Block 8 (Montag 26.2.2024)

VIDEO: `video07a_perkolation_r` | VIDEO: `video07b_cluster_r`

9 Percolation

Model for conducting material: partition a block of material into small cubes, where each cube is "non-conducting" with probability $1-p$. Remaining fraction p is "conducting". In general: sites of interest = "occupied", here the conducting ones.

Question: Value of p such that current can run from one side to the other? ("percolating") $\rightarrow$ Phase transition at critical concentration $p = p_c$ above which current runs. Percolation: important (toy) model for phase transitions. Many phase transitions can be explained by hidden percolation transitions.

Literature: Stauffer and Arahony, Percolation theory (1994) [4].

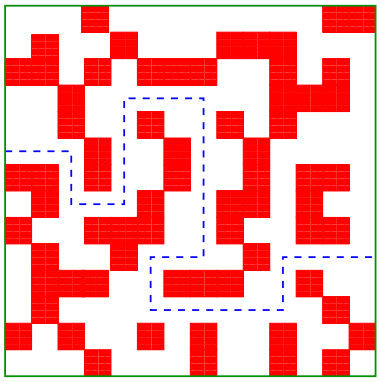


Figure 15: Percolation in two dimensions: there is a path of conducting sites (light), such that current runs through full system (broken line). The system "percolates".

Cluster := connected region of occupied sites $\Rightarrow$ Percolation = largest cluster spans full lattice.

Order parameter: size S of largest cluster divided by total number N of sites.

$p > p_c$: $S/N \to \text{const}(p) > 0$ for $N \to \infty$
$p < p_c$: $S/N \to 0$ for $N \to \infty$

The value of p_c depends on the dimension of the system and on the lattice structure.

[Activator]

How large is p_c in one dimension (system =line)?

For larger dimensions usually no analytical statement → computer simulations (square lattice: $p_c \approx 0.592746$, cubic: $p_c \approx 0.3116$, ... [4]). Corresponding algorithms (to find clusters): applied in MANY areas of computational physics.

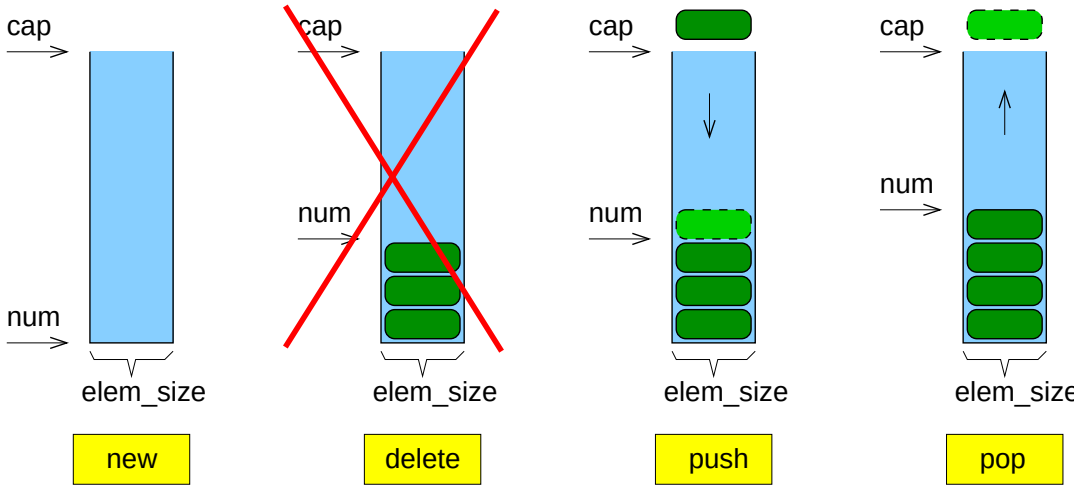
9.1 Stacks

For implementation needed: special Data type: *stacks*.

Stack = one can put elements on top and remove from the top.
→ LIFO (last in first out).
Remark: waiting queue = FIFO principle.

Basic operations:

- `lstack_new`: create stack of elements of given size with maximum number of elements..
- `lstack_delete`: delete stack.
- `lstack_push`: Put one element on top of the stack.
- `lstack_pop`: Remove top element from stack.



9.2 Cluster Algorithm

Representation of a d-dim systems in computer:
e.g., d-dim array `site`, e.g.. 3-dim `site[x][y][z]` = 1 if site (x,y,z) occupied. Also higher dimensions of theoretical interest, thus
`site [x1][x2][x3][x4][x5][x6][x7]` → too complicated, inflexible (also for changing lattice structure).

[Activator]

Think for 3 minutes about how one can represent a system betters.

ATTENTION: Do NOT read on before you found something

Solution: Number sites from 1 to N and use 1-dimensional (!!) array `site`:
`site[t]=1` if site `t` occupied.
Realization of lattice structure: variable '`num_n`' (=number of neighbors) and array `next`.
Each site '`t`' has `num_n` neighbors, stored in `next[t*num_n]`... `next[(t+1)*num_n-1]`.
order (simple cubic): +x,-x,+y,-y,... directions.
Access conveniently by macros:

```
#define INDEX(t, r, nn) ((t)*(nn)+r)
#define NEXT(t, r) next[INDEX(t, r, num_n)]
```

Attention: whenever using the macros, the variables `next` and `num_n` have be available with exactly these names. This can be made sure easily if all variables are used locally. The macros make the coding a bit less flexible but

more convenient and better readable (runs also faster as if using function calls).

Initialize array `next[]` only once in the beginning, can be used everywhere. Here: periodic boundary conditions:

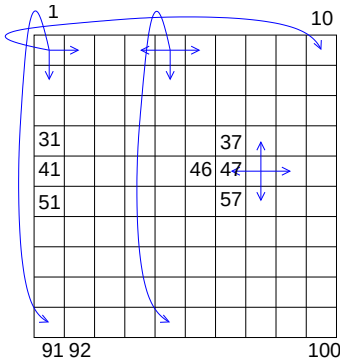


Figure 16: A 10x10 square lattice with periodic boundary conditions. The arrows point to the neighbors.

Example: $L \times L$ square lattice for $L = 10$. Site $i = 48$ has the neighbors $i + 1 = 49$ (+x), $i - 1 = 47$ (-x), $i + L = 58$ (+y) and $i - L = 38$. Site $i = 1$ has the neighbors 2 (+x), 10 (-x), 11 (+y), 91 (-y).
Main question of Percolation: is there a *percolating* cluster, i.e. running through the system:
First, determine *clusters* = connected regions (see below). Next:

a) Check whether there is cluster spanning through system

b) In the percolating region: average size of larger clusters grows linearly with size $N \to$: Calculate fraction of sites in largest cluster (for different system sizes N).

Here: first method b), easier to implement.

Idea for determination of clusters:

[Activator]

Think about a possible solution for three minutes, then discuss for 3 minutes with your neighbor.

ATTENTION: Do NOT read on before you head some idea.

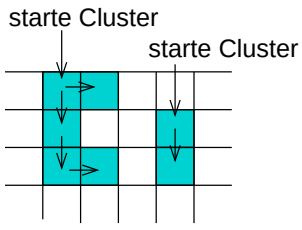
Starting point: a lattice with some sites occupied.
The occupied neighbors of an occupied site belong to the same cluster.
Their occupied neighbors also.
Implementation: The neighbors are store in a stack and the treated one after the other. Take care than every site is put on stack at most once.

algorithm Cluster
begin
  while there are "untreated" occupied sites t **do**
  begin
    push t on stack
    start new cluster with t
    while stack is not empty **do**
    begin
      pop one site c from stack
      for all neighbors n of c **do**
        if n occupied and not yet "treated" **then**
          push n on stack, add to cluster, mark as "treated"
    end
  end
end

Store in `cluster[]` array, cluster ID of each site (-1: not yet treated).

Since each site is treated only once: run time $O(N)$.

Example:



Subroutine for cluster determination:

```
/****************** percol_cluster() ******************/
/** Calculates the connected clusters of the lattice   **/
/** 'site'. Occupied sites have 'site[i]=1' (i=1..N).  **/
/** Neigbours of occupied sites form clusters.         **/
/** For each site, in 'cluster[i]' the id of the       **/
/** cluster (starting at 0) is stored                  **/
/** PARAMETERS: (*)= return-paramter                   **/
/**       num_n: number of neighbours                  **/
/**           N: number of sites                       **/
/**        next: gives neighbours (0..N)x(0..num_n-1)  **/
/**              0 not used here. Use NEXT() to access **/
/**        site: tells whether site is occupied        **/
/** (*) cluster: id of clusters sites are contained in **/
/** RETURNS: number of clusters                        **/
/******************************************************/
int percol_cluster(int num_n, int N, int *next,
                   short int *site, int *cluster)
{
  int num_clusters=0;
  int t, r;       /* loop counters over sites, directions */
  int current, neighbour;                        /* sites */
  lstack_t *members;      /* stack of members for cluster */
  for(t=1; t<=N; t++)    /* initialise all clusters empty */
    cluster[t] = -1;
  members = lstack_new(N, sizeof(int));
  for(t=1; t<=N; t++)              /* loop over all sites */
  {
    if((site[t] == 1)&&(cluster[t]==-1)) /* new cluster ? */
    {
      lstack_push(members, &t);          /* start cluster */
      cluster[t] = num_clusters;
      while(!lstack_is_empty(members))  /* extend cluster */
      {
        lstack_pop(members, &current);
        for(r=0; r<num_n; r++)   /* loop over neighbours */
        {
          neighbour = NEXT(current, r);
          if((site[neighbour]==1)&&(cluster[neighbour]==-1))
          {          /* neighbour belongs to same cluster */
            lstack_push(members, &neighbour);
            cluster[neighbour] = num_clusters;
          }
        }
      }
      num_clusters++;
    }
  }
  lstack_delete(members); return(num_clusters);
}
```

Example run for 2 dimensions, side length $L = 10$, $p = 0.5$). Output cluster IDs of site:

```
 0  0  0  0  0  0  0  0     0
 0                0     0  0  0
                  0        0
    1           0  0  0  0
    1     2           0  0     3
    1     2     4           5
       6     4  4  4  4
 0                            0  0
 0  0                  7        0  0
 0  0     0  0  0        0  0
```

9.3 Ergebnisse

Many program calls done by script `run_percol.scr`:

```
#!/bin/bash
L=$1
for p in 0.1 0.2 0.3 0.4 0.45 0.5 0.52 0.54 0.56 0.57 0.58 0.59 \
         0.60 0.61 0.62 0.64 0.66 0.68 0.7 0.8 0.9 1.0
do
  percolation1 $L $p 100
done
```

Possible you have to make the script executable by: `chmod u+x run_percol.scr`.

[Activator]

Compile the program by
`cc -o percolation1 percolation1.c percol.c stacks.c`
Run the script for different system sizes (e.g. 10, 20, 50, 100, 200) and redirect each time the results to a file, e.g., by `run_percol.scr 10 > perc10.dat`. Plot the data using `gnuplot`.