4. miniworkshop difrakce a ultraperiferálních srážek

A very brief introduction to percolation

Guillermo Contreras

Czech Technical University in Prague

Děčín

September 14, 2021

Definitions

Percolate: filter gradually through a porous surface or substance.

Oxford Dictionary

Definitions

Percolate: filter gradually through a porous surface or substance.

Oxford Dictionary

Percolation theory: describes the behaviour of a network when nodes or links are added.

Wikipedia

Definitions

Percolate: filter gradually through a porous surface or substance.

Oxford Dictionary

Percolation theory: describes the behaviour of a network when nodes or links are added.

Wikipedia

There is a geometric type of phase transition, since at a critical fraction of addition the network of small, disconnected clusters merge into significantly larger, connected, so-called spanning cluster.

Wikipedia

Definitions

Percolate: filter gradually through a porous surface or substance.

Oxford Dictionary

Percolation theory: describes the behaviour of a network when nodes or links are added.

Wikipedia

The numerical value of the fraction is determined by the local structure of the graph, whereas the behaviour near the critical threshold is characterised by universal critical exponents

Wikipedia

There is a geometric type of phase transition, since at a critical fraction of addition the network of small, disconnected clusters merge into significantly larger, connected, so-called spanning cluster.

Wikipedia

The birth of percolation (1957)

Mathematical Proceedings of the Cambridge Philosophical Society , Volume 53 , Issue 3 , July 1957 , pp. 629 - 641

PERCOLATION PROCESSES

I. CRYSTALS AND MAZES

By S. R. BROADBENT and J. M. HAMMERSLEY

Received 15 August 1956

ABSTRACT. The paper studies, in a general way, how the random properties of a 'medium' influence the percolation of a 'fluid' through it. The treatment differs from conventional diffusion theory, in which it is the random properties of the fluid that matter. Fluid and medium bear general interpretations: for example, solute diffusing through solvent, electrons migrating over an atomic lattice, molecules penetrating a porous solid, disease infecting a community, etc.

The birth of percolation (1957)

Mathematical Proceedings of the Cambridge Philosophical Society , Volume 53 , Issue 3 , July 1957 , pp. 629 - 641

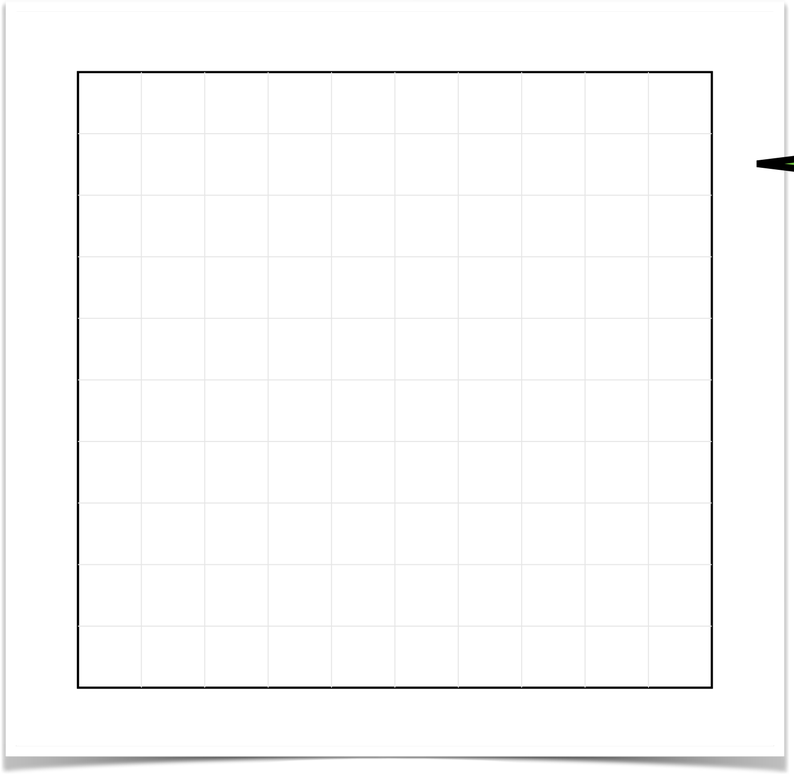
1. *Introduction*. There are many physical phenomena in which a *fluid* spreads randomly through a *medium*. Here fluid and medium bear general interpretations: we may be concerned with a solute diffusing through a solvent, electrons migrating over an atomic lattice, molecules penetrating a porous solid, or disease infecting a community. Besides the random mechanism, external forces may govern the process, as with water percolating through limestone under gravity. According to the nature of the problem, it may be natural to ascribe the random mechanism either to the fluid or to the medium. Most mathematical analyses are confined to the former alternative, for which we retain the usual name of *diffusion process*: in contrast, there is (as far as we know) little published work on the latter alternative, which we shall call a *percolation process*. The present paper is a preliminary exploration of percolation processes; and, although our conclusions are somewhat scanty, we hope we may encourage others to investigate this terrain, which has both pure mathematical fascinations and many practical applications.

Percolation in a lattice

Different types of lattices: dimensionality (2D, 3D, …) and shape (square, honeycomb, …)

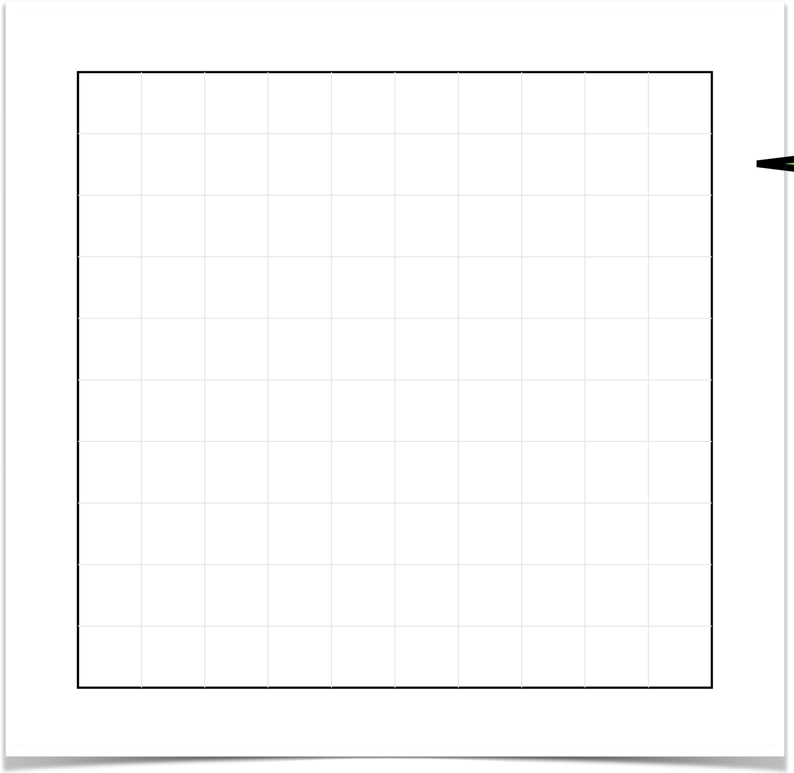
Type of percolation: bond or site

2D site percolation in a lattice



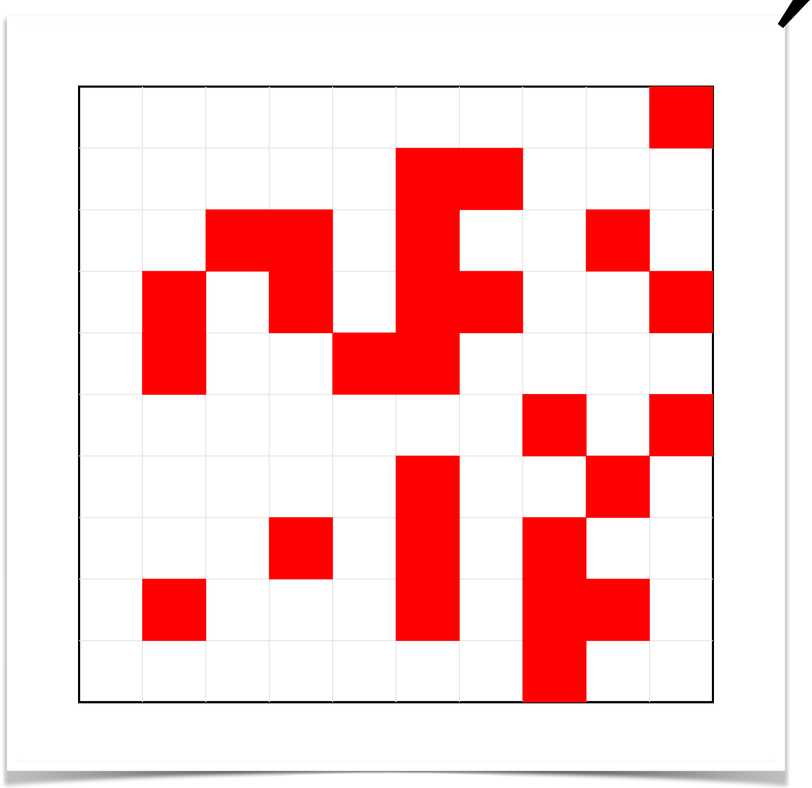
Start with an empty lattice

2D site percolation in a lattice

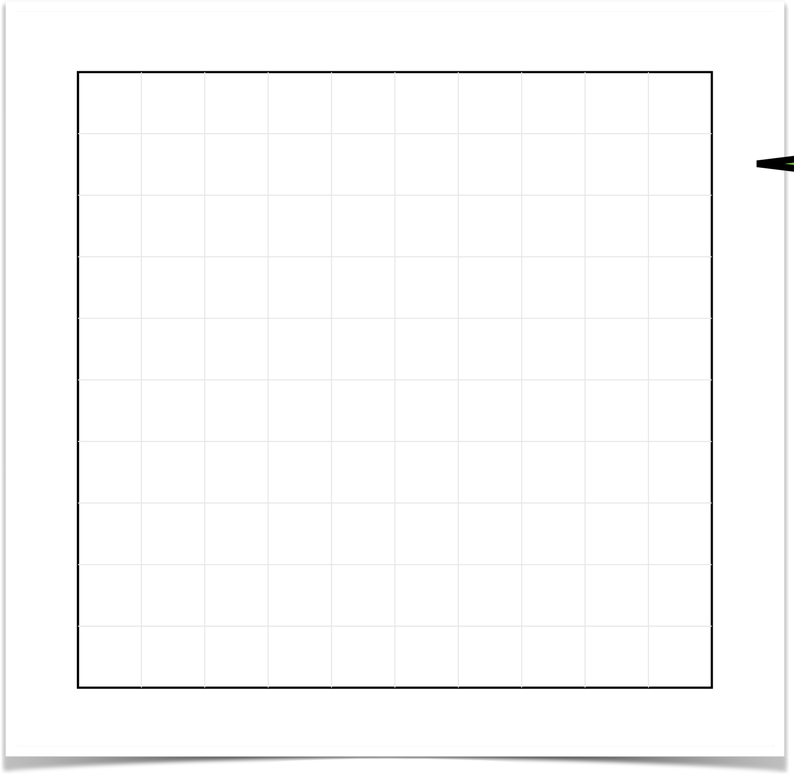


Start with an empty lattice

Fill randomly a fraction of the sites.

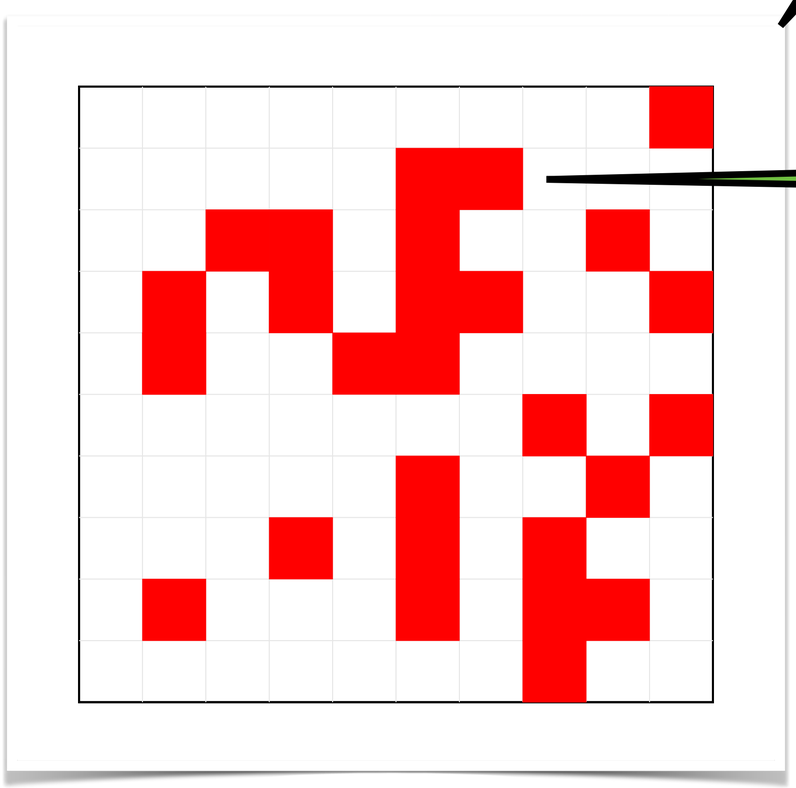


2D site percolation in a lattice



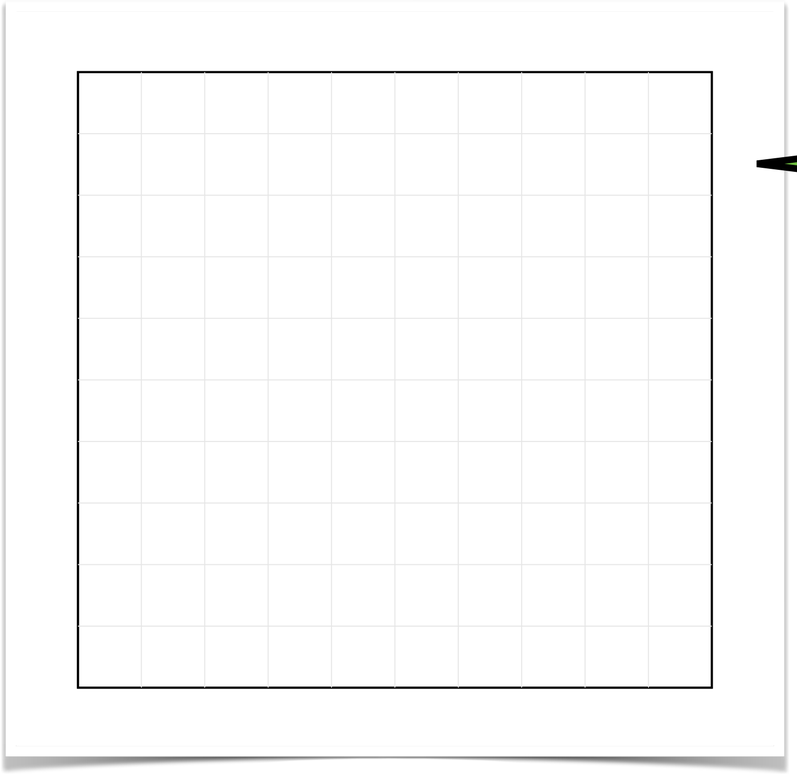
Start with an empty lattice

Fill randomly a fraction of the sites.



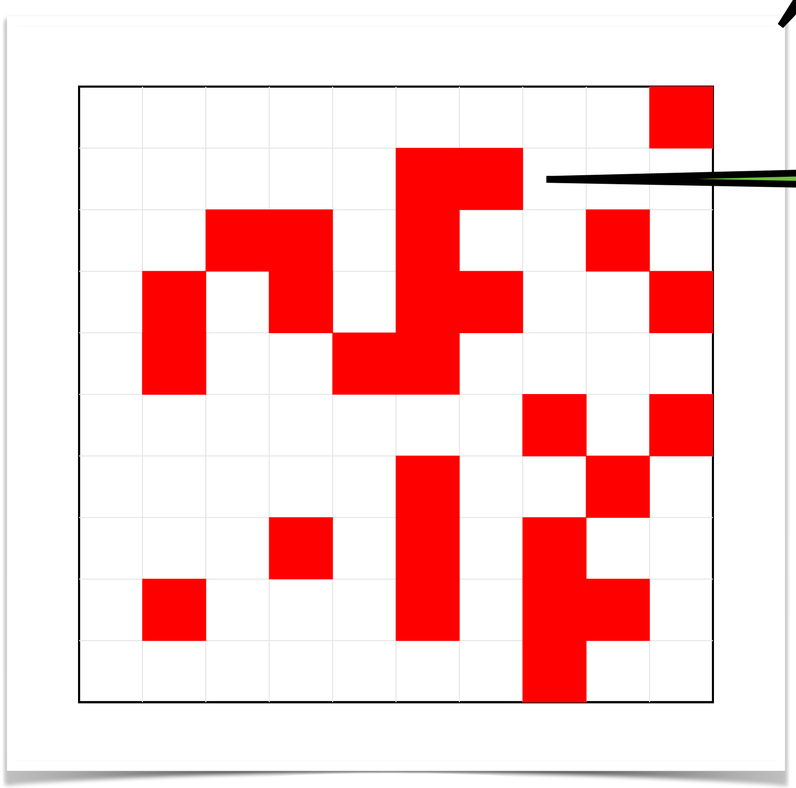
Define clusters by joining **occupied neighbouring** sites.

2D site percolation in a lattice



Start with an empty lattice

Fill randomly a fraction of the sites.



Define clusters by joining **occupied neighbouring** sites.

Percolation deals with the properties of the clusters

Parenthesis: code to produce the plots

I have added to the agenda a piece of code to draw the lattices shown in the previous (and next) pages.

```
DrawLattice(int length, float frac)
```

Parenthesis: code to produce the plots

I have added to the agenda a piece of code to draw the lattices shown in the previous (and next) pages.

Creates a lattice of size (length, length)

And fills a fraction frac of the sites

```
DrawLattice(int length, float frac)
```

Parenthesis: code to produce the plots

I have added to the agenda a piece of code to draw the lattices shown in the previous (and next) pages.

Creates a lattice of size (length, length)

And fills a fraction frac of the sites

```
DrawLattice(int length, float frac)
```

To use it, enter root, compile and run. E.g.:

Parenthesis: code to produce the plots

I have added to the agenda a piece of code to draw the lattices shown in the previous (and next) pages.

Creates a lattice of size (length, length)

And fills a fraction frac of the sites

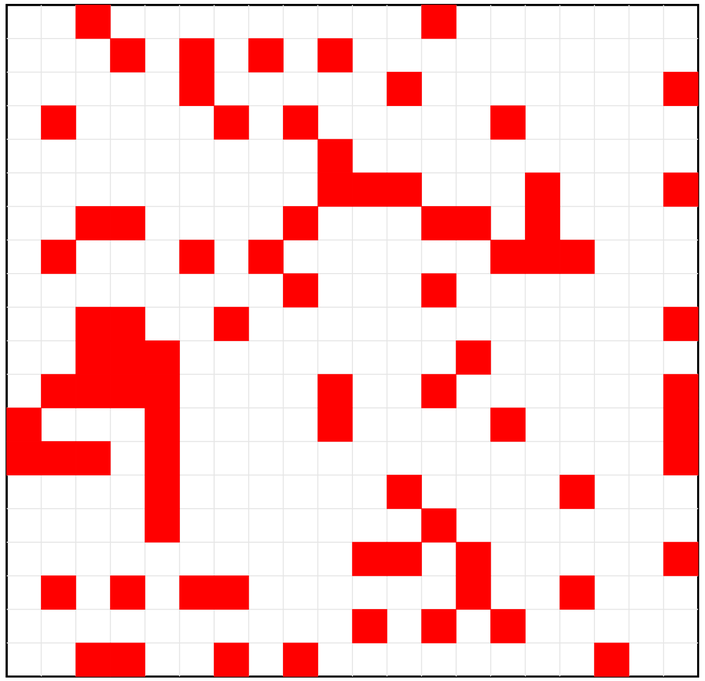
```
DrawLattice(int length, float frac)
```

To use it, enter root, compile and run. E.g.:

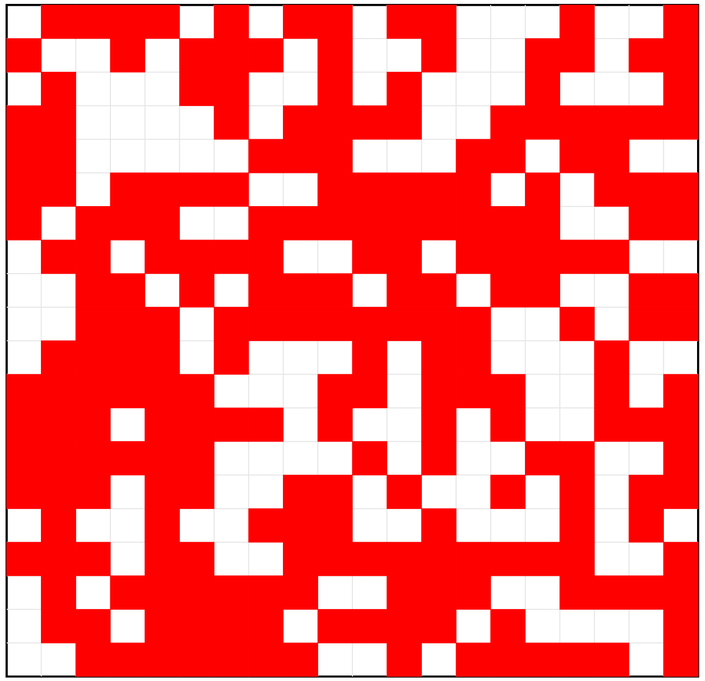
```
root [0] .L DrawLattice.C+g
root [1] DrawLattice(20,0.60)
root [2] DrawLattice(20,0.20)
```

Behaviour at different fractions

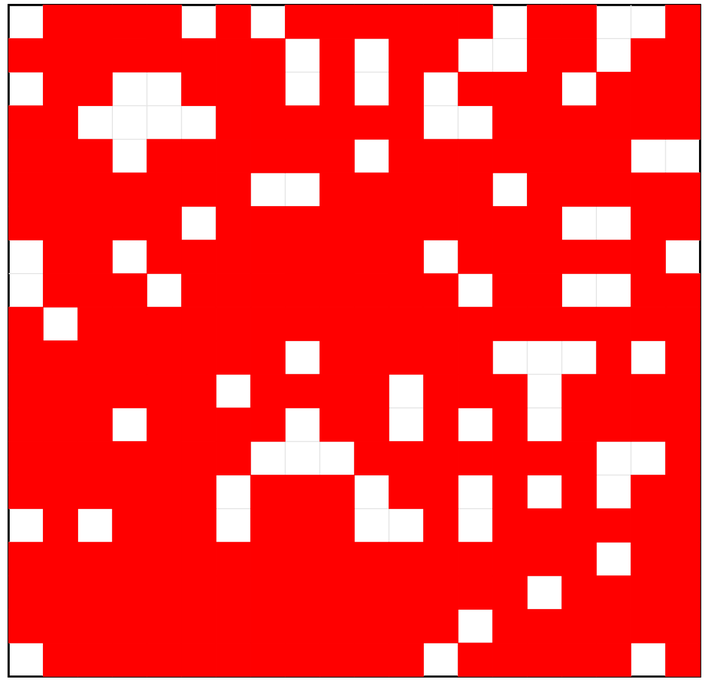
f = 0.2



f = 0.6

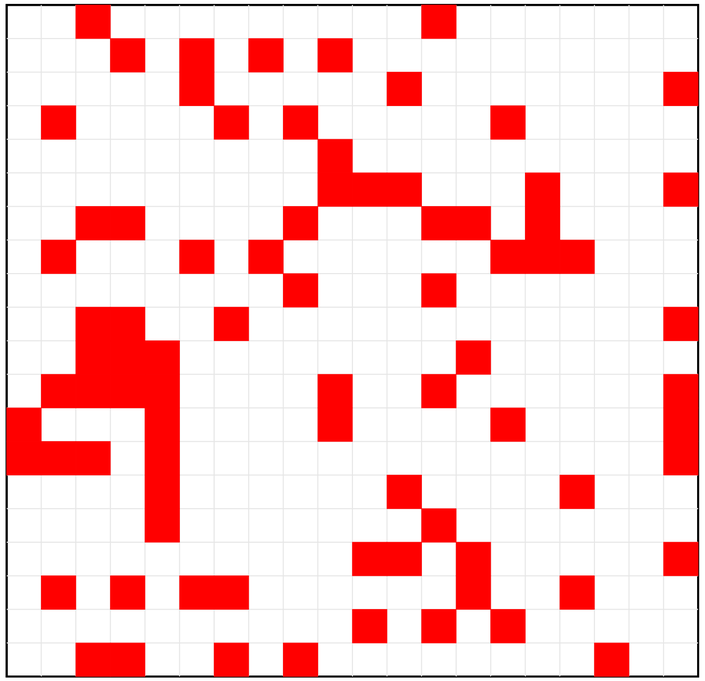


f = 0.8

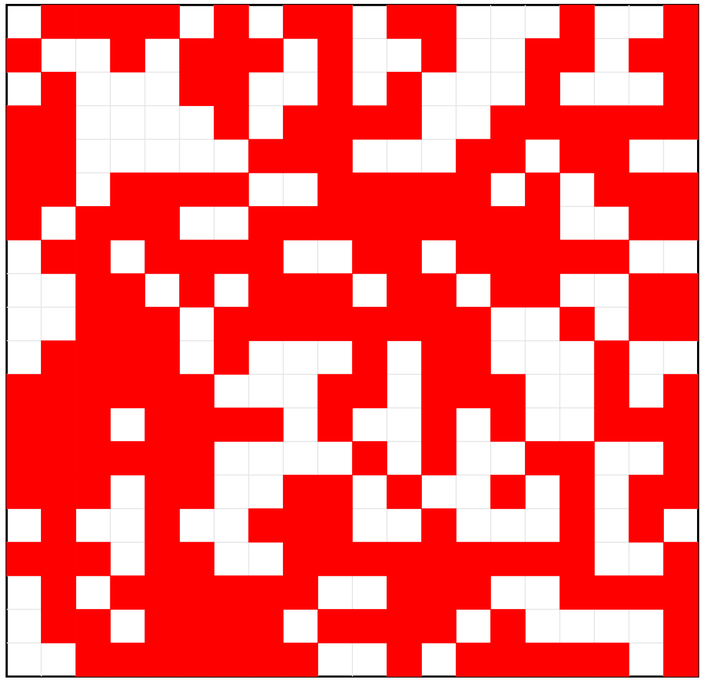


Behaviour at different fractions

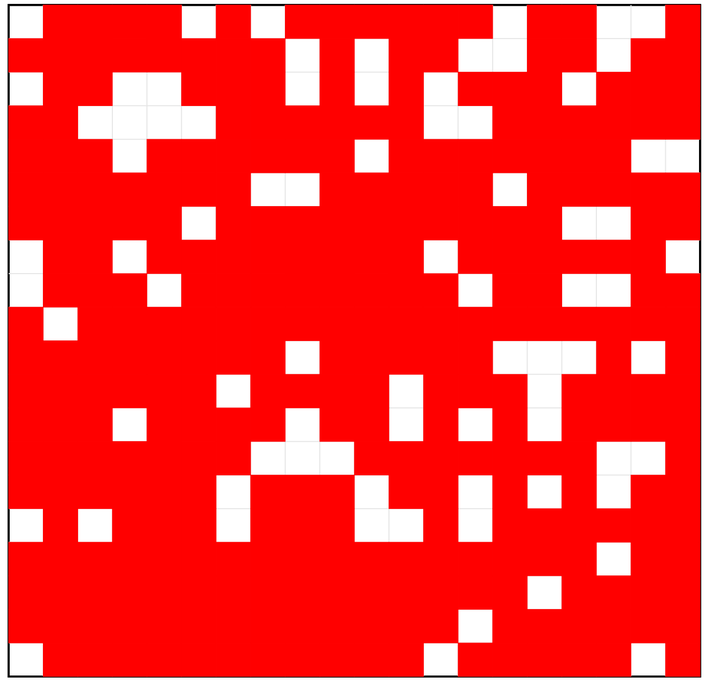
f = 0.2



f = 0.6

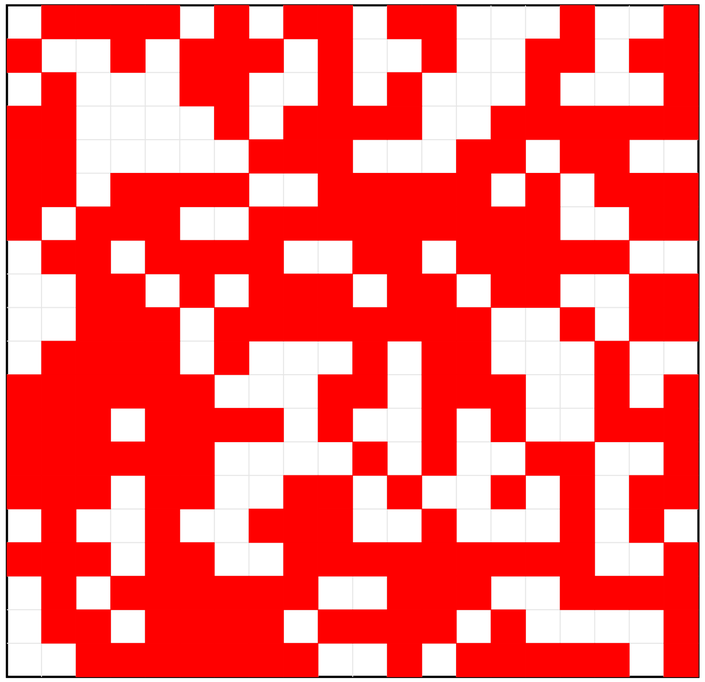


f = 0.8



Critical behaviour at ≈0.59: a cluster spans from one border to the other.

Behaviour at the critical concentration

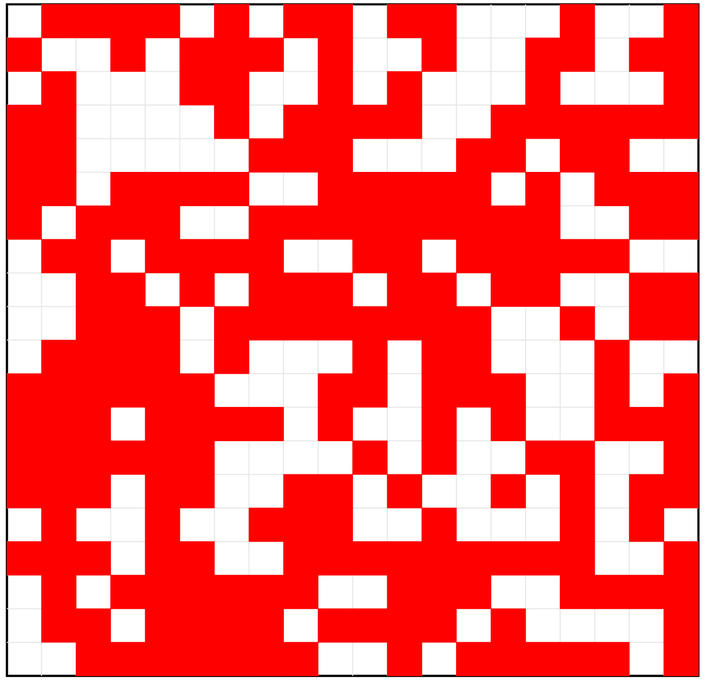


The spanning cluster

- can be ‘destroyed’ by removing few sites
- Is ‘infinite’ but contains a ‘vanishing’ fraction of the occupied sites
- It contains large holes: its mass varies as a power law with fractal exp.

Behaviour at the critical concentration

Is infinite in the thermodynamic limit

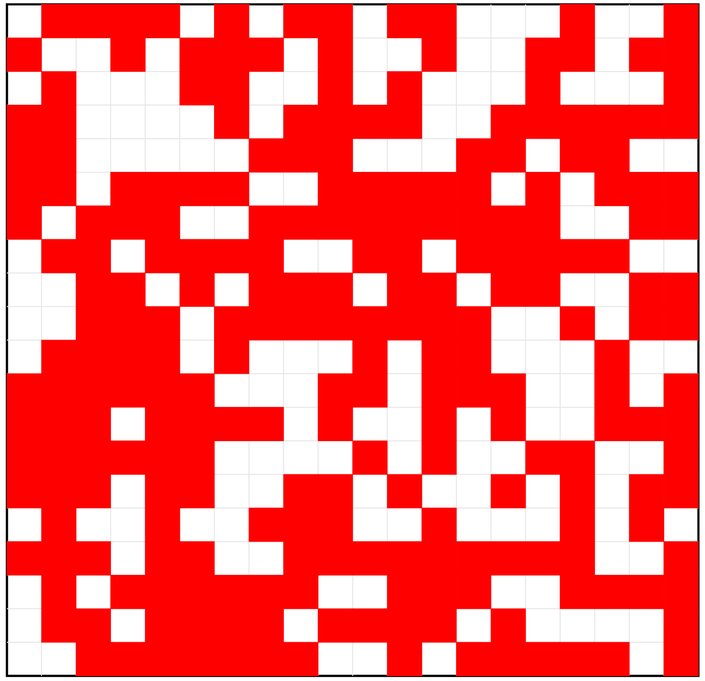


The spanning cluster

- can be ‘destroyed’ by removing few sites
- Is ‘infinite’ but contains a ‘vanishing’ fraction of the occupied sites
- It contains large holes: its mass varies as a power law with fractal exp.

Behaviour at the critical concentration

Is infinite in the thermodynamic limit



The spanning cluster

can be 'destroyed' by removing few sites

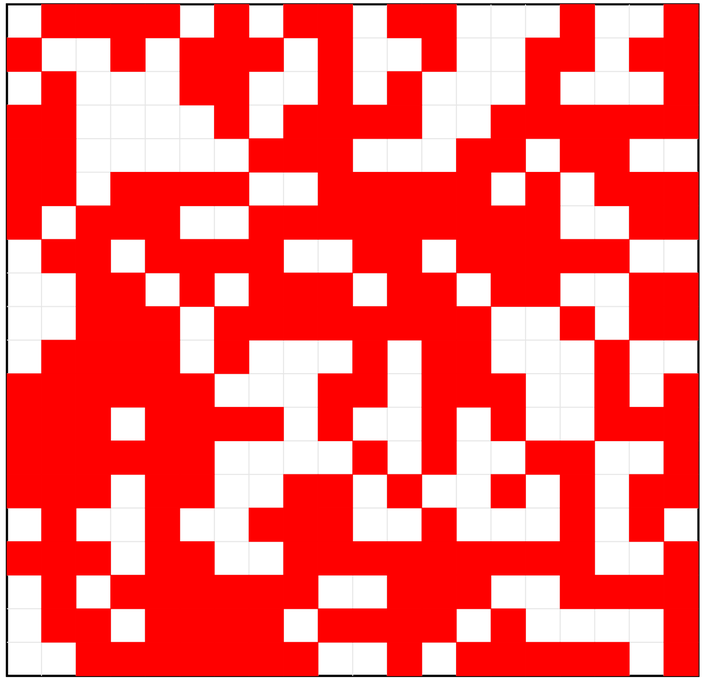
Is 'infinite' but contains a 'vanishing' fraction of the occupied sites

It contains large holes: its mass varies as a power law with fractal exp.

The mass (number of occupied sites) of a cluster varies as a function of r (the distance from a site to another site) as $m(r) \sim r^{D}$, where D = 2 for a regular 2D cluster but is 91/48≈1.9 for site percolation.

Behaviour at the critical concentration

Is infinite in the thermodynamic limit



The spanning cluster

can be 'destroyed' by removing few sites

Is 'infinite' but contains a 'vanishing' fraction of the occupied sites

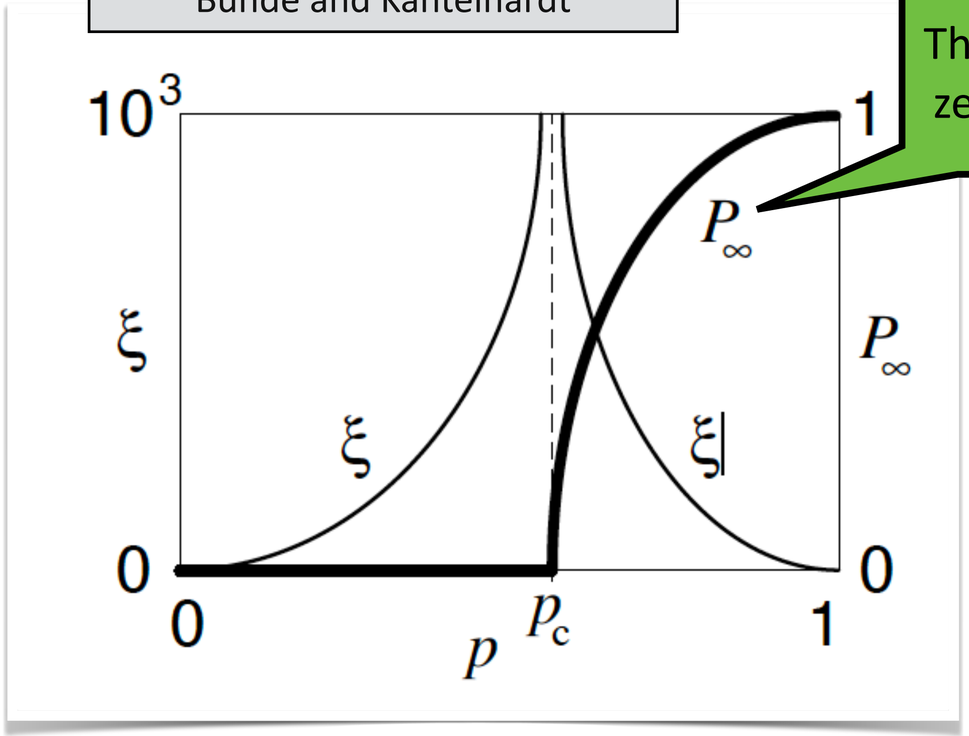
It contains large holes: its mass varies as a power law with fractal exp.

The mass (number of occupied sites) of a cluster varies as a function of r (the distance from a site to another site) as $m(r) \sim r^{D}$, where D = 2 for a regular 2D cluster but is 91/48≈1.9 for site percolation.

This fractal dimension (d_f) is one of the critical exponents mentioned before.

Other critical exponents

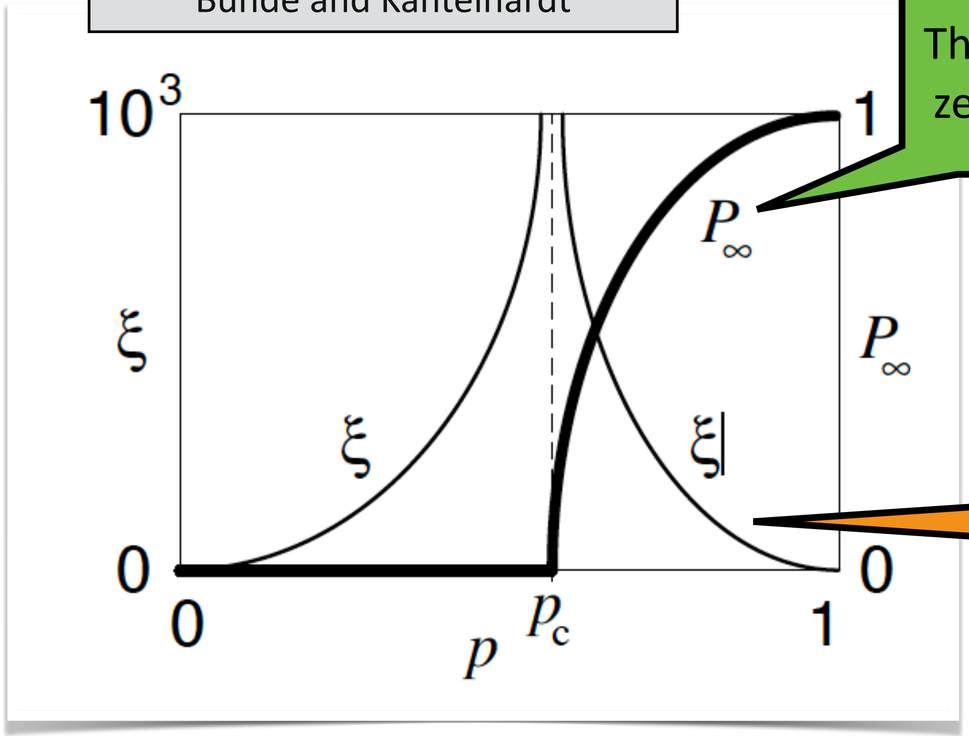
Introduction to percolation theory, Bunde and Kantelhardt



The probability p that a site belongs to the infinite cluster (order parameter) is zero below the critical concentration p_C, and increases above it as $P_\infty \sim (p-p_C)^\beta$

Other critical exponents

Introduction to percolation theory, Bunde and Kantelhardt

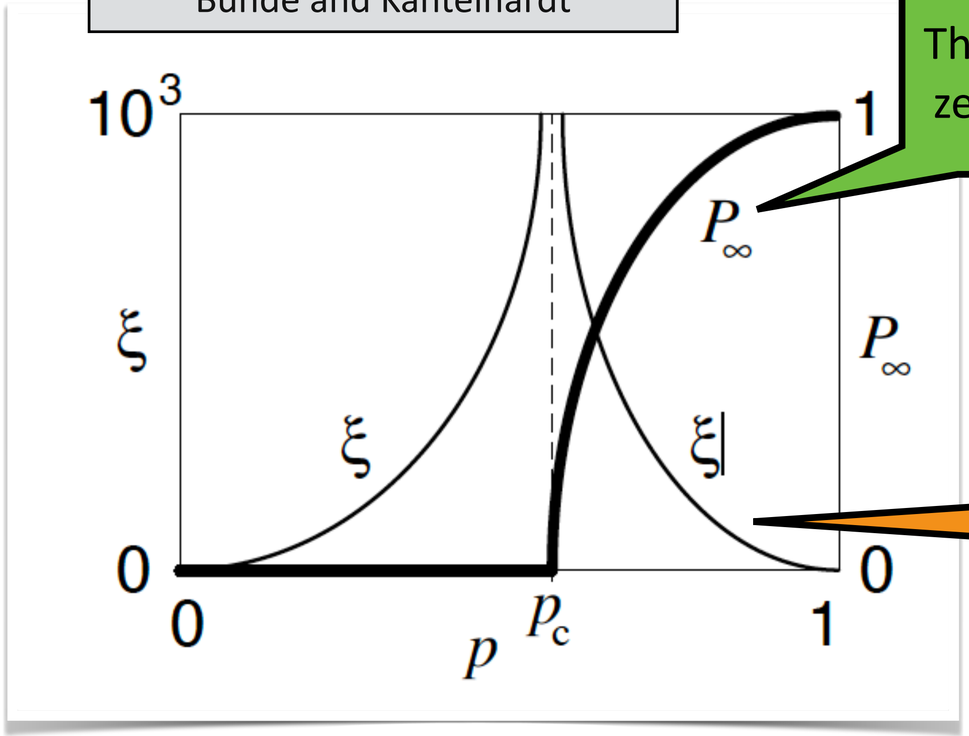


The probability p that a site belongs to the infinite cluster (order parameter) is zero below the critical concentration p_C, and increases above it as $P_\infty \sim (p-p_C)^\beta$

The mean distance between two sites in the same finite cluster defines the correlation length that increases towards the critical concentration as $\chi \sim |p-p_C|^{-\nu}$

Other critical exponents

Introduction to percolation theory, Bunde and Kantelhardt



The probability p that a site belongs to the infinite cluster (order parameter) is zero below the critical concentration p_C, and increases above it as $P_\infty \sim (p-p_C)^\beta$

The mean distance between two sites in the same finite cluster defines the correlation length that increases towards the critical concentration as $\chi \sim |p-p_C|^{-\nu}$

These critical exponents depend only on the dimension, but not the type of the lattice.

The three coefficients are related: $d_f = d - \beta/\nu$

Distribution of cluster sizes and susceptibility

Let $n(p,m)$ be the number of clusters of size m at the concentration p.

One can define a 'free energy' using the generation function h:

$$F = \Sigma n(p,m) \exp(-h\, m)$$

(where the sum is over all clusters except the infinite one)

Distribution of cluster sizes and susceptibility

Let $n(p,m)$ be the number of clusters of size m at the concentration p.

One can define a 'free energy' using the generation function h:

$$F = \Sigma n(p,m) \exp(-h\, m)$$

(where the sum is over all clusters except the infinite one)

The first derivative, evaluated at h=0, is related to P_∞ introduced in the previous slide

The second derivative is related to the susceptibility

$$\chi(p) = \Sigma m^2 n(p,m) / p \sim |p-p_C|^{-\gamma}$$

Percolation in practice

To learn something about percolation one studies random lattices of different sizes at different concentrations.

E.g. the critical exponents can be obtained by plotting their behaviour as a function of the lattice size.

Percolation in practice

To learn something about percolation one studies random lattices of different sizes at different concentrations.

E.g. the critical exponents can be obtained by plotting their behaviour as a function of the lattice size.

How to do this efficiently?

Algorithms

Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm

J. Hoshen and R. Kopelman
Phys. Rev. B **14**, 3438 – Published 15 October 1976

Article | References | Citing Articles (1,452) | PDF | Export Citation

Analysis of a lattice in O(N) operations

Algorithms

Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm

J. Hoshen and R. Kopelman
Phys. Rev. B **14**, 3438 – Published 15 October 1976

Article | References | Citing Articles (1,452) | PDF | Export Citation

Analysis of a lattice in O(N) operations

Observables for all p at once in O(N) operations

Efficient Monte Carlo Algorithm and High-Precision Results for Percolation

M. E. J. Newman and R. M. Ziff
Phys. Rev. Lett. **85**, 4104 – Published 6 November 2000

Article | References | Citing Articles (330) | PDF | Export Citation

Fast Monte Carlo algorithm for site or bond percolation

M. E. J. Newman and R. M. Ziff
Phys. Rev. E **64**, 016706 – Published 27 June 2001

Article | References | Citing Articles (317) | PDF | Export Citation

The Newman-Ziff algorithm

Key insight: Instead of computing for a given p (canonical ensamble) compute for a fixed number of occupied states (microcanonical ensamble) and convolute with a Binomial distribution to obtain an observable Q(p)

$$Q(p) = \sum_n \binom{N}{n} p^n (1-p)^{N-n} Q_n \,.$$

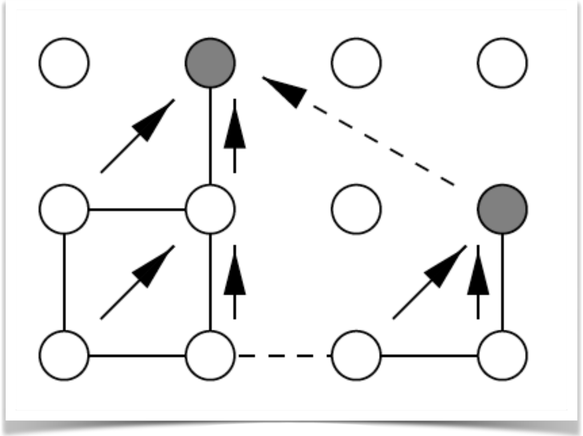
The Newman-Ziff algorithm

Key insight: Instead of computing for a given p (canonical ensamble) compute for a fixed number of occupied states (microcanonical ensamble) and convolute with a Binomial distribution to obtain an observable Q(p)

$$Q(p) = \sum_n \binom{N}{n} p^n (1-p)^{N-n} Q_n \,.$$

Algorithm for Qn

- Sites are added in a random order starting with an empty lattice
- → each site gets a unique label and a weight
- An added site can form a new cluster, join a single cluster or join together several clusters
- → use a tree structure to keep track of the belonging of a site to a cluster
- → use the weight to keep track of the size of the cluster
- Measure Qn



The Newman-Ziff algorithm

Key insight: Instead of computing for a given p (canonical ensamble) compute for a fixed number of occupied states (microcanonical ensamble) and convolute with a Binomial distribution to obtain an observable Q(p)

$$Q(p) = \sum_n \binom{N}{n} p^n (1-p)^{N-n} Q_n \,.$$

The full algorithm is 73 lines in C++ (found in the Appendix of PRE64 (2001) 016706)

Algorithm for Qn

Sites are added in a random order starting with an empty lattice

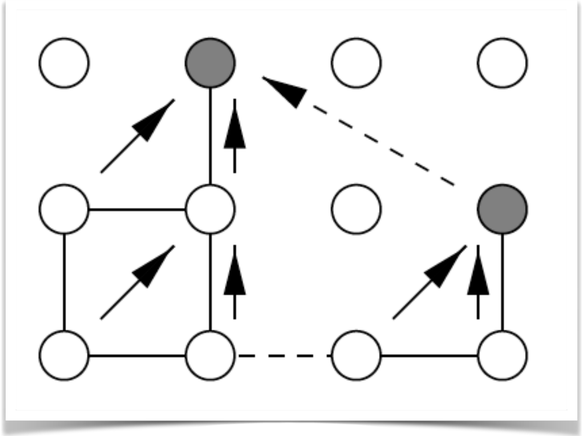
→ each site gets a unique label and a weight

An added site can form a new cluster, join a single cluster or join together several clusters

→ use a tree structure to keep track of the belonging of a site to a cluster

→ use the weight to keep track of the size of the cluster

Measure Qn



The Newman-Ziff algorithm: my code (1/3)

```
void sitePercolation(int length)
// length = size of one side of a 2D square lattice
{
  // initialise some internal variables
  int latticeSize = length*length; // lattice size
  int emptyCell = -(latticeSize+1); // mark empty cells
  std::vector<int> cellPointer(latticeSize,emptyCell); // assign cells to clusters

  // initialise observables
  int biggestCluster = 0;

  // initialise order of occupying the cells in the lattice
  std::vector<int> cellOrder(latticeSize); // create vector
  std::iota(cellOrder.begin(), cellOrder.end(), 0); // fill it form 0 to latticeSize-1
  std::shuffle(cellOrder.begin(), cellOrder.end(), std::random_device()); // suffle order

  // initialise neighbors of cell
  std::vector<int> cellNeighbours(latticeSize*4); // each cell has 4 neighbours
  setCellNeigbhours(length,latticeSize,cellNeighbours);

  // percolate
  int s1, s2, r1, r2; // internal indices for the pointers vector
  for(int i=0; i<latticeSize; i++) { // loop over cells
```

The Newman-Ziff algorithm: my code (1/3)

If empty, has a default value.
If part of a cluster points to its 'parent'.
If root of cluster has minus the size of the cluster.

```
void sitePercolation(int length)
// length = size of one side of a 2D square lattice
{
  // initialise some internal variables
  int latticeSize = length*length; // lattice size
  int emptyCell = -(latticeSize+1); // mark empty cells
  std::vector<int> cellPointer(latticeSize,emptyCell); // assign cells to clusters

  // initialise observables
  int biggestCluster = 0;

  // initialise order of occupying the cells in the lattice
  std::vector<int> cellOrder(latticeSize); // create vector
  std::iota(cellOrder.begin(), cellOrder.end(), 0); // fill it form 0 to latticeSize-1
  std::shuffle(cellOrder.begin(), cellOrder.end(), std::random_device()); // suffle order

  // initialise neighbors of cell
  std::vector<int> cellNeighbours(latticeSize*4); // each cell has 4 neighbours
  setCellNeigbhours(length,latticeSize,cellNeighbours);

  // percolate
  int s1, s2, r1, r2; // internal indices for the pointers vector
  for(int i=0; i<latticeSize; i++) { // loop over cells
```

The Newman-Ziff algorithm: my code (1/3)

```
void sitePercolation(int length)
// length = size of one side of a 2D square lattice
{
  // initialise some internal variables
  int latticeSize = length*length; // lattice size
  int emptyCell = -(latticeSize+1); // mark empty cells
  std::vector<int> cellPointer(latticeSize,emptyCell); // assign cells to clusters

  // initialise observables
  int biggestCluster = 0;

  // initialise order of occupying the cells in the lattice
  std::vector<int> cellOrder(latticeSize); // create vector
  std::iota(cellOrder.begin(), cellOrder.end(), 0); // fill it form 0 to latticeSize-1
  std::shuffle(cellOrder.begin(), cellOrder.end(), std::random_device()); // suffle order

  // initialise neighbors of cell
  std::vector<int> cellNeighbours(latticeSize*4); // each cell has 4 neighbours
  setCellNeigbhours(length,latticeSize,cellNeighbours);

  // percolate
  int s1, s2, r1, r2; // internal indices for the pointers vector
  for(int i=0; i<latticeSize; i++) { // loop over cells
```

If empty, has a default value.
If part of a cluster points to its ‘parent’.
If root of cluster has minus the size of the cluster.

Occupy the cells in a random order

The Newman-Ziff algorithm: my code (1/3)

```
void sitePercolation(int length)
// length = size of one side of a 2D square lattice
{
  // initialise some internal variables
  int latticeSize = length*length; // lattice size
  int emptyCell = -(latticeSize+1); // mark empty cells
  std::vector<int> cellPointer(latticeSize,emptyCell); // assign cells to clusters

  // initialise observables
  int biggestCluster = 0;

  // initialise order of occupying the cells in the lattice
  std::vector<int> cellOrder(latticeSize); // create vector
  std::iota(cellOrder.begin(), cellOrder.end(), 0); // fill it form 0 to latticeSize-1
  std::shuffle(cellOrder.begin(), cellOrder.end(), std::random_device()); // suffle order

  // initialise neighbors of cell
  std::vector<int> cellNeighbours(latticeSize*4); // each cell has 4 neighbours
  setCellNeigbhours(length,latticeSize,cellNeighbours);

  // percolate
  int s1, s2, r1, r2; // internal indices for the pointers vector
  for(int i=0; i<latticeSize; i++) { // loop over cells
```

If empty, has a default value.
If part of a cluster points to its ‘parent’.
If root of cluster has minus the size of the cluster.

Occupy the cells in a random order

Precompute the neighbours of each cell

The Newman-Ziff algorithm: my code (2/3)

```
void setCellNeigbhours(int length, int latticeSize, vector<int> &cellNeighbours)
{
  for(int i=0;i<latticeSize;i++) {
    cellNeighbours[i] = (i+1)%latticeSize;
    cellNeighbours[i+latticeSize] = (i+latticeSize-1)%latticeSize;
    cellNeighbours[i+2*latticeSize] = (i+length)%latticeSize;
    cellNeighbours[i+3*latticeSize] = (i+latticeSize-length)%latticeSize;
    // wrap horizontally
    if(i%length==0) cellNeighbours[i+latticeSize] = i+length-1;
    if((i+1)%length==0) cellNeighbours[i] = i-length+1;
  }
}
```

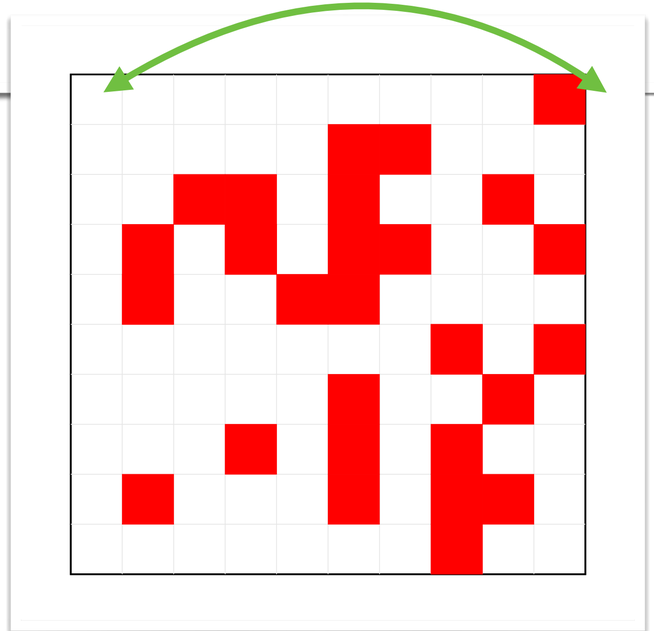
Fill each neighbour

The Newman-Ziff algorithm: my code (2/3)

```
void setCellNeigbhours(int length, int latticeSize, vector<int> &cellNeighbours)
{
  for(int i=0;i<latticeSize;i++) {
    cellNeighbours[i] = (i+1)%latticeSize;
    cellNeighbours[i+latticeSize] = (i+latticeSize-1)%latticeSize;
    cellNeighbours[i+2*latticeSize] = (i+length)%latticeSize;
    cellNeighbours[i+3*latticeSize] = (i+latticeSize-length)%latticeSize;
    // wrap horizontally
    if(i%length==0) cellNeighbours[i+latticeSize] = i+length-1;
    if((i+1)%length==0) cellNeighbours[i] = i-length+1;
  }
}
```

Fill each neighbour

If needed, wrap around.



The Newman-Ziff algorithm: my code (3/3)

Loop over cells

```
// percolate
int s1, s2, r1, r2; // internal indices for the pointers vector
for(int i=0; i<latticeSize; i++) { // loop over cells
  r1 = s1 = cellOrder[i]; // new cell
  cellPointer[s1] = -1; // current size of the cluster
  for(int j=0;j<4;j++) {// loop over neighbours
    s2 = cellNeighbours[s1+j*latticeSize]; // index of neighbouring cell
    if(cellPointer[s2] != emptyCell) { // cell not empty, form a cluster
      r2 = findroot(s2,cellPointer); // find representative of the cluster of this cell
      if (r2 != r1) {// merge clusters: smaller cluster is absorved
        if (cellPointer[r1] > cellPointer[r2]) { // cluster size is negative for root nodes!
          cellPointer[r2] += cellPointer[r1];
          cellPointer[r1] = r2;
          r1 = r2;
        } else {
          cellPointer[r1] += cellPointer[r2];
          cellPointer[r2] = r1;
        }
        // fill the observable
        if (-cellPointer[r1]>biggestCluster) biggestCluster = -cellPointer[r1];
      } // end of merging
    } // end cell not empty
  } // end loop over neighbours
  // print out the observable
  std::cout << i << "  " << biggestCluster  << endl;
} // end loop over cells
```

The Newman-Ziff algorithm: my code (3/3)

```
// percolate
int s1, s2, r1, r2; // internal indices for the pointers vector
for(int i=0; i<latticeSize; i++) { // loop over cells
  r1 = s1 = cellOrder[i]; // new cell
  cellPointer[s1] = -1; // current size of the cluster
  for(int j=0;j<4;j++) {// loop over neighbours
    s2 = cellNeighbours[s1+j*latticeSize]; // index of neighbouring cell
    if(cellPointer[s2] != emptyCell) { // cell not empty, form a cluster
      r2 = findroot(s2,cellPointer); // find representative of the cluster of this cell
      if (r2 != r1) {// merge clusters: smaller cluster is absorved
        if (cellPointer[r1] > cellPointer[r2]) { // cluster size is negative for root nodes!
          cellPointer[r2] += cellPointer[r1];
          cellPointer[r1] = r2;
          r1 = r2;
        } else {
          cellPointer[r1] += cellPointer[r2];
          cellPointer[r2] = r1;
        }
        // fill the observable
        if (-cellPointer[r1]>biggestCluster) biggestCluster = -cellPointer[r1];
      } // end of merging
    } // end cell not empty
  } // end loop over neighbours
  // print out the observable
  std::cout << i << "  " << biggestCluster  << endl;
} // end loop over cells
```

Loop over cells

Study next cell

The Newman-Ziff algorithm: my code (3/3)

```
// percolate
int s1, s2, r1, r2; // internal indices for the pointers vector
for(int i=0; i<latticeSize; i++) { // loop over cells
  r1 = s1 = cellOrder[i]; // new cell
  cellPointer[s1] = -1; // current size of the cluster
  for(int j=0;j<4;j++) {// loop over neighbours
    s2 = cellNeighbours[s1+j*latticeSize]; // index of neighbouring cell
    if(cellPointer[s2] != emptyCell) { // cell not empty, form a cluster
      r2 = findroot(s2,cellPointer); // find representative of the cluster of this cell
      if (r2 != r1) {// merge clusters: smaller cluster is absorved
        if (cellPointer[r1] > cellPointer[r2]) { // cluster size is negative for root nodes!
          cellPointer[r2] += cellPointer[r1];
          cellPointer[r1] = r2;
          r1 = r2;
        } else {
          cellPointer[r1] += cellPointer[r2];
          cellPointer[r2] = r1;
        }
        // fill the observable
        if (-cellPointer[r1]>biggestCluster) biggestCluster = -cellPointer[r1];
      } // end of merging
    } // end cell not empty
  } // end loop over neighbours
  // print out the observable
  std::cout << i << "  " << biggestCluster  << endl;
} // end loop over cells
```

Loop over cells

Study next cell

Check each neighbour in turn

The Newman-Ziff algorithm: my code (3/3)

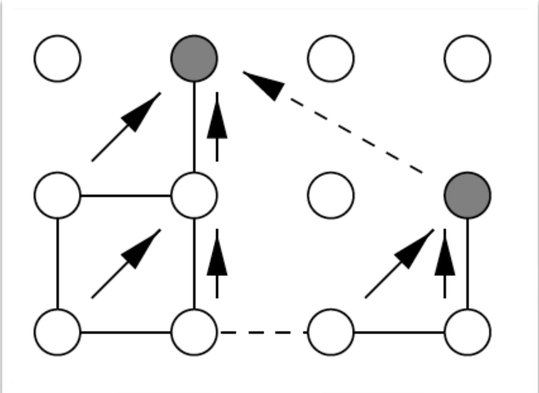
```
// percolate
int s1, s2, r1, r2; // internal indices for the pointers vector
for(int i=0; i<latticeSize; i++) { // loop over cells
  r1 = s1 = cellOrder[i]; // new cell
  cellPointer[s1] = -1; // current size of the cluster
  for(int j=0;j<4;j++) {// loop over neighbours
    s2 = cellNeighbours[s1+j*latticeSize]; // index of neighbouring cell
    if(cellPointer[s2] != emptyCell) { // cell not empty, form a cluster
      r2 = findroot(s2,cellPointer); // find representative of the cluster of this cell
      if (r2 != r1) {// merge clusters: smaller cluster is absorved
        if (cellPointer[r1] > cellPointer[r2]) { // cluster size is negative for root nodes!
          cellPointer[r2] += cellPointer[r1];
          cellPointer[r1] = r2;
          r1 = r2;
        } else {
          cellPointer[r1] += cellPointer[r2];
          cellPointer[r2] = r1;
        }
        // fill the observable
        if (-cellPointer[r1]>biggestCluster) biggestCluster = -cellPointer[r1];
      } // end of merging
    } // end cell not empty
  } // end loop over neighbours
  // print out the observable
  std::cout << i << "  " << biggestCluster  << endl;
} // end loop over cells
```

Loop over cells

Study next cell

Check each neighbour in turn

If empty, done;
If not, get the root of its cluster



The Newman-Ziff algorithm: my code (3/3)

```
// percolate
int s1, s2, r1, r2; // internal indices for the pointers vector
for(int i=0; i<latticeSize; i++) { // loop over cells
  r1 = s1 = cellOrder[i]; // new cell
  cellPointer[s1] = -1; // current size of the cluster
  for(int j=0;j<4;j++) {// loop over neighbours
    s2 = cellNeighbours[s1+j*latticeSize]; // index of neighbouring cell
    if(cellPointer[s2] != emptyCell) { // cell not empty, form a cluster
      r2 = findroot(s2,cellPointer); // find representative of the cluster of this cell
      if (r2 != r1) {// merge clusters: smaller cluster is absorved
        if (cellPointer[r1] > cellPointer[r2]) { // cluster size is negative for root nodes!
          cellPointer[r2] += cellPointer[r1];
          cellPointer[r1] = r2;
          r1 = r2;
        } else {
          cellPointer[r1] += cellPointer[r2];
          cellPointer[r2] = r1;
        }
        // fill the observable
        if (-cellPointer[r1]>biggestCluster) biggestCluster = -cellPointer[r1];
      } // end of merging
    } // end cell not empty
  } // end loop over neighbours
  // print out the observable
  std::cout << i << "  " << biggestCluster  << endl;
} // end loop over cells
```

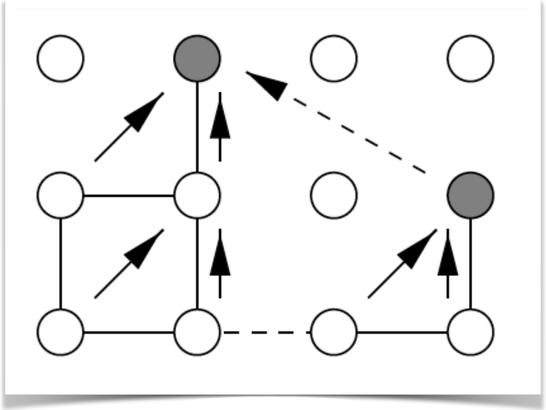
Loop over cells

Study next cell

Check each neighbour in turn

If empty, done;
If not, get the root of its cluster

```
int findroot(int i, vector<int> &cellPointer)
// implements path compression
{
  if (cellPointer[i]<0) return i;
  return cellPointer[i] = findroot(cellPointer[i],cellPointer);
}
```



The Newman-Ziff algorithm: my code (3/3)

```
// percolate
int s1, s2, r1, r2; // internal indices for the pointers vector
for(int i=0; i<latticeSize; i++) { // loop over cells
  r1 = s1 = cellOrder[i]; // new cell
  cellPointer[s1] = -1; // current size of the cluster
  for(int j=0;j<4;j++) {// loop over neighbours
    s2 = cellNeighbours[s1+j*latticeSize]; // index of neighbouring cell
    if(cellPointer[s2] != emptyCell) { // cell not empty, form a cluster
      r2 = findroot(s2,cellPointer); // find representative of the cluster of this cell
      if (r2 != r1) {// merge clusters: smaller cluster is absorved
        if (cellPointer[r1] > cellPointer[r2]) { // cluster size is negative for root nodes!
          cellPointer[r2] += cellPointer[r1];
          cellPointer[r1] = r2;
          r1 = r2;
        } else {
          cellPointer[r1] += cellPointer[r2];
          cellPointer[r2] = r1;
        }
        // fill the observable
        if (-cellPointer[r1]>biggestCluster) biggestCluster = -cellPointer[r1];
      } // end of merging
    } // end cell not empty
  } // end loop over neighbours
  // print out the observable
  std::cout << i << "  " << biggestCluster  << endl;
} // end loop over cells
```

Loop over cells

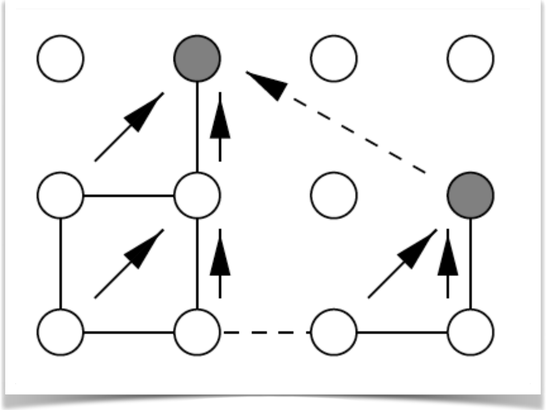
Study next cell

Check each neighbour in turn

If empty, done;
If not, get the root of its cluster

If needed, merge clusters

```
int findroot(int i, vector<int> &cellPointer)
// implements path compression
{
  if (cellPointer[i]<0) return i;
  return cellPointer[i] = findroot(cellPointer[i],cellPointer);
}
```



The Newman-Ziff algorithm: my code (3/3)

```
// percolate
int s1, s2, r1, r2; // internal indices for the pointers vector
for(int i=0; i<latticeSize; i++) { // loop over cells
  r1 = s1 = cellOrder[i]; // new cell
  cellPointer[s1] = -1; // current size of the cluster
  for(int j=0;j<4;j++) {// loop over neighbours
    s2 = cellNeighbours[s1+j*latticeSize]; // index of neighbouring cell
    if(cellPointer[s2] != emptyCell) { // cell not empty, form a cluster
      r2 = findroot(s2,cellPointer); // find representative of the cluster of this cell
      if (r2 != r1) {// merge clusters: smaller cluster is absorved
        if (cellPointer[r1] > cellPointer[r2]) { // cluster size is negative for root nodes!
          cellPointer[r2] += cellPointer[r1];
          cellPointer[r1] = r2;
          r1 = r2;
        } else {
          cellPointer[r1] += cellPointer[r2];
          cellPointer[r2] = r1;
        }
        // fill the observable
        if (-cellPointer[r1]>biggestCluster) biggestCluster = -cellPointer[r1];
      } // end of merging
    } // end cell not empty
  } // end loop over neighbours
  // print out the observable
  std::cout << i << "  " << biggestCluster  << endl;
} // end loop over cells
```

Loop over cells

Study next cell

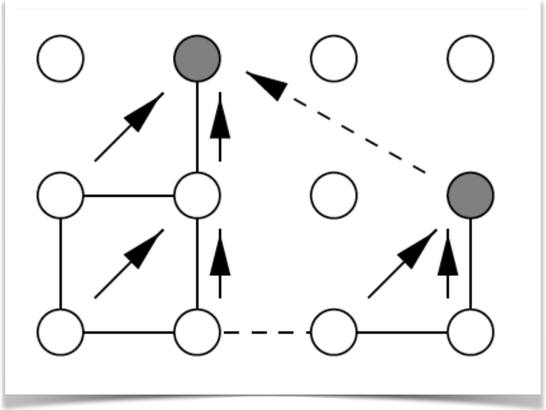
Check each neighbour in turn

If empty, done;
If not, get the root of its cluster

If needed, merge clusters

Compute whatever you want

```
int findroot(int i, vector<int> &cellPointer)
// implements path compression
{
  if (cellPointer[i]<0) return i;
  return cellPointer[i] = findroot(cellPointer[i],cellPointer);
}
```



The Newman-Ziff algorithm: my code (3/3)

```
// percolate
int s1, s2, r1, r2; // internal indices for the pointers vector
for(int i=0; i<latticeSize; i++) { // loop over cells
  r1 = s1 = cellOrder[i]; // new cell
  cellPointer[s1] = -1; // current size of the cluster
  for(int j=0;j<4;j++) {// loop over neighbours
    s2 = cellNeighbours[s1+j*latticeSize]; // index of neighbouring cell
    if(cellPointer[s2] != emptyCell) { // cell not empty, form a cluster
      r2 = findroot(s2,cellPointer); // find representative of the cluster of this cell
      if (r2 != r1) {// merge clusters: smaller cluster is absorved
        if (cellPointer[r1] > cellPointer[r2]) { // cluster size is negative for root nodes!
          cellPointer[r2] += cellPointer[r1];
          cellPointer[r1] = r2;
          r1 = r2;
        } else {
          cellPointer[r1] += cellPointer[r2];
          cellPointer[r2] = r1;
        }
        // fill the observable
        if (-cellPointer[r1]>biggestCluster) biggestCluster = -cellPointer[r1];
      } // end of merging
    } // end cell not empty
  } // end loop over neighbours
  // print out the observable
  std::cout << i << "  " << biggestCluster  << endl;
} // end loop over cells
```

Loop over cells

Study next cell

Check each neighbour in turn

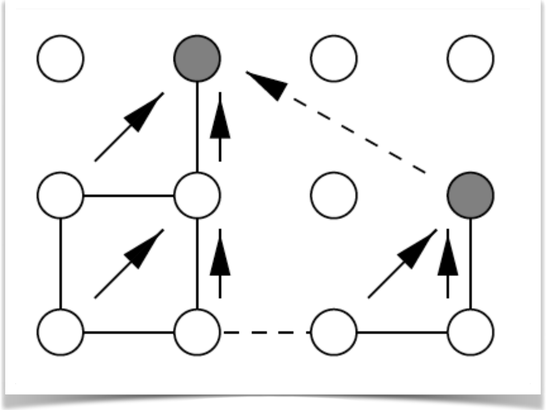
If empty, done;
If not, get the root of its cluster

If needed, merge clusters

Compute whatever you want

Show the results

```
int findroot(int i, vector<int> &cellPointer)
// implements path compression
{
  if (cellPointer[i]<0) return i;
  return cellPointer[i] = findroot(cellPointer[i],cellPointer);
}
```



Continuum percolation

Most real problems do not occur in a lattice, but in a continuum.
The Newman-Ziff algorithm has been extended to this case

Continuum percolation thresholds in two dimensions

Stephan Mertens and Cristopher Moore
Phys. Rev. E **86**, 061109 – Published 7 December 2012

Article | References | Citing Articles (105) | PDF | HTML | Export Citation

Continuum percolation

Continuum and lattice percolation are in the same universality class.
They have the same critical exponents but different transition points.

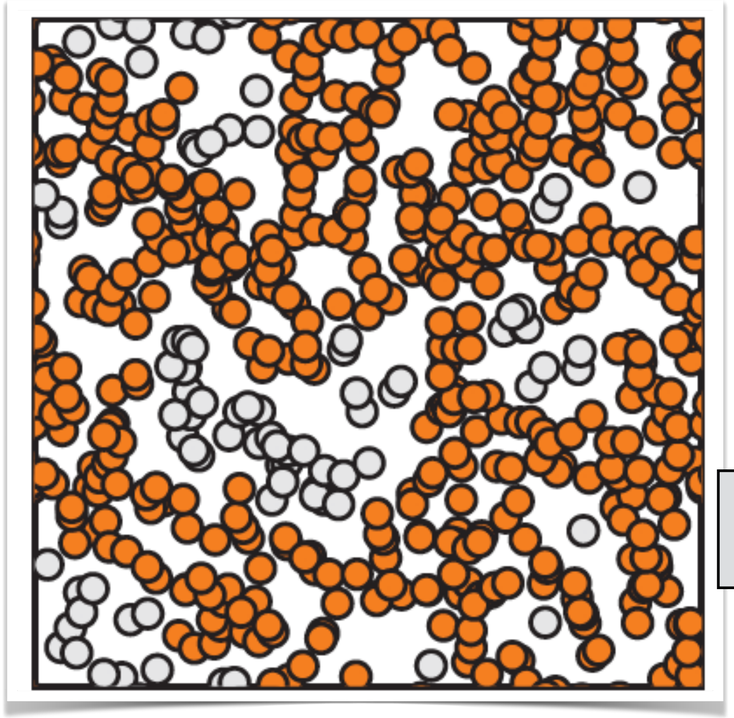
Most real problems do not occur in a lattice, but in a continuum.
The Newman-Ziff algorithm has been extended to this case

Continuum percolation thresholds in two dimensions

Stephan Mertens and Cristopher Moore
Phys. Rev. E **86**, 061109 – Published 7 December 2012

Article | References | Citing Articles (105) | PDF | HTML | Export Citation

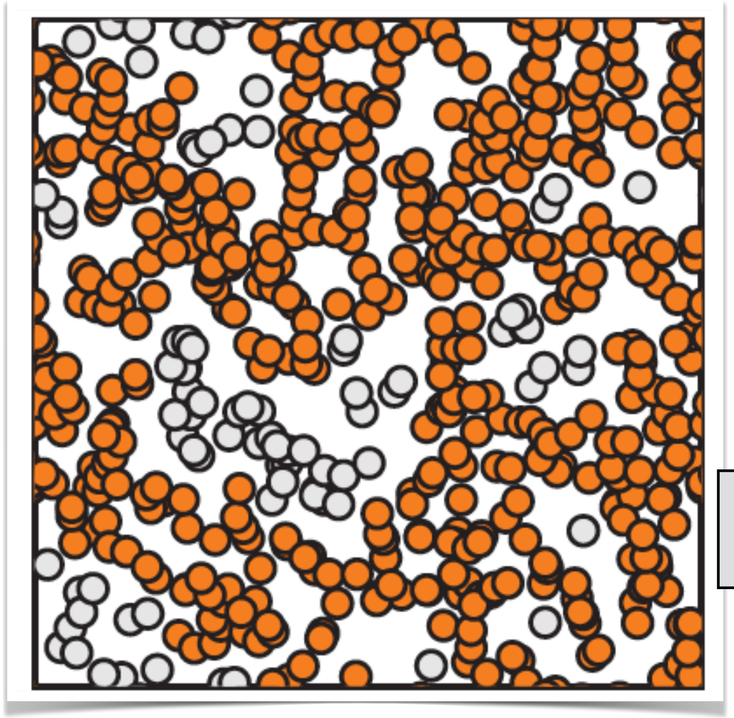
Continuum percolation: changes to the algorithm



In this case the cells can overlap and there is not a fixed number of neighbours.

Mertens and Moore, PRE 86, 061109 (2012)

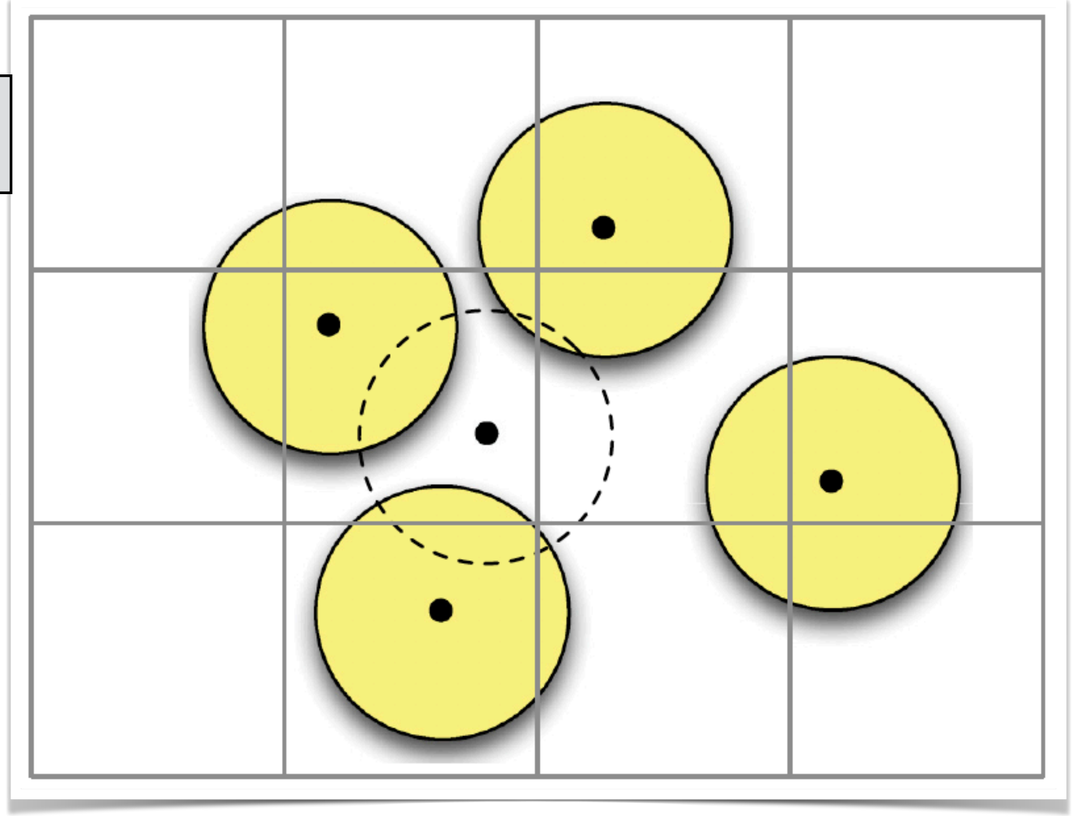
Continuum percolation: changes to the algorithm



In this case the cells can overlap and there is not a fixed number of neighbours.

Mertens and Moore, PRE 86, 061109 (2012)

Use a grid to make an efficient search for neighbours.



Some applications of interest today: Forest fire

Statics of a "self-organized" percolation model

Christopher L. Henley
Phys. Rev. Lett. **71**, 2741 – Published 25 October 1993

Article | References | Citing Articles (74) | PDF | Export Citation

ABSTRACT

A stochastic "forest-fire" model is considered. Sites are filled individually at a constant mean rate; also, "sparks" are dropped at a small rate k, and instantaneously burn up the entire cluster they hit. I find *nontrivial* critical exponents in the self-organized critical limit $k \to 0$, contrary to earlier results of Drossel and Schwabl. Spatial correlation functions and a site occupancy correlation exponent are measured for the first time. Scaling relations, derived by analogy to uncorrelated percolation, are used extensively as numerical checks. Hyperscaling is violated in this system.

Some applications of interest today: Ecology

Ecology, 76(8), 1995, pp. 2446–2459

CRITICAL THRESHOLDS IN SPECIES' RESPONSES TO LANDSCAPE STRUCTURE[1]

KIMBERLY A. WITH[2]
Environmental Sciences Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831–6038 USA

THOMAS O. CRIST[3]
Department of Biology, Colorado State University, Fort Collins, Colorado 80523 USA

Abstract. Critical thresholds are transition ranges across which small changes in spatial pattern produce abrupt shifts in ecological responses. Habitat fragmentation provides a familiar example of a critical threshold. As the landscape becomes dissected into smaller parcels of habitat, landscape connectivity—the functional linkage among habitat patches—may suddenly become disrupted, which may have important consequences for the distribution and persistence of populations. Landscape connectivity depends not only on the abundance and spatial patterning of habitat, but also on the habitat specificity and dispersal abilities of species. Habitat specialists with limited dispersal capabilities presumably have a much lower threshold to habitat fragmentation than highly vagile species, which may perceive the landscape as functionally connected across a greater range of fragmentation severity.

To determine where threshold effects in species' responses to landscape structure are likely to occur, we developed a simulation model modified from percolation theory. Our simulations predicted the distributional patterns of populations in different landscape mosaics, which we tested empirically using two grasshopper species (Orthoptera: Acrididae) that occur in the shortgrass prairie of north-central Colorado. Increasing degree of habitat

Some applications of interest today: Ecology

Ecology, 76(8), 1995, pp. 2446–2459

CRITICAL THRESHOLDS IN SPECIES' RESPONSES TO LANDSCAPE STRUCTURE[1]

KIMBERLY A. WITH[2]
Environmental Sciences Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831–6038 USA

THOMAS O. CRIST[3]
Department of Biology, Colorado State University, Fort Collins, Colorado 80523 USA

Abstract. Critical thresholds are transition ranges across which small changes in spatial pattern produce abrupt shifts in ecological responses. Habitat fragmentation provides a familiar example of a critical threshold. As the landscape becomes dissected into smaller parcels of habitat, landscape connectivity—the functional linkage among habitat patches—may suddenly become disrupted, which may have important consequences for the distribution and persistence of populations. Landscape connectivity depends not only on the abundance and spatial patterning of habitat, but also on the habitat specificity and dispersal abilities of species. Habitat specialists with limited dispersal capabilities presumably have a much lower threshold to habitat fragmentation than highly vagile species, which may perceive the landscape as functionally connected across a greater range of fragmentation severity.

To determine where threshold effects in species' responses to landscape structure are likely to occur, we developed a simulation model modified from percolation theory. Our simulations predicted the distributional patterns of populations in different landscape mosaics, which we tested empirically using two grasshopper species (Orthoptera: Acrididae) that occur in the shortgrass prairie of north-central Colorado. Increasing degree of habitat

Two different types of grasshoppers simulated.
Simulations describe observations.

Some applications of interest today: Ecology

Ecology, 76(8), 1995, pp. 2446–2459

CRITICAL THRESHOLDS IN SPECIES' RESPONSES TO LANDSCAPE STRUCTURE[1]

KIMBERLY A. WITH[2]
Environmental Sciences Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831–6038 USA

THOMAS O. CRIST[3]
Department of Biology, Colorado State University, Fort Collins, Colorado 80523 USA

Abstract. Critical thresholds are transition ranges across which small changes in spatial pattern produce abrupt shifts in ecological responses. Habitat fragmentation provides a familiar example of a critical threshold. As the landscape becomes dissected into smaller parcels of habitat, landscape connectivity—the functional linkage among habitat patches—may suddenly become disrupted, which may have important consequences for the distribution and persistence of populations. Landscape connectivity depends not only on the abundance and spatial patterning of habitat, but also on the habitat specificity and dispersal abilities of species. Habitat specialists with limited dispersal capabilities presumably have a much lower threshold to habitat fragmentation than highly vagile species, which may perceive the landscape as functionally connected across a greater range of fragmentation severity.

To determine where threshold effects in species' responses to landscape structure are likely to occur, we developed a simulation model modified from percolation theory. Our simulations predicted the distributional patterns of populations in different landscape mosaics, which we tested empirically using two grasshopper species (Orthoptera: Acrididae) that occur in the shortgrass prairie of north-central Colorado. Increasing degree of habitat

Two different types of grasshoppers simulated.
Simulations describe observations.

Three types of cells

Different probabilities of moving on the lattice.

Some applications of interest today: Media industry

ELSEVIER

Physica A: Statistical Mechanics and its Applications

Volume 277, Issues 1–2, 1 March 2000, Pages 239-247

Social percolation models

Sorin Solomon [a, b], Gerard Weisbuch [a], Lucilla de Arcangelis [c, d], Naeem Jan [c], Dietrich Stauffer [c, e]

Show more

Share Cite

https://doi.org/10.1016/S0378-4371(99)00543-9

Get rights and content

Abstract

We here relate the occurrence of extreme market shares, close to either 0 or 100%, in the media industry to a percolation phenomenon across the social network of customers. We further discuss the possibility of observing self-organized criticality when customers and cinema producers adjust their preferences and the quality of the produced films according to previous experience. Comprehensive computer simulations on square lattices do indeed exhibit self-organized criticality towards the usual percolation threshold and related scaling behaviour.

Some applications of interest today: Media industry

ELSEVIER

Physica A: Statistical Mechanics and its Applications

Volume 277, Issues 1–2, 1 March 2000, Pages 239-247

Social percolation models

Sorin Solomon [a, b], Gerard Weisbuch [a], Lucilla de Arcangelis [c, d], Naeem Jan [c], Dietrich Stauffer [c, e]

Show more

Share Cite

https://doi.org/10.1016/S0378-4371(99)00543-9

Get rights and content

Abstract

We here relate the occurrence of extreme market shares, close to either 0 or 100%, in the media industry to a percolation phenomenon across the social network of customers. We further discuss the possibility of observing self-organized criticality when customers and cinema producers adjust their preferences and the quality of the produced films according to previous experience. Comprehensive computer simulations on square lattices do indeed exhibit self-organized criticality towards the usual percolation threshold and related scaling behaviour.

Bi-modal distribution of market shares.

Model with a lattice populated with agents.

Quality as a weight for transport.

Some applications of interest today: Media industry

ELSEVIER

Physica A: Statistical Mechanics and its Applications

Volume 277, Issues 1–2, 1 March 2000, Pages 239-247

Social percolation models

Sorin Solomon [a, b], Gerard Weisbuch [a], Lucilla de Arcangelis [c, d], Naeem Jan [c], Dietrich Stauffer [c, e]

Show more

Share Cite

https://doi.org/10.1016/S0378-4371(99)00543-9

Get rights and content

Abstract

We here relate the occurrence of extreme market shares, close to either 0 or 100%, in the media industry to a percolation phenomenon across the social network of customers. We further discuss the possibility of observing self-organized criticality when customers and cinema producers adjust their preferences and the quality of the produced films according to previous experience. Comprehensive computer simulations on square lattices do indeed exhibit self-organized criticality towards the usual percolation threshold and related scaling behaviour.

Bi-modal distribution of market shares.

Model with a lattice populated with agents.

Quality as a weight for transport.

Play with quality, preferences.

Some applications of interest today: disease propagation

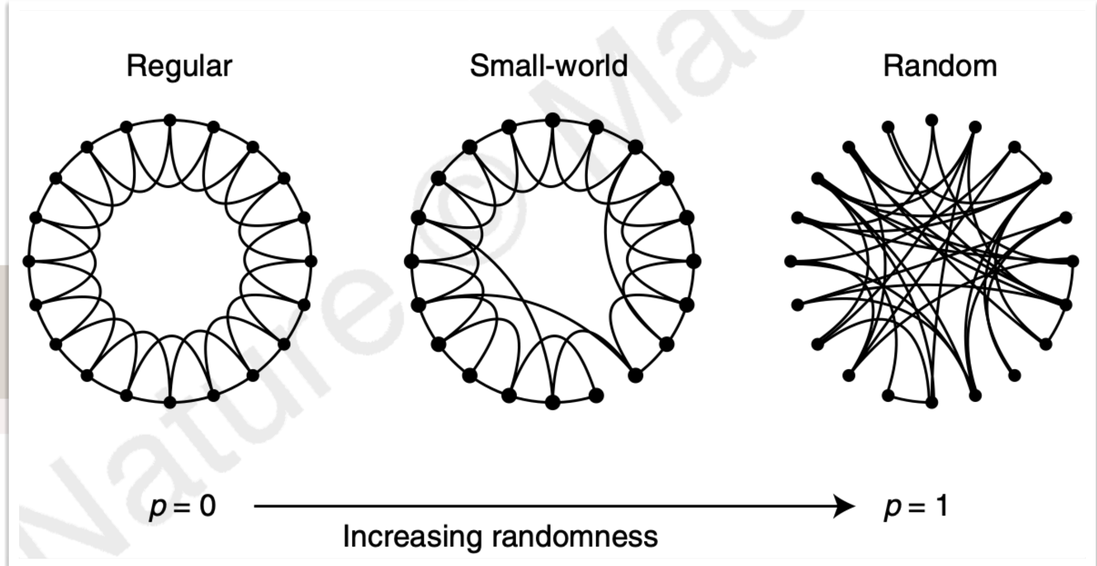
Exact solution of site and bond percolation on small-world networks

Cristopher Moore and M. E. J. Newman
Phys. Rev. E **62**, 7059 – Published 1 November 2000

Article | References | Citing Articles (123) | PDF | Export Citation

ABSTRACT

We study percolation on small-world networks, which has been proposed as a simple model of the propagation of disease. The occupation probabilities of sites and bonds correspond to the susceptibility of individuals to the disease, and the transmissibility of the disease respectively. We give an exact solution of the model for both site and bond percolation, including the position of the percolation transition at which epidemic behavior sets in, the values of the critical exponents governing this transition, the mean and variance of the distribution of cluster sizes (disease outbreaks) below the transition, and the size of the giant component (epidemic) above the transition.



Some applications of interest today: disease propagation

Exact solution of site and bond percolation on small-world networks

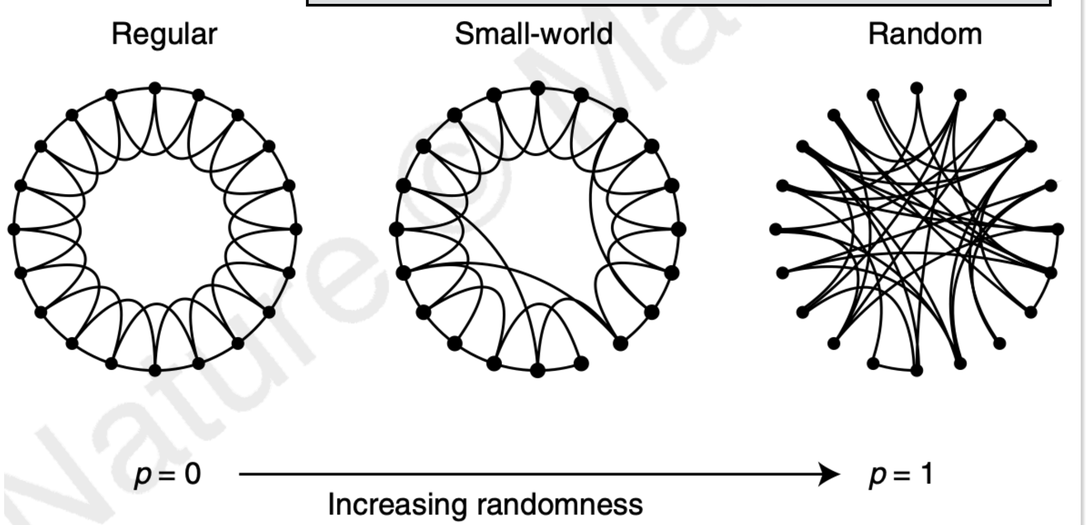
Cristopher Moore and M. E. J. Newman
Phys. Rev. E **62**, 7059 – Published 1 November 2000

Article | References | Citing Articles (123) | PDF | Export Citation

ABSTRACT

We study percolation on small-world networks, which has been proposed as a simple model of the propagation of disease. The occupation probabilities of sites and bonds correspond to the susceptibility of individuals to the disease, and the transmissibility of the disease respectively. We give an exact solution of the model for both site and bond percolation, including the position of the percolation transition at which epidemic behavior sets in, the values of the critical exponents governing this transition, the mean and variance of the distribution of cluster sizes (disease outbreaks) below the transition, and the size of the giant component (epidemic) above the transition.

Watts, Strogat, Nature 393, 440–442 (1998)



Some applications of interest today: Network robustness

Network Robustness and Fragility: Percolation on Random Graphs

Duncan S. Callaway, M. E. J. Newman, Steven H. Strogatz, and Duncan J. Watts
Phys. Rev. Lett. **85**, 5468 – Published 18 December 2000

Article | References | Citing Articles (1,511) | PDF | Export Citation

ABSTRACT

Recent work on the Internet, social networks, and the power grid has addressed the resilience of these networks to either random or targeted deletion of network nodes or links. Such deletions include, for example, the failure of Internet routers or power transmission lines. Percolation models on random graphs provide a simple representation of this process but have typically been limited to graphs with Poisson degree distribution at their vertices. Such graphs are quite unlike real-world networks, which often possess power-law or other highly skewed degree distributions. In this paper we study percolation on graphs with completely general degree distribution, giving exact solutions for a variety of cases, including site percolation, bond percolation, and models in which occupation probabilities depend on vertex degree. We discuss the application of our theory to the understanding of network resilience.

Some applications of interest today: QCD and the QGP

Percolation Approach to Quark-Gluon Plasma and J/ψ Suppression

N. Armesto, M. A. Braun, E. G. Ferreiro, and C. Pajares
Phys. Rev. Lett. **77**, 3736 – Published 28 October 1996

Article | References | Citing Articles (130) | PDF | Export Citation

ABSTRACT

It is shown that the critical threshold for percolation of the overlapping strings exchanged in heavy ion collisions can naturally explain the sharp strong suppression of J/ψ shown by the experimental data on central Pb-Pb collisions, which does not occur in central O-U and S-U collisions.