical physicists (see e.g. "Fractals in Physics", L. Pietronero and E. Tosatti (eds.), Elsevier Amsterdam 1986), and much progress has been made in their study, as well as in the study of percolation and diffusion limited aggregation. The application of these results to structure and dynamics of porous media promises therefore to be of great interest.

Lung: Gas Diffusion through the Fractal Landscape of the Lung

Fractal Flow II

- Diffusion on Fractals: Three-dimensional percolation threshold.
  Particles are assumed to diffuse randomly in a porous medium. This medium is a simple cubic lattice consisting of pores (empty sites) and rock (occupied sites). Diffusion can take place only in the empty space, and empty and occupied sites are distributed randomly. Increasing the fraction of empty sites, we change from an impermeable to a permeable medium at a percolation threshold of 31 percent of empty sites. Right at this threshold, the pore space is fractal, and the diffusion becomes anomalous, i.e. the squared distance increases weaker than linearly with time.
  - M.J.Velgakis, Physica A 159, 167 (1989)
- Hydrodynamical Cellular automata
  The cellular automata approximation of hydrodynamics (Frisch, Hasslacher, and Pomeau, 1986) is presently restricted to two dimensions. Basically, molecules fly with unit velocity along the lattice directions of a triangular lattice, scattering on each other according to the law of momentum conservation. Brosa's implementation of this algorithm on the HLRZ Cray supercomputer is an order of magnitude faster than the diffusion simulation mentioned above. Duarte and Brosa simulated the flow around a cylinder between two plates and found good agreement with old experimental data. The same algorithm is presently applied to many such obstacles as a model for a porous medium. 2000 * 666 lattices can be simulated within a minute on a Cray YMP; we average over 8 such lattices with the same porous structure but a different initial velocity distribution.

Fractal Flow III

- U.Brosa and D.Stauffer, J.Statistical Physics 57, 399 (1989)
- J.A.M.S.Duarte+U.Brosa, J.Statistical Physics 59, 501 (1990)
- D.Stauffer, invited talk A4a, EPS Condensed Matter Conference, Lisbon, April 1990

Excercises I

Exercise 1: **Percolation Threshold**
$p_c = 0.59274601$ percolation threshold [3]

Exercise 2: **Bond Percolation**

Exercise 3: **Correlated Site-Bond Percolation**

Exercise 4: **Snowflake Cellular Automaton** hfill
abc

Exercise 5: **Multiple Reduction Copy Machine Algorithms – MRCM** Iterated Function System:
Many fractals can be described with iterated function systems

Excercises II

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

An initial image is transformed by a set of affine transformations (functions) producing a new image. The new image is then transformed by the same affine transformations producing another new image. Thus, each time the image is transformed, an iteration occurs. If the transformation is contractive–that is, the transformation brings points closer together–, then the image will begin to converge. After infinitely many iterations, assuming a contractive transformation, the image will converge to what is called an attractor.
http://pages.cs.wisc.edu/ ergreen/honors thesis/IFS.html
Pythagorean Tree (or Pythagoras Tree):
http://ecademy.agnesscott.edu/ lriddle/ifs/pythagorean/pythTree.htm

Bibliography I

[1] J. S. Kim, K.-I. Goh, B. Kahng, and D. Kim. A box-covering algorithm for fractal scaling in scale-free networks. *Chaos*, 17(2), 2007.

[2] www.stsci.edu/ lbradley/seminar/ifs.html, 2010.

[3] Jesper Lykke Jacobsen. High-precision percolation thresholds and Potts-model critical manifolds from graph polynomials. *Journal of Physics A Mathematical General*, 47(13):135001, April 2014.

Index I

, 14
box-covering method, 8
cluster-growing method, 8
fractal, 4
fractal dimension, 4
iterated function systems, 10
Koch curve, 5
percolation threshold, 14
self-similar, 4
snow flakes, 6
statistical self-similarity, 4
topological dimension, 4