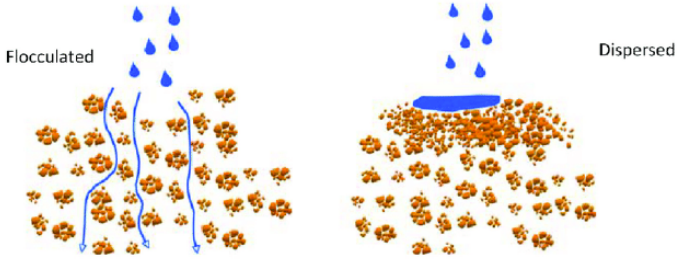


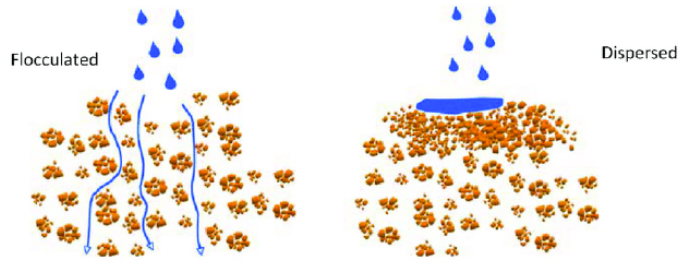
Percolation!

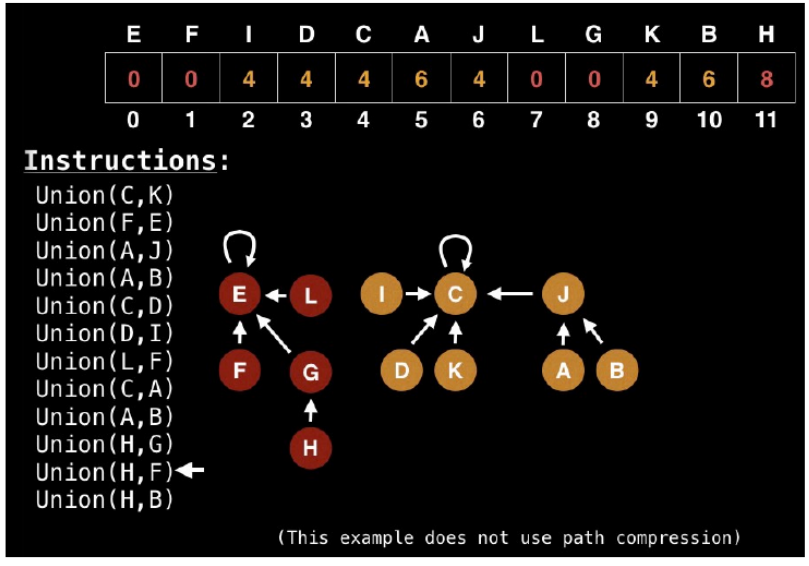
A Game for all ages

Ryan Levy

Today's ~~Menu~~ Algorithms

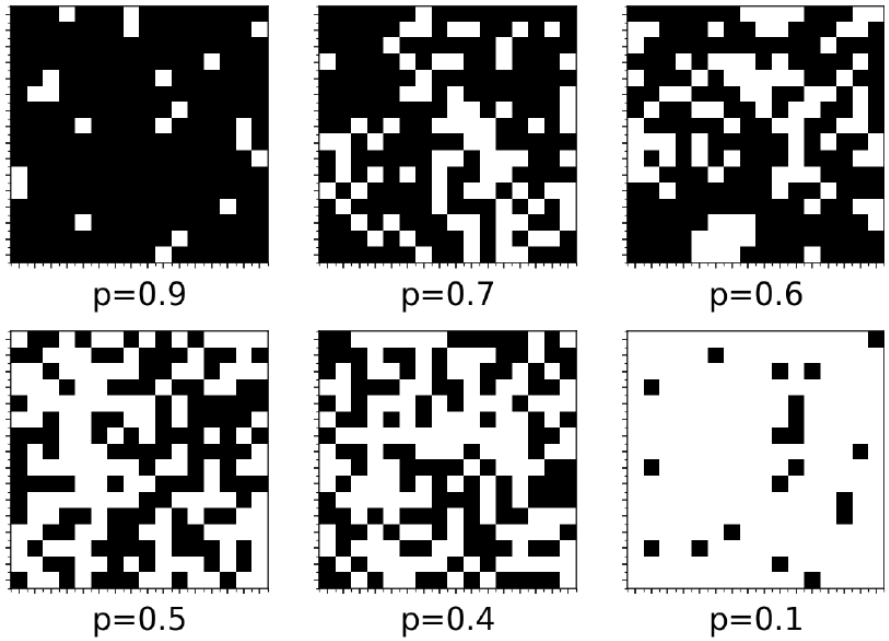
- Percolation – Union/Find
  - Hoshen–Kopelman Algorithm
  - Newman-Ziff Algorithm





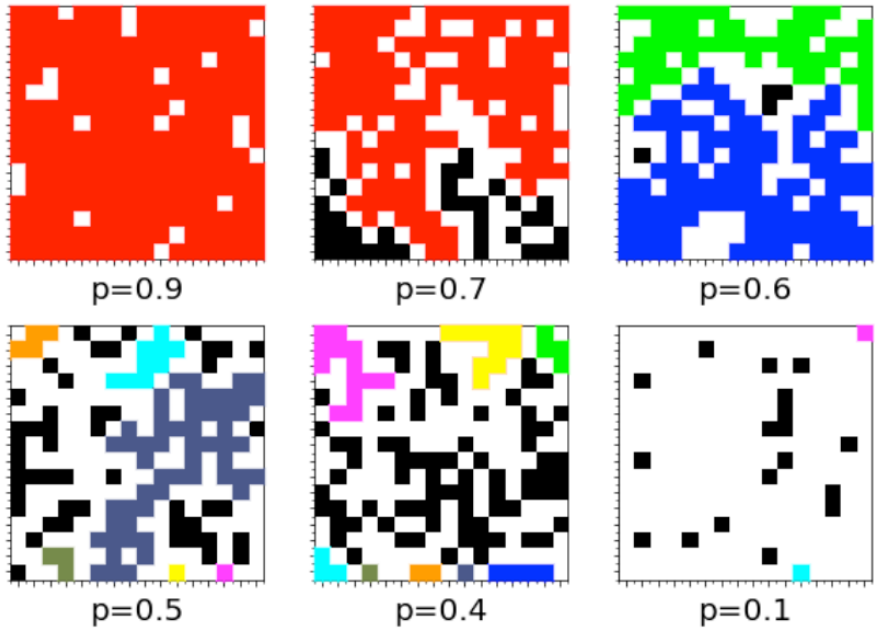
Introduction

- Consider a lattice (or graph) with binary options of occupied or not
- Percolation – Is there a cluster that spans the entire system?
- Question of the day – determine when there's always a percolating cluster



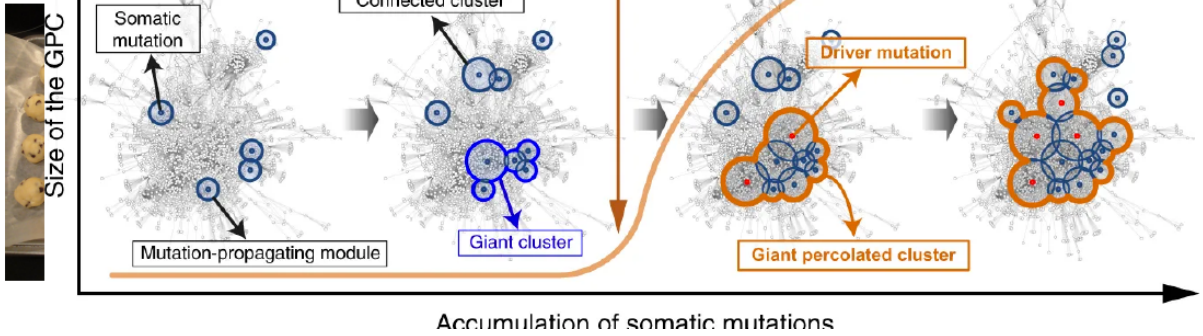
Introduction

- Consider a lattice (or graph) with binary options of occupied or not
- Percolation – Is there a cluster that spans the entire system?
- Question of the day – determine when there's always a percolating cluster



Examples in the world (!)

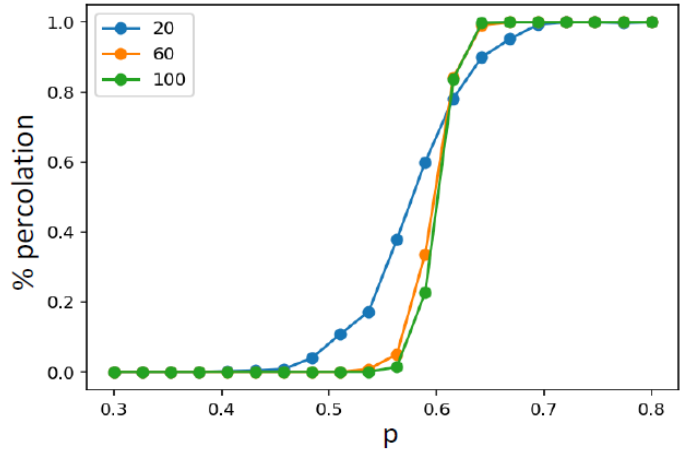
- Oil fields
- Forest Fires
- Spread of Diseases
- Wire Meshes
- Baking of cookies
- Cancer

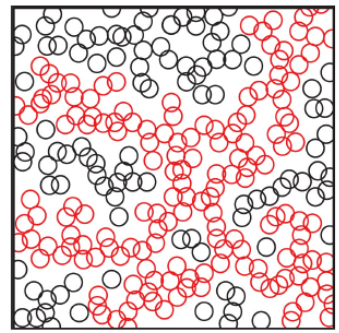


Gould and Tobochnik

Extra Cool Stuff ™

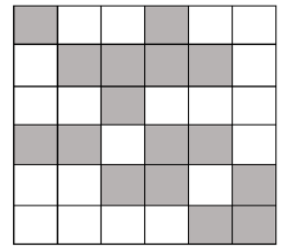
- (2nd order) Phase transition!
  - When you're at the critical point the clusters are scale invariant
  - Critical exponents are measurable
  - Universal behavior for other models
- Numerics needed to determine critical point for many lattices
- **Things we won't talk about:**
  - There's lots of non–numeric work (e.g. dual lattices)
  - Scaling collapse / RG / critical exponents
  - Mapping problems to percolation problems
  - Other percolations
    - Continuum
    - Bond*
    - Random Graphs





How to check for percolation?

Cluster Labeling

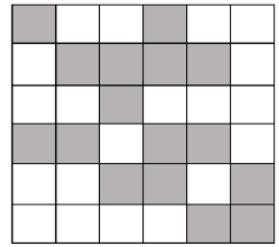


- Hoshen–Kopelman Algorithm (union/find)
  - Initialize
    - An array to hold labels for each site
    - Count for max label
    - Array/dict to convert between labels and proper labels
  - Algorithm:
    - For each site on the lattice:

    1. Check if any neighbors are labeled, if not assign new label (label=maxLabel++)
    2. Else, if all neighbors have the same label assign it to the current site
    3. Else, determine the proper label for the site
       a) If you aren't lazy, update all proper labels to point to the "root" (I am)

How to check for percolation?

Cluster Labeling

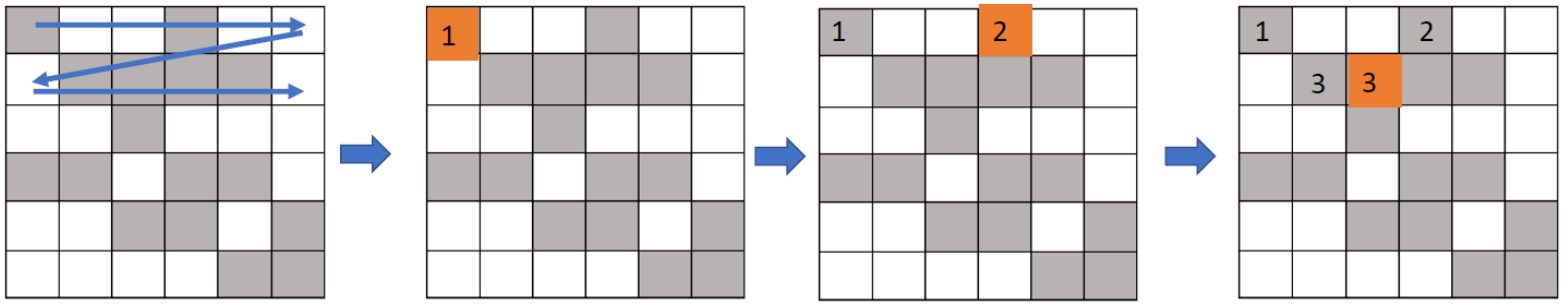


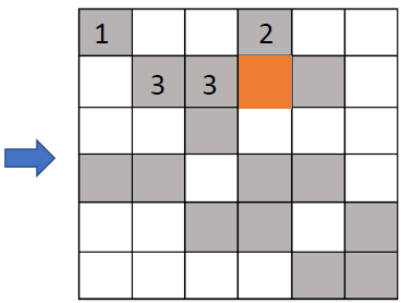
- Hoshen–Kopelman Algorithm (union/find)
  - Initialize
    - An array to hold labels for each site
    - Count for max label
    - Array/dict to convert between labels and proper labels
  - Algorithm:
    - For each site on the lattice:

    1. Check if any neighbors are labeled, if not assign new label (label=maxLabel++)
    2. Else, if all neighbors have the same label assign it to the current site - *Find*
    3. Else, determine the proper label for the site – *Union+Find*
       a) If you aren't lazy, update all proper labels to point to the "root" (I am)

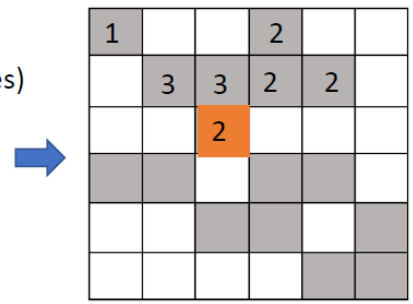
How to check for percolation?

Example - Labeling



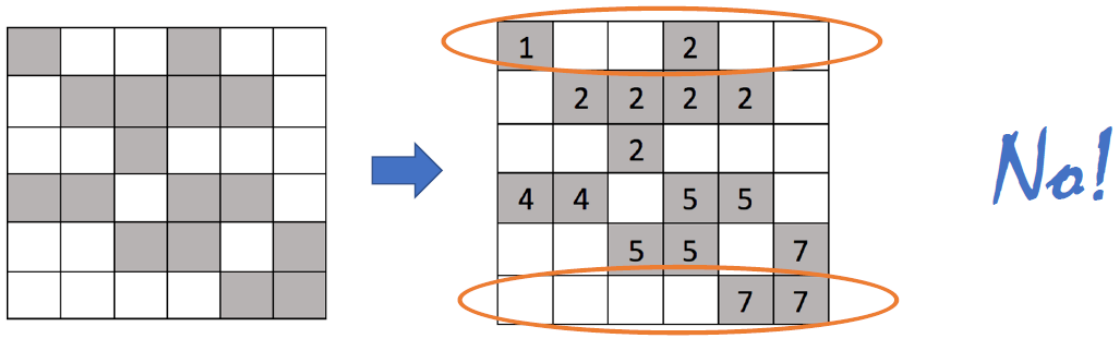


- neighborLabels= [2,3]
- Label = min(neighborLables)
  = 2
- properLabel[2] = 2
  properLabel[3] = 2

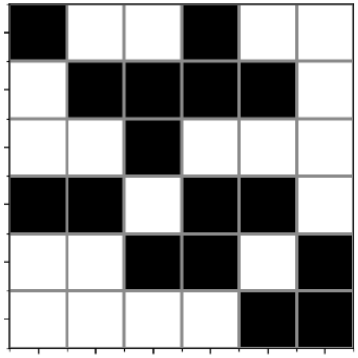


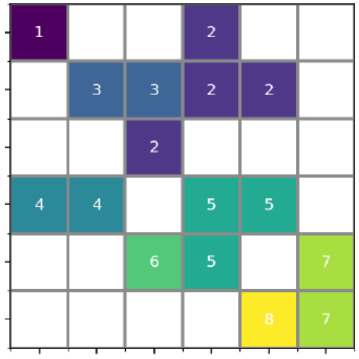
How to check for percolation?

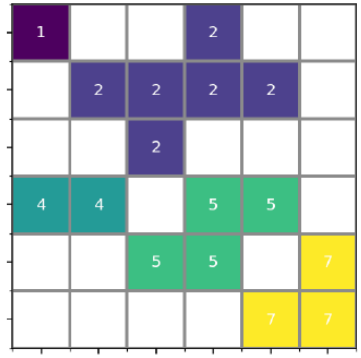
Example - Checking

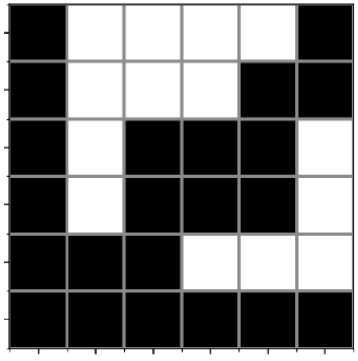


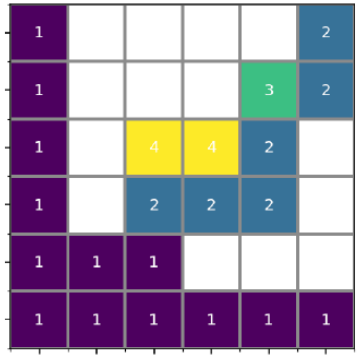
Code break!

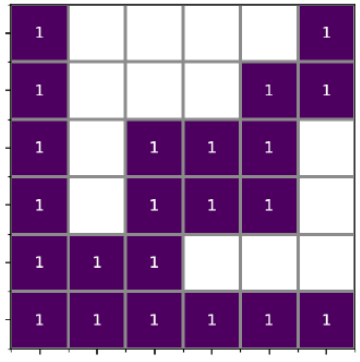



1 2
3 3 2 2
2
4 4 5 5
6 5 7
8 7


1 2
2 2 2 2
2
4 4 5 5
5 5 7
7 7



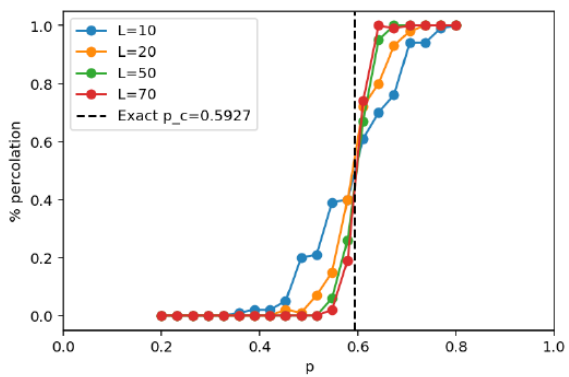

1 2
1 3 2
1 4 4 2
1 2 2 2
1 1 1
1 1 1 1 1 1


1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1
1 1 1 1 1 1

Percolation Transition

Naïve Approach

- For a given probability of site occupation, generate a random lattice
  - Determine if percolating
  - Repeat, record percentage of percolations
- Increase system size, use finite size scaling techniques to determine transition



Note: p_c has many more digits…

Percolation Transition

Naïve Approach

- Why is this not the best?
  - Imagine we have many labels pointing to another (cluster merges)
    - 10->6->4->2
    - Determining the true label takes a lot of time
    - (But this has an easy fix)
  - The current implementation, one lattice gives one sample for a given *p*, can we do better?

> The current fastest algorithm for percolation was published in 2000 by Mark Newman and Robert Ziff.[1]

A fast Monte Carlo algorithm for site or bond percolation

M. E. J. Newman[1] and R. M. Ziff[2]
[1] *Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501*
[2] *Michigan Center for Theoretical Physics and Department of Chemical Engineering, University of Michigan, Ann Arbor, MI 48109*

```c
void percolate()
{
  int i,j;
  int s1,s2;
  int r1,r2;
  int big=0;

  for (i=0; i<N; i++) ptr[i] = EMPTY;
  for (i=0; i<N; i++) {
    r1 = s1 = order[i];
    ptr[s1] = -1;
    for (j=0; j<4; j++) {
      s2 = nn[s1][j];
      if (ptr[s2]!=EMPTY) {
        r2 = findroot(s2);
        if (r2!=r1) {
          if (ptr[r1]>ptr[r2]) {
            ptr[r2] += ptr[r1];
            ptr[r1] = r2;
            r1 = r2;
          } else {
            ptr[r1] += ptr[r2];
            ptr[r2] = r1;
          }
          if (-ptr[r1]>big) big = -ptr[r1];
        }
      }
    }
    printf("%i %i\n",i+1,big);
  }
}
```

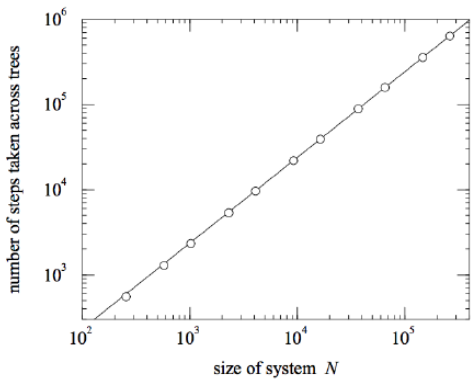


FIG. 5. Total number of steps taken through trees during a single run of the our algorithm as a function of system size. Each point is averaged over 1000 runs. Statistical errors are much smaller than the data points. The solid line is a straight-line fit to the last five points, and gives a slope of 1.006 ± 0.011.

algorithm	time in seconds	
depth-first search	4 500 000	
unweighted relabeling	16 000	
weighted relabeling	4.2	
tree-based algorithm	2.9	*0.545 s*

TABLE I. Time in seconds for a single run of each of the algorithms discussed in this paper, for bond percolation on a square lattice of 1000×1000 sites.

"(In actual fact, if one measures the performance of the algorithm in real ("wallclock") time, it will on most computers (circa 2001) not be precisely linear in system size because of the increasing incidence of cache misses as N becomes large. This however is a hardware failing, rather than a problem with the algorithm.)"

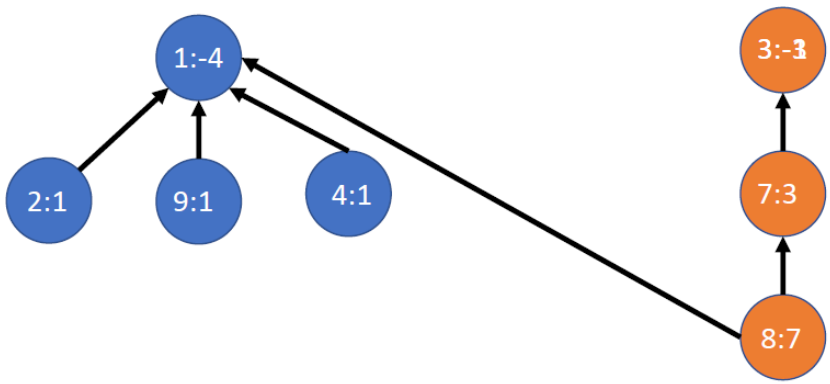
A fast Monte Carlo algorithm for site or bond percolation

M. E. J. Newman[1] and R. M. Ziff[2]
[1] *Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501*
[2] *Michigan Center for Theoretical Physics and Department of Chemical Engineering, University of Michigan, Ann Arbor, MI 48109*

- Find
  - Store labels as a tree; find reports the ultimate root of the tree
  - As paths are traversed, reduce depth as root is found (*path compression*)
    - Recursion is your friend here
  - Roots keep track of cluster size
- Union
  - When adding a cluster, add the smaller one as a subtree of the larger

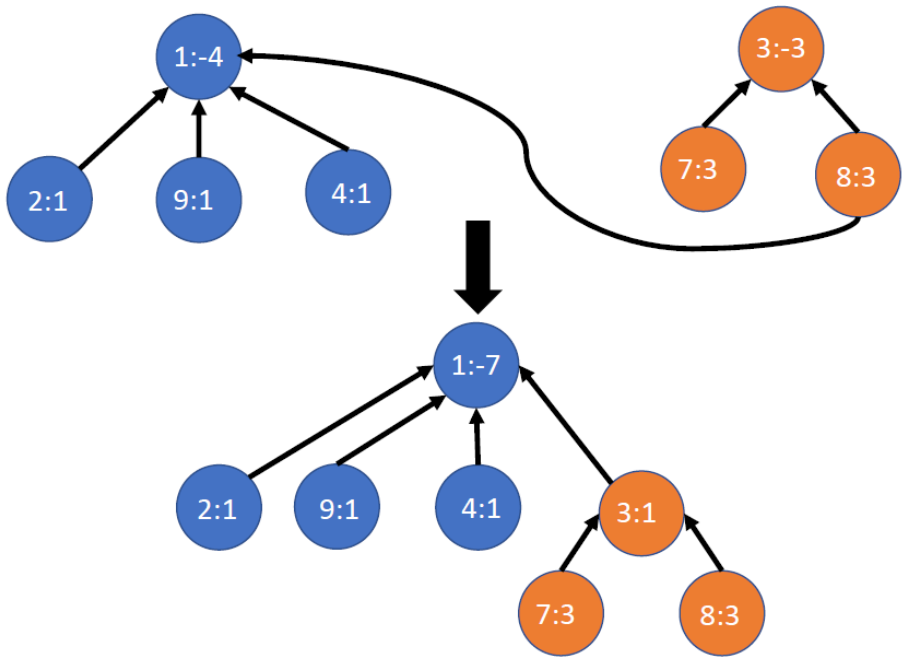
A fast Monte Carlo algorithm for site or bond percolation

M. E. J. Newman[1] and R. M. Ziff[2]
[1] *Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501*
[2] *Michigan Center for Theoretical Physics and Department of Chemical Engineering, University of Michigan, Ann Arbor, MI 48109*



A fast Monte Carlo algorithm for site or bond percolation

M. E. J. Newman[1] and R. M. Ziff[2]
[1] *Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501*
[2] *Michigan Center for Theoretical Physics and Department of Chemical Engineering, University of Michigan, Ann Arbor, MI 48109*



Path Compression

Subtree Union

A fast Monte Carlo algorithm for site or bond percolation

M. E. J. Newman[1] and R. M. Ziff[2]
[1] *Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501*
[2] *Michigan Center for Theoretical Physics and Department of Chemical Engineering, University of Michigan, Ann Arbor, MI 48109*

- Find
  - Store labels as a tree; find reports the ultimate root of the tree
  - As paths are traversed, reduce depth as root is found (*path compression*)
    - Recursion is your friend here
  - Roots keep track of cluster size
- Union
  - When adding a cluster, add the smaller one as a subtree of the larger

- Algorithm:
  1. Randomly occupy sites
  2. Occupied sites are clusters of size 1
  3. Go through each occupied site and check which clusters its connected to, updating the tree

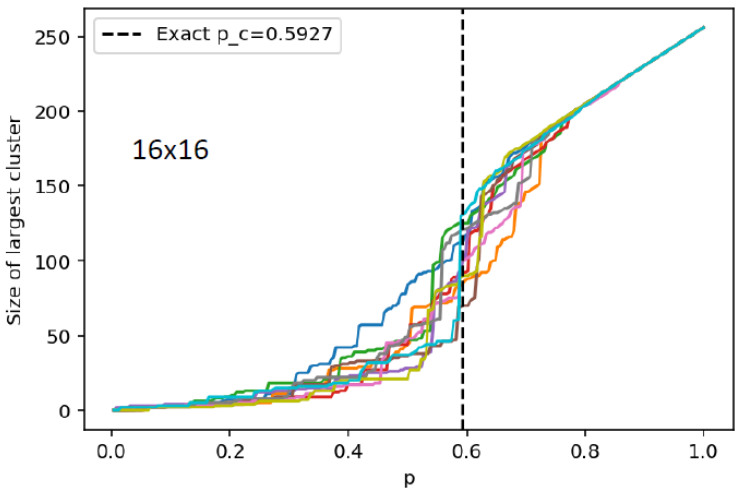
Newman-Ziff Algorithm

Subtle Improvement

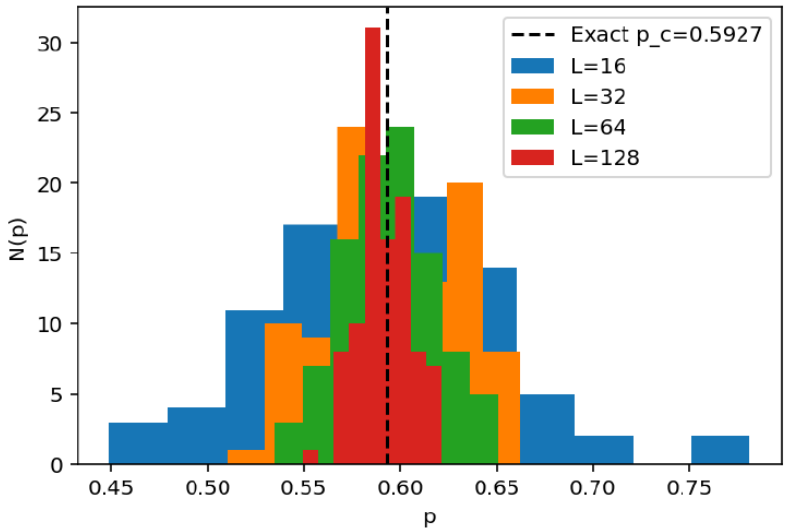
- Instead of randomly filling a lattice and determining a label, randomly order what sites will be occupied
  - As you label clusters, you also calculate percolation percentage for a smaller *p* on the way
  - If all you care about is percolation, stop once you find a percolating cluster (potentially saves time)
  - We keep track of the size of each cluster for free

Newman-Ziff Algorithm – Code Break!

Subtle Improvement



1000x1000 time: >>2.9 seconds in python



Took 400 “samples”

Thanks for listening!

Questions?