Simulations in Statistical Physics

Course for MSc physics students

Janos Török

Department of Theoretical Physics

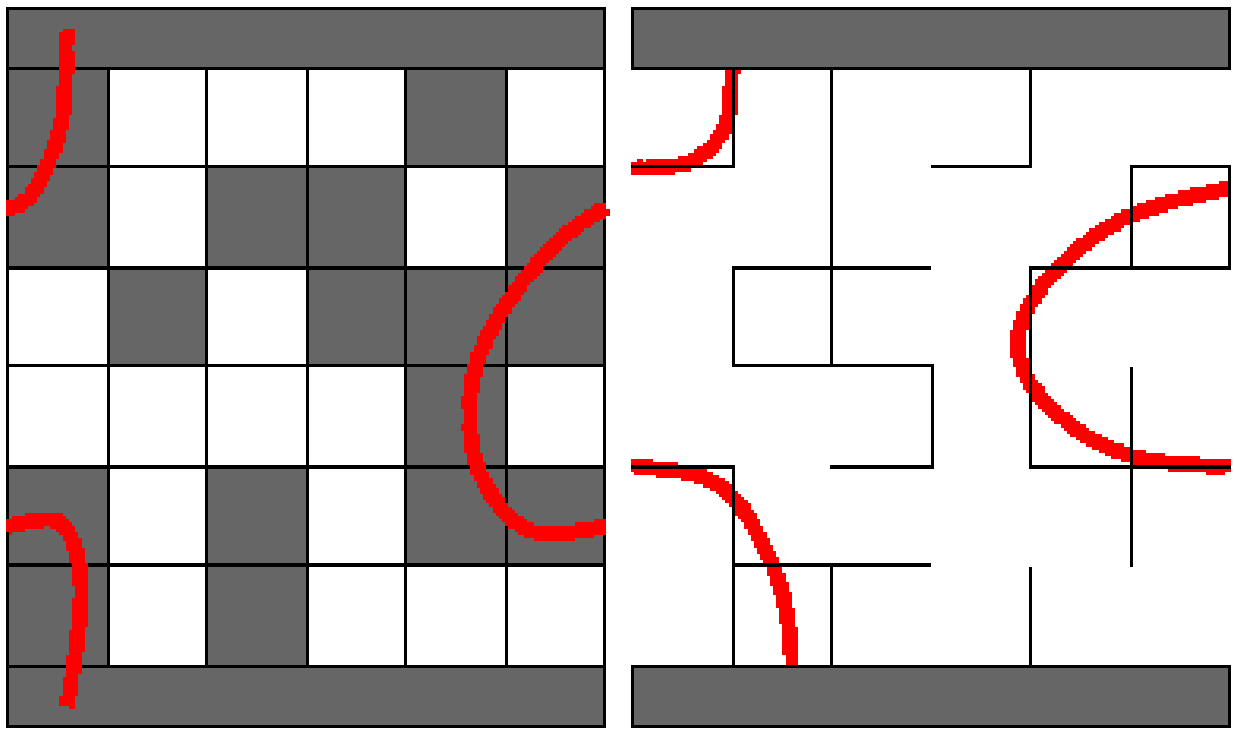
September 22, 2015

Percolation

Percolation

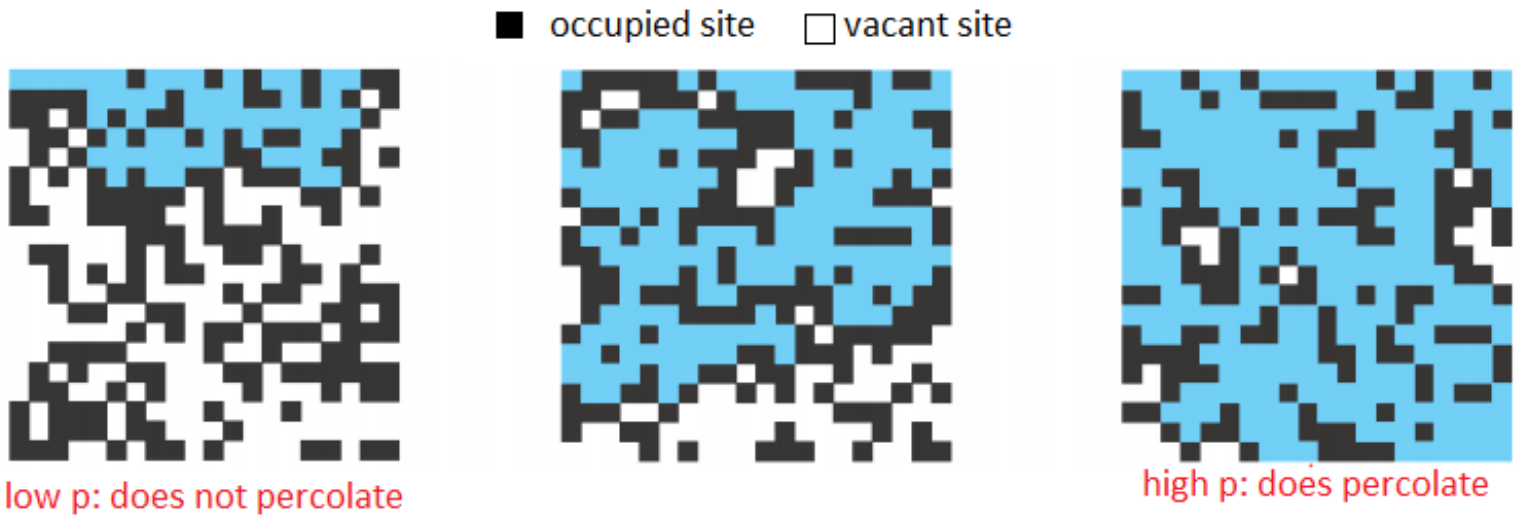
Behavior of connected cluster

- Site percolation
- Bond percolation



Percolation model

- Random environment
- With probability p site vacant (conducts)
- Two states: percolates or not!



Percolation theory

Questions (in infinite systems):

1. Is there an infinite cluster in infinite systems?
2. How many infinite clusters are there?
3. Mean cluster size (without the inifinte one)?
4. Cluster size distribution

Answers:

1. Above a critical density with probability 1 below it with probability 0
2. Only 1!
3. Decreases as a power low away from the critical density
4. Power law

Percolation theory

Questions (in infinite systems):

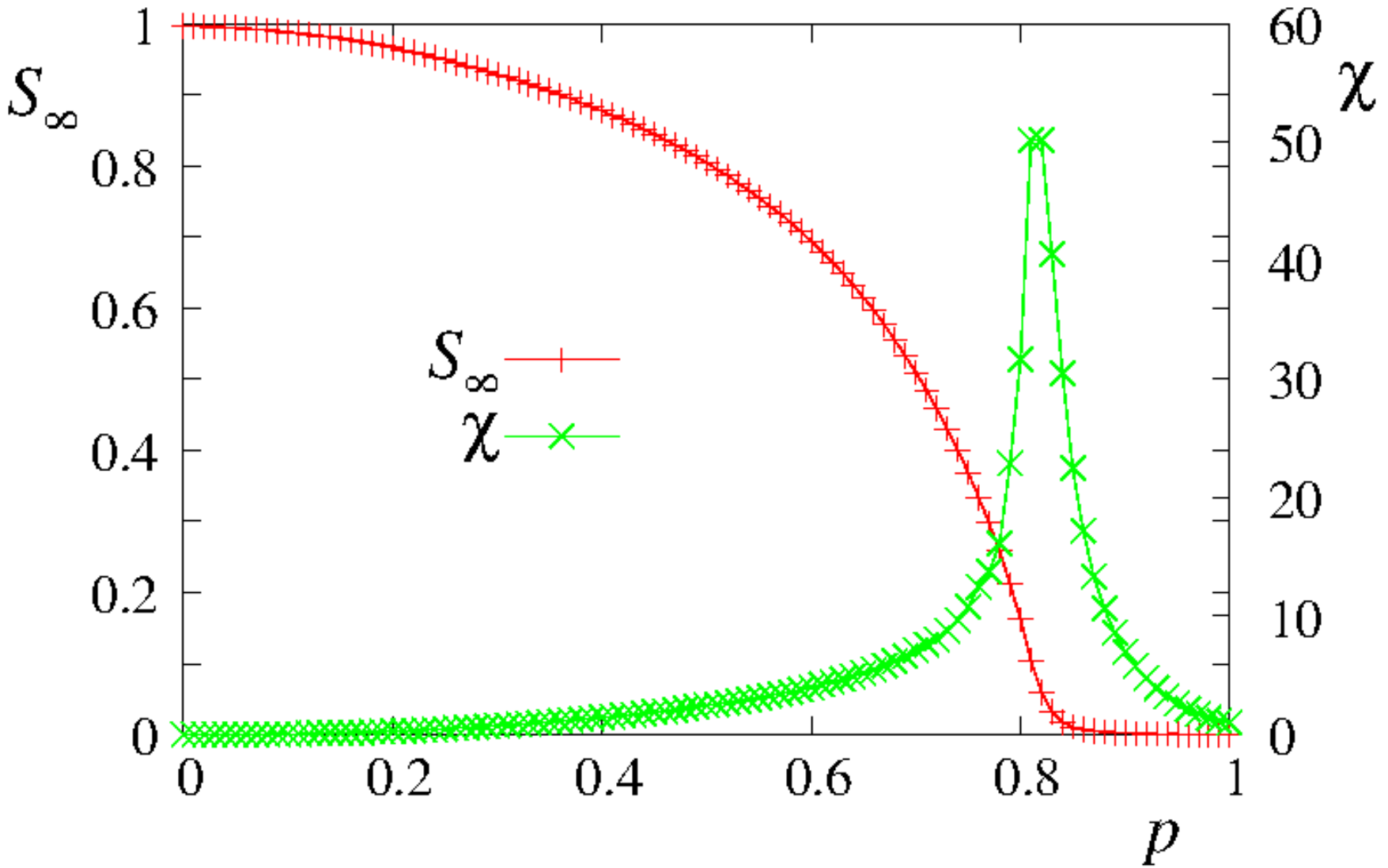
1. Is there an infinite cluster in infinite systems?
2. How many infinite clusters are there?
3. Cluster size distribution (n_s)
4. Mean cluster size (without the inifinte one)? ($S = \sum_s s^2 n_s$)

Answers:

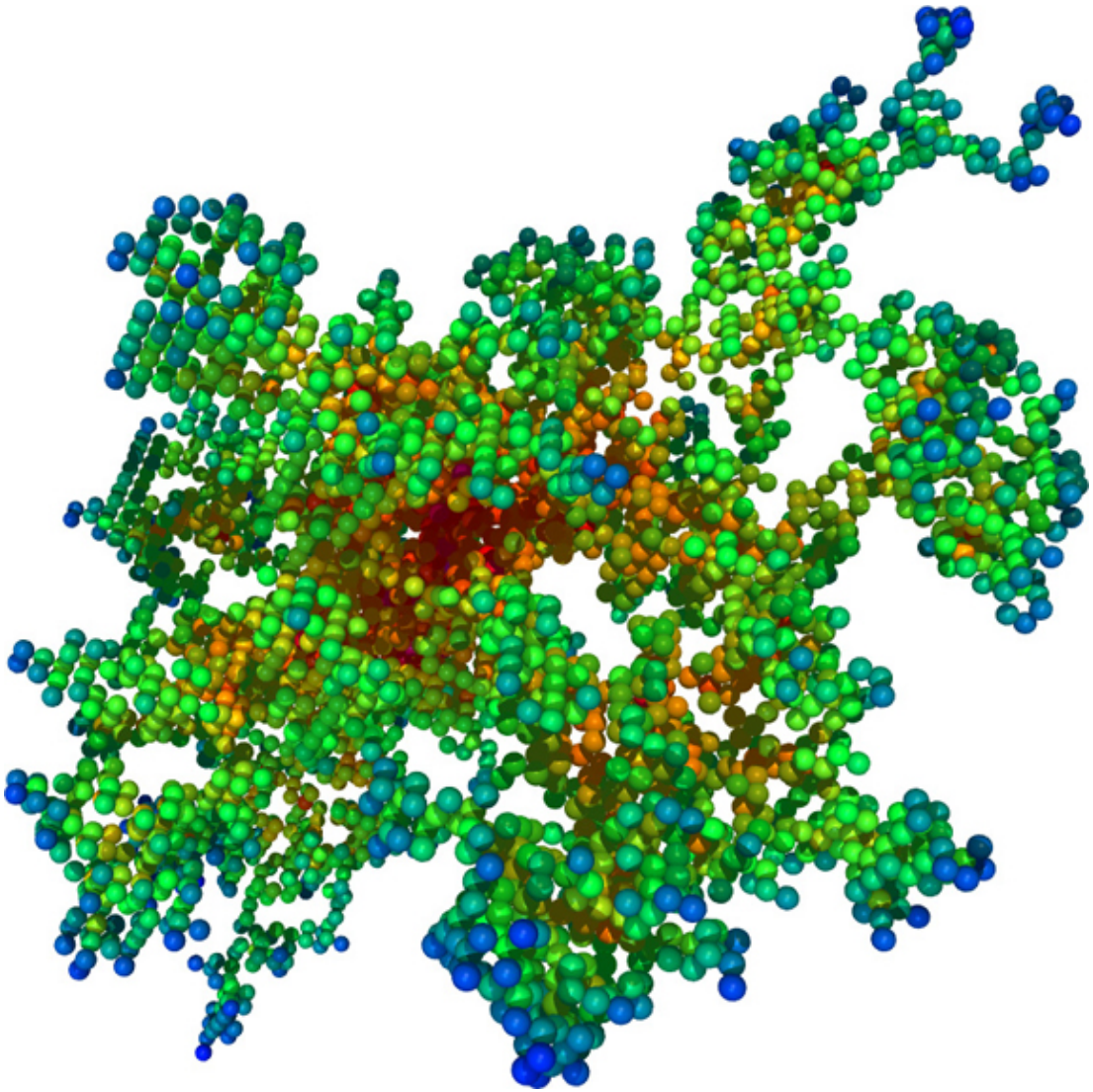
1. if $p > p_c$ then yes, otherwise no
2. Only 1!
3. $n_s \sim s^{-\tau}$
4. $S \sim |p - p_c|^{-\gamma}$

Like a second order phase transition in a geometric system!

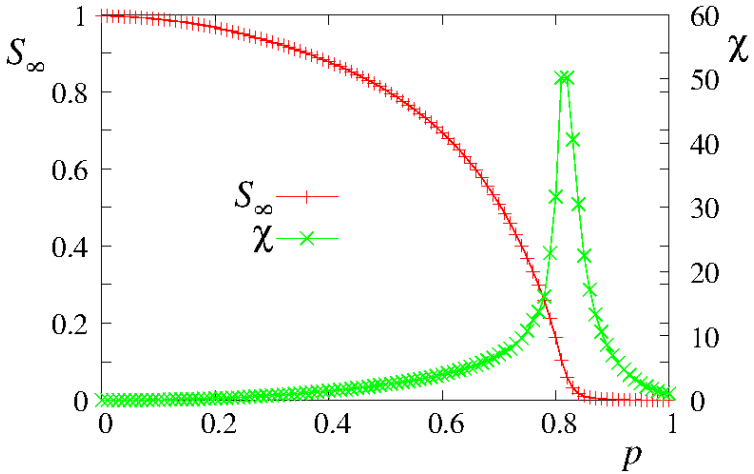
Percolation model

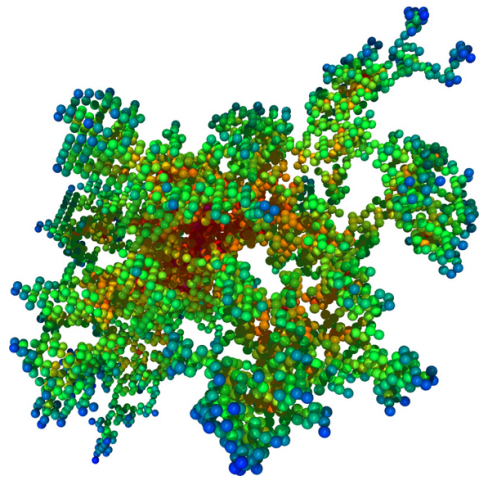


Percolation model



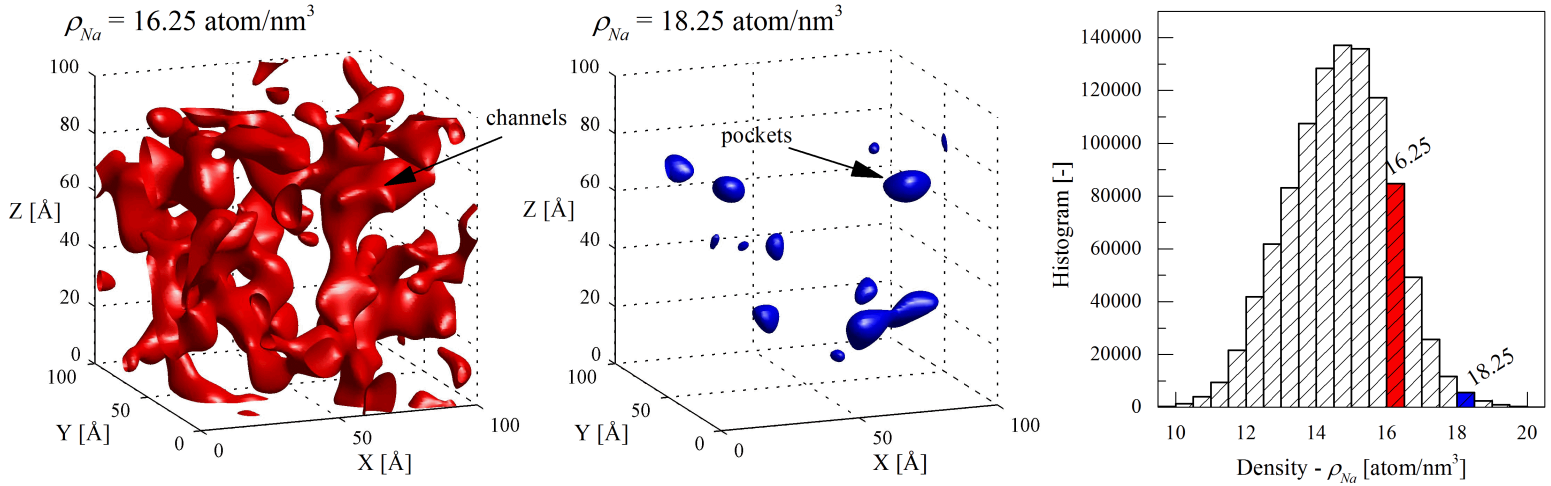
Percolating cluster

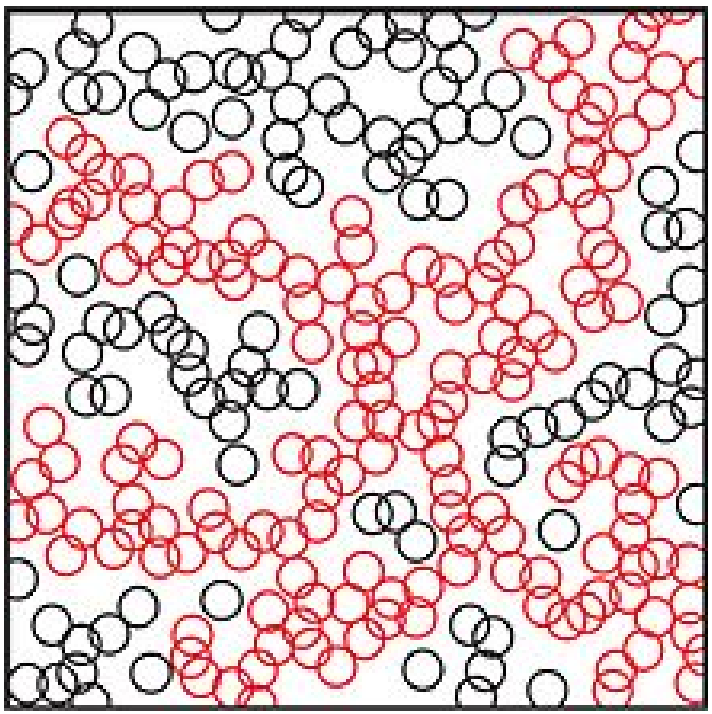


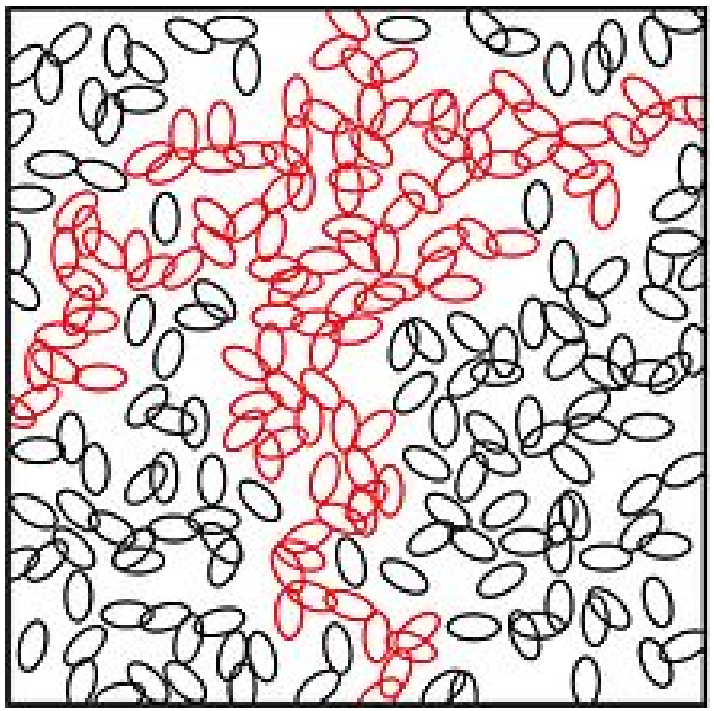


- Largest cluster
  - fractal with fractal dimension of d_f
  - $$S_\infty \sim \begin{cases} \xi_f^d \log(N/\xi^d) & p < p_c \\ N^{d_f/d} & p = p_c \\ NP_\infty(p) & p > p_c \end{cases}$$
- Largest not infinite cluster: size $\sim |p - p_c|^{-\nu}$

Percolation theory: Importance







Percolation theory: Importance

- COFFEE!!!!
- Non-equilibrium statistical physics
- Image analysis
- Percolation on networks: Phase transitions
- Percolation on networks: robustness, fragility
- Flodings

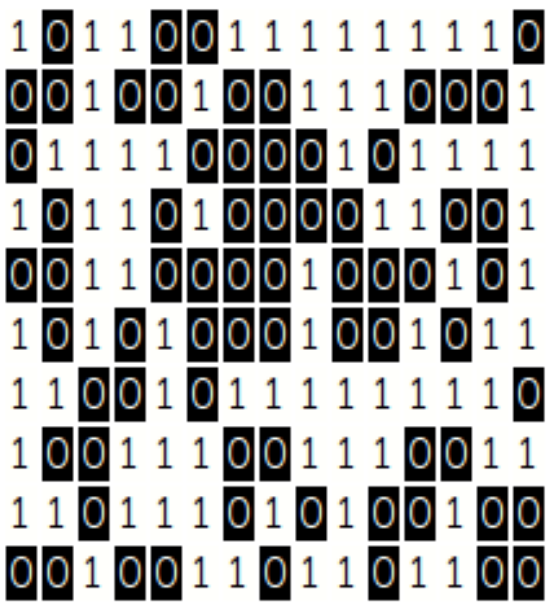
Percolation model

Bond [site] percolation

- Let us have a lattice (network)
- Each bond [site] is occupied with probability p
- (unoccupied with probability $1 - p$)
- A cluster is a set of sites connected by occupied bonds
  [A cluster is a set of occupied sites]

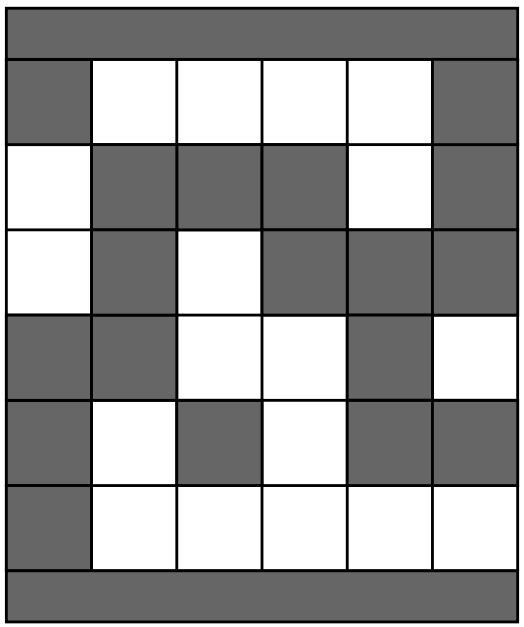
Hoshen-Kopelman Algorithm

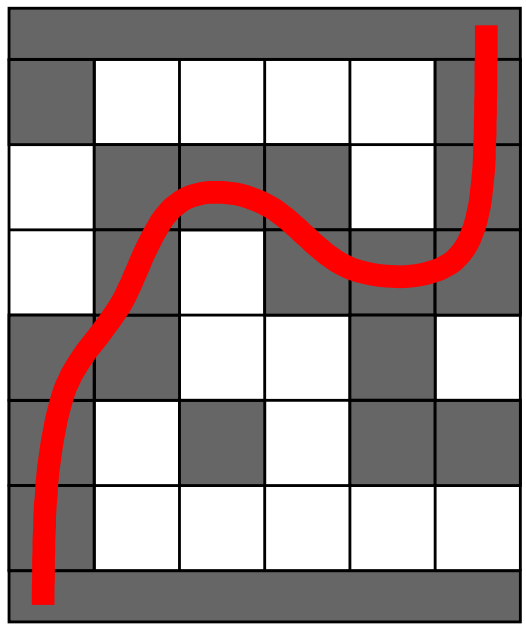
- Numerical task: find clusters
- Identify clusters
- Visit all sites
- Mark them with numbers



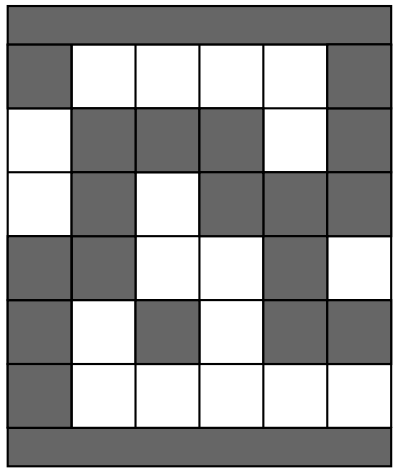
Hoshen-Kopelman Algorithm

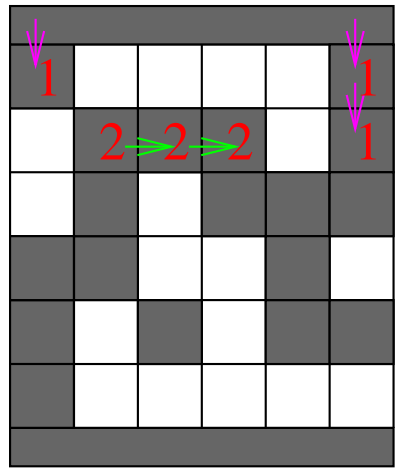
- Site percolation
- Open boundary conditions
- Go through site in typewriter style
- Check left and above



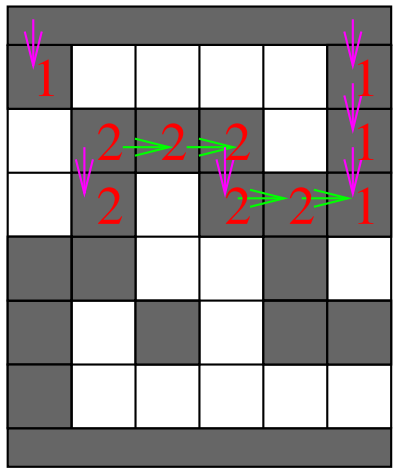


Hoshen-Kopelman Algorithm

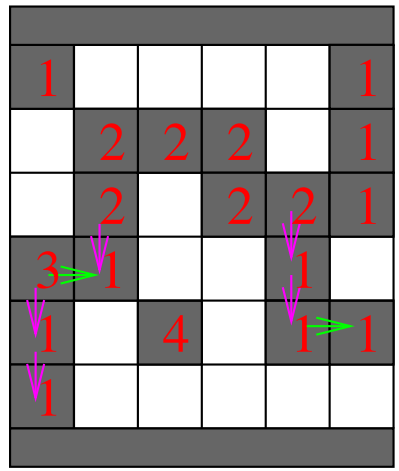




link[1]=1
link[2]=2



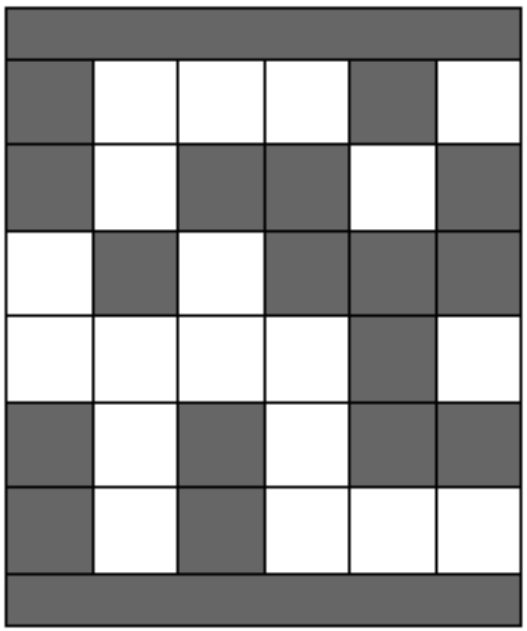
link[1]=1
link[2]=1

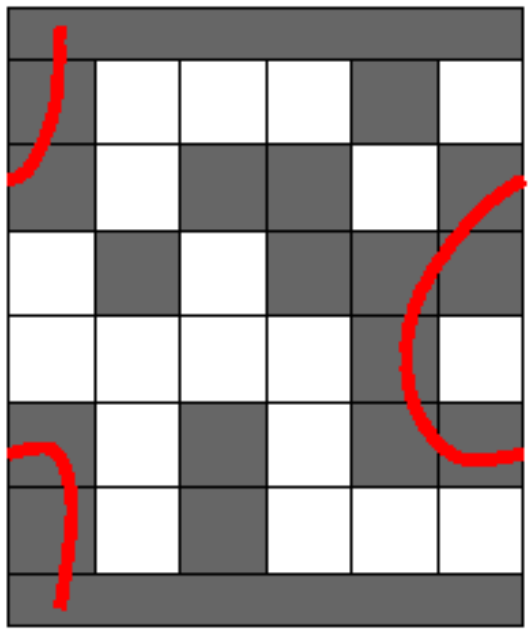


link[1]=1
link[2]=1
link[3]=1
link[4]=4

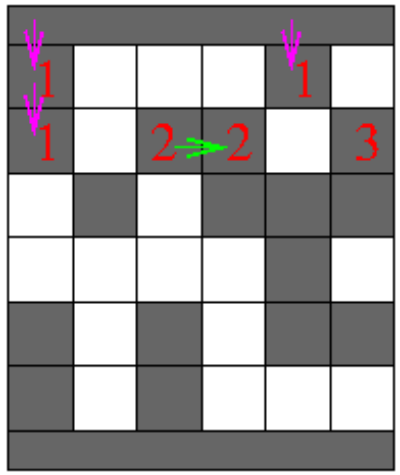
Hoshen-Kopelman Algorithm

- Site percolation
- Helical boundary conditions
- Go through site in typewriter style
- Check left and above

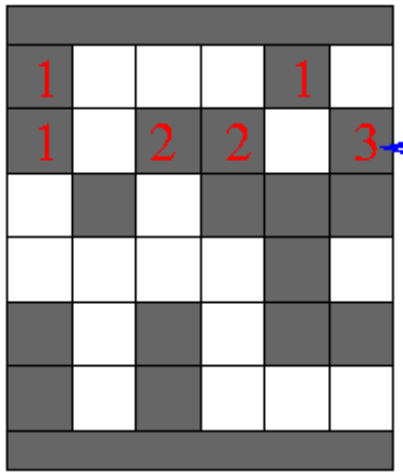




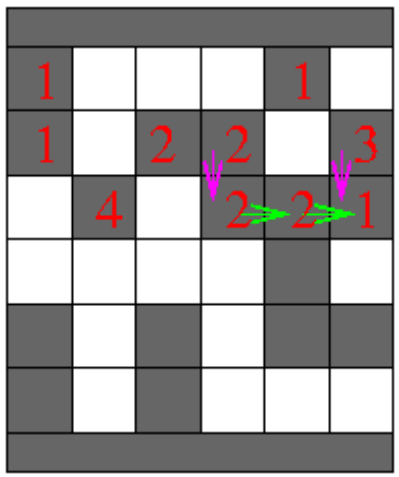
Hoshen-Kopelman Algorithm, Helical BC



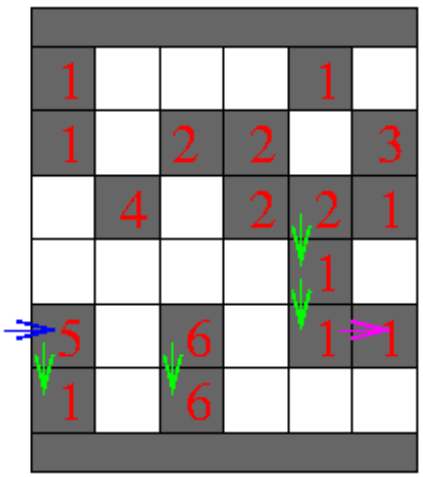
link[1]=1
link[2]=2
link[3]=3



link[1]=1
link[2]=2
link[3]=1



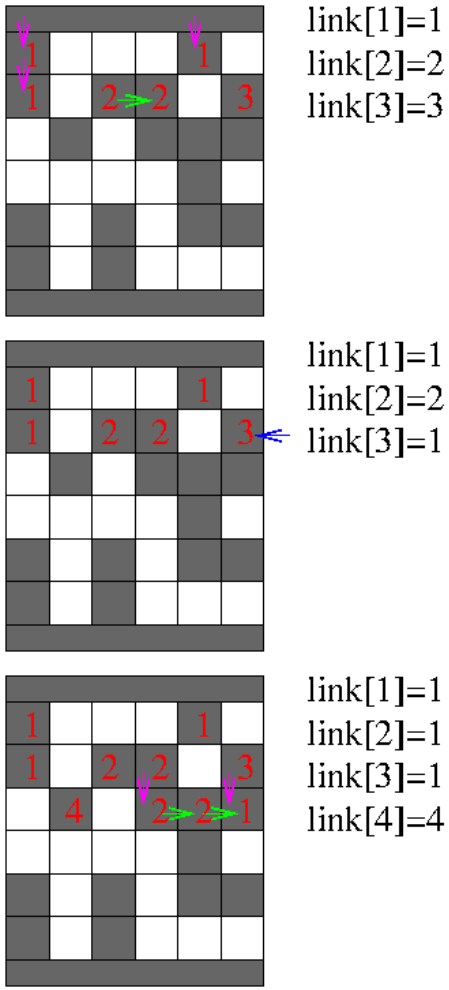
link[1]=1
link[2]=1
link[3]=1
link[4]=4



link[1]=1
link[2]=1
link[3]=1
link[4]=4
link[5]=1
link[6]=6

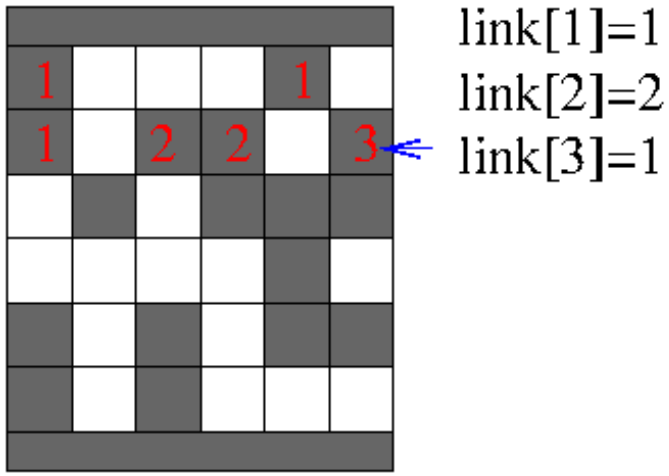
Hoshen-Kopelman Algorithm

```
largest_label = 0;
for (y = 0;y < n_rows;y++) {
  for (x = 0;x < n_columns;x++) {
    if (occupied[x][y]) {
      left = occupied[x-1][y];
      above = occupied[x][y-1];
      if (left == 0) && (above == 0) {
        largest_label ++;
        label[x][y] = largest_label;

      } else if (left != 0) && (above == 0) {
        label[x,y] = find(left);

      } else if (left == 0) && (above != 0) {
        label[x,y] = find(above);

      } else {
        label[x,y] = union(left,above);
      }
    }
  }
  /* Helical boundary conditions */
  if (occupied[n_columns-1][y]) && (occupied[0][y]) {
    union(occupied[n_columns-1][y],occupied[0][y])
  }
}
```



Hoshen-Kopelman Algorithm

```
largest_label = 0;
for x in 0 to n_columns {
 for y in 0 to n_rows {
   if occupied[x,y] then
     left = occupied[x-1,y];
     above = occupied[x,y-1];
     if (left == 0) and (above == 0) then
       largest_label = largest_label + 1;
       label[x,y] = largest_label;
     else {
       if (left != 0) {
         if (right != 0)
           UNION(left,above);
         label[x,y] = FIND(above);
       } else
       label[x,y] = FIND(right);
     }
   }
 }
```




```
int link[N];

int find(int x) {
 while (link[x] != x)
   x = link[x];
 return x;
}
```

```
int union(int x, int y) {
  int fx = find(x);
  int fy = find(y);
  if (fx < fy) {
    link[fy] = fx;
    return (fx);
  } else {
    link[fx] = fy;
    return (fy);
  }
}
```

Hoshen-Kopelman Algorithm

- Go through lattice as typewriter
- Check neighbors
- Resolve conflicts by linking clusters together
- Original trick: use link[] array for cluster size measure
  - link[] positive: number of sites in the cluster
  - link[] negative: cluster is linked to on other cluster
  - Not necessary faster than a seperate arrey for size

Percolation on networks (graphs)

- Network is defined by nodes and links
- Two arrays:
  - node[] list of nodes
  - link[i][] list of links of node i
  - link[i][j] is a link between i and j
- Cluster: nodes connected with links
- Links can be directed link[i][j] is a link from i $\rightarrow$ j

Stack (Verem – Hole/Pitfall)

- Last in forst out (LIFO)
- Code:

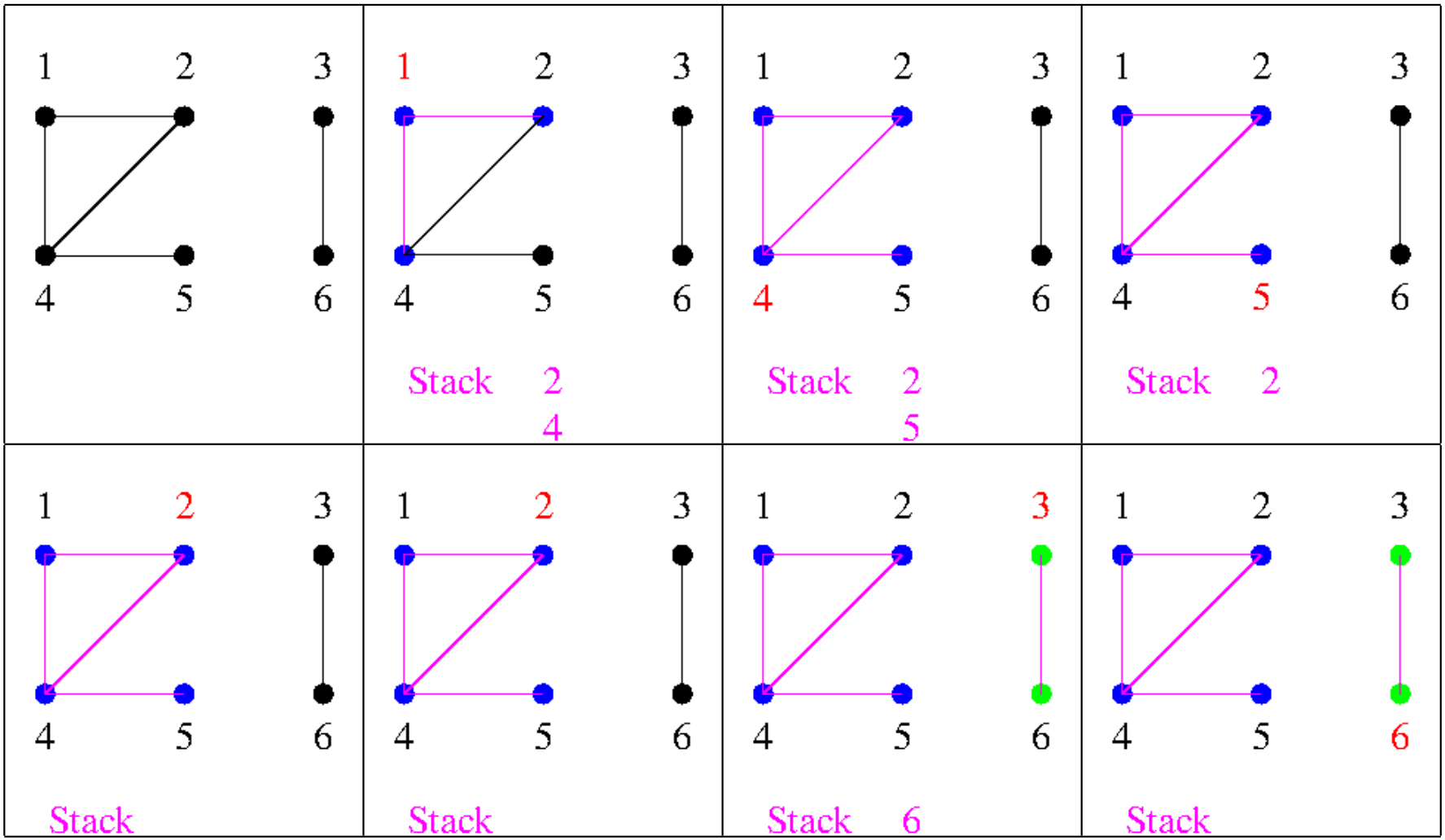
```
int Stack_size = Hopefully_large_enough_number;
int stack[Stack_size];
int sp=0;

void push(int item) {
  stack[sp++] = item;
  if (sp == Stack_size) enlarge_array(stack);
}

int pop() {
  return(stack[--sp]);
}
```

- Error handling?
- Size of the stack?

Algorithm percolation on networks (graphs)

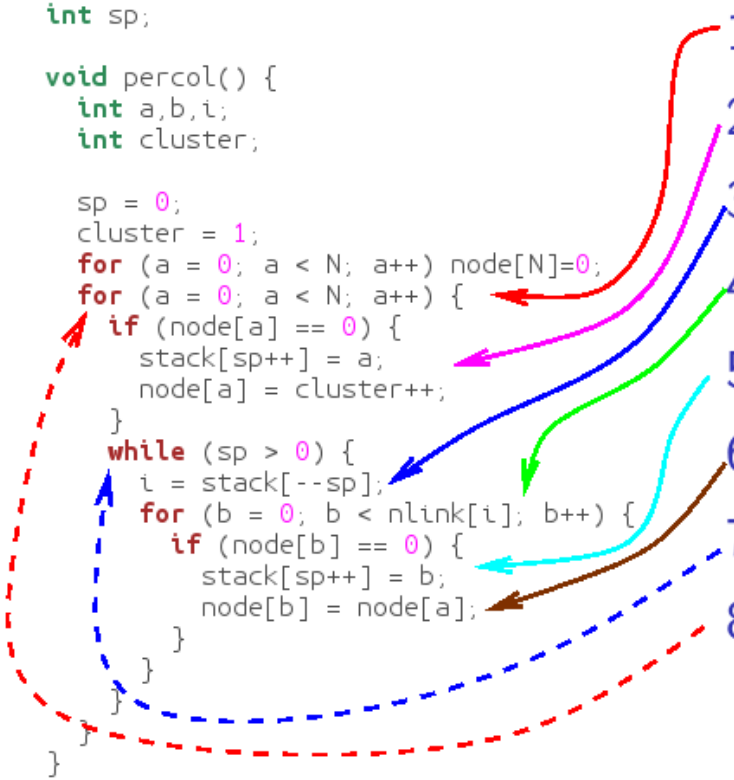
1. Go through each node
2. Put node in the stack
3. Get a node from the stack
4. Go through each unmarked link of the node
5. Put other end of links in the stack if it is not marked
6. Mark nodes
7. If the stack not empty Go to 3.
8. If the stack empty Go to 1.

Algorithm percolation on networks (graphs)



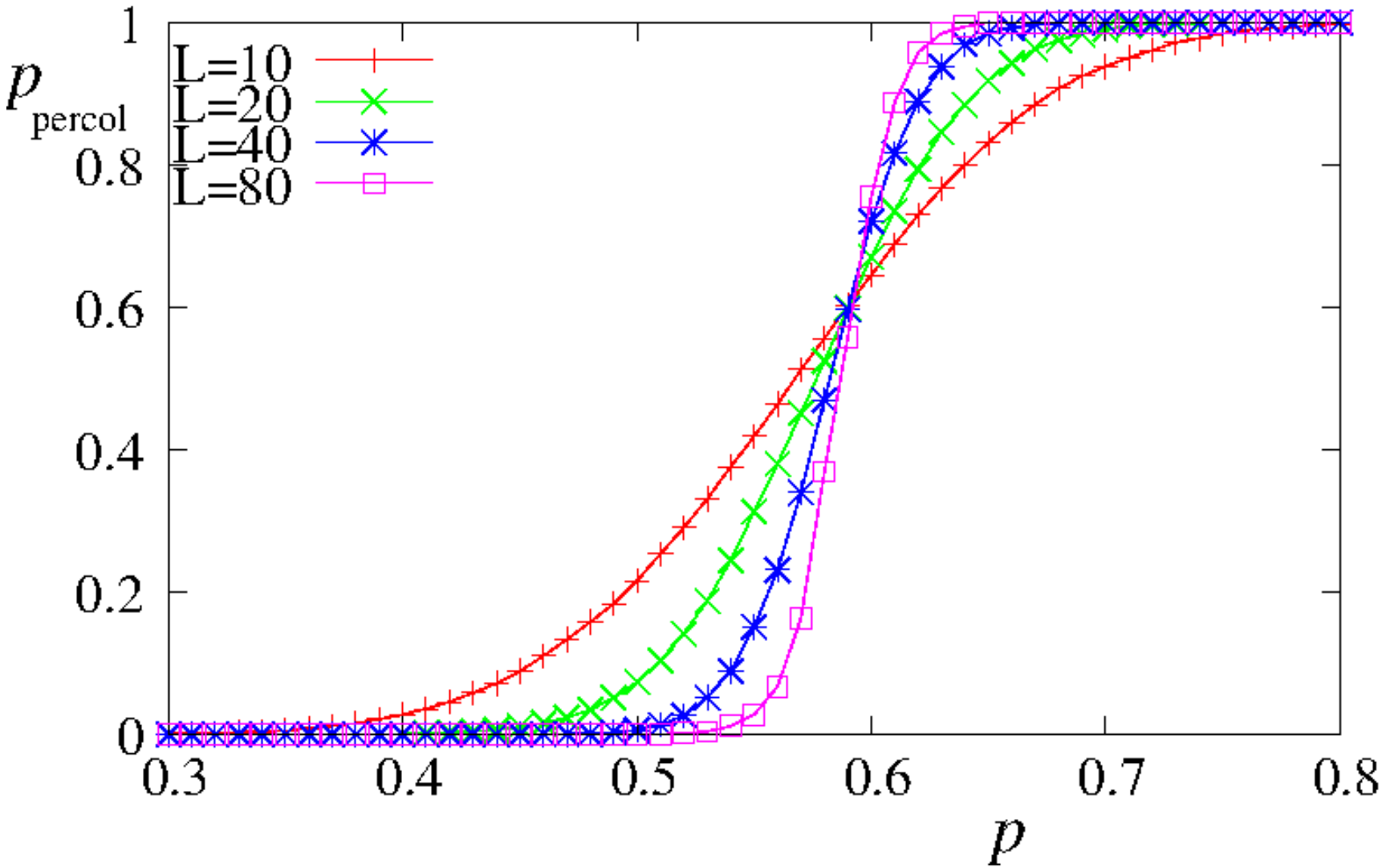
Algorithm percolation on networks (graphs)

```
int node[N];
int nlink[N];
int link[N][N];
int stack[N]
int sp;

void percol() {
  int a,b,i;
  int cluster;

  sp = 0;
  cluster = 1;
  for (a = 0; a < N; a++) node[N]=0;
  for (a = 0; a < N; a++) {
   if (node[a] == 0) {
      stack[sp++] = a;
      node[a] = cluster++;
    }
    while (sp > 0) {
      i = stack[--sp];
      for (b = 0; b < nlink[i]; b++) {
        if (node[b] == 0) {
          stack[sp++] = b;
          node[b] = node[a];
        }
      }
    }
  }
}
```



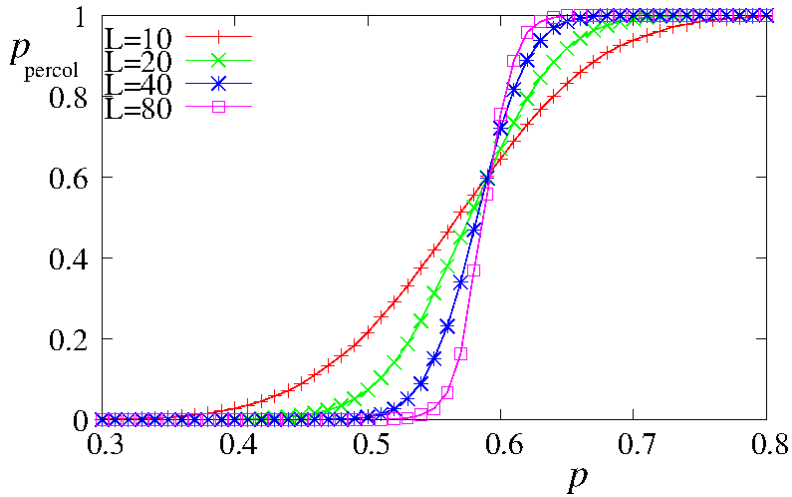
1. Go through each node
2. Put node in the stack
3. Get a node from the stack
4. Go through each unmarked link of the node
5. Put other end of links in the stack if it is not
6. Mark nodes
7. if the stack not empty Go to 3.
8. if the stack empty Go to 1.

Result

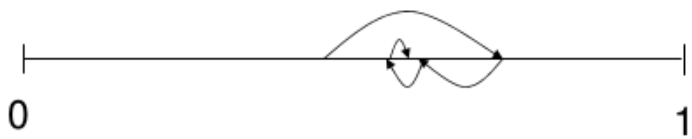


Determine p_c

- From order parameter:

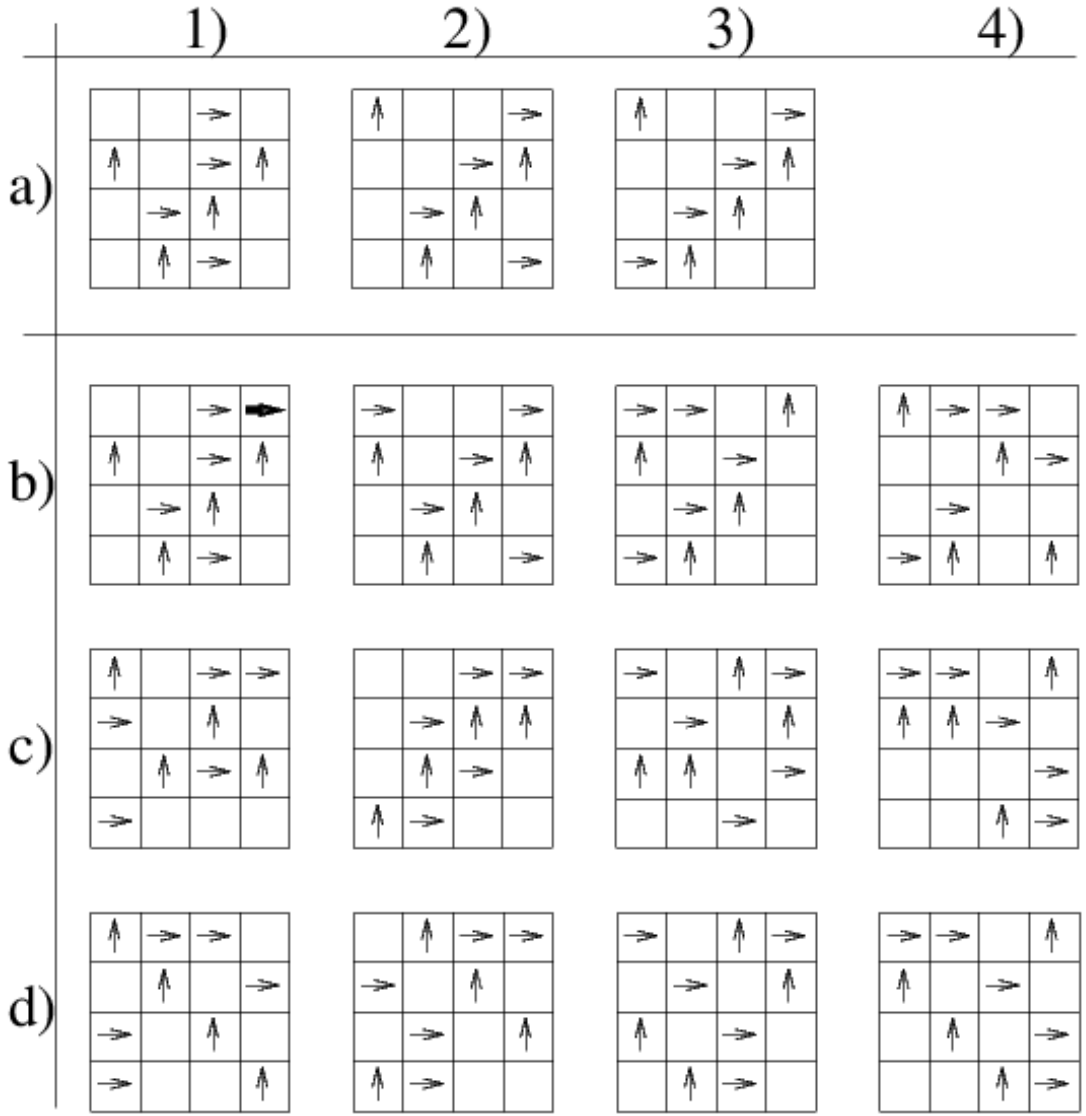


- Increase and decrease p by $p/2$ to converge to p_c
- Use the monotonity of the percolation
- Same random number sequence can be generated!



Monotonity

Not always true!



9. ábra: Az a/1 helyen található konfigurációból kiindulva blokkolt határciklushoz jutunk (a/3). A b/1 helyen az a/1 konfigurációhoz hozzávettük még a vastagon kihúzott nyilat, így a b/1-ben a sűrűség nagyobb lett, mint az a/1-ben. Innét indítva a modellt Szabadon mozgó fázishoz jut (d/4).