Percolation on Trees of Valence k

Qianfan Chen, Rachel Fuller, Charlie Gagnon

October 2020

Contents

0 Acknowledgements

We owe thanks especially to Tom Stone, our intrepid graduate student mentor for this project, and thanks also to the organizers of this inaugural Brown Undergraduate Mathematics Program, Sam Freedman and Sarah Griffith-Siqueira.

1 Problem Statement

> To model percolation on a graph Γ, we choose some probability p. Each edge will be "open" with probability p, and "closed" with probability $1 - p$. Once the open and closed edges have been determined, we can start at any vertex v and form the "cluster" Γ which consists of all vertices connected to v by using only open edges. The basic question is: what is the expected size of Γ? With what probability will it be finite (or infinite)? More generally, what is the "mass spectrum", that is, the probability distribution of sizes of clusters? How do all these properties vary with p?

Project Laboratory in Mathematics: Project descriptions
MIT Course 18.821
September 7, 2016

2 Overview

In [1], percolation is described as a global process, which operates on a graph to form a collection of subgraphs.

The problem statement is primarily interested in percolation applied to infinite graphs (graphs with an infinite number of vertices and edges). As a result, it is possible for the process to form a cluster of any size. Therefore, the expected size of a cluster is a sum of an *infinite* number of terms, where each term corresponds to a particular size. Cluster sizes can be measured in either their number of vertices or their number of open edges. **We measure cluster size as the number of vertices included in the cluster.**

Larger clusters are less likely to be formed, since the openness/closedness of each edge is independent of that of any other edge, and therefore the probability of finding each successive open edge within a given region of the graph is smaller than the probability of finding the previous open edge.

3 Simulating Percolation on Regular Trees of Valence k

This code is available at this GitHub repository. We modeled percolation on infinite regular trees of valence d (using code) in two distinct ways:

3.1 Estimating Expected Cluster Size by Partitioning, k=2

Our first approach (in Java) constructs a very large (but not infinite) initial graph, and models the $k = 2$ case, in which all vertices have two neighbors, except for the first and last vertex, which each have only one neighbor. Once the graph is constructed, we iterate over all edges and assign them to be open or closed. We then count the number of clusters and the sum of the sizes of the clusters (where the size of a cluster is the cluster's number of vertices). We take the sum of all cluster sizes divided by the number of clusters as the mean cluster size. That mean cluster size should be a decent estimate of the expected cluster size, as long as our initial graph is very large. To summarize:

$$\mathbb{E}[X_i] \approx \frac{\sum_{i=1}^{n} X_i}{n} \tag{1}$$

where each X_i is the average number of vertices in a particular graph. This is a good approximation if n is large and all X_i are independent and identically distributed.

3.1.1 Initial Results

For probability p = 0.5 on a regular tree of valence 2, we predicted an expectation of 3 vertices per cluster (see [4]). Over 100 trials of the simulation, the average expected cluster size was measured to be 1.46 vertices.

3.1.2 Interpretation and Adjustments

We considered two possible explanations for the discrepancy between our observed and expected result.

Initial Graph Size: The program seemed to consistently underestimate our predicted expected size. We assumed that increasing the size of the initial graph would improve the estimate, by making the finite simulation graph a better approximation of an infinite graph. However, increasing the size of the initial graph did not improve our estimate noticeably. We conclude that our initial graph size (50,000 vertices), is so large that slight increases to the initial graph size do not meaningfully affect the accuracy of the estimate of the expected cluster size. Certainly, initial graph size does not explain the discrepancy between our observed and expected result.

Clusters at Endpoints: We wondered whether the endpoints of the simulation graph (the simulated regular tree of valence 2 has 2 endpoints) affected our result, because any cluster that included an endpoint would be terminated “prematurely.” To better approximate the behavior of an infinite graph, we made the endpoints of our finite graph “fuzzy.” With this adjustment, clusters that include an endpoint of the graph can extend the graph by one vertex at a time (with probability p for each extension to take place). The cluster stops “growing” (there are no additional vertices added) after the first time that an extension ”request” is rejected (which for every request has probability $1 - p$ of occurring).

With fuzzy endpoints, we consistently observe an average cluster size of 2.0 vertices. Therefore, endpoint effects do not explain the discrepancy between our observed and expected result for the partitioning approach.

3.2 Estimating Expected Cluster Size by Growth from a Seed

Our second approach (in Python) models percolation as a local process. While the Java approach constructs multiple clusters within a finite graph of predetermined size (and then counts the size of each cluster to calculate an average) the Python approach ”builds” a single cluster from a ”seed” vertex, measures the size of the cluster once it is ”fully grown,” and repeats this process for a fixed number of trials.

The Java approach places a fixed (though fuzzy) size on the **total** number of vertices in the initialized graph, while the “growth-from-seed” approach places a fixed maximum on the number of vertices in *each* cluster (set by the recursion limit).

3.2.1 Results

For a regular tree of valence 2, ($k = 2$) and $p = 0.5$, the Python approach measures the average cluster size to be approximately 3.0 vertices. This is consistent with our predicted expectation of cluster size.

3.2.2 Interpretation

Since this process constructs clusters in layers (with each layer corresponding to a particular “level” of the Python script’s recursive calls), we can describe the expected size of clusters formed by this process as a geometric series (in terms of the number of layers in the cluster). The expectation of the size of clusters formed through this process is described in [4].

3.3 Comparing the Two Approaches

In the partitioning (Java) approach, in order for a particular cluster to "gain" a vertex, the neighboring cluster is prevented from containing that vertex (since if the neighboring cluster is connected to this border vertex, then the two clusters are actually parts of the same cluster). By contrast, in the Python approach, the structure of each cluster does not affect the structure of any other clusters, since the Python approach does not consider the "locations" of the clusters at all - clusters are not embedded on a shared tree.

3.3.1 A Brief Detour

The code structure of the Python approach makes it easy to modify the shape of the constructed clusters. The parameter `additional_main_branches` can be adjusted to create "spokes" of additional regular trees of valence k that all attach to the same root node. For p=0.5, $k = 2$, and `additional_main_branches = n`, the average cluster size is 3+n. This pattern suggests that additional "main branches" may be thought of as independent "opportunities" for vertices to grow from the root.

While adding "fuzzy endpoints" to the Java code, we initially wondered whether this pattern in the Python code may explain why the Java approach gives an expectation of 2.0. At first, we adjusted the Java code so that only the final vertex was fuzzy (with the root vertex not yet fuzzy). That model *felt* similar to the Python approach with $k = 2$ and `additional_main_branches = -1`, because in both cases, clusters can grow infinitely far from the root, but only in a single direction. (When `adm=0` and $k = 2$, clusters can grow infinitely far from the root in *two* directions.) Both of these models give an average cluster size of 2.0 vertices.

However, after the Java code was adjusted so that **both** graph endpoints were fuzzy (which would seem to visually correspond to the Python code with $k = 2$ and `adm=0`, the Java program still gave an average cluster size of 2.0 vertices, instead of our predicted result of 3.0 vertices. **This result suggests that the partitioning (Java) approach and growth-from-seed (Python) approach are fundamentally different.**

3.3.2 Explaining the Discrepancy

In the growth-from-seed approach, each trial measures the number of vertices in a graph grown from a root vertex. At the beginning of each trial, a minimum of k "coin flips" are performed, in order to determine whether each of the root's k edges are open or closed.

In the partitioning approach, **not every root is granted k independent "coin flips"**, because in this approach, each "trial" involves multiple clusters that share the same (partitioned) graph. As a result, the right-side neighbor of the first vertex shares an edge (and therefore shares a "coin flip") with the left-side neighbor of the second vertex. If that edge is closed, the first and second vertex are part of separate clusters, and **the size of the first vertex's cluster is no longer independent of the size of the second vertex's cluster**. By contrast, in the growth-from-seed approach, the size of **every** cluster is not influenced by that of all other clusters, because each cluster exists on its own graph.

Another way to interpret the difference is that, the partition approach is estimating the expected size of a cluster (where the expectation is taken over all possible clusters), while the growth-from-seed approach is estimating the expected size of the cluster to which a vertex belongs (where the expectation is taken over all possible vertices). At first glance, they seem to be calculating the same expectation, just with a different way of counting. However, there is an important difference: in the growth-from-seed method, a cluster of size t is counted t times, by each of its t vertices, while in the partition approach, any cluster is counted just once. As a consequence, the result by growth-from-seed method is larger than that by partition approach, as larger clusters are counted more times.

In this view, these two processes are fundamentally different (i.e. give different results) when applied to a finite graph. A major remaining question is whether, given an infinite graph to partition, the Java approach would also yield an average cluster size of 3.0 vertices in the $k = 2$ case. On an infinite graph, it seems reasonable that cluster sizes might behave as if determined independently: no matter where a closed edge would occur, there would be regions of the graph sufficiently far away to "escape" the effect of that partition.

3.3.3 Numerical Evidence of the Fundamental Difference

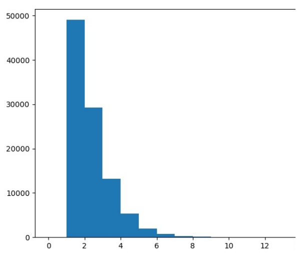
The following table summarizes findings that suggest that the Java approach (with **two** fuzzy endpoints) mimics the Python approach with `additional_main_branches` (adm) set to -1, which means that the "seed" (root node) has only $k - 1$ children, instead of k children. For the Java simulations, the expectations are the average over 100 runs on graphs with approximately 5000 vertices (this number is approximate due to fuzzy endpoints), for each value of p. For the Python simulations, the expectations are the average cluster size for 1.000,000 growth-from-seed trials for each value of p.

We have not developed a "visual" explanation of why the Java code behaves like the Python code with `adm` $= -1$.

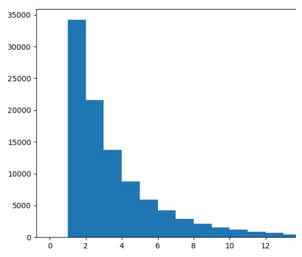
	Expected Cluster Size (number of vertices)	
p	Java, $k = 2$	Python, $k = 2, adm = -1$
0.1	1.1110585966611612	1.111508
0.2	1.2498430716611117	1.250301
0.3	1.4285463583578437	1.428301
0.4	1.6677872496040012	1.666358
0.5	2.000825151857754	1.999642
0.6	2.499215130408659	2.502326
0.7	3.336351644758741	3.33435
0.8	5.001444912246423	4.998291
0.9	10.020327691669797	9.97686

3.4 Visualization of Some Simulations

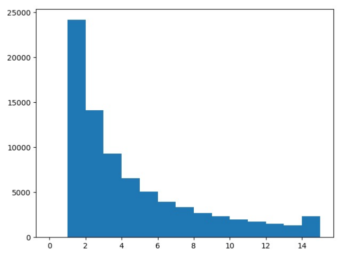
Here are the histograms from several simulations for a more intuitive presentation of the distributions across different P and different k's :



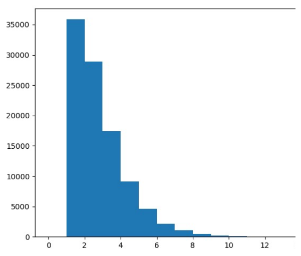
(a) k = 2, p = 0.3



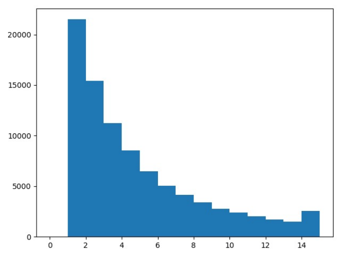
(b) k = 3, p = 0.3



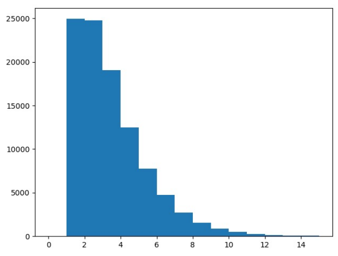
(c) k = 4, p = 0.3



(d) k = 2, p = 0.4



(e) k = 3, p = 0.4



(f) k = 2, p = 0.5

4 Expectation of Cluster Size

In this section, we calculate the expected result of the growth-from-seed approach, which is equivalent to the expectation of the size of a cluster "grown" from a single root vertex.

We mention in [3.3.2] that the growth-from seed method estimates the expectation of the size of the cluster to which a vertex belongs. This is an equivalent description of the same probability, since in both descriptions, a vertex is fixed, and we consider the expected size of cluster containing that vertex.

To calculate the size of a cluster, let the layer 0 be the set containing just the root vertex, layer 1 be the set containing the children of the root, layer 2 the set containing the grandchildren of the root, etc. In general, layer i, (denoted as L_i) is defined to be the set containing the vertices that are i edges away from the root.
Then, the size the cluster is

$$S = \sum_{i=0}^{\infty} |L_i|$$

Then,

$$\mathbb{E}[S] = \mathbb{E}[\sum_{i=0}^{\infty} |L_i|] = \sum_{i=0}^{\infty} \mathbb{E}[|L_i|]$$

For every layer i vertex in this tree, the expected number of its children (in layer $i+1$) is

$$\begin{cases} k \cdot p & i = 0 \\ (k-1) \cdot p & i > 0 \end{cases}$$

Therefore, we derive the recursive relation

$$\mathbb{E}[|L_{i+1}| \mid L_i] = \begin{cases} k \cdot p \cdot |L_i| & i = 0 \\ (k-1) \cdot p \cdot |L_i| & i > 0 \end{cases}$$

Thus, by rule of total probability,

$$\mathbb{E}[|L_{i+1}|] = \begin{cases} k \cdot p \cdot \mathbb{E}[|L_i|] & i = 0 \\ (k-1) \cdot p \cdot \mathbb{E}[|L_i|] & i > 0 \end{cases}$$

Therefore,

$$\mathbb{E}[|L_i|] = \begin{cases} 1 & i = 0 \\ kp & i = 1 \\ kp((k-1)p)^{i-1} & i > 1 \end{cases}$$

The expectation of S is finite if and only if $(k-1)p < 1$.
In that case,

$$\mathbb{E}[S] = 1 + \frac{kp}{1-(k-1)p}$$

5 Distribution of Cluster Sizes

We tried to find the probability distribution of such percolation. That is, to find the probability of having exactly n vertices grown from a "seed" in a given $k-$valent tree. The probability distribution P_n proposed in this section are under the interpretation of the growth-from-seed approach.

We tried random simulation, and the result shows that this distribution converges slower than the geometric sequence (as n increases), which is what we expected at first. Then we tried to derive the formula by hand.

5.1 Probability of a Particular Tree

Lemma 1 *On any of model of trees of valence k, the probability of getting a particular tree with n vertices is*

$$P_{particular\ shape} = p^{n-1}(1-p)^{k+(n-1)(k-2)}$$

The reason is as follows:
Let's use the phrase closed edges to refer to the edges that connects an included vertex with an excluded node, and use open edge to refer to the edges that connect two included vertices. When we just have the root (1 node), we have k closed edges and 0 open edges. Each time we include a node, we include 1 more open edge and $k-2$ more closed edges. Therefore, when we have N vertices in total, we would have $k+(n-1)(k-2)$ closed edges, and $n-1$ open edges. **Therefore, the probability of obtaining an n-vertex tree that has a *particular* assembly of open and closed edges is exactly the probability of closing those $k+(n-1)(k-2)$ edges, and opening those $n-1$ edges, which is**

$$P_{\text{particular shape}} = p^{n-1}(1-p)^{k+(n-1)(k-2)}$$

5.2 Counting the Number of Distinct Trees with n Vertices

To calculate the probability of getting *some* tree with n vertices, we just need to count the number of trees with n vertices.
When $k = 2$, this is fairly easy, as there are n different trees with n vertices, starting from a certain root (think of sliding a segment along the axis, always covering the root). Thus, for $k = 2$, the distribution is just

$$P_n = np^{n-1}(1-p)^2$$

For $k \geq 2$, however, this becomes significantly more complicated. We write a recurrence relation using the notation $C_k(n)$ to represent the number of trees with n vertices in a model of trees where each vertex has d children (thus, our original model of valence $k+1$ consists of $k+1$ such trees and a shared root), and use $S_k(n)$ to denote the number of trees in a model of k-valent trees. For a tree T with t nodes, we write $t-1$ as a sum of non-negative $x_1, \ldots, x_{k-1}$, which represents a way of dividing the $t-1$ vertices (t vertices minus the root) into the subtrees, whose roots are children of the root of T. Then, we could use the recurrence relation to calculate the number of possible trees with x_i nodes, which is the number of ways to form each of those subtrees. Multiply the number of ways to form each subtree, and we get the number of possible trees represented by a specific summation. Then we consider all the possible ways to write $t-1$ as a sum of non-negative $x_1, \ldots, x_{k-1}$, and we sum up all the results to get $C_k(t)$. So we have

$$C_k(1) = C_k(0) = 1$$

$$C_k(t) = \sum_{x_1+\ldots x_{k-1}=t-1, x_i \in \mathbb{N}} \prod_{i=1}^{k-1} C_k(x_i)$$

$$S_k(t) = \sum_{x_1+\ldots x_k=t-1, x_i \in \mathbb{N}} \prod_{i=1}^{k} C_k(x_i)$$

The calculation of $S_k(t)$ is different from that of $C_k(t)$ only in that the "seed" has k, not $k-1$.

5.3 Getting a Distribution of Cluster Sizes, Given n

Plugging $P_{\text{particular shape}}$ [5.1] into the formula for $S_k(n)$ [5.2], we get that

Proposition 1 *The probability of observing a growth-from-seed cluster with n vertices (on a tree of valence k and probability of an included edge p) is*

$$P_n = S_k(n)p^{n-1}(1-p)^{k+(n-1)(k-2)}$$