# Percolation of Matérn II Point Processes on a Random Graph

Ryan Honea, Matthew Jones

April 27, 2017

**Abstract**

Consider a graph on $n$ order Poisson distributed points in a $2 \times 2$ square. I remove any point denoted $x_n$ that is within distance $a$ from $x_{n+1}$ such that no point is with a set distance from another. Each point is connected edge-wise with probability $p$ if the distance between points is less than $r$. After setting a point at the origin, I attempt to find the probability that a point will be connected edgewise on some path to another point outside a unit circle inscribed within the $2 \times 2$ square. With a high enough Poisson intensity in the square, I use this to approximate the likelihood $\Phi$ of an infinite cluster forming with parameters $p$, $r$, and $a$. After simulating these results in C++ and analysis of data with R through logistic regression, $\Phi(p, r, a) = -11.94 + .016p_c + 1.657r_c + 2.657a_c + .962p_cr_c - .013p_ca_c - 1.58r_ca_c - .217p_cr_ca_c$ (where $p_c$, $r_c$, and $a_c$ are coded variables) is found. With a decision boundary of $\Phi \geq .5$, our function is accurate with 96-98% accuracy. Not only that, but barring the presence of a mathematical proof, it appears that a critical point in which $\Phi$ changes dramatically for $a$ is $\frac{r}{2}$ which is of strong interest.

## 1 Introduction

Random graphs are random ensembles of vertices and edges, and are often used to model ad-hoc wireless networks, spread of disease in populations, optimal travel routes, and many other natural or man- made systems. The study of graphs often entails rules being applied such as the probability of edge-wise connections between vertices, the application of different edge weights, or different distributions of points.

The classical geometric random graph is constructed on a set of randomly placed vertices in a metric space by drawing an edge between any two vertices in the graph, independently, if and only if their nodes are within a certain distance apart.

A Matérn II point process (Figure 1) is a random ensemble of points constructed by first generating a Poisson process in the plane, then ordering the vertices according to their arrival times, and deleting earlier vertices that exist in a set proximity to later vertices [2]. The result is a random set of points that aren't too close to each other. These models are attractive because they model random locations aspects of many systems while acknowledging that physical objects cannot occupy the same space.

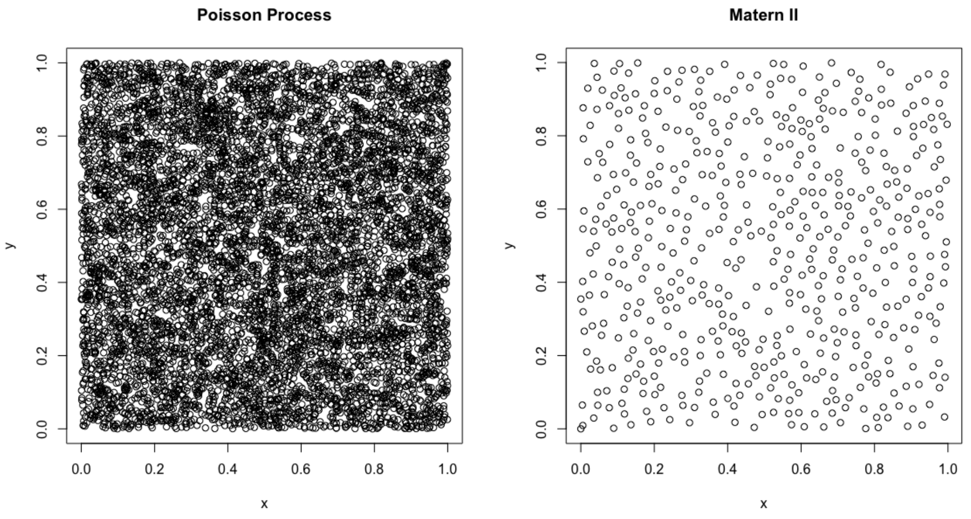

Figure 1: Matern II Thinning Example

Percolation theory was invented by Broadbent and Hammersly [1] to try to answer the question of whether or not the center of a porous stone would get wet when the stone is submerged in water. A classical model for this is the integer lattice with each edge removed independently with some probability. The origin is in a percolating cluster if it belongs to an infinitely connected cluster. The central problem is to identify the probability of a percolating cluster at the origin as a function of your probability. Often, these structures exhibit a phase transition, meaning the percolation probability is equal to 0 for $p < p_c$ where $p_c$ is some critical value of your probability where the likelihood of a percolating cluster starts to increase.

## 2 Restatement of the Problem

I will be studying random graphs constructed on Matérn II point processes by connecting points by an edge with probability $p$, provided the points are within a certain maximum radius $r$. This is similar to Penrose's Model [3E] where the connection probability $p$ is constant if the points are within $r$, and 0 otherwise, except that the point process governing the locations of the certices in the model is not Poisson (neighboring points cannot be too close together). This subtle distinction makes the already difficult mathematical analysis even more so. I will study the effects of the Mat'ern II deletion radius $a$ which governs the allowed proximity of points (or half the radius of a hard sphere), the maximum connection radius $r > a$, and connection probability $p$ on the overall probability of the presence of a percolating cluster $\Phi$. That is, by varying these

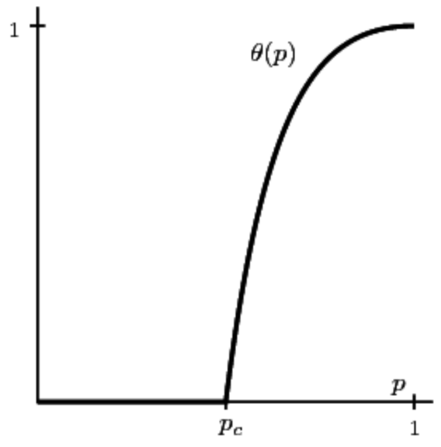

Figure 2: Example of a Phase Shift

parameters and running simulations, I aim to approximate the chance that there exists an infinite cluster that touches the origin.

## 3 Assumptions and Variables Used

I will generalize *infinity*, or the infinite cluster, by generating these Matérn II points across a square that is approximately $2 \times 2$ but extended for higher $r$. If a series of vertices are connected edgewise from the origin to outside the unit circle, I will say that it is a percolating cluster. I can assume this for small $r$ and high $\lambda$ (intensity of the Poisson process). I assume that if the origin will percolate this far, it will likely percolate even further.

I also assume for now that the data can be analyzed linearly. Granted, this is a dangerous assumption, but for the purposes of the initial model, the results should help in developing future models.

The variables below are in the order of their appearance.

- $p =$ probability of edge-wise connection given distance $d$ between points is less than or equal to $r$.
- $d =$ distance between two points.
- $r =$ radius required for edge-wise connection
- $a =$ Matérn II thinning radius or the allowed proximity of points
- $\Phi =$ the probability that a percolating cluster exists.
- $\lambda =$ intensity of the Poisson point distribution
- $\Delta x_\mu =$ the expected average distance between any two points.
- $n =$ the initial number of points.
- $n_{II} =$ the number of points after Matérn II thinning has occurred.
- $p_c =$ the coded value of the probability
- $r_c =$ the coded value of the radius
- $a_c =$ the coded value of the deletion radius
-

## 4 Formulating the Model

I will develop the simulation of this process in C++ and analyze it **R**. Before determining which factor levels to check for the three explanatory variables, that is $p$, $r$, and $a$, I need to check some important characters of the data. Namely, I need to k now what the average distance between any two neighboring points will be given that the points are uniformly distributed across a $2 \times 2$ graph. If $\lambda = 7500$, then $\sqrt{7500}$ should be approximately the number of points spanning a line across the graph, and therefore the average horizontal and vertical distance will be $\frac{2}{\sqrt{7500}}$. The average diagonal distance is $\sqrt{2(\frac{2}{\sqrt{7500}})^2}$. Therefore,

$$\Delta x_\mu = \frac{\frac{2}{\sqrt{7500}} + \sqrt{2(\frac{2}{\sqrt{7500})^2}}}{2} = .027877$$

This can be easily modified for different $\lambda$. I find this because there isn't much value to check for $r < \Delta x_\mu$, and indeed, for $a$ to be analyzed, its minimum value should be at least $\frac{2}{10}\Delta x_\mu$ as it is within reason that two points can be within that proximity. So, I will set the minimum $r$ value to be $\Delta x_\mu$ and vary $a$ with four levels: $\frac{1}{5}r$, $\frac{2}{5}r$, $\frac{3}{5}r$, and $\frac{4}{5}r$. The deletion radius $a$ and Poisson Intensity $\lambda$ will both be functions of $r$ so that the concentration of points remain similar across factor level changes. Based on this analysis, I will use the following parameters to estimate $\Phi$. Table 1 shows the variables' values with their coded variable equivalent.

| Coded Values | Connection Probability ($p$) | Connection Radius ($r$) | Deletion Radius($a$) |
|---|---|---|---|
| 1 | 1/11 | $\Delta x_\mu$ | $(1/5)r$ |
| 2 | 2/11 | $(6/5)\Delta x_\mu$ | $(2/5)r$ |
| 3 | 3/11 | $(7/5)\Delta x_\mu$ | $(3/5)r$ |
| 4 | 4/11 | $(8/5)\Delta x_\mu$ | $(4/5)r$ |
| 5 | 5/11 | $(9/5)\Delta x_\mu$ | - |
| 6 | 6/11 | $(10/5)\Delta x_\mu$ | - |
| 7 | 7/11 | $(11/5)\Delta x_\mu$ | - |
| 8 | 8/11 | $(12/5)\Delta x_\mu$ | - |
| 9 | 9/11 | $(13/5)\Delta x_\mu$ | - |
| 10 | 10/11 | $(14/5)\Delta x_\mu$ | - |

Table 1: Factor Levels and Values

In order to construct a model with reasonable accuracy., I will simulate using C++ (see Appendix) each of the factor levels 2500 times giving a total of 1,000,000 different data points. I will then plot each variable against the probability that a cluster appeared (i.e. number of appearances over total number of replicates on factor level). I will also create interaction plots in order to determine the necessity of interaction variables.

Following the results of a visual analysis, I will construct a logistic regression on the parameters and their interactions.

### 4.1 Algorithmic Analysis of C++ Code

The primary operations in the algorithm are the comparison of points for the Matérn II thinning process, the comparing of points in the Adjacency Matrix construction, and the graph traversal. In the case of comparing points in the thinning process (i.e. Distance(point1, point2) $< a$), the algorithm operates on $i$ from 1 to $n-1$ with a nested $j$ from $i+1$ to $n$. The number of computations then is $\frac{n(n+1)}{2}$. After this, we express the number of points remaining as $n_{II}$. In allocating the adjacency matrix, there are two operations: distance(point1, point2) $< r$ and (random number between 0 and 1) $< p$. The loop is the same as the thinning but with $n_{II}$ points. So $2\frac{n_{II}(n_{II}+1)}{2}$. Finally, note that the average graph traversal time is $n$. Our algorithm's operations then are

$$\frac{n(n+1)}{2} + n_{II}(n_{II}+1) + n_{II} \in \Theta(n^2)$$

For further analysis of the C++ code, see Appendix.

## 5 Results

### 5.1 Visual Analysis of Parameters

Following the simulation of results, the graphs in Figure 3, 4, and 5 were constructed.

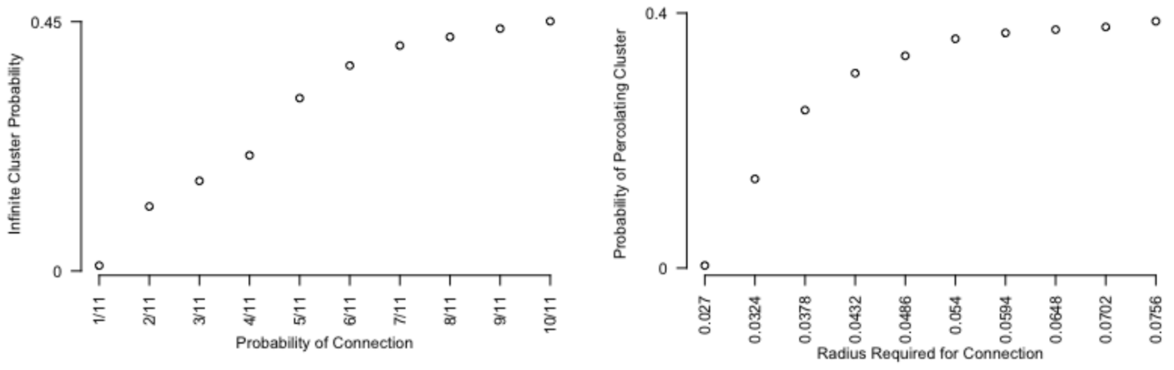

Figure 3: Results of Simulation for different levels of $p$ and $r$

In the cases of Figure 3, it initially appears as if the probability has a somewhat linear relationship while the connection radius is a part of a polynomial family of curves.

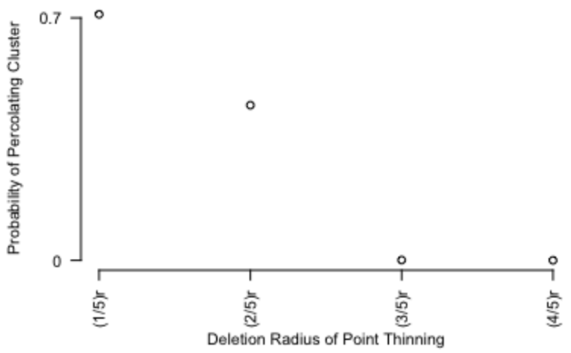

Figure 4: Results of Simulation for different levels of $a$

The plot of the deletion radius in Figure 4 suggests that there is likely a phase shift in the range $(\frac{2}{5}r, \frac{3}{5}r)$. With the presence of a phase shift, it is incredibly likely that there is a strong interaction with the other variables. This is made immediately clear by the appearance of the graphs in Figure 5.

From a visual perspective, it appears as if the probability of connection is almost linear when viewed independent of the other variables. The connection radius appears to be from a polynomial family of curves, and the deletion radius definitely seems to exhibit a phase shift. The interaction graphs provide significantly more information in regards to the connection probability. When accounting for multiple levels of $a$, it also has a polynomial curve and in the case of $a = 2$, a visible phase shift. The cause of this is made more apparent by Figure 6.

With the strength of $a$'s interaction, the interaction terms will need to be included in the logistic model results.

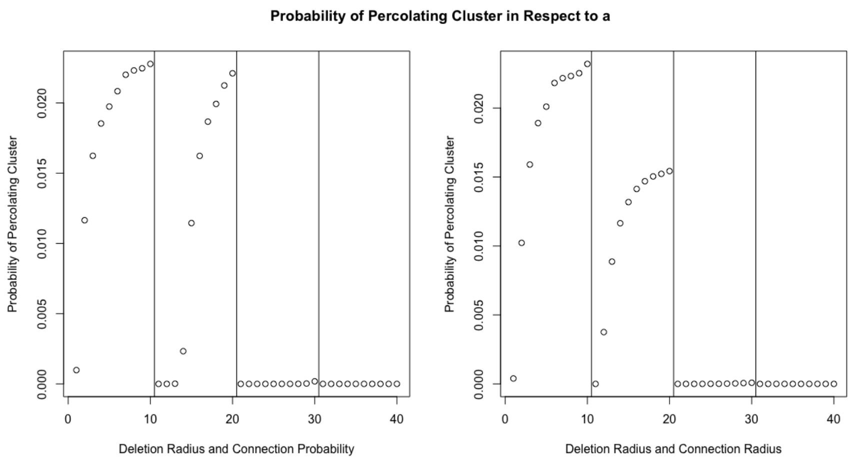

Figure 5: Results of Simulation for different levels of $p$ and $r$ in respect to $a$

## 5.2 Logistic Model of $\Phi$

The full logistic model of $\Phi(p, r, a)$ revealed that $p$ and surprisingly the interaction $p : a$ were not significant. Following the removal was a slightly better model with equivalent accuracy as shown in Table 2.

| Model for $\Phi(p, r, a)$ | | |
|---|---|---|
| Coefficient | Full Model | Updated Model |
| intercept | -11.94349 | -11.822189 |
| $p_c$ | 0.016554 | |
| $r_c$ | 1.657618 | 1.640490 |
| $a_c$ | 2.657233 | 2.562077 |
| $p_c : a_c$ | 0.961634 | |
| $p_c : r_c$ | -0.013140 | 0.963367 |
| $a_c : r_c$ | -1.580816 | -1.566532 |
| $p_c : r_c : a_c$ | -0.217565 | -0.219323 |
| *Accuracy* | 97.91% | 97.91% |

Table 2: Model Results

Accuracy was determined by using the different factor levels in the test dataset and predicting their outcome with the coefficients of the logistic model. Because these values will come back as a percentage between 0 and 1, the decision boundary $\mathbb{1}\{\Phi > .5\}$ is used. These predictions are then compared with the actual results of the testing data set. With 97.91% of those prediction matching, and a confidence interval at 95% of (97.86, 97.97), the model is fairly strong in its predictive abilities.

# 6 Conclusion

A final improvement on the model in Table 2 is to account for the phase shift that appears to be occurring in $a$. The final equation developed then is

$$\Phi(p_c, r_c, a_c) = -11.8222 + 1.6405r_c + 2.5621a_c + 0.9634p_cr_c - 1.5665a_cr_c - 0.1293p_cr_ca_c\mathbb{1}\{a < 1/2\}$$

While this improved model only provides a marginal increase in the accuracy of the model, its assurance of accounting for the phase shift is valuable in analyzing any other phase shifts.

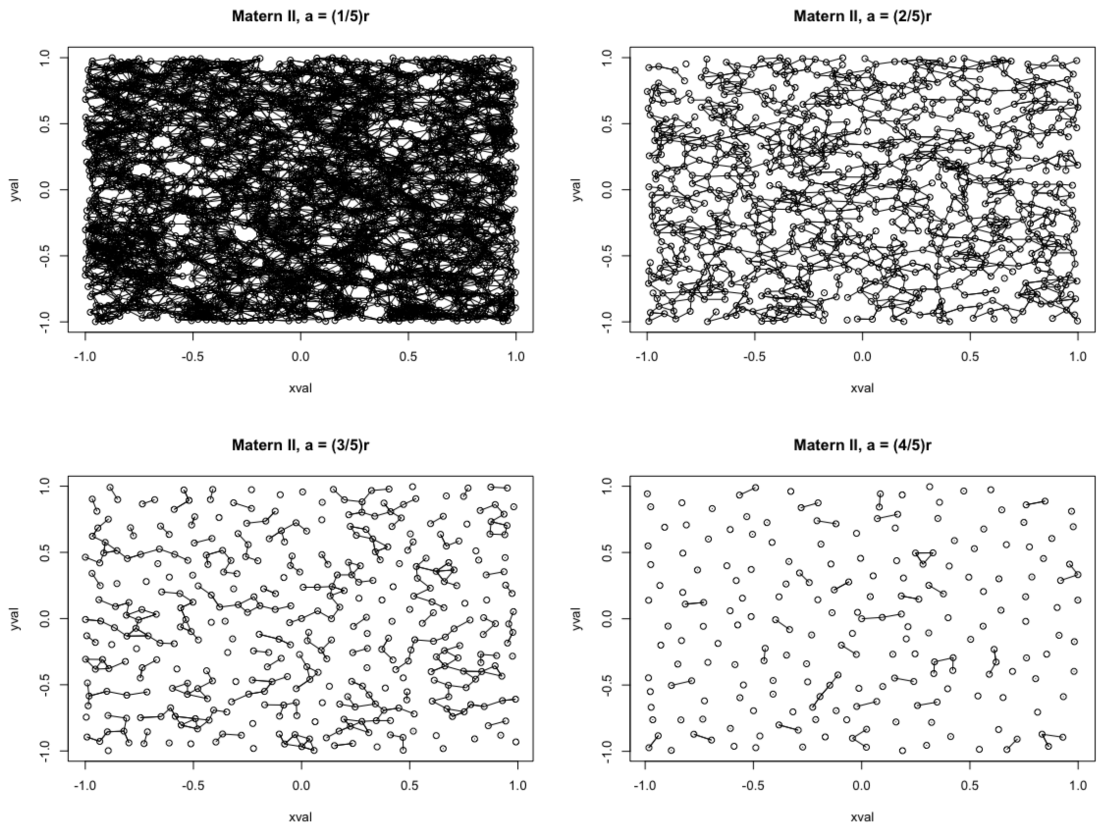

Figure 6: Visual Appearance of a Matérn II Graph for varying deletion radii at $p = .5$ and $r = .1$

## 7 References

1. Broadbent, S.; Hammersly, J. (1957), Percolation processes I. Crystals and mazes, Proceedings of the Cambridge Philosophical Society 53: 629641

2. Mat ern, B.; Spatial Variation

3. Penrose, M. Connectivity of soft random geometric graphs. The Annals of Applied Probability. 2016, 2, 968-1028

# Appendix

## 7.1 C++ Code Used

### 7.1.1 Simulation Driver

```cpp
#include <iostream>
#include <time.h>
#include <iomanip>
#include <fstream>
#include <random>
#include <omp.h>
#include "Perc_Graph.h"

using namespace std;

void Run(int);

int main() {

  int timer1, timer2, finaltime, i, factors, reps;
  //  double r, a, p;
  /*
  cout << setw(35) << left << "Enter Connection Radius" << ": ";
  cin >> r;
  cout << setw(35) << left << "Enter Matern II Thinning Radius" << ": ";
  cin >> a;
  cout << setw(35) << left << "Enter Connection Probability" << ": ";
  cin >> p; */
  //cout << setw(35) << left << "Enter number of factors" << ": ";
  //cin >> factors;
  cout << setw(35) << left << "Enter number of replicates" << ": ";
  cin >> reps;

  timer1 = time(0);

  Run(reps);

  timer2 = time(0);
  finaltime = timer2 - timer1;
  double simulationNum = reps;
  cout << "Simulation of " << simulationNum << " data points completed in ";
  cout << finaltime / 60 << " minutes and " << finaltime % 60 << " seconds." << endl;
  cout << "Average time per simulation: " << double(finaltime) / double(simulationNum) << " seconds." << endl;
}

void Run(int reps) {
  std::default_random_engine generator(time(NULL));
  ofstream datafile;
  double p,r,a;
  int runs = 0;
  datafile.open("data2.txt");
  datafile << "p r a infclus\n";
  //bool clusterPres = false;
#pragma omp parallel for
  for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 5; j++) {
    for (int k = 0; k < 9; k++) {
        for (int l = 0; l < reps; l++) {
        //  p = (i+1)*(1.0/11);  so it runs from 1/11 to 10/11
        p = (i+1)*.333333;
      r = .02702 + j*(2.0/5)*.02702; // so it runs from the avg dist to 3*avg dist
      a = ((k+16.0)/40)*r; // so it runs from 2/5 r to 3/5 r

      Graph* percolation = new Graph;
      percolation->createPoints(r, generator);
      percolation->thinPoints(a);
      percolation->createAdjMatrix(r, p);
      bool clusterPres = percolation->hasInfiniteCluster();
      if (clusterPres) {
          cout << "Simulation " << runs + 1 << " Complete. " << "Cluster Exists." << endl << flush;
          runs++;
      }
      else {
          cout << "Simulation " << runs + 1 << " Complete. " << "No cluster.    " << endl << flush;
          runs++;
      }
      percolation->reset();
#pragma omp critical
      datafile << i+1 << " " << j+1 << " " << k+1 << " " << clusterPres << endl << flush;
      delete percolation;
        }
    }
      }
  }
        datafile.close();
}
```

driver.cpp

### 7.1.2 Graph Class

```cpp
#include "Perc_Graph.h"
#include "Perc_Point.h"
#include <random>
#include <iostream>
#include <time.h>
```

```
Graph::Graph() {
  length_ = 0;
  visited_.push_back(0);
  infCluster_ = false;
}

void Graph::createPoints(double r, std::default_random_engine generator) {

  double min = -1.0 - r;
  double max = 1.0 + r;
  double randn;
  std::poisson_distribution<int> distribution(7500*((2.0+2.0*r)*(2.0+2.0*r)/4.0));

  // distributes poisson points of lambda intensity

  lambdaIntensity_ = distribution(generator);
  points_ = new Point[lambdaIntensity_];

  for (int i = 0; i < lambdaIntensity_; i++) {
    randn = (((double)rand()*(max - min) / RAND_MAX) + min);
    points_[i].setX(randn);
    randn = (((double)rand()*(max - min) / RAND_MAX) + min);
    points_[i].setY(randn);
  }
}


/* This method does a number of things. The first thing it does is create
a Point array of max size lambdaIntensity with the first point being initialized
as the origin of the graph. Following that, it checks the original array and any
point that is within a deletion radius is set as (0,0). After that check, any point
that is not (0,0) is transferred into the new array. Following that, our points_
array will be set to the new one.*/
void Graph::thinPoints(double a) {
  Point* temparray;
  temparray = new Point[(lambdaIntensity_+1)];

  // setting first point of thinned array at the origin
  temparray[0].setX(0.0);
  temparray[0].setY(0.0);
  length_ = 1;

  for (int i = 0; i < (lambdaIntensity_-1); i++) {
    for (int j = (i + 1); j < lambdaIntensity_; j++) {
      if (distance_(points_[i], points_[j]) <= a) {
        points_[i].setX(0.0);
        points_[i].setY(0.0);
      }
    }
  }

  for (int i = 0; i < lambdaIntensity_; i++) {
    if (!(points_[i].getX() == 0.0 && points_[i].getY() == 0.0)) {
      temparray[length_].setX(points_[i].getX());
      temparray[length_].setY(points_[i].getY());
      temparray[length_].checkOutside();
      length_++;
    }
  }

  delete points_;

  points_ = new Point[length_];
  for (int i = 0; i < length_; i++) {
    points_[i] = temparray[i];
  }


  delete temparray;
}

void Graph::createAdjMatrix(double r, double p) {
  adjMatrix_ = new bool[length_ * length_];
  // So any reference to the array will be
  // in the form i*length_+j]

  // initializes diagonal as 0s
  for (int i = 0; i < length_; i++) {
    adjMatrix_[i*length_ + i] = false;
  }

  // constructs remainder of adjacency matrix
  for (int i = 0; i < (length_ - 1); i++) {
    for (int j = (i + 1); j < length_; j++) {
      if ((distance_(points_[i], points_[j]) <= r) && (((double)rand() / RAND_MAX) <= p)) {
        adjMatrix_[i*length_ + j] = true;
        adjMatrix_[j*length_ + i] = true;
      }
      else {
        adjMatrix_[i*length_ + j] = false;
        adjMatrix_[j*length_ + i] = false;
      }
    }
  }
}


bool Graph::hasInfiniteCluster() {

  hasEdge(points_[0], 0);

  return infCluster_;
}

void Graph::reset() {
  visited_.clear();
  visited_.push_back(0);
  delete [] points_;
  delete [] adjMatrix_;
  int length_ = 0;
  infCluster_ = false;
}
```

Perc_Graph.cpp

```
#ifndef Perc_Graph_h
#define Perc_Graph_h

#include<vector>
#include<iostream>
#include "Perc_Point.h"
#include <random>
#include <cmath>
#include <vector>

class Graph {
private:
  Point* points_;
  bool* adjMatrix_;
  int lambdaIntensity_;
  int length_;
  bool infCluster_;
  std::vector<int> visited_;


  double distance_(Point p1, Point p2) {
    double xdist, ydist, dist;
    xdist = (p1.getX() - p2.getX())*(p1.getX() - p2.getX());
    ydist = (p1.getY() - p2.getY())*(p1.getY() - p2.getY());
    dist = sqrt(xdist + ydist);
    return dist;
  }


  bool alreadyVisited(int val) {
    for (int i = 0; i < visited_.size(); i++) {
      if (visited_[i] == val) {
        return true;
      }
    }
    return false;
  }


  bool hasEdge(Point p1, int row) {
    int edges = 0;
    if (infCluster_) { return true; }
    for (int i = 0; i < length_; i++) {
      if (adjMatrix_[row*length_ + i]) {
        edges++;
        //std::cout << "Node " << row << " had edge with " << i << std::endl;
      }
    }
    for (int i = 0; i < edges; i++) {
      for (int i = 1; i < length_; i++) {
        if (alreadyVisited(i) == false) { //checks to make sure no back tracking
          //std::cout << adjMatrix_[row*length_ + i] << std::endl;

            if (adjMatrix_[row*length_ + i]) {
              visited_.push_back(i);
              //std::cout << "Point " << row << " has edge with " << i;
              //std::cout << " at location " << points_[i].getX() << " " << points_[i].getY();
              //std::cout << " which is " << distance_(points_[i], points_[0]) << " from origin." << std::endl;

              if (distance_(points_[i], points_[0]) >= 1.0) {
                infCluster_ = true;
                return true;
              }
              else if (infCluster_) { return true; }
              else { hasEdge(points_[i], i); }
            }
        }
      }
    }

    return false;
  }

public:
  Graph();
  void createPoints(double, std::default_random_engine);
  void thinPoints(double);
  void createAdjMatrix(double, double);
  bool hasInfiniteCluster();
  void reset();
};



#endif
```

Perc_Graph.h

## 7.2 Point Class

```cpp
#include "Perc_Point.h"
#include<cmath>

Point::Point() {

  x_ = 0.0;
  y_ = 0.0;
  isOutside_ = false;
}

Point::Point(double x, double y) {
  double squareSum;
  x_ = x;
  y_ = y;
  if (sqrt(x*x + y*y) > 1) { isOutside_ = true; }
}

double Point::getX() {
  return x_;
}

double Point::getY() {
  return y_;
}

bool Point::isOutside() {
  return isOutside_;
}

void Point::setX(double x) {
  x_ = x;
}

void Point::setY(double y) {
  y_ = y;
}

void Point::checkOutside() {
  if (sqrt(x_*x_ + y_*y_) >= 1.0) { isOutside_ = true; }
}

bool Point::Outside() {
  if (sqrt(x_*x_ + y_*y_) >= 1.0) { isOutside_ = true; }
  return false;
}
```

Perc_Point.cpp

```cpp
#ifndef Perc_Point_hpp
#define Perc_Point_hpp

class Point {
private:
  double x_;
  double y_;
  bool isOutside_;




public:
  //Constructors
  Point();
  Point(double, double);

  //Getters
  double getX();
  double getY();
  bool isOutside();

  //Setters
  void setX(double);
  void setY(double);
  void checkOutside();

  bool Outside();
};






#endif
```

Perc_Point.h

## 7.3 R Code Used

```r
p <-  .5 #Multiplied by 1.1 so that probability is never 1
r <- .1
a <- (4/5)*r
pois <- rpois(1, 3500) #Generates a point of poisson variance for following lines.
xval <- runif(pois, min=-1, max=1)
yval <- runif(pois, min=-1,max=1)
adj <- diag(pois+1) #initializing this matrix
  for(i in 1:(pois-1)){
    for(j in (i+1):pois){
      if((sqrt((xval[j]-xval[i])^2+(yval[j]-yval[i])^2)) < a){
          xval[i] = 0
          yval[i] = 0
      }
     }
  }
xval <- xval[xval != 0] #Removing all deleted points
yval <- yval[yval != 0] #Removing all deleted points
xval <- c(0,xval) #Creating a center (origin) point
yval <- c(0,yval) #Creating a center (origin) point
plot(xval,yval,main="Matern II, a = (4/5)r")
for(i in 1:length(xval)) {
  for(j in 1:length(xval)){
    if((sqrt((xval[i]-xval[j])^2+(yval[i]-yval[j])^2)) < r && runif(1,0,1) < p){
      adj[i,j] = 1
      segments(xval[j],yval[j],xval[i],yval[i])
    }
  }
}
```

GraphicPoisson.R

```r
####Matern II EX
a <- .025
pois <- rpois(1, 7500) #Generates a point of poisson variance for following lines.
xval <- runif(pois, min=0, max=1)
yval <- runif(pois, min=0,max=1)
par(mfrow=c(1, 2))
plot(xval, yval,main="Poisson Process",xlab="x", ylab="y")
adj <- diag(pois+1) #initializing this matrix
for(i in 1:(pois-1)){
  for(j in (i+1):pois){
    if((sqrt((xval[j]-xval[i])^2+(yval[j]-yval[i])^2)) < a){
      xval[i] = 0
      yval[i] = 0
    }
  }
}
xval <- xval[xval != 0] #Removing all deleted points
yval <- yval[yval != 0] #Removing all deleted points
xval <- c(0,xval) #Creating a center (origin) point
yval <- c(0,yval) #Creating a center (origin) point
plot(xval,yval,main="Matern II",xlab="x", ylab="y")
```

PoissonvMatern.R

```r
setwd("~/Documents/Undergraduate/Research/")
library(Rcpp)
library(ggplot2)
library(aod)

data1 <- read.table("datasim.txt", header=TRUE)
data2 <- read.table("datasim2.txt", header=TRUE)

train <- data2
test <- data1

infdata <- rbind(data1,data2)

pprob <- c(); rprob <- c(); aprob <- c(); approb <- c(); arprob <- c();

for (i in 1:10) {
  pprob[i] =  length(which((infdata$p == i) & (infdata$infclus == 1)))/length(which(infdata$p == i));
  rprob[i] =  length(which((infdata$r == i) & (infdata$infclus == 1)))/length(which(infdata$r == i));
}

for (i in 1:4) {
  aprob[i] =  length(which((infdata$a == i) & (infdata$infclus == 1)))/length(which(infdata$a == i));
  for (j in 1:10) {
    approb[(i-1)*10 + j] = length(which((infdata$p == j) & (infdata$a == i) &
          (infdata$infclus == 1)))/length(which(infdata$p == j) & (infdata$a == i));
    arprob[(i-1)*10 + j] = length(which((infdata$r == j) & (infdata$a == i) &
          (infdata$infclus == 1)))/length(which(infdata$r == j) & (infdata$a == i));

  }
}

mylogit <- glm(infclus ~ p*r*a, data = train,family = binomial(logit))
summary(mylogit)
mylogitupd <- glm(infclus ~ r + a + p:r + r:a + p:r:a, data = train, family = binomial(logit))
summary(mylogitupd)

fitted.results <- predict(mylogit,test,type='response')
fitted.results <- ifelse(fitted.results > 0.5,1,0)
misClasificError <- mean(fitted.results != test$infclus)
print(paste('Accuracy',1-misClasificError))

fitted.results1 <- predict(mylogitupd,test,type='response')
fitted.results1 <- ifelse(fitted.results1 > 0.5,1,0)
misClasificError <- mean(fitted.results1 != test$infclus)
print(paste('Accuracy',1-misClasificError))

fitted.results2 <- predict(mylogitupd, test, type='response')
fitted.results2 <- fitted.results2*ifelse(test$a < 3,1,0)
fitted.results2 <- ifelse(fitted.results2 > .5, 1, 0)

library(caret)
confusionMatrix(data=fitted.results, reference=test$infclus)
confusionMatrix(data=fitted.results1, reference=test$infclus)

#### Graphing Individual Factor Probabilities
```

```r
par(mfrow=c(1,2))

plabels <- c("1/11","2/11","3/11","4/11","5/11","6/11","7/11","8/11","9/11","10/11")
plot(pprob ~ c(1:10), xlab = "Probability of Connection",
     ylab = "Infinite Cluster Probability", axes=FALSE)
axis(1, at=c(1:10),labels=plabels, las=2)
axis(2, at=c(0,.45),labels=c(0,.45), las=2)

rlabels = c(expression(paste(Delta, x[mu])),
            expression(paste(1.2, Delta, x[mu])),
            expression(paste(1.4, Delta, x[mu])),
            expression(paste(1.6, Delta, x[mu])),
            expression(paste(1.8, Delta, x[mu])),
            expression(paste(2.0, Delta, x[mu])),
            expression(paste(2.2, Delta, x[mu])),
            expression(paste(2.4, Delta, x[mu])),
            expression(paste(2.6, Delta, x[mu])),
            expression(paste(2.8, Delta, x[mu])))

plot(rprob ~ c(1:10), xlab = "", ylab = "Probability of Percolating Cluster", axes = FALSE)
axis(1, at = c(1:10), labels = rlabels, las=2)
title(xlab="Radius Required for Connection", line=3.5, cex.lab=1.0)
axis(2, at=c(0,.4), labels=c(0,.4),las=2)


par(mfrow=c(1,1))
alabels = c("(1/5)r","(2/5)r","(3/5)r","(4/5)r")
plot(aprob ~ c(1:4), xlab = "Deletion Radius of Point Thinning",
     ylab = "Probability of Percolating Cluster", axes=FALSE)
axis(1, at = c(1:4), labels = alabels, las=2)
axis(2, at = c(0,.7), labels=c(0,.7),las=2)

par(mfrow=c(1,2))

plot(approb ~ c(1:40), xlab = "Deletion Radius and Connection Probability",
     ylab = "Probability of Percolating Cluster")
abline(v=10.5); abline(v=20.5); abline(v=30.5);
plot(arprob ~ c(1:40), xlab = "Deletion Radius and Connection Radius",
     ylab = "Probability of Percolating Cluster")
abline(v=10.5); abline(v=20.5); abline(v=30.5)
title(main="Probability of Percolating Cluster in Respect to a", outer = T, line =-2)
```

Research.R