CSE 7350/5350

Wireless Sensor Network Project

Instructor: David W. Matula

Author: Tiancong Zhou

Contents

1. Executive Summary

1.1 Introduction and Summary

A wireless sensor network (WSN) of spatially distributed autonomous sensors is used to monitor physical or environmental conditions, such as temperature, sound, pressure, etc. and to cooperatively pass their data through the network to a main location.

It has widespread applications such as process management, health care monitoring, environmental/ earth sensing, industrial monitoring, etc. There could be numerous sensors all around the world or spread out in a certain region.

There may be certain types of questions to ask. For example, if one sensor node is responsible for a region within a distance of it, how can we choose among multiple nodes to form independent sensor groups to serve the whole system, and how to rotate the groups for a certain period of time since they may have limited battery lives.

This project applies graph theory and algorithm engineering to the field of WSN, by using algorithm engineering, we design and implement algorithm applications to solve real problems, like the ones mentioned above.

Graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. A "graph" in this context is made up of "vertices" or "nodes" and lines called edges that connect them.

In this project, each sensor is a node or vertex in the graphs. Firstly, we simulate the sensors by displaying the nodes in a sphere or in a square using random geometric graph (RGG) generation. We can connect the nodes by edges if they are within the threshold, to indicate they have direct connections. We demonstrate the RGGs according to the number of nodes and thresholds.

Secondly, after each node's connecting with its neighbors, we get the degree (number of adjacent nodes) for each node. We then reorder it by an algorithm "smallest last ordering", and apply a coloring algorithm to the nodes so that each vertex has a different color with its direct neighbor and we also use minimal number of colors. In the meantime, we plot the degree distribution diagrams, original degree diagrams, and the degrees when deleted using "smallest last ordering".

Thirdly, we choose 2 out of all the colors to form the backbone selection, which covers all or nearly all the vertices, and facilitates the communication throughout the whole network system. We also want to have different backbone color sets to rotate in order for the sensing operation to be rotated over time so that a single backbone is always operable while the others are rested to preserve server lifetime. The backbone pictures are illustrated and color sets data are recorded.

1.2 Programming Environment Description (20 points)

The programming environment consisted of a 15-inch MacBook Pro of Late 2011 running OS X. The following table outlines the hardware and software architecture.

Processor	2.2Ghz Intel Core i7
Memory	8GB 1067 MHz DDR3
Operating System	OS X 10.9.1(13B42), 64bit
Language	Java, Processing
Programming IDE	Eclipse
Graphics	AMD Radeon HD 6750M 512 MB
Statistic plot tool	Microsoft Excel

Processing was chosen as the displaying tool or graphic package in this project for its advantages in displaying 2D, 3D graphic designs and, simplicity of the syntax, and the Object-Oriented feature.

Processing is a programming language, and the same name applies to the IDE, as well as the online community. Though Processing does not have a long history, it has promoted software literacy within the visual arts and visual literacy within technology.

From a wider point of view, Processing builds on Java language and every Processing sketch is actually a subclass of PApplet Java class which implements most of the Processing language's features.

Java and Eclipse IDE provide a better debugging environment, console displaying, and clearer file organization. Therefore, I chose Java to set up a Processing PApplet, implement the algorithms, save data into txt files (for later plots), and demonstrate the graphic results.

In this project report, graphic drawings and displays are in the form of png files produced from the saveFrame() feature of Processing applet.

Microsoft Excel is used to plot the diagrams of data generated from the algorithms' implementation procedures. I chose Excel because it is widely accessible, highly convenient to use, and intuitive to show.

1.3 References (30 points)

[1] D.W. Matula, Wireless Sensor Network Project, www.lyle.smu.edu/cse/7350/, 2012

[2] Zizhen Chen, David W. Matula, "Partitioning RGG's Into Disjoint (1 - ε) Dominant Bipartite Subgraphs", in SIAM, 07, 2014, pp.48-50

[3] Dhia Mahjoub, David W. Matula, "Building (1 − ε) dominating sets partition as backbones in wireless sensor networks using distributed graph coloring," in Proc. of DCOSS '10, 2010, pp. 144–157.

[4] Dhia Mahjoub, David W. Matula, "Constructing efficient rotating backbones in wireless sensor networks using graph coloring", In Computer Communications, vol. 35, no. 9, pp. 1086-1097, 2012.

[5] D. Mahjoub and D. W. Matula, "Experimental study of independent and dominating sets in wireless sensor networks," in Proc. Of WASA '09, ser. LNCS, vol. 5682, 2009, pp. 32–42.

[6] D. Mahjoub and D. W. Matula. "Employing (1-epsilon) Dominating Set Partitions as Backbones in Wireless Sensor Networks." In ALENEX, pp. 98-111. 2010.

[7] Amdouni, I.; Minet, P.; Adjih, C., "Node coloring for dense wireless sensor networks", INRIA Research Report 7588, Paris-Rocquencourt, France, March 2011.

[8] lteris Murat Derici, Mihai Tudor Panu, "Determining the maximum clique size in large random geometric graphs", HPCS , 2011

[9] J. P. Baril, Wireless Sensor Network Project

[10] Zizhen Chen, Wireless Sensor Network Project

[11] Tao Zhong, Wireless Sensor Network Project

[12] Mihai Tudor Panu, Wireless Sensor Network Project

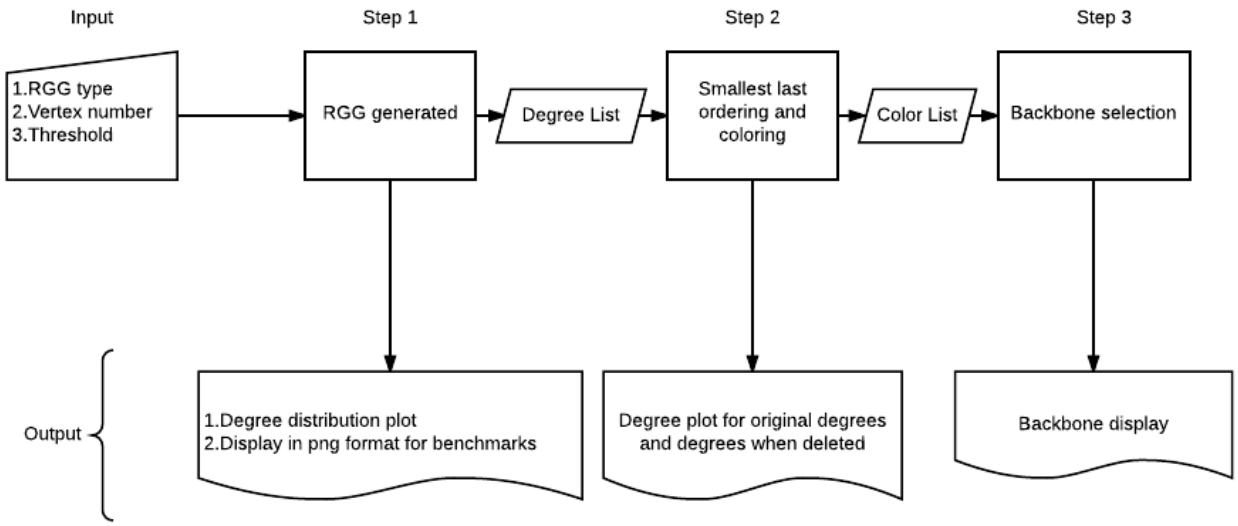
2. Wireless Sensor Network Backbone Report

2.1 Reduction to Practice (120 points)

2.1.1 Practice Introduction

As introduced above, this project has three steps: RGG generation; smallest last ordering and coloring; and backbone selection. RGG generation and display is in the requirement document Part I, and the other two steps are in Part II. In order to achieve a highly effective running procedure for multiple data group, we need to design specific data structure and implement effective algorithms to walk through the entire process.

The detailed process is illustrated in the following flow chart:



2.1.2 Data structure Design:

Custom Data Type

The basic component of RGG is the vertex or nodes; each node stores a bundle of information. According to this feature, I create a class called Node as the type for each node object. The following table describes the fields in the class.

Node

Fields	Note
int no	Identify the number of the node
int color	Indicate the color of the node
float x,y,z	Coordinates of x, y, z for the node
Boolean colored	Indicate it has been colored or not

int oridegree	The original degree when RGG is generated
int degdeleted	The degree when deleted by smallest last ordering
ArrayList <Node> nb	The ArrayList that contains the node's neighbors

ArrayList, Array

ArrayList is used to add, store, and delete nodes in the RGG and also for each node's neighbor information storage. After deletion, we put the nodes into an array to store since we don't need to have any manipulation with the nodes.

DegreeList(dl)

DegreeList is an ArrayList of LinkedLists. Its index represents how many degrees the nodes are.

For example, in the walkthrough later, the original DegreeList is like this:

```
degree 1 [Node 0]
degree 2 [Node 21]
degree 3 [Node 3, Node 19, Node 20]
degree 4 [Node 1, Node 4, Node 22, Node 23]
degree 5 [Node 2, Node 8, Node 9, Node 17, Node 18]
degree 6 [Node 7, Node 10, Node 12, Node 14]
degree 7 [Node 5, Node 11, Node 13, Node 15, Node 16]
degree 8 [Node 6]
degree 0 is set to null as empty
```

It means there is one node that has the degree of 1 whose number is 0, there is one node that has the degree of 2 whose number is 21, there are 3 nodes that have the degree of 3, whose names are 3,19,20, etc.

ColorList(cl)

ColorList is similar to DegreeList. Every node with a color *n* is grouped into the ColorList with index *n*. For the following ColorList:

```
color 0: null
```

```
color 1: [Node 0, Node 21, Node 3, Node 19, Node 22, Node 8, Node 9, Node
14]
color 2: [Node 20, Node 1, Node 4, Node 23, Node 17, Node 10]
color 3: [Node 2, Node 18, Node 7, Node 11]
color 4: [Node 12, Node 5, Node 13]
color 5: [Node 15, Node 6]
color 6: [Node 16]
color 7: null
color 8: null
```

We can tell that nodes with numbers {0,21,3,19,22,8,9, 14} are grouped into the ColorList index 1 because they all are colored with color 1, and {20, 1, 4, 23, 17, 10} with color 2… etc.

2.1.3 Algorithm Descriptions:

Smallest last vertex ordering:

Since the input is the DegreeList, we can simply remove one of the nodes with smallest degree available and put it into the array with the last available index; and at the same time, we remove this node from all its neighbors' (nb) ArrayList, updating the DegreeList by moving the neighbors from the previous degree to that of one less degree. That gives an O(1) time efficiency.

Graph coloring:

Once the vertex ordering is done, we need to find the colors for each vertex, or node. I hereby created a method getmincolor (Node n, ArrayList <Node> nodes) to get the color for each node n in ArrayList nodes. For this node n, we check if each of its neighbors has been colored, and then we assign a minimal number available color to it. That gives an O(N) time efficiency.

Counting faces using Euler's formula:

According the Euler's formula of counting faces on the sphere, we let the F= faces, V= vertices, and E= edges.

F-E+V=2

Then F = E-V+2. We then use this formula to calculate the number of faces for each benchmark.

2.1.4 Algorithm Engineering:

We cannot avoid walking through each node's neighbors when we are implementing the algorithms. So the question comes up as: what is the bound for traversing the neighbors?

The following formulas represent the relation between the *average degree* (ad), the number of vertices (N), the threshold (r).

For a square:

$\pi r^2/ad = 1/N$

$ad = N\pi r^2$

For a sphere:

$\pi r^2/ad = 4\pi r^2/N$

$ad = Nr^2/4$

In both case the bound for average degree is $O(Nr^2)$.

The total number of edges of the graph (E) is the average degree multiplied by the numbers of vertices and then divided by 2.

$$E = ad\ N = O(N^2 r^2)$$

For smallest last ordering:

As discussed above, the smallest last vertex ordering process uses ArrayList and Array data structures.

In java, the ArrayList takes O(N) space, and the Array takes O(N) space as well. So the space complexity for smallest last ordering is O(N).

In implementation, we find one node from the DegreeList with least index, that gives time O(1); then we walk through the neighbors of this node and remove this neighbor from their neighbor list, that gives time $O(Nr^2)$; we then move the nodes into minus one of previous indices, that gives time $O(Nr^2)$. So in general for N nodes the time complexity is $O(N^2r^2)$. Which is O(N+E).

For graph coloring:

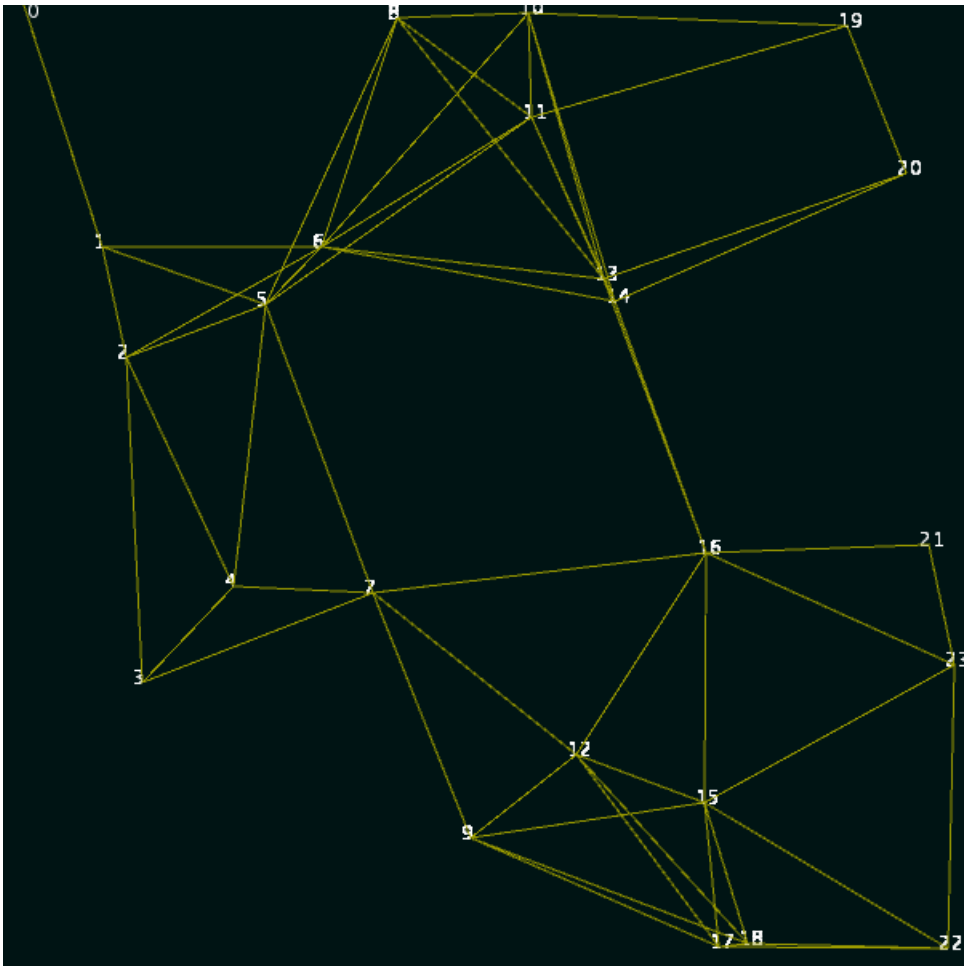
In the graph coloring algorithm, first the getmincolor() method visits each neighbor of a node, that gives time $O(Nr^2)$, for n nodes $O(N^2r^2)$. Then assigning the color gives time O(1). Which is also O(N+E).

2.1.5 Verification Walkthrough:

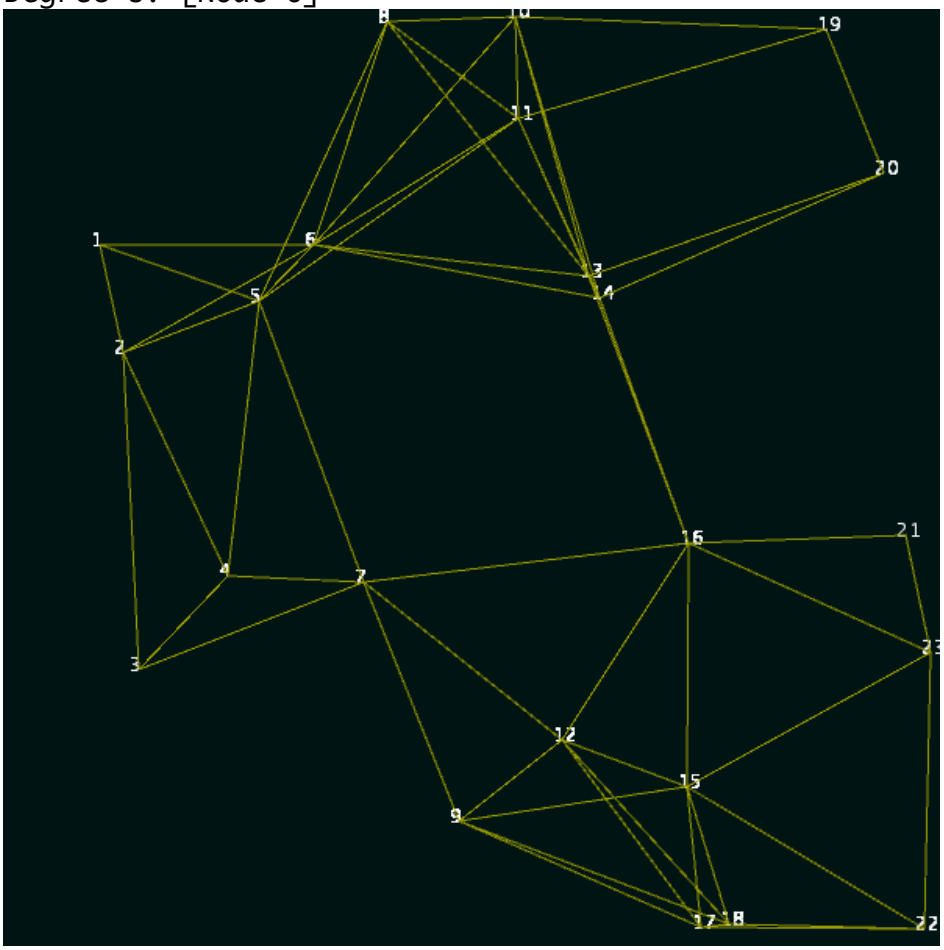
Here begins the Verification Walkthrough of RGG N=24, R=0.35 smallest last ordering process:

The graph display and the DegreeList are printed in each step to show the process.

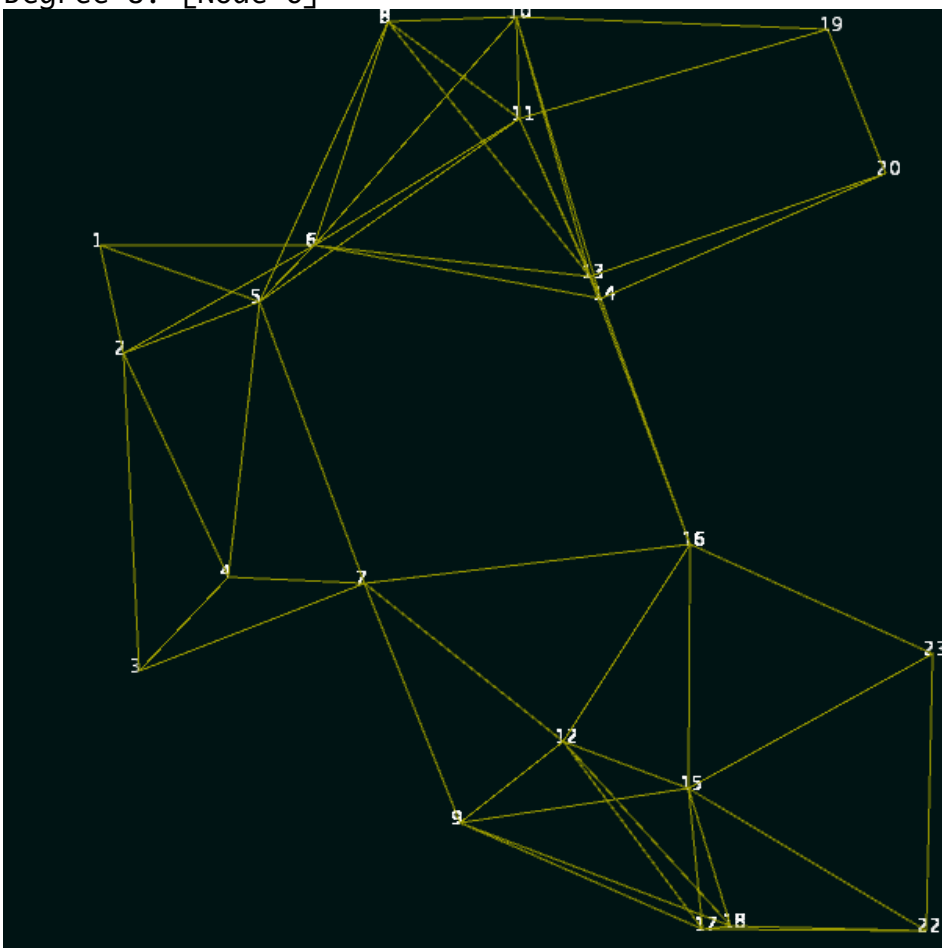
```
Step 0:
degree 1 [Node 0]
degree 2 [Node 21]
degree 3 [Node 3, Node 19, Node 20]
degree 4 [Node 1, Node 4, Node 22, Node 23]
degree 5 [Node 2, Node 8, Node 9, Node 17, Node 18]
degree 6 [Node 7, Node 10, Node 12, Node 14]
degree 7 [Node 5, Node 11, Node 13, Node 15, Node 16]
degree 8 [Node 6]
degree 0 is set to null as empty
```



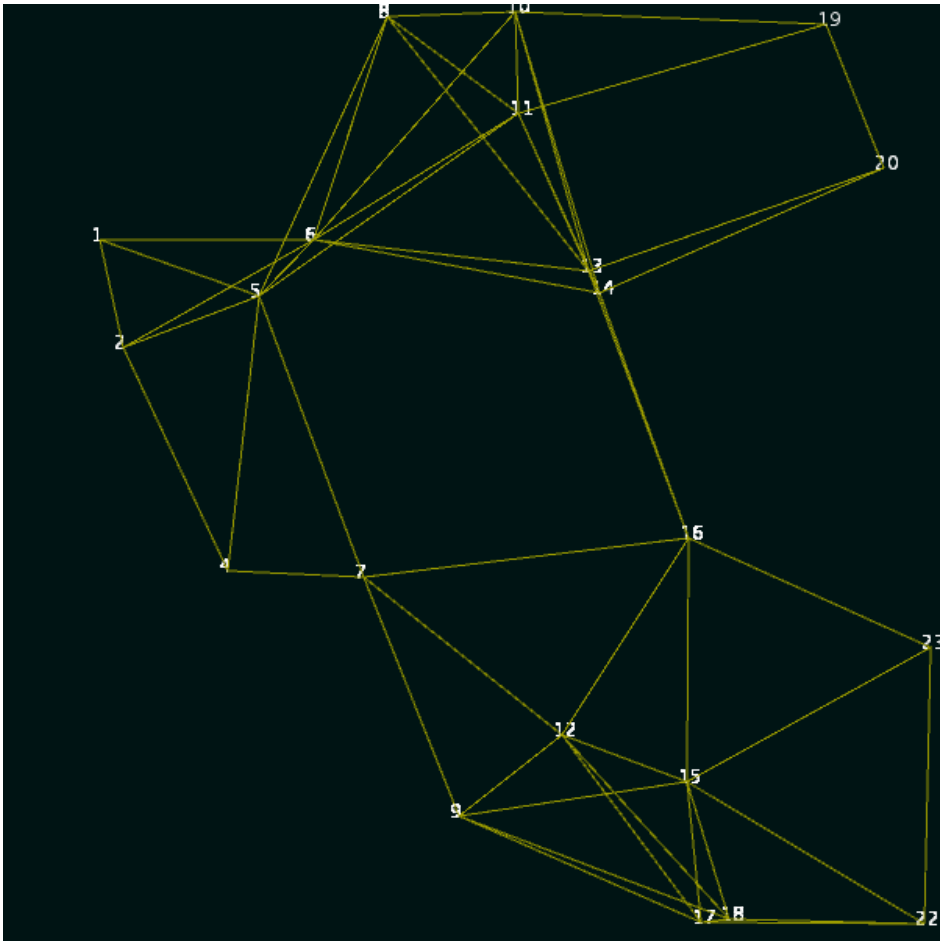
```
Step 1:
Node 0 with degree 1 is being removed to n[23]; it has nb of [Node 1]
Node 1 with degree 3 is removed from dl index 4; Node 0 is also removed
from Node 1 neighbor
Degree 0: null
Degree 1: []
Degree 2: [Node 21]
Degree 3: [Node 3, Node 19, Node 20, Node 1]
Degree 4: [Node 4, Node 22, Node 23]
Degree 5: [Node 2, Node 8, Node 9, Node 17, Node 18]
Degree 6: [Node 7, Node 10, Node 12, Node 14]
Degree 7: [Node 5, Node 11, Node 13, Node 15, Node 16]
Degree 8: [Node 6]
```



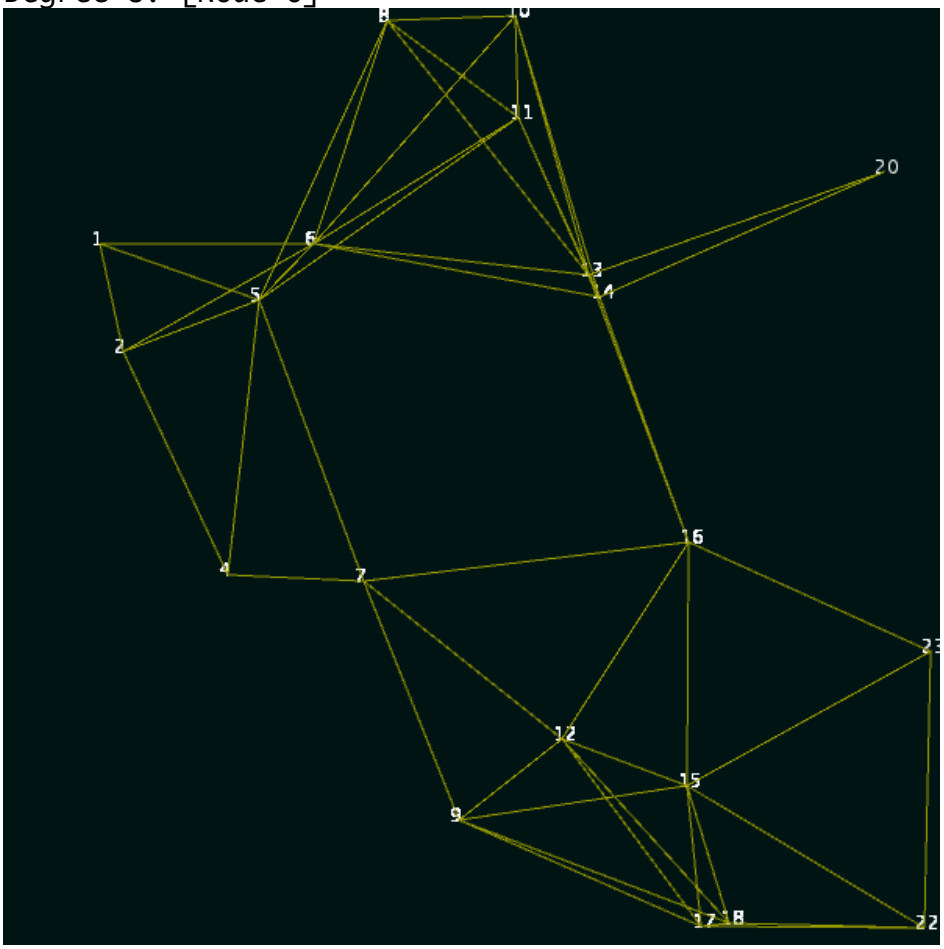
```
Step 2:
Node 21 with degree 2 is being removed to n[22]; it has nb of [Node 16,
Node 23]
Node 23 with degree 3 is removed from dl index 4; Node 21 is also removed
from Node 23 neighbor
Node 16 with degree 6 is removed from dl index 7; Node 21 is also removed
from Node 16 neighbor
Degree 0: null
Degree 1: []
Degree 2: []
Degree 3: [Node 3, Node 19, Node 20, Node 1, Node 23]
Degree 4: [Node 4, Node 22]
Degree 5: [Node 2, Node 8, Node 9, Node 17, Node 18]
Degree 6: [Node 7, Node 10, Node 12, Node 14, Node 16]
Degree 7: [Node 5, Node 11, Node 13, Node 15]
Degree 8: [Node 6]
```



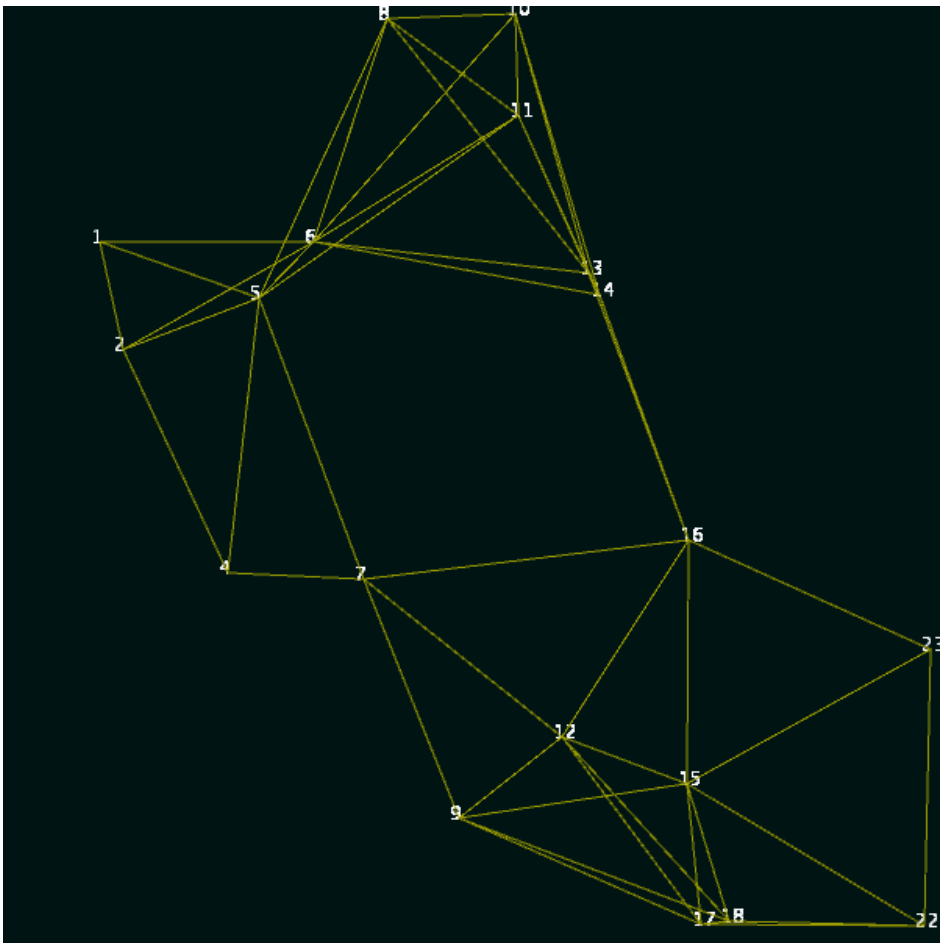
```
Step 3:
Node 3 with degree 3 is being removed to n[21]; it has nb of [Node 2, Node
4, Node 7]
Node 7 with degree 5 is removed from dl index 6; Node 3 is also removed
from Node 7 neighbor
Node 4 with degree 3 is removed from dl index 4; Node 3 is also removed
from Node 4 neighbor
Node 2 with degree 4 is removed from dl index 5; Node 3 is also removed
from Node 2 neighbor
Degree 0: null
Degree 1: []
Degree 2: []
Degree 3: [Node 19, Node 20, Node 1, Node 23, Node 4]
Degree 4: [Node 22, Node 2]
Degree 5: [Node 8, Node 9, Node 17, Node 18, Node 7]
Degree 6: [Node 10, Node 12, Node 14, Node 16]
Degree 7: [Node 5, Node 11, Node 13, Node 15]
Degree 8: [Node 6]
```



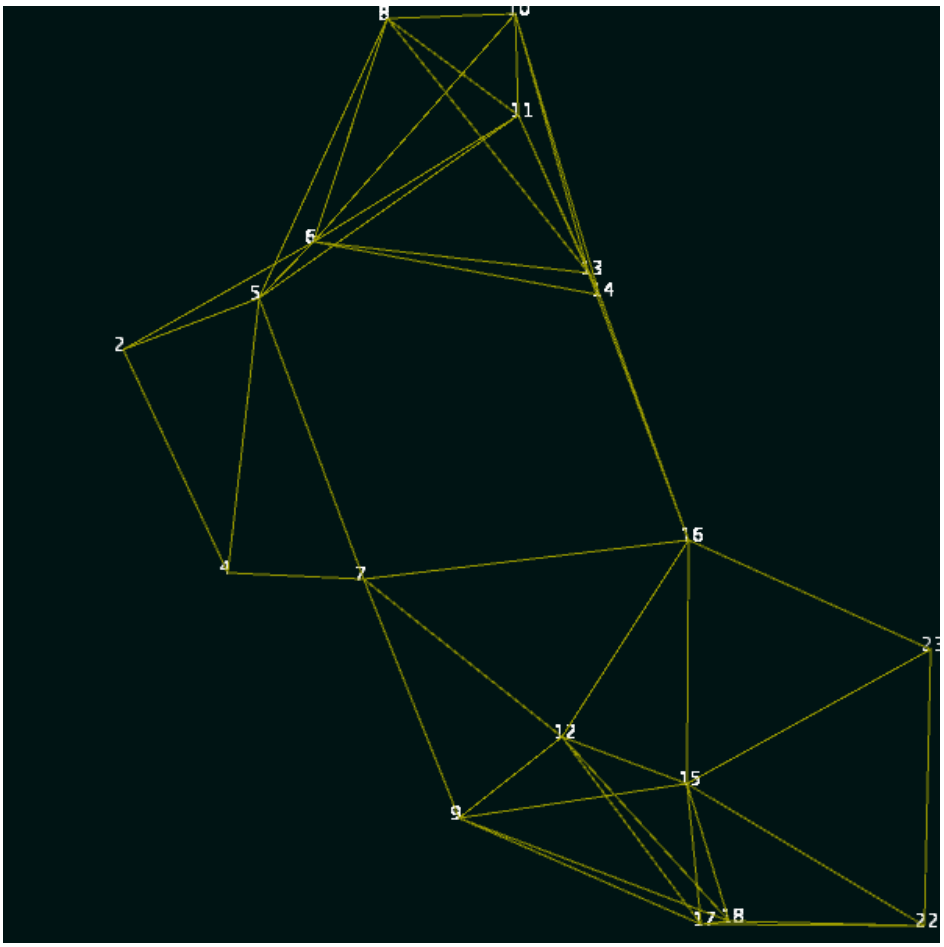
Step 4:
Node 19 with degree 3 is being removed to n[20]; it has nb of [Node 10, Node 11, Node 20]
Node 20 with degree 2 is removed from dl index 3; Node 19 is also removed from Node 20 neighbor
Node 11 with degree 6 is removed from dl index 7; Node 19 is also removed from Node 11 neighbor
Node 10 with degree 5 is removed from dl index 6; Node 19 is also removed from Node 10 neighbor
Degree 0: null
Degree 1: []
Degree 2: [Node 20]
Degree 3: [Node 1, Node 23, Node 4]
Degree 4: [Node 22, Node 2]
Degree 5: [Node 8, Node 9, Node 17, Node 18, Node 7, Node 10]
Degree 6: [Node 12, Node 14, Node 16, Node 11]
Degree 7: [Node 5, Node 13, Node 15]
Degree 8: [Node 6]



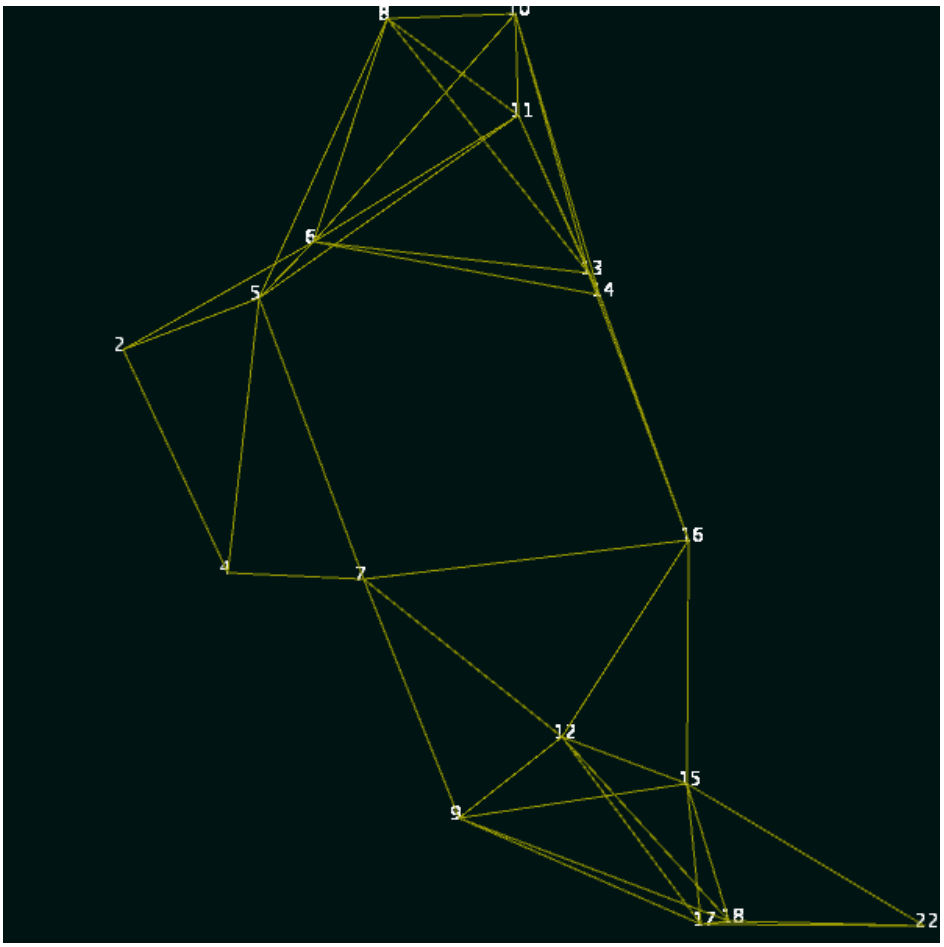
```
Step 5:
Node 20 with degree 2 is being removed to n[19]; it has nb of [Node 13,
Node 14]
Node 14 with degree 5 is removed from dl index 6; Node 20 is also removed
from Node 14 neighbor
Node 13 with degree 6 is removed from dl index 7; Node 20 is also removed
from Node 13 neighbor
Degree 0: null
Degree 1: []
Degree 2: []
Degree 3: [Node 1, Node 23, Node 4]
Degree 4: [Node 22, Node 2]
Degree 5: [Node 8, Node 9, Node 17, Node 18, Node 7, Node 10, Node 14]
Degree 6: [Node 12, Node 16, Node 11, Node 13]
Degree 7: [Node 5, Node 15]
Degree 8: [Node 6]
```



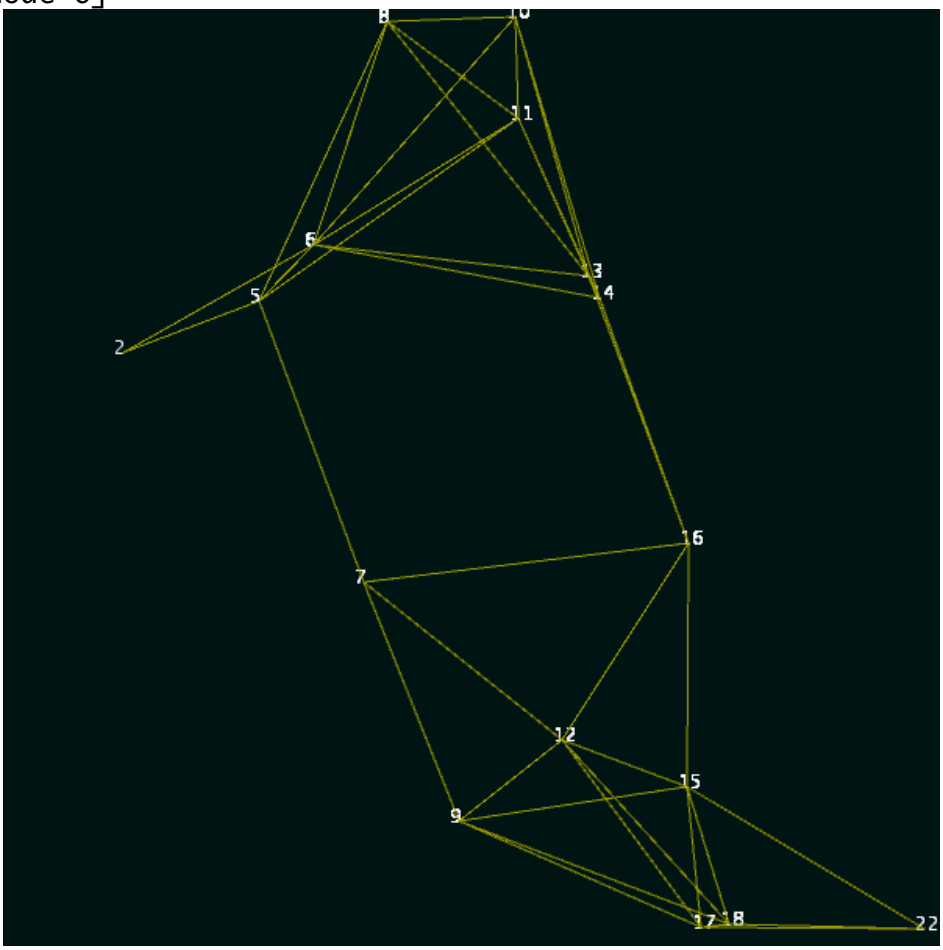
```
Step 6:
Node 1 with degree 3 is being removed to n[18]; it has nb of [Node 2, Node
5, Node 6]
Node 6 with degree 7 is removed from dl index 8; Node 1 is also removed
from Node 6 neighbor
Node 5 with degree 6 is removed from dl index 7; Node 1 is also removed
from Node 5 neighbor
Node 2 with degree 3 is removed from dl index 4; Node 1 is also removed
from Node 2 neighbor
Degree 0: null
Degree 1: []
Degree 2: []
Degree 3: [Node 23, Node 4, Node 2]
Degree 4: [Node 22]
Degree 5: [Node 8, Node 9, Node 17, Node 18, Node 7, Node 10, Node 14]
Degree 6: [Node 12, Node 16, Node 11, Node 13, Node 5]
Degree 7: [Node 15, Node 6]
Degree 8: []
```



Step 7:
Node 23 with degree 3 is being removed to n[17]; it has nb of [Node 15, Node 16, Node 22]
Node 22 with degree 3 is removed from dl index 4; Node 23 is also removed from Node 22 neighbor
Node 16 with degree 5 is removed from dl index 6; Node 23 is also removed from Node 16 neighbor
Node 15 with degree 6 is removed from dl index 7; Node 23 is also removed from Node 15 neighbor
Degree 0: null
Degree 1: []
Degree 2: []
Degree 3: [Node 4, Node 2, Node 22]
Degree 4: []
Degree 5: [Node 8, Node 9, Node 17, Node 18, Node 7, Node 10, Node 14, Node 16]
Degree 6: [Node 12, Node 11, Node 13, Node 5, Node 15]
Degree 7: [Node 6]
Degree 8: []

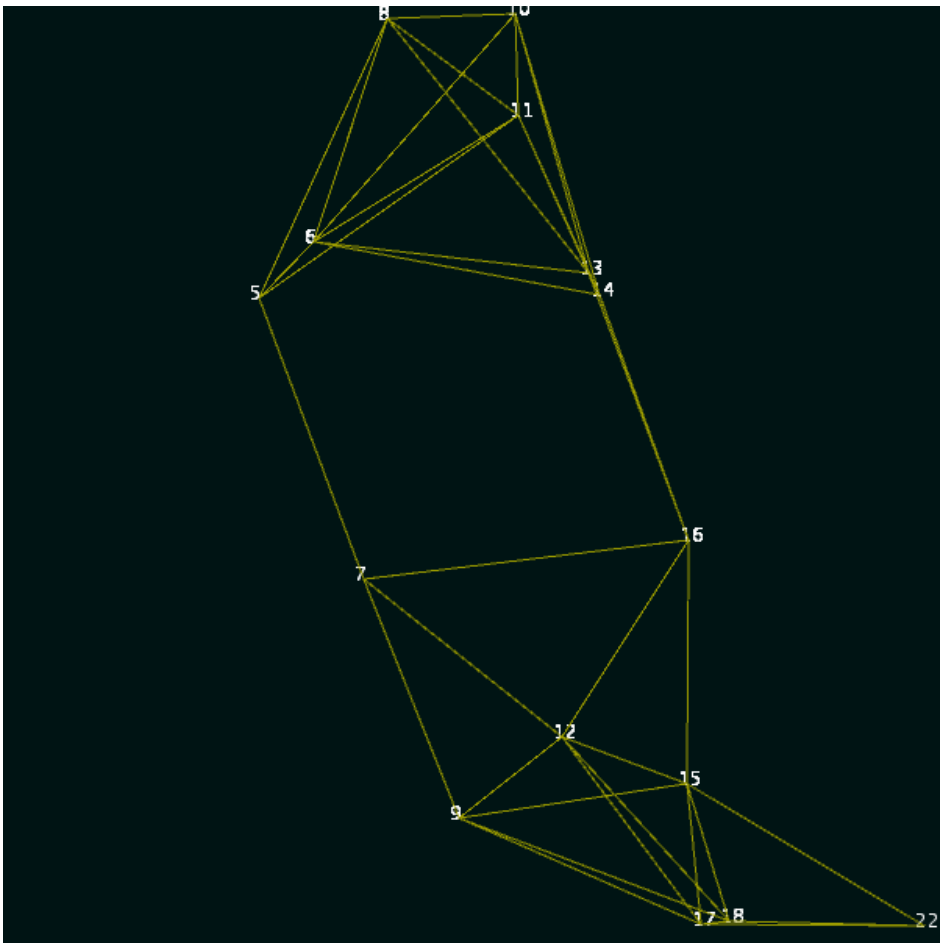


Step 8:
Node 4 with degree 3 is being removed to n[16]; it has nb of [Node 2, Node 5, Node 7]
Node 7 with degree 4 is removed from dl index 5; Node 4 is also removed from Node 7 neighbor
Node 5 with degree 5 is removed from dl index 6; Node 4 is also removed from Node 5 neighbor
Node 2 with degree 2 is removed from dl index 3; Node 4 is also removed from Node 2 neighbor
Degree 0: null
Degree 1: []
Degree 2: [Node 2]
Degree 3: [Node 22]
Degree 4: [Node 7]
Degree 5: [Node 8, Node 9, Node 17, Node 18, Node 10, Node 14, Node 16, Node 5]
Degree 6: [Node 12, Node 11, Node 13, Node 15]
Degree 7: [Node 6]

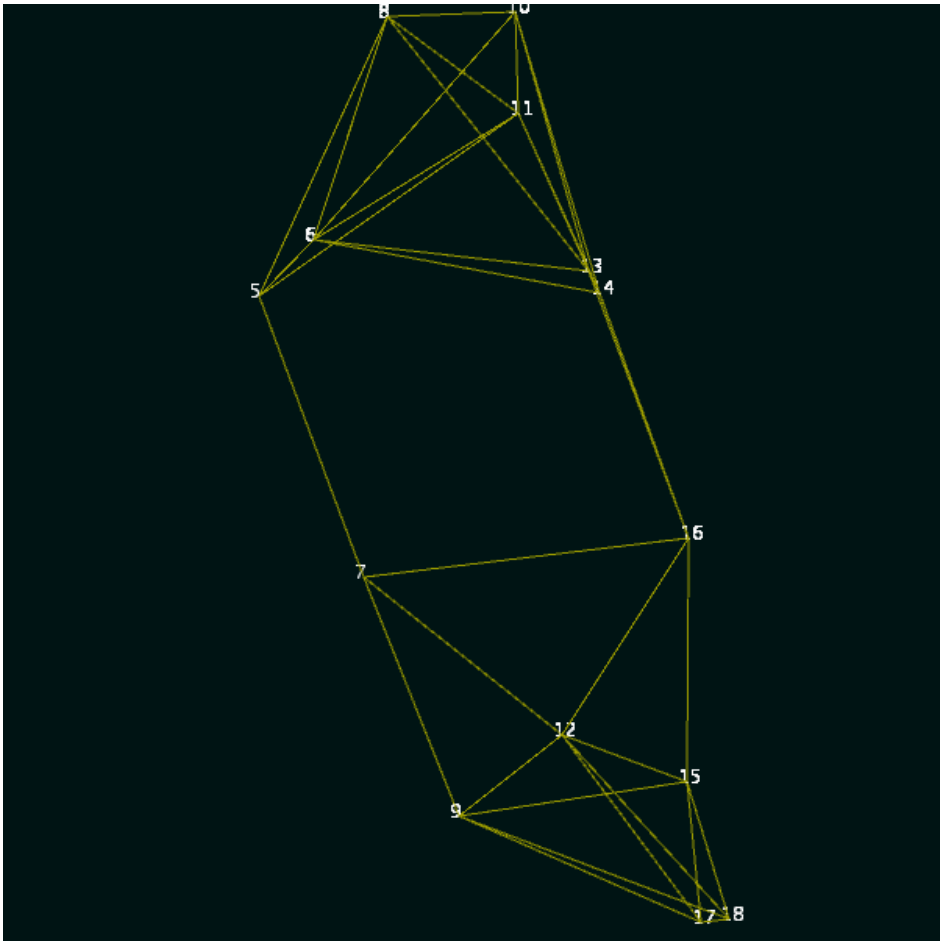


Degree 8: []

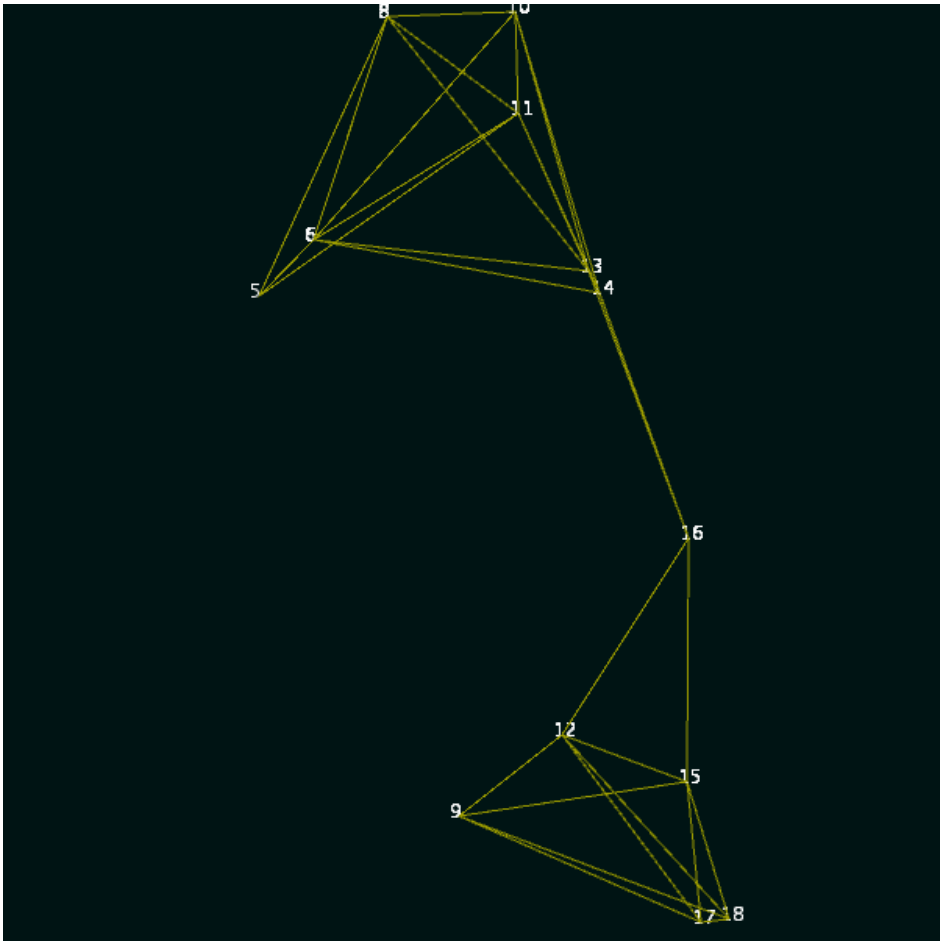
Step 9:
Node 2 with degree 2 is being removed to n[15]; it has nb of [Node 5, Node 6]
Node 6 with degree 6 is removed from dl index 7; Node 2 is also removed from Node 6 neighbor
Node 5 with degree 4 is removed from dl index 5; Node 2 is also removed from Node 5 neighbor
Degree 0: null
Degree 1: []
Degree 2: []
Degree 3: [Node 22]
Degree 4: [Node 7, Node 5]
Degree 5: [Node 8, Node 9, Node 17, Node 18, Node 10, Node 14, Node 16]
Degree 6: [Node 12, Node 11, Node 13, Node 15, Node 6]
Degree 7: []
Degree 8: []



```
Step 10:
Node 22 with degree 3 is being removed to n[14]; it has nb of [Node 15,
Node 17, Node 18]
Node 18 with degree 4 is removed from dl index 5; Node 22 is also removed
from Node 18 neighbor
Node 17 with degree 4 is removed from dl index 5; Node 22 is also removed
from Node 17 neighbor
Node 15 with degree 5 is removed from dl index 6; Node 22 is also removed
from Node 15 neighbor
Degree 0: null
Degree 1: []
Degree 2: []
Degree 3: []
Degree 4: [Node 7, Node 5, Node 18, Node 17]
Degree 5: [Node 8, Node 9, Node 10, Node 14, Node 16, Node 15]
Degree 6: [Node 12, Node 11, Node 13, Node 6]
Degree 7: []
Degree 8: []
```

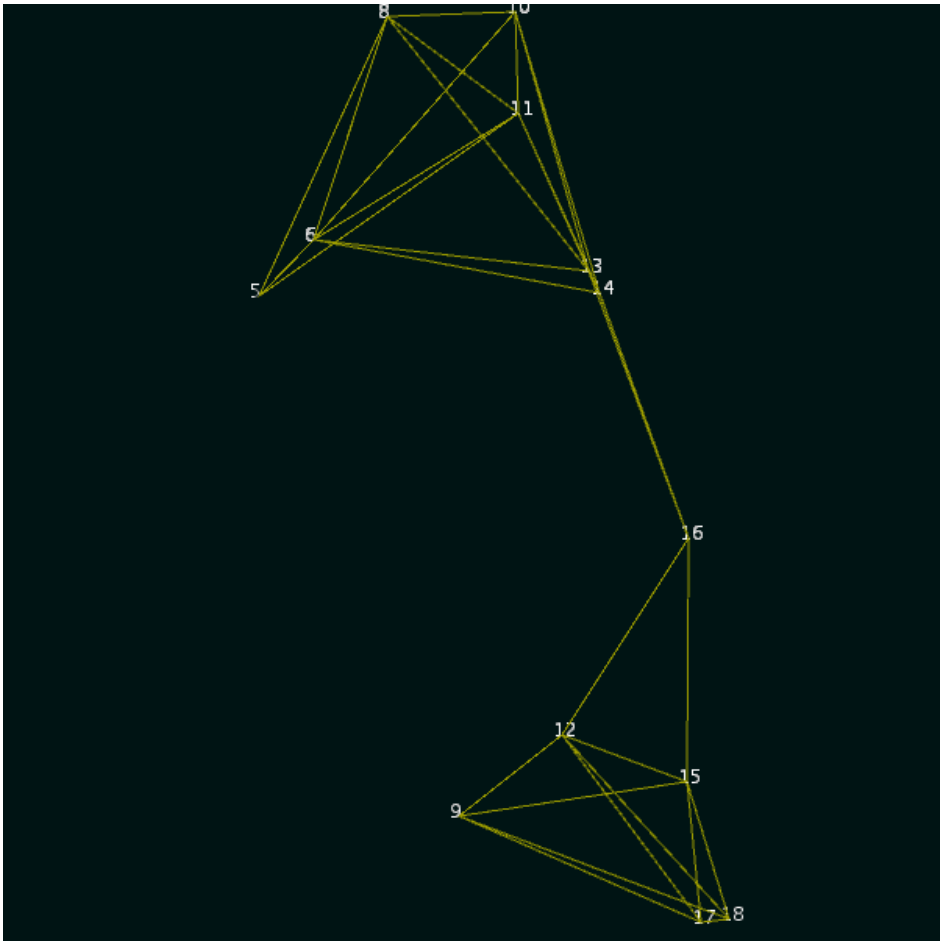


Step 11:
Node 7 with degree 4 is being removed to n[13]; it has nb of [Node 5, Node 9, Node 12, Node 16]
Node 16 with degree 4 is removed from dl index 5; Node 7 is also removed from Node 16 neighbor
Node 12 with degree 5 is removed from dl index 6; Node 7 is also removed from Node 12 neighbor
Node 9 with degree 4 is removed from dl index 5; Node 7 is also removed from Node 9 neighbor
Node 5 with degree 3 is removed from dl index 4; Node 7 is also removed from Node 5 neighbor
Degree 0: null
Degree 1: []
Degree 2: []
Degree 3: [Node 5]
Degree 4: [Node 18, Node 17, Node 16, Node 9]
Degree 5: [Node 8, Node 10, Node 14, Node 15, Node 12]
Degree 6: [Node 11, Node 13, Node 6]
Degree 7: []



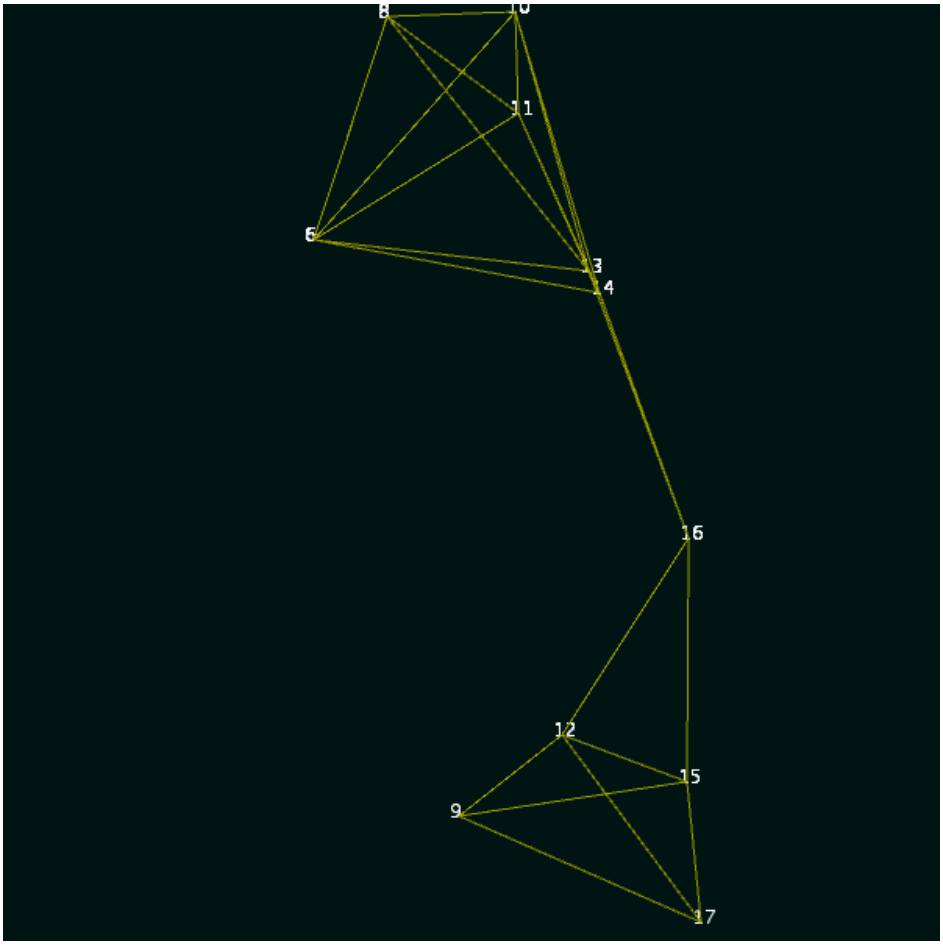
Degree 8: []

Step 12:
Node 5 with degree 3 is being removed to n[12]; it has nb of [Node 6, Node 8, Node 11]
Node 11 with degree 5 is removed from dl index 6; Node 5 is also removed from Node 11 neighbor
Node 8 with degree 4 is removed from dl index 5; Node 5 is also removed from Node 8 neighbor
Node 6 with degree 5 is removed from dl index 6; Node 5 is also removed from Node 6 neighbor
Degree 0: null
Degree 1: []
Degree 2: []
Degree 3: []
Degree 4: [Node 18, Node 17, Node 16, Node 9, Node 8]
Degree 5: [Node 10, Node 14, Node 15, Node 12, Node 11, Node 6]
Degree 6: [Node 13]
Degree 7: []



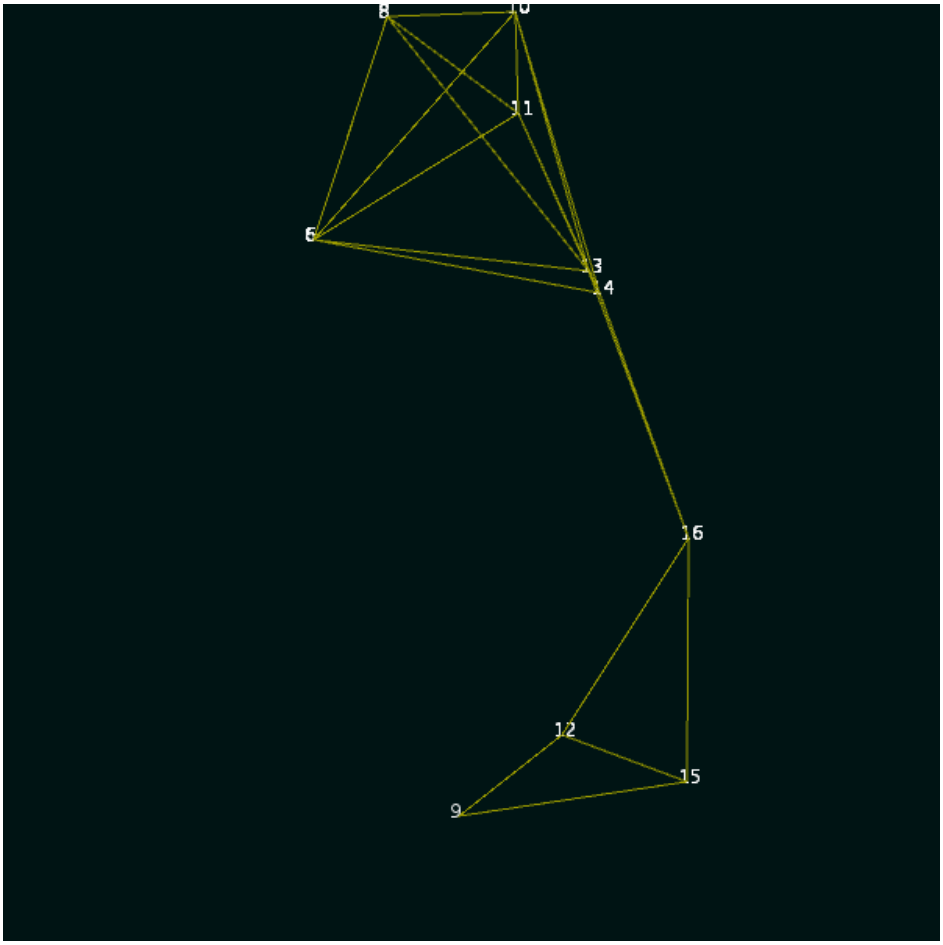
Degree 8: []

Step 13:
Node 18 with degree 4 is being removed to n[11]; it has nb of [Node 9, Node 12, Node 15, Node 17]
Node 17 with degree 3 is removed from dl index 4; Node 18 is also removed from Node 17 neighbor
Node 15 with degree 4 is removed from dl index 5; Node 18 is also removed from Node 15 neighbor
Node 12 with degree 4 is removed from dl index 5; Node 18 is also removed from Node 12 neighbor
Node 9 with degree 3 is removed from dl index 4; Node 18 is also removed from Node 9 neighbor
Degree 0: null
Degree 1: []
Degree 2: []
Degree 3: [Node 17, Node 9]
Degree 4: [Node 16, Node 8, Node 15, Node 12]
Degree 5: [Node 10, Node 14, Node 11, Node 6]
Degree 6: [Node 13]
Degree 7: []



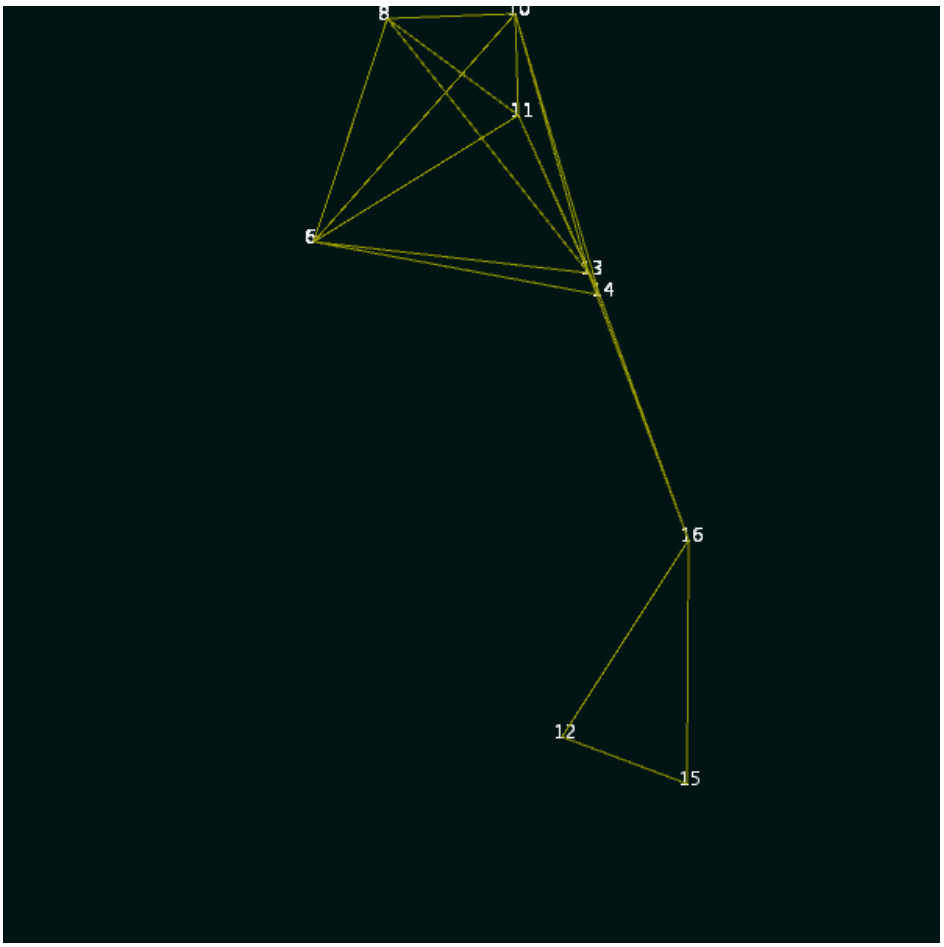
Degree 8: []

Step 14:
Node 17 with degree 3 is being removed to n[10]; it has nb of [Node 9, Node 12, Node 15]
Node 15 with degree 3 is removed from dl index 4; Node 17 is also removed from Node 15 neighbor
Node 12 with degree 3 is removed from dl index 4; Node 17 is also removed from Node 12 neighbor
Node 9 with degree 2 is removed from dl index 3; Node 17 is also removed from Node 9 neighbor
Degree 0: null
Degree 1: []
Degree 2: [Node 9]
Degree 3: [Node 15, Node 12]
Degree 4: [Node 16, Node 8]
Degree 5: [Node 10, Node 14, Node 11, Node 6]
Degree 6: [Node 13]
Degree 7: []

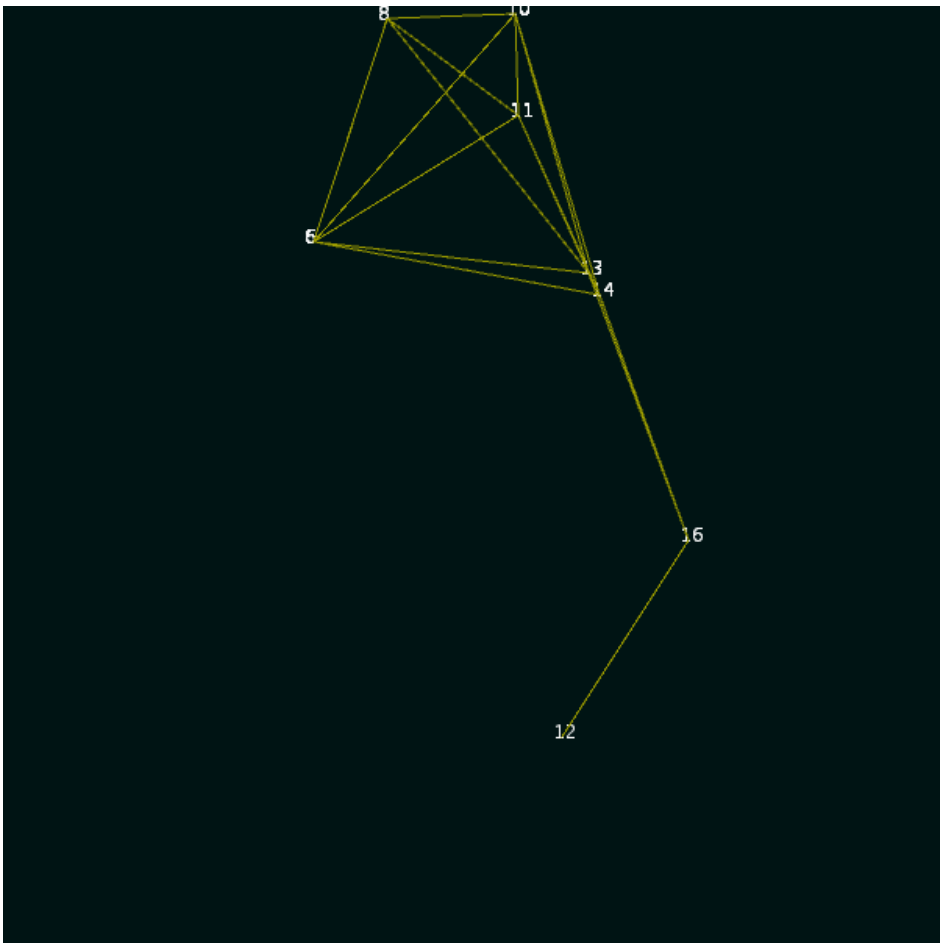


Degree 8: []

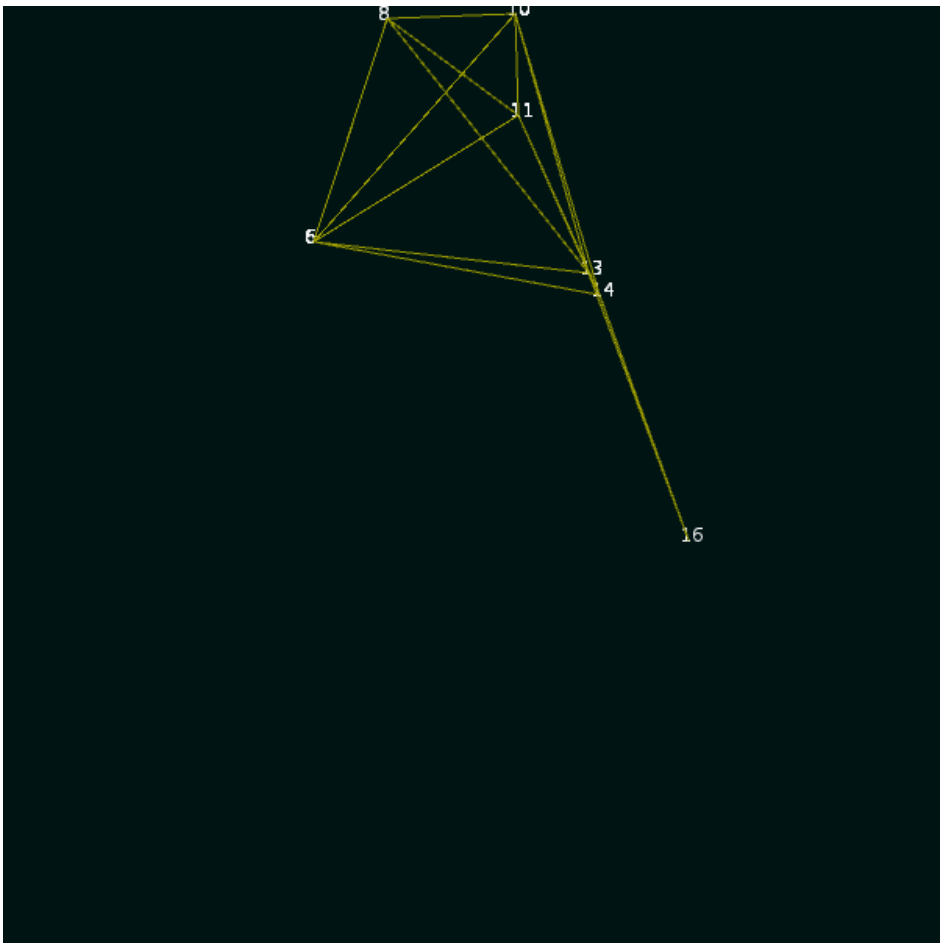
```
Step 15:
Node 9 with degree 2 is being removed to n[9]; it has nb of [Node 12, Node
15]
Node 15 with degree 2 is removed from dl index 3; Node 9 is also removed
from Node 15 neighbor
Node 12 with degree 2 is removed from dl index 3; Node 9 is also removed
from Node 12 neighbor
Degree 0: null
Degree 1: []
Degree 2: [Node 15, Node 12]
Degree 3: []
Degree 4: [Node 16, Node 8]
Degree 5: [Node 10, Node 14, Node 11, Node 6]
Degree 6: [Node 13]
Degree 7: []
Degree 8: []
```



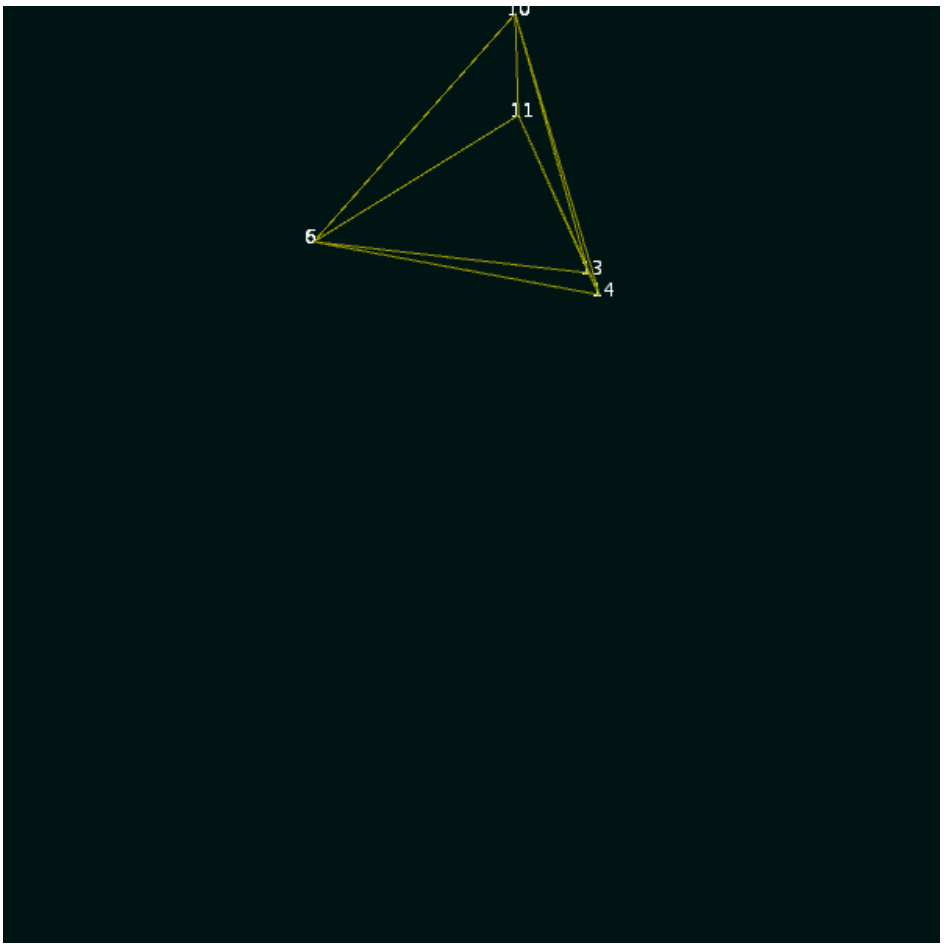
```
Step 16:
Node 15 with degree 2 is being removed to n[8]; it has nb of [Node 12, Node
16]
Node 16 with degree 3 is removed from dl index 4; Node 15 is also removed
from Node 16 neighbor
Node 12 with degree 1 is removed from dl index 2; Node 15 is also removed
from Node 12 neighbor
Degree 0: null
Degree 1: [Node 12]
Degree 2: []
Degree 3: [Node 16]
Degree 4: [Node 8]
Degree 5: [Node 10, Node 14, Node 11, Node 6]
Degree 6: [Node 13]
Degree 7: []
Degree 8: []
```



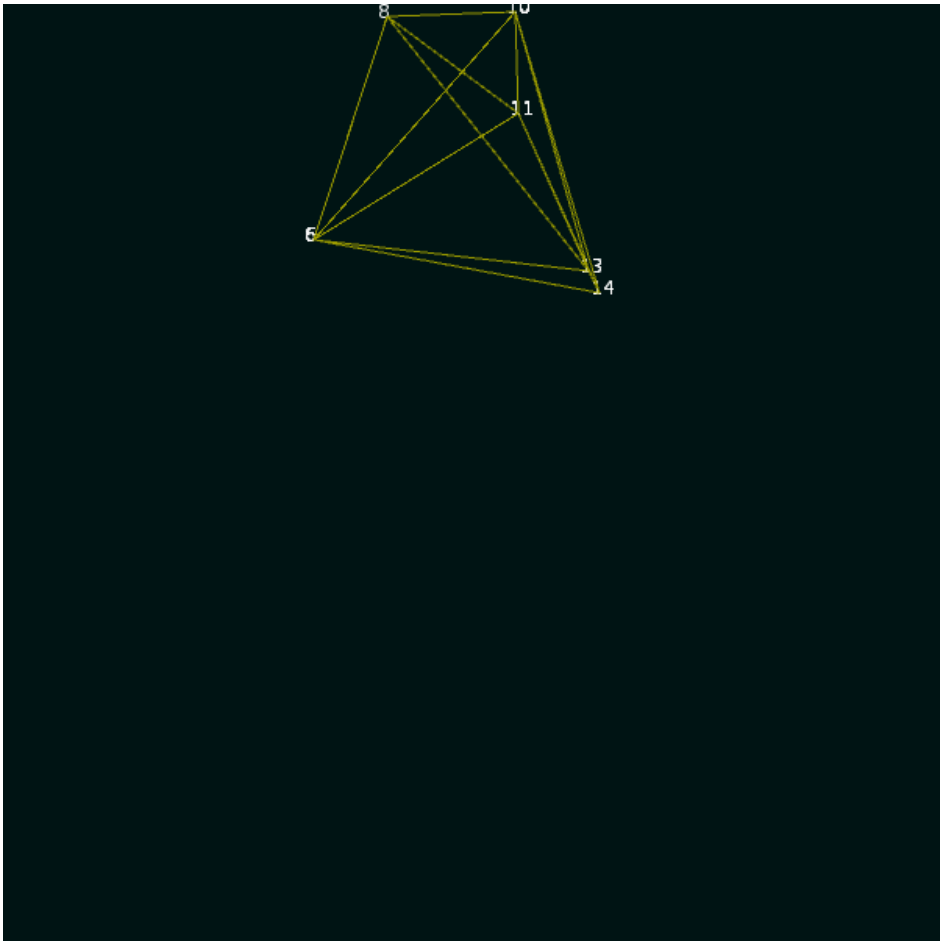
Step 17:
Node 12 with degree 1 is being removed to n[7]; it has nb of [Node 16]
Node 16 with degree 2 is removed from dl index 3; Node 12 is also removed from Node 16 neighbor
Degree 0: null
Degree 1: []
Degree 2: [Node 16]
Degree 3: []
Degree 4: [Node 8]
Degree 5: [Node 10, Node 14, Node 11, Node 6]
Degree 6: [Node 13]
Degree 7: []
Degree 8: []



Step 18:
Node 16 with degree 2 is being removed to n[6]; it has nb of [Node 13, Node 14]
Node 14 with degree 4 is removed from dl index 5; Node 16 is also removed from Node 14 neighbor
Node 13 with degree 5 is removed from dl index 6; Node 16 is also removed from Node 13 neighbor
Degree 0: null
Degree 1: []
Degree 2: []
Degree 3: []
Degree 4: [Node 8, Node 14]
Degree 5: [Node 10, Node 11, Node 6, Node 13]
Degree 6: []
Degree 7: []
Degree 8: []

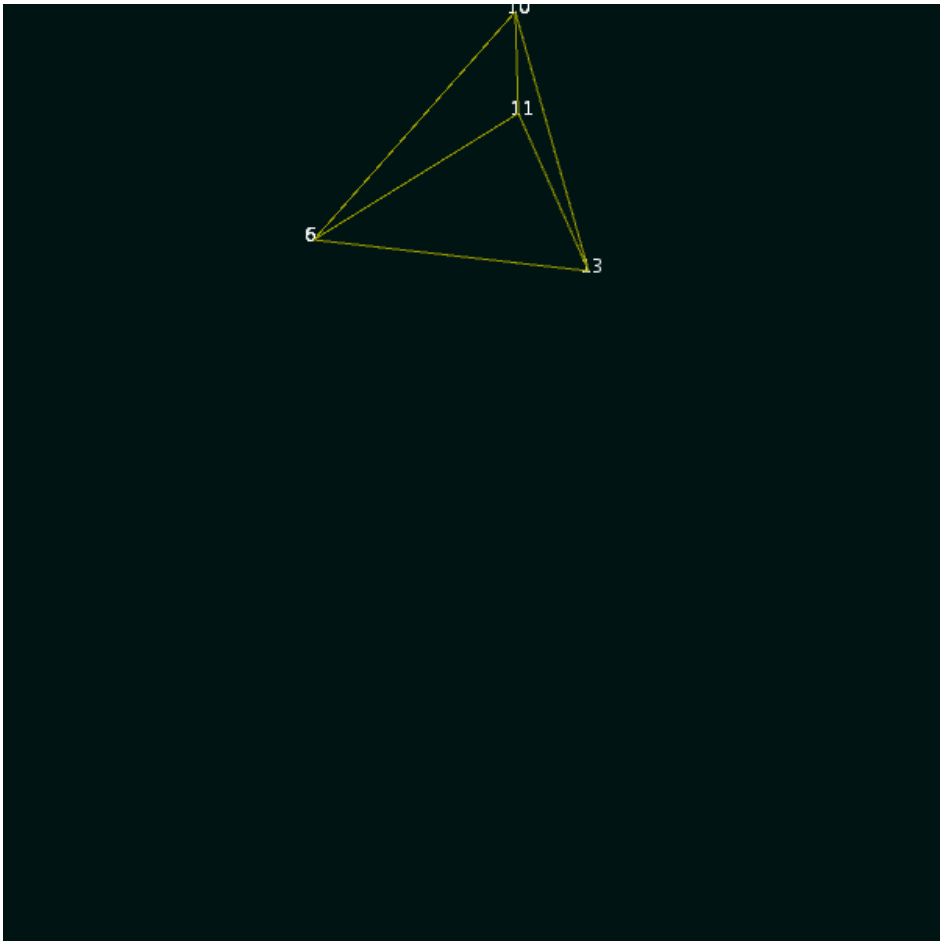


```
Step 19:
Node 8 with degree 4 is being removed to n[5]; it has nb of [Node 6, Node
10, Node 11, Node 13]
Node 13 with degree 4 is removed from dl index 5; Node 8 is also removed
from Node 13 neighbor
Node 11 with degree 4 is removed from dl index 5; Node 8 is also removed
from Node 11 neighbor
Node 10 with degree 4 is removed from dl index 5; Node 8 is also removed
from Node 10 neighbor
Node 6 with degree 4 is removed from dl index 5; Node 8 is also removed
from Node 6 neighbor
Degree 0: null
Degree 1: []
Degree 2: []
Degree 3: []
Degree 4: [Node 14, Node 13, Node 11, Node 10, Node 6]
Degree 5: []
Degree 6: []
Degree 7: []
```



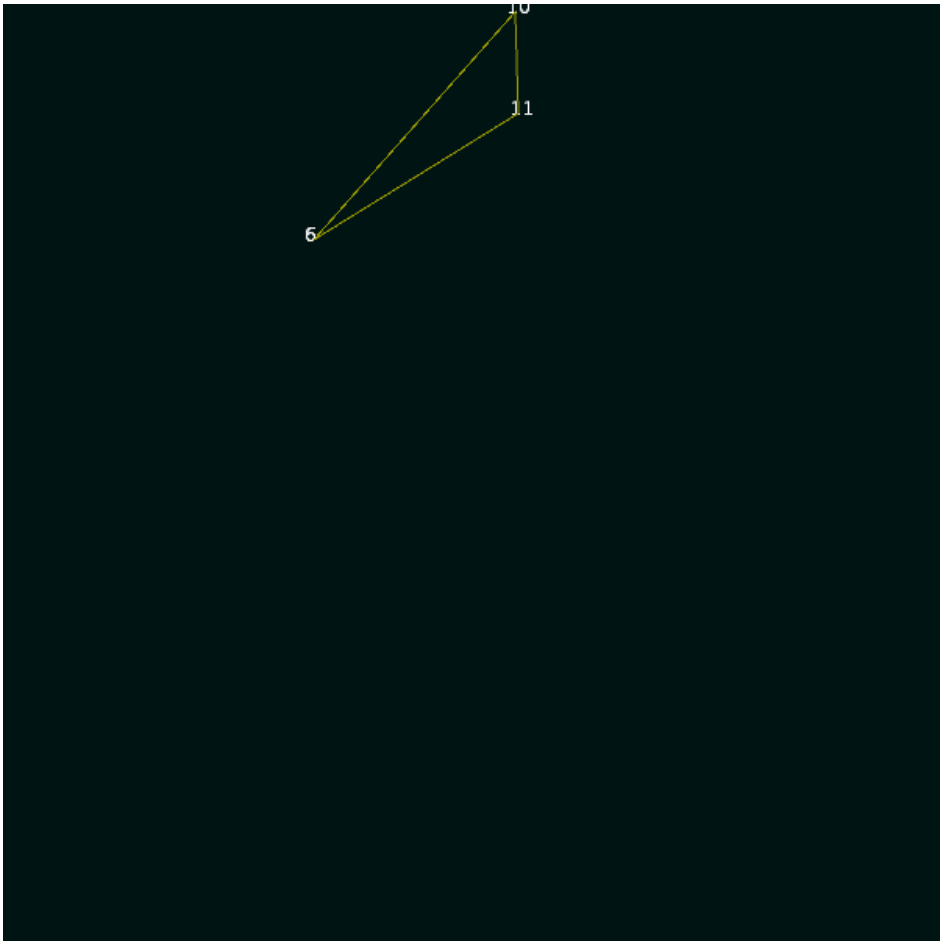

```
Degree 8: []
```

Step 20:
Node 14 with degree 4 is being removed to n[4]; it has nb of [Node 6, Node 10, Node 11, Node 13]
Node 13 with degree 3 is removed from dl index 4; Node 14 is also removed from Node 13 neighbor
Node 11 with degree 3 is removed from dl index 4; Node 14 is also removed from Node 11 neighbor
Node 10 with degree 3 is removed from dl index 4; Node 14 is also removed from Node 10 neighbor
Node 6 with degree 3 is removed from dl index 4; Node 14 is also removed from Node 6 neighbor
Degree 0: null
Degree 1: []
Degree 2: []
Degree 3: [Node 13, Node 11, Node 10, Node 6]
Degree 4: []
Degree 5: []
Degree 6: []
Degree 7: []

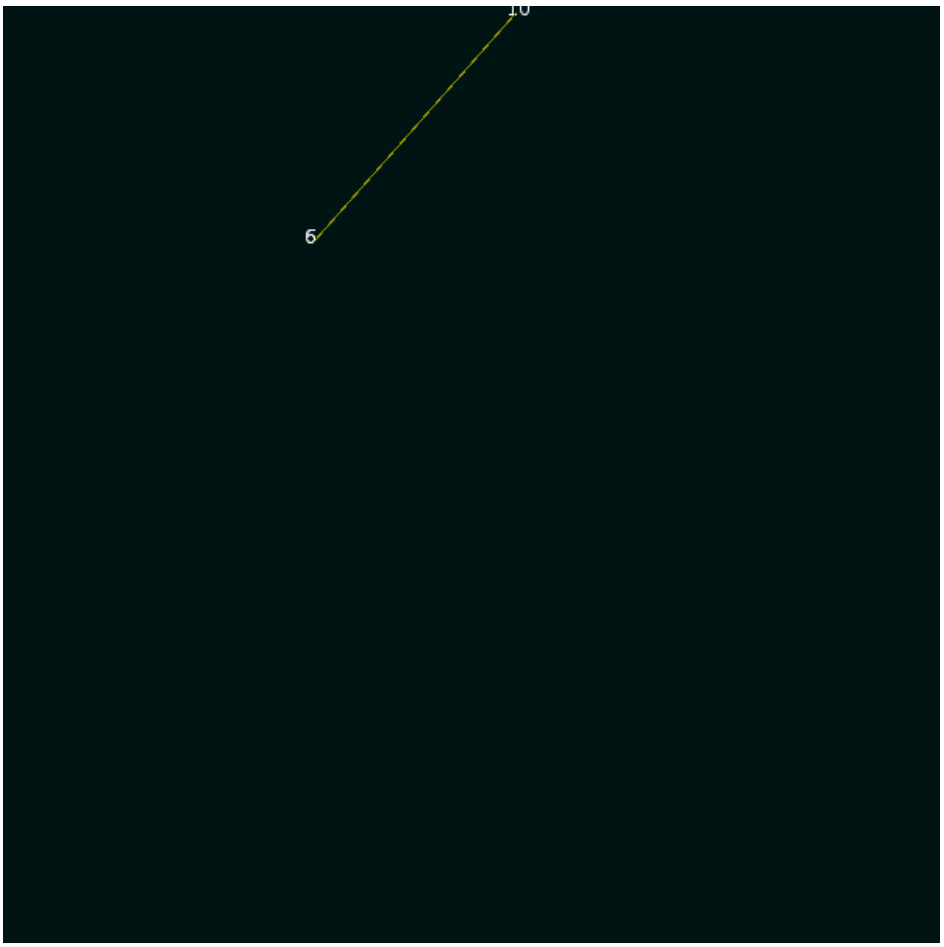


Degree 8: []

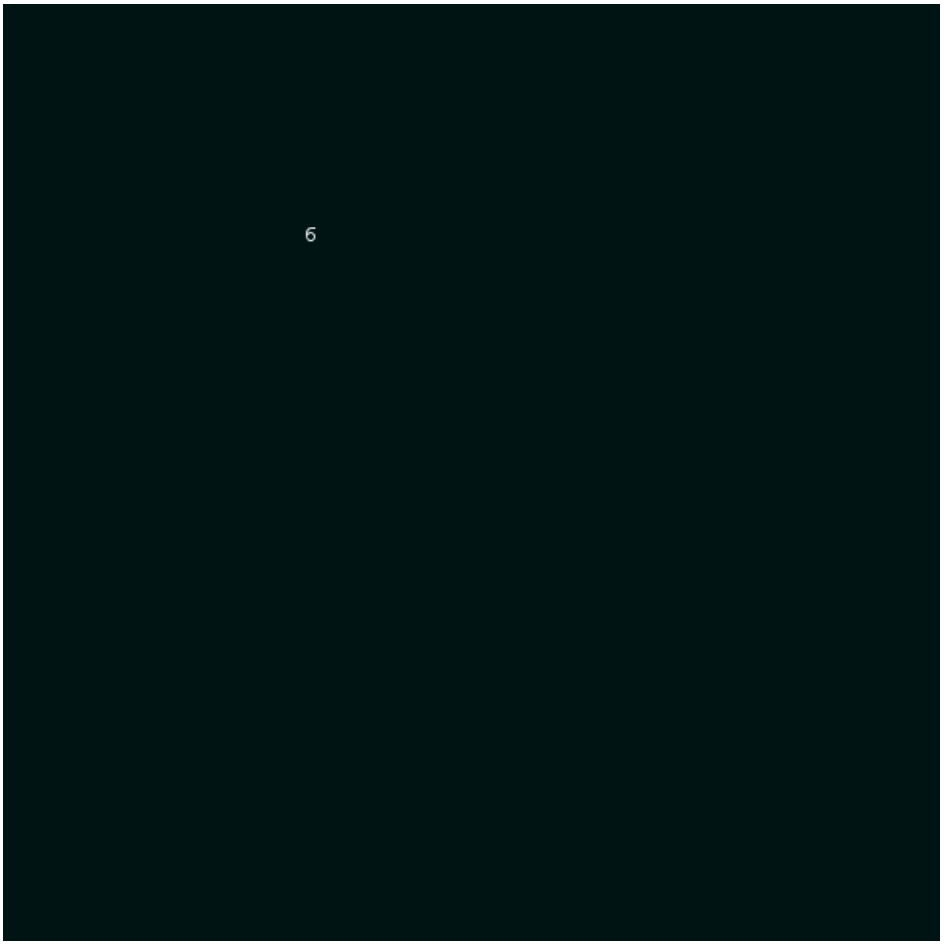
Step 21:
Node 13 with degree 3 is being removed to n[3]; it has nb of [Node 6, Node 10, Node 11]
Node 11 with degree 2 is removed from dl index 3; Node 13 is also removed from Node 11 neighbor
Node 10 with degree 2 is removed from dl index 3; Node 13 is also removed from Node 10 neighbor
Node 6 with degree 2 is removed from dl index 3; Node 13 is also removed from Node 6 neighbor
Degree 0: null
Degree 1: []
Degree 2: [Node 11, Node 10, Node 6]
Degree 3: []
Degree 4: []
Degree 5: []
Degree 6: []
Degree 7: []
Degree 8: []



```
Step 22:
Node 11 with degree 2 is being removed to n[2]; it has nb of [Node 6, Node
10]
Node 10 with degree 1 is removed from dl index 2; Node 11 is also removed
from Node 10 neighbor
Node 6 with degree 1 is removed from dl index 2; Node 11 is also removed
from Node 6 neighbor
Degree 0: null
Degree 1: [Node 10, Node 6]
Degree 2: []
Degree 3: []
Degree 4: []
Degree 5: []
Degree 6: []
Degree 7: []
Degree 8: []
```



Step 23:
Node 10 with degree 1 is being removed to n[1]; it has nb of [Node 6]
Node 6 with degree 0 is removed from dl index 1; Node 10 is also removed from Node 6 neighbor
Degree 0: [Node 6]
Degree 1: []
Degree 2: []
Degree 3: []
Degree 4: []
Degree 5: []
Degree 6: []
Degree 7: []
Degree 8: []



```
Step 24:
Node 6 with degree 0 is being removed to n[0]; it has nb of []
Degree 0: []
Degree 1: []
Degree 2: []
Degree 3: []
Degree 4: []
Degree 5: []
Degree 6: []
Degree 7: []
Degree 8: []
```

After the smallest last ordering, all the nodes are removed and here we begin the graph coloring process.

I present this procedure by walking through each node, printing each node's number and the process of deciding which color it should be colored. The detailed procedure is described in the *Algorithm Descriptions* section.

```
Node 0 gets the color of 1
Node 21 gets the color of 1
Node 3 gets the color of 1
Node 19 gets the color of 1
Node 20's nb Node 19 has been colored with color 1
Node 20 gets the color of 2
Node 1's nb Node 0 has been colored with color 1
Node 1 gets the color of 2
Node 4's nb Node 3 has been colored with color 1
Node 4 gets the color of 2
Node 22 gets the color of 1
Node 23's nb Node 21 has been colored with color 1
Node 23's nb Node 22 has been colored with color 1
Node 23 gets the color of 2
Node 2's nb Node 1 has been colored with color 2
Node 2's nb Node 3 has been colored with color 1
Node 2's nb Node 4 has been colored with color 2
Node 2 gets the color of 3
Node 8 gets the color of 1
Node 9 gets the color of 1
Node 17's nb Node 9 has been colored with color 1
Node 17's nb Node 22 has been colored with color 1
Node 17 gets the color of 2
Node 18's nb Node 9 has been colored with color 1
Node 18's nb Node 17 has been colored with color 2
```

```
Node 18's nb Node 22 has been colored with color 1
Node 18 gets the color of 3
Node 7's nb Node 3 has been colored with color 1
Node 7's nb Node 4 has been colored with color 2
Node 7's nb Node 9 has been colored with color 1
Node 7 gets the color of 3
Node 10's nb Node 8 has been colored with color 1
Node 10's nb Node 19 has been colored with color 1
Node 10 gets the color of 2
Node 12's nb Node 7 has been colored with color 3
Node 12's nb Node 9 has been colored with color 1
Node 12's nb Node 17 has been colored with color 2
Node 12's nb Node 18 has been colored with color 3
Node 12 gets the color of 4
Node 14's nb Node 10 has been colored with color 2
Node 14's nb Node 20 has been colored with color 2
Node 14 gets the color of 1
Node 5's nb Node 1 has been colored with color 2
Node 5's nb Node 2 has been colored with color 3
Node 5's nb Node 4 has been colored with color 2
Node 5's nb Node 7 has been colored with color 3
Node 5's nb Node 8 has been colored with color 1
Node 5 gets the color of 4
Node 11's nb Node 5 has been colored with color 4
Node 11's nb Node 8 has been colored with color 1
Node 11's nb Node 10 has been colored with color 2
Node 11's nb Node 14 has been colored with color 1
Node 11's nb Node 19 has been colored with color 1
Node 11 gets the color of 3
Node 13's nb Node 8 has been colored with color 1
Node 13's nb Node 10 has been colored with color 2
Node 13's nb Node 11 has been colored with color 3
Node 13's nb Node 14 has been colored with color 1
Node 13's nb Node 20 has been colored with color 2
Node 13 gets the color of 4
Node 15's nb Node 9 has been colored with color 1
Node 15's nb Node 12 has been colored with color 4
Node 15's nb Node 17 has been colored with color 2
Node 15's nb Node 18 has been colored with color 3
Node 15's nb Node 22 has been colored with color 1
Node 15's nb Node 23 has been colored with color 2
Node 15 gets the color of 5
Node 16's nb Node 7 has been colored with color 3
Node 16's nb Node 12 has been colored with color 4
```

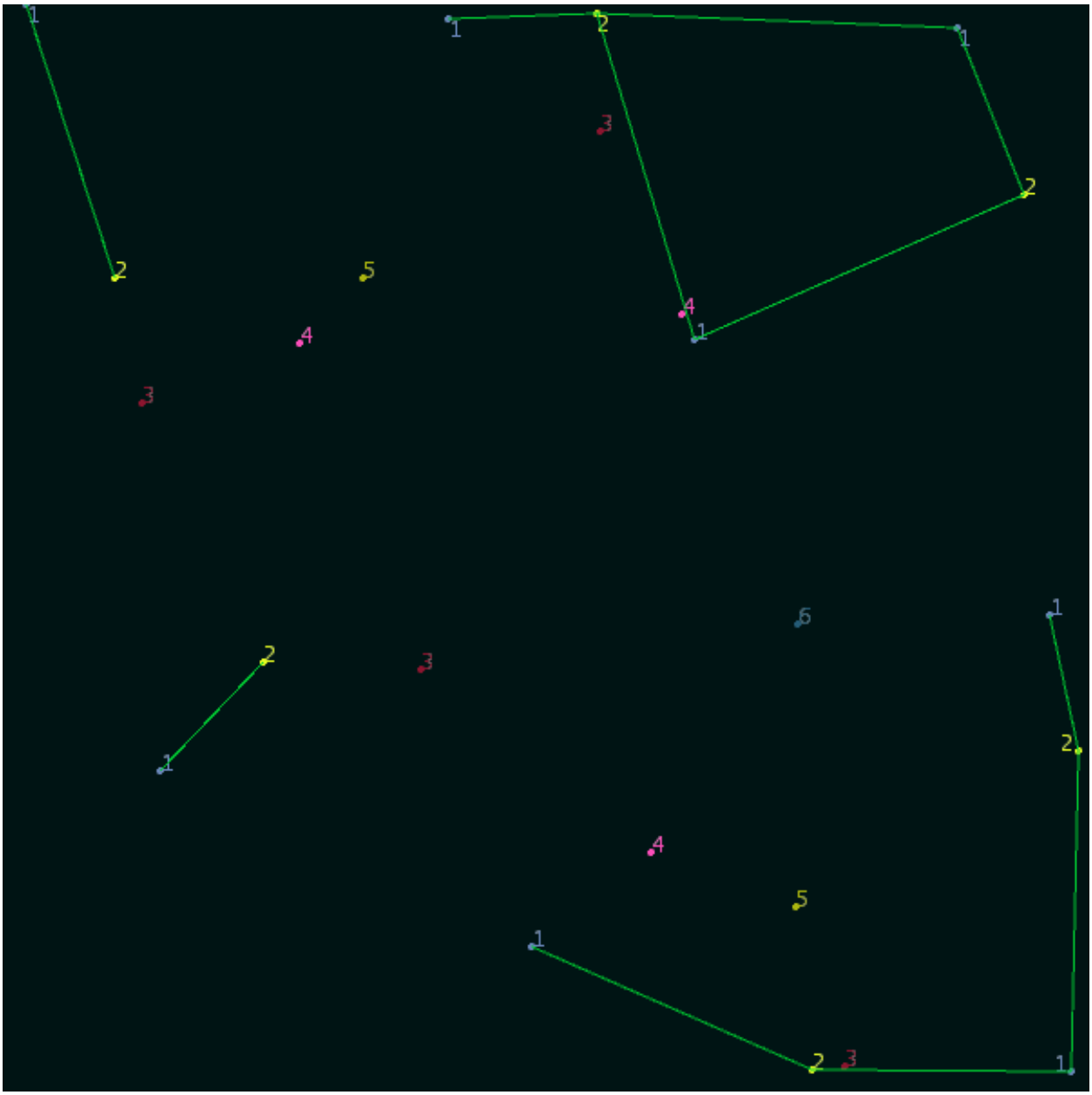
```
Node 16's nb Node 13 has been colored with color 4
Node 16's nb Node 14 has been colored with color 1
Node 16's nb Node 15 has been colored with color 5
Node 16's nb Node 21 has been colored with color 1
Node 16's nb Node 23 has been colored with color 2
Node 16 gets the color of 6
Node 6's nb Node 1 has been colored with color 2
Node 6's nb Node 2 has been colored with color 3
Node 6's nb Node 5 has been colored with color 4
Node 6's nb Node 8 has been colored with color 1
Node 6's nb Node 10 has been colored with color 2
Node 6's nb Node 11 has been colored with color 3
Node 6's nb Node 13 has been colored with color 4
Node 6's nb Node 14 has been colored with color 1
Node 6 gets the color of 5
```

After the coloring process, The ColorList is produced as following:

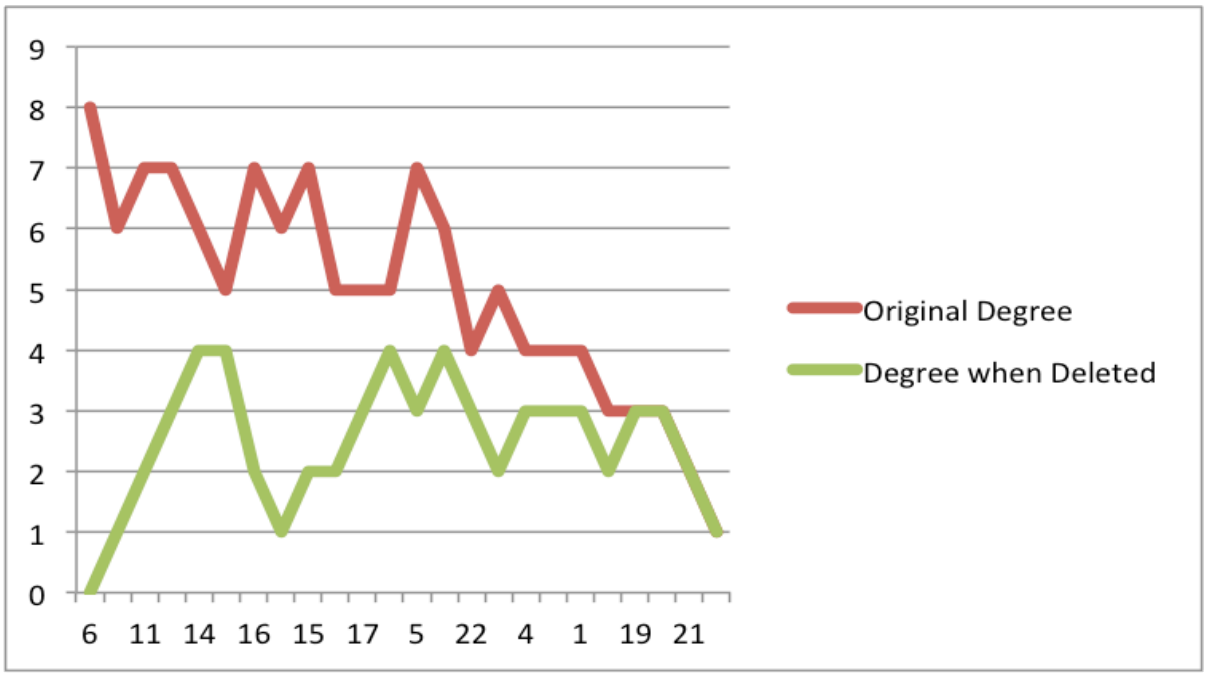
```
color 0: null
color 1: [Node 0, Node 21, Node 3, Node 19, Node 22, Node 8, Node 9, Node
14]
color 2: [Node 20, Node 1, Node 4, Node 23, Node 17, Node 10]
color 3: [Node 2, Node 18, Node 7, Node 11]
color 4: [Node 12, Node 5, Node 13]
color 5: [Node 15, Node 6]
color 6: [Node 16]
color 7: null
color 8: null
```

Here we display the backbone bipartite subgraph illustrating the domination provided by the first color set and the planarity of the bipartite subgraph.

The following picture shows the subgraph that contains the nodes with color set (1,2) and the edges between them.



Here is the plot for the degree when deleted values and the original degree values for the vertices ordered by the Smallest Last order.



2.1.6 Algorithm Effectiveness:

We hereby present the algorithm effectiveness by showing the graph coloring summary table, which is included in the project statement.

Graph Coloring Summary Table

G=Graph type (sp for sphere, sq for square)

N=number of vertex.

R=radius.

Min=Minimum degree

Max=Max degree

NE=number of edges.

AVG= average degree

MD=maximum degree when deleted

NC=number of colors.

MC1= max color class 1 size.

MC2= max color class 2 size

LC= largest component size

BE= number of edges in largest connected bipartite subgraph

NF= number of faces in sphere

	G	N	R	Min	Max	NE	AVE	MD	NC	MC1	MC2	LC	BE	NF
1	sq	400	0.2	9	65	8278	41.39	27	38	25	22	47	42	
2	sq	1000	0.12	8	61	20441	40.89	26	34	60	56	116	97	
3	sq	2000	0.12	20	112	82036	82.0	49	65	63	59	122	121	
4	sq	4000	0.08	24	113	150036	75.02	45	61	124	125	249	231	
5	sq	10000	0.06	26	158	534914	107.0	63	84	220	216	436	396	
6	sq	20000	0.04	30	137	961862	96.19	58	77	479	478	957	866	
7	sp	1000	0.12	0	18	3997	7.99	10	8	411	257	668	387	2999
8	sp	2000	0.12	0	18	6733	6.04	10	13	521	422	943	704	4735
9	sp	10000	0.04	0	12	16254	3.25	7	9	4163	2663	6826	3826	6256
10	sp	20000	0.04	0	15	64951	6.50	11	13	5286	4326	9612	40	44953
11	sq	40000	0.029	27	138	1928256	96.41	58	79	940	944	1884	792	
12	sq	40000	0.03	31	154	2176341	108.8	64	86	844	833	1677	392	
13	sq	60000	0.022	23	126	2536600	84.55	52	73	1600	1602	3202	324	

14	sp	20000	0.06	0	36	156920	15.69	16	21	2573	2419	4992	179	136922
15	sp	20000	0.08	0	51	289645	28.46	20	32	1479	1437	2916	477	269647
16	sp	20000	0.12	0	100	673837	67.74	40	57	653	655	1308	381	653839
17	sp	20000	0.2	0	204	1620038	162.0	116	115	282	274	556	234	1600040
18	sp	40000	0.16	0	290	4622726	231.1	178	160	396	390	786	363	4602728
19	sp	80000	0.11	0	274	8654520	216.4	162	153	839	843	1682	584	8574522
20	sp	100000	0.075	0	179	6481498	129.6	89	97	1738	1718	3456	422	6381500

Conclusion and Analysis

The strengths and weaknesses of the RGG generation, coloring, and bipartite backbone selection algorithms (BBSAs), are shown in the following table, and the implementation in another table.

	RGG generation	**Coloring**	**BBSAs**
Strength	1.The use of sweeping method facilitates the edge generation; 2.random numbers generated by random seed	1.The use of color list structure; 2.specific methods for coloring process in different stages	1.Meets all the criteria and requirements 2.Has the full functionality and effectiveness
Weakness	Not as fast as the “cell” method	Colors are not evenly distributed for some benchmarks	Could not product largely connected bipartite sub-graphs;

	Implementation
Strength	For input with a relatively large number, a medium threshold within a small range could make a competitive result
Weakness	For input with fairly small threshold, the backbone mainly consists with distributed “islands” with small number of vertices connected

According to the practices that have been experimented and the result data, we found the biggest connected bipartite appeared at the (2000,0.12) benchmark for square, and for sphere it appeared at the (40000,0.16) benchmark.

So from the author’s perspective, there may be some kind of linear formula for the number of vertices and the threshold to produce the largest connected bipartite sub-graph.

Therefore, we could do more test sets and perform some linear regression, or curvilinear regression to optimize the formula, as one of the further study directions.

For the bipartite sub-graph rotating purposes, there are enough color sets to choose from and the coverage does not change much for most benchmark data sets. However, most of them have multiple “islands”, meaning the nodes are not grouped in one connected sub-graph, so we need to set up signal sources in different “islands” to facilitates the communication. In which way could there be more nodes connected to one bipartite sub-graph would be one of the future fields of study as well.

2.2 Benchmark Result Summary (80 points)

As required in the project statement, following results are provided in this section for all 10 benchmarks listed in the project statement, every benchmark has 9 pictures:

1. Degree Distribution:

   X coordinates for the degree values 1,2…maxdegree, y coordinates for numbers of nodes.

2. RGG Display

   For Benchmarks 1-3, the points in the x, y plane are drawn (since the average degree is greater than 30, the edges are not drawn). For Benchmarks 5-6 only the points were shown. For the sphere display (Benchmarks 7-10), the points projected on the x, y plane corresponding to positive z values (a hemisphere) are drawn. The edges were drawn for Benchmark 7.

3. Sequential Coloring Plot

   X coordinates for the "ID"s of the nodes, y coordinates for the degrees. Red is for the original degrees, and green is for the degrees when deleted.

4. Color Class Size Distribution

   X coordinates for the color number 1,2…maxcolor, y coordinates for the number of nodes.

5. Benchmark Selection Procedure and Display

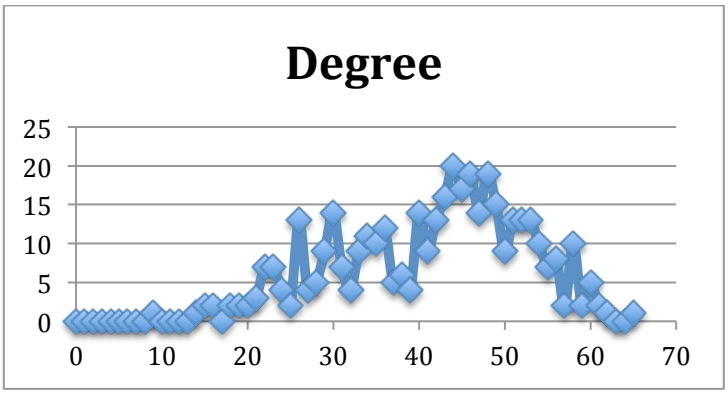
   The nodes with color set (1,2) and the edges between them are displayed.

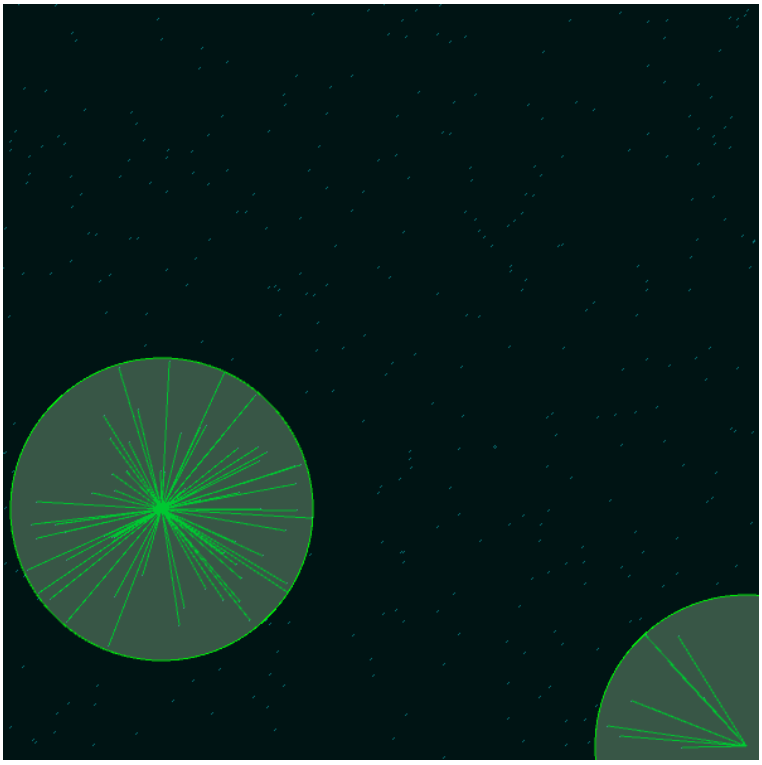
6. Largest Connected Component Display

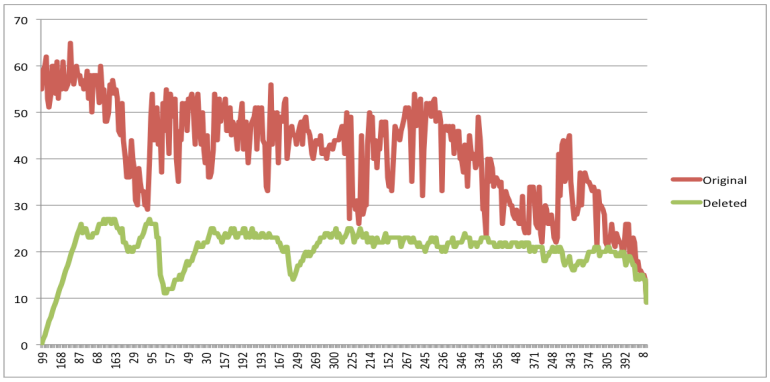
7. 3 Other Color Sets for each Benchmark

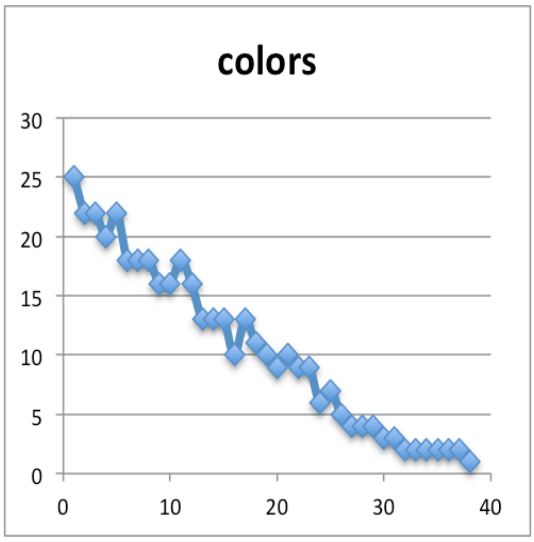
   As required by the project statement, there should be 4 color pairs displayed, mainly based on the coverage, connectivity, and the number of isolated "islands".

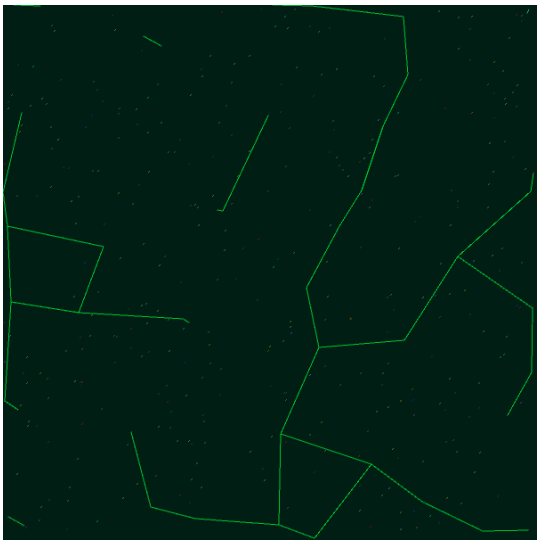
Benchmark 1: (N=400, R=0.20); average Degree is 41. Backbone coverage is 1.0.

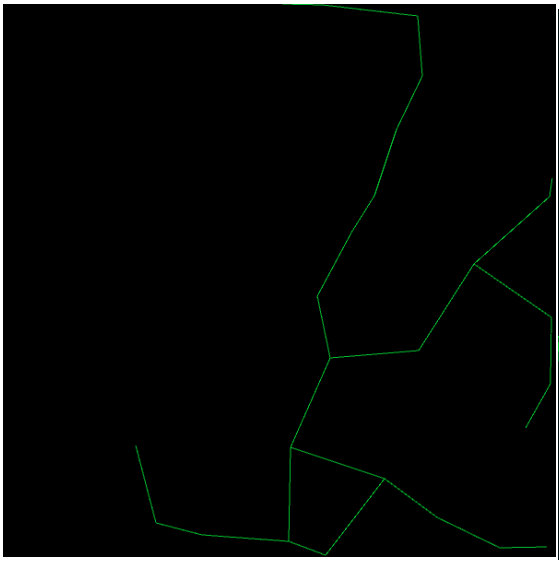


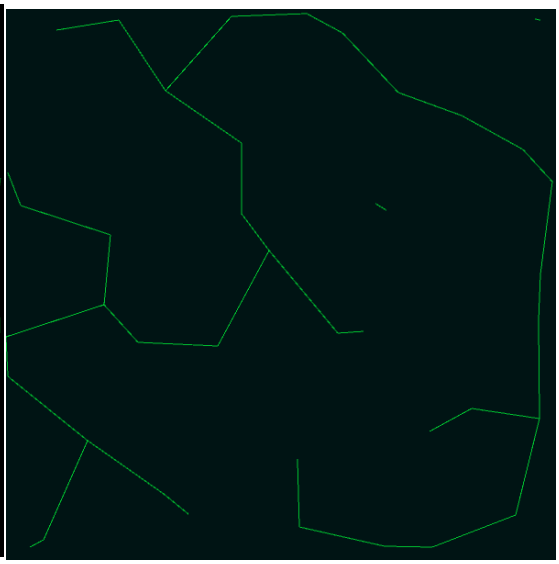


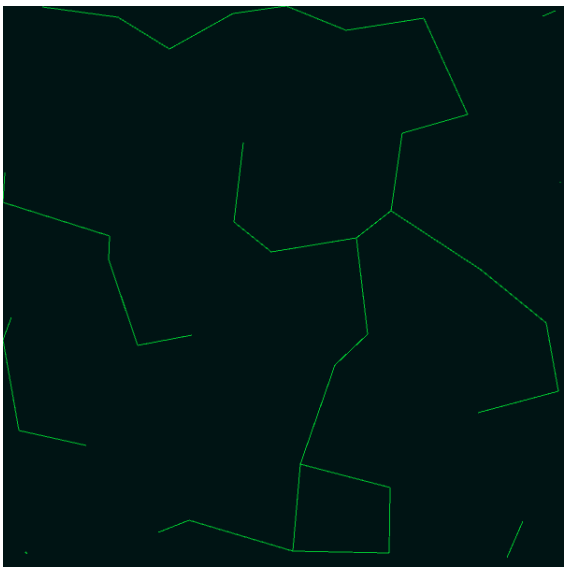


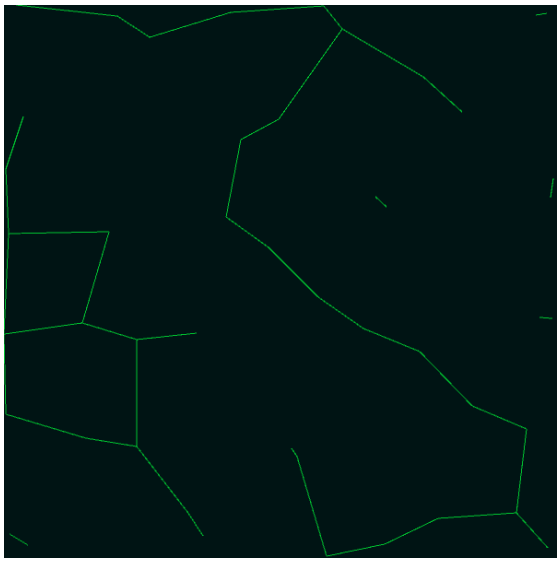

colors
30
25
20
15
10
5
0
0
10
20
30
40



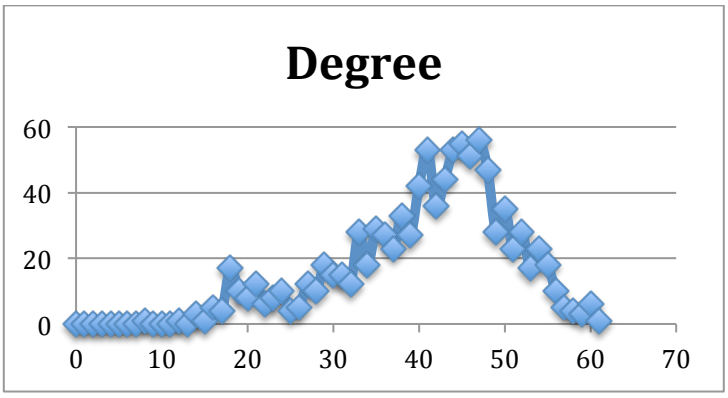


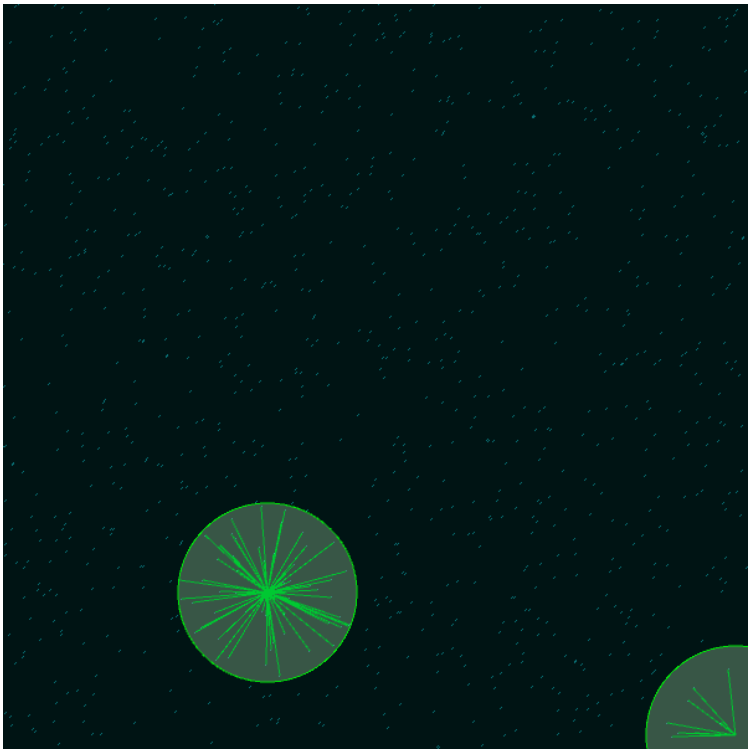


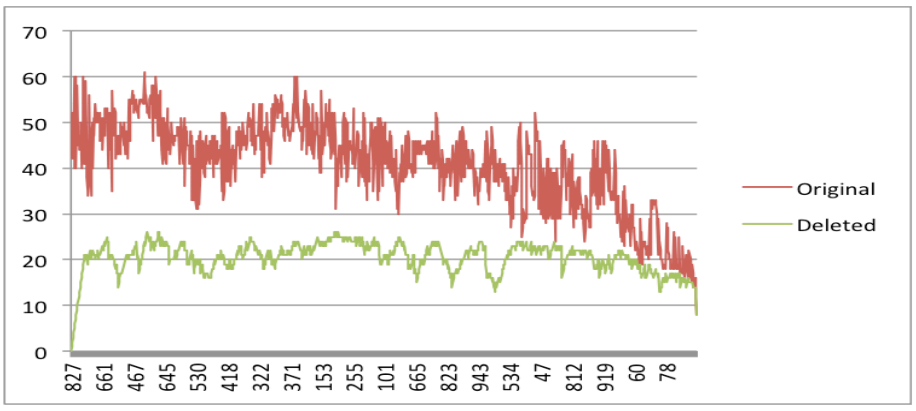


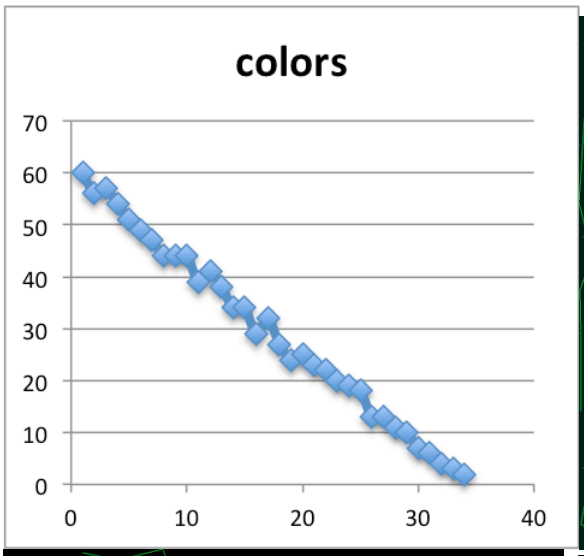


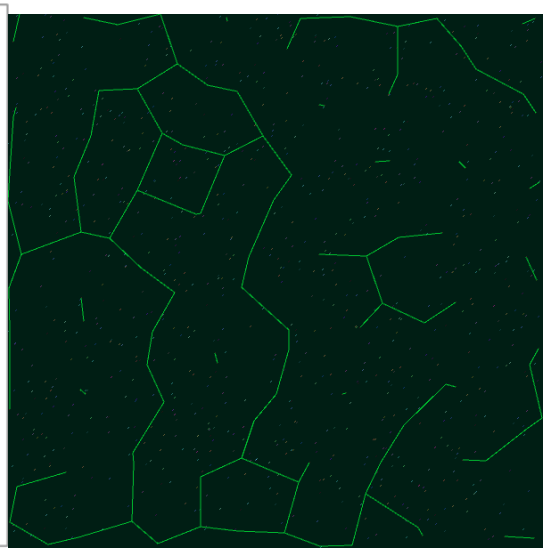
Benchmark 2: (N=1000, R=0.12); average Degree is 40. Backbone coverage is 1.0.

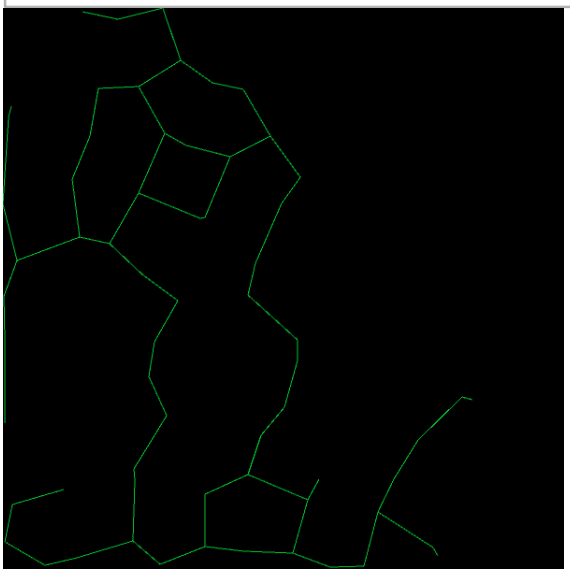


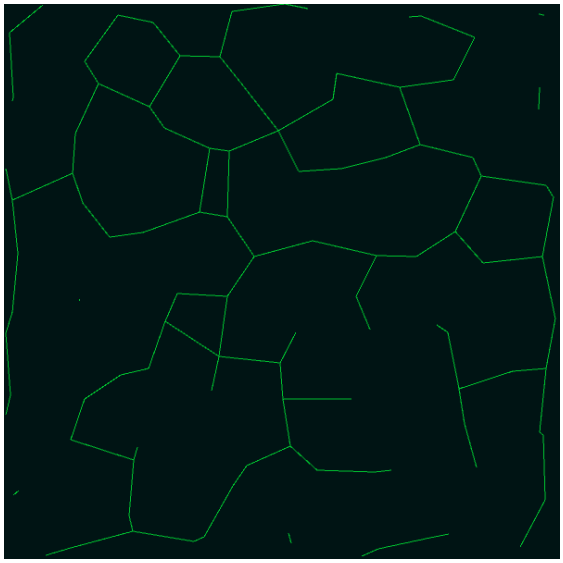


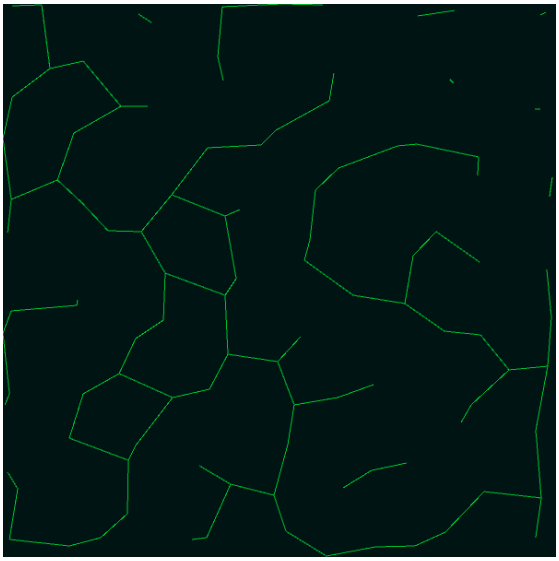


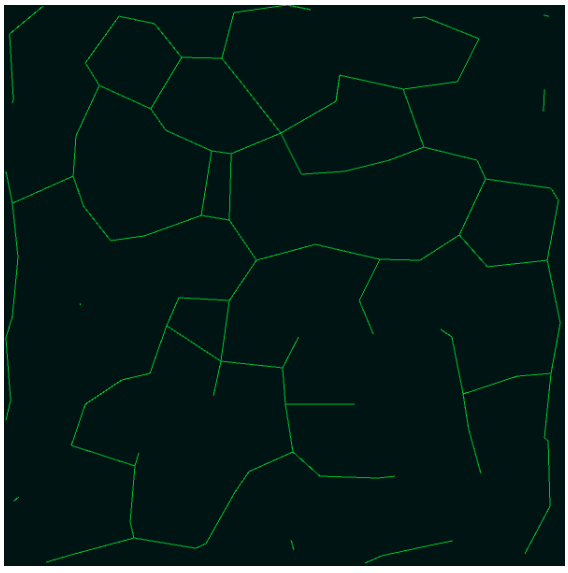

colors
70
60
50
40
30
20
10
0
0
10
20
30
40



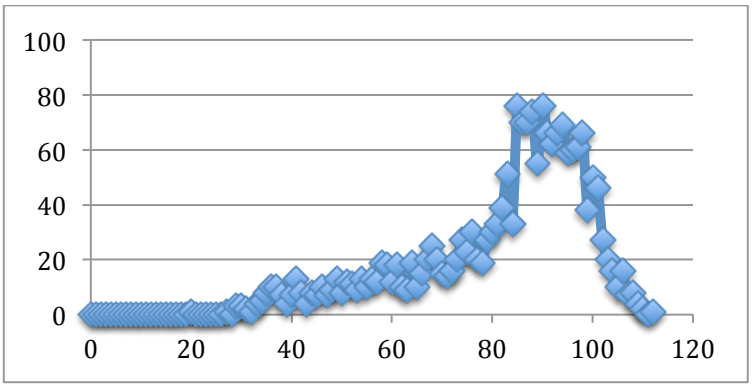


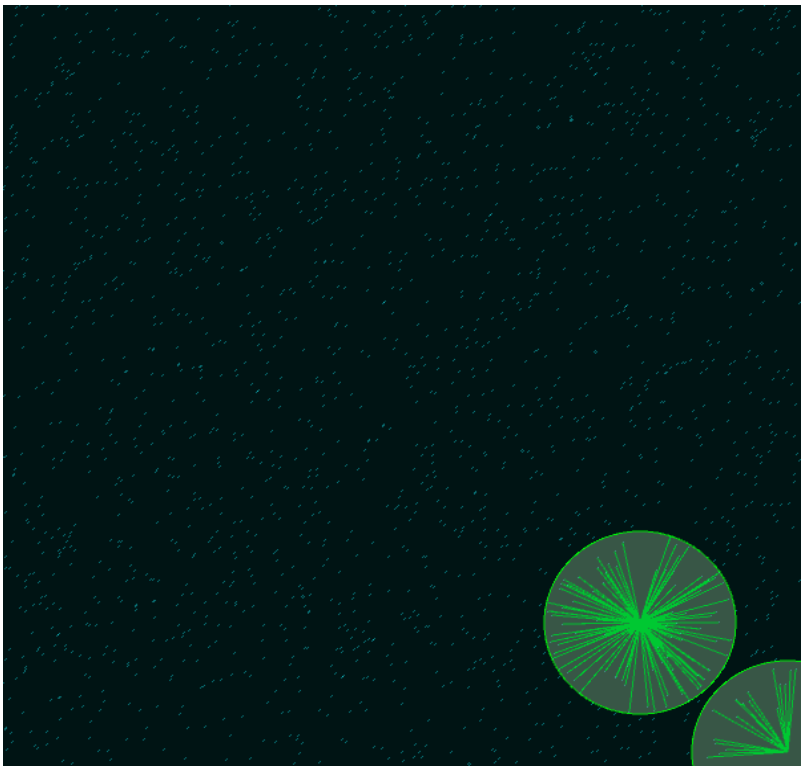


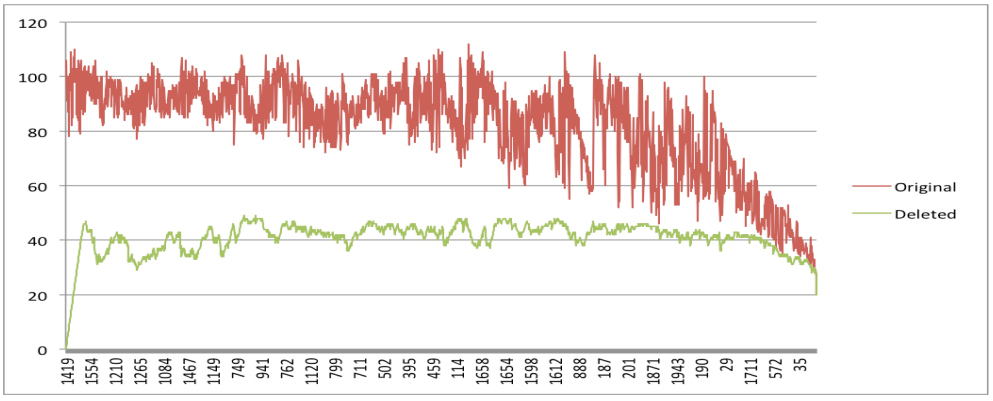


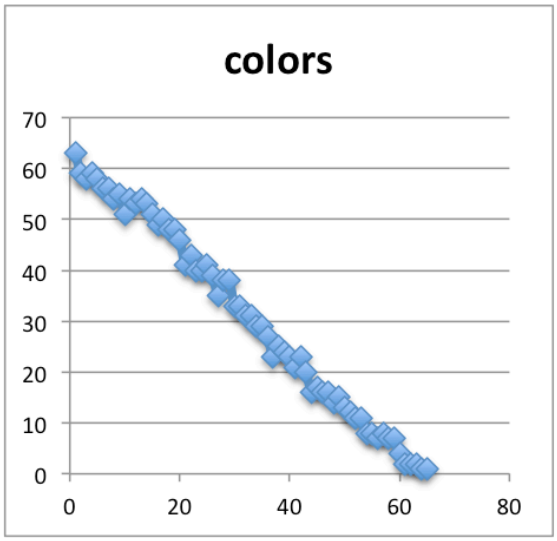


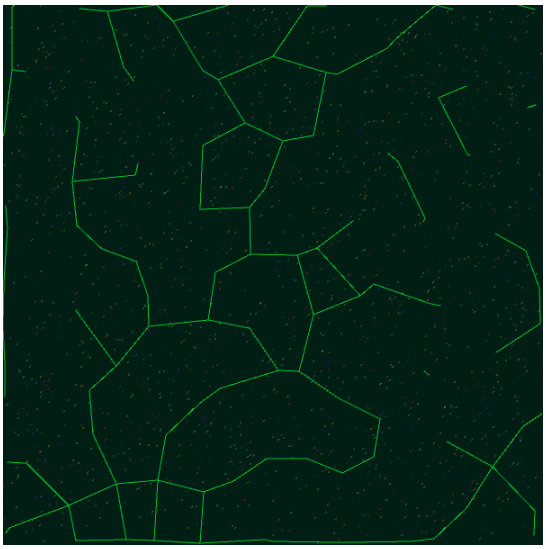
Benchmark 3: (N=2000, R=0.12); average degree is 82. Backbone coverage is 1.0.

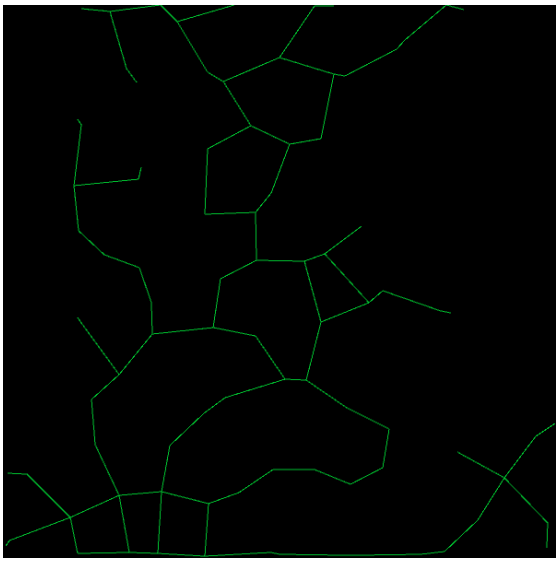


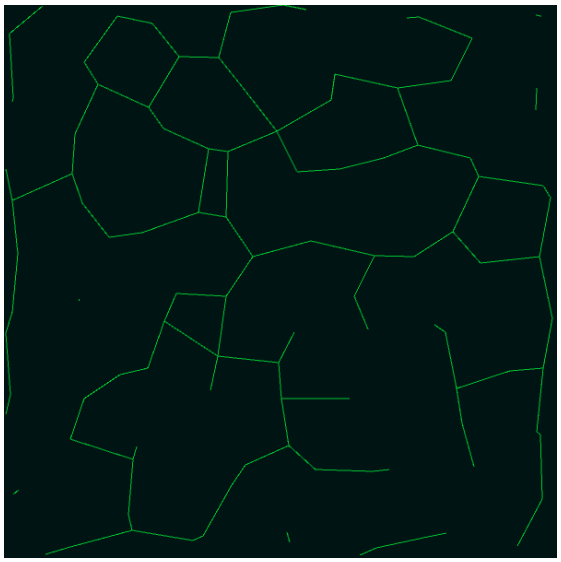


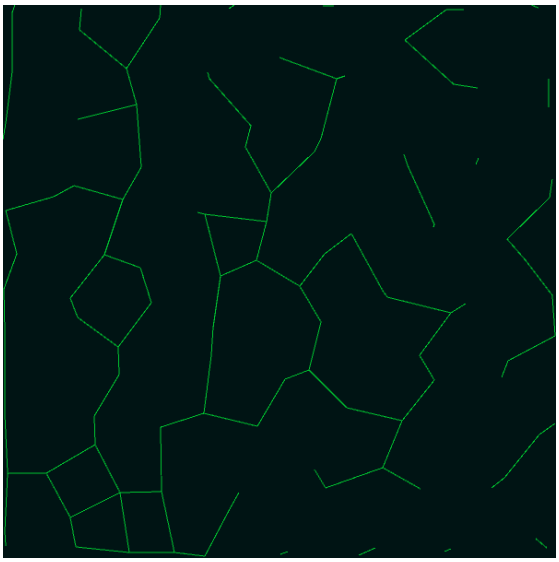


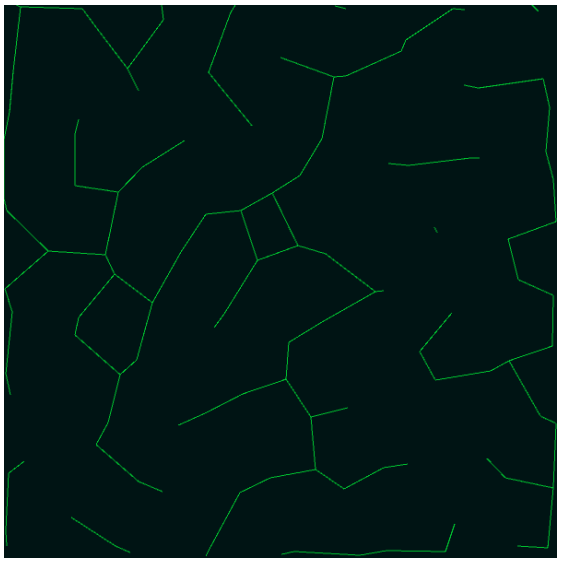

colors
70
60
50
40
30
20
10
0
0
20
40
60
80



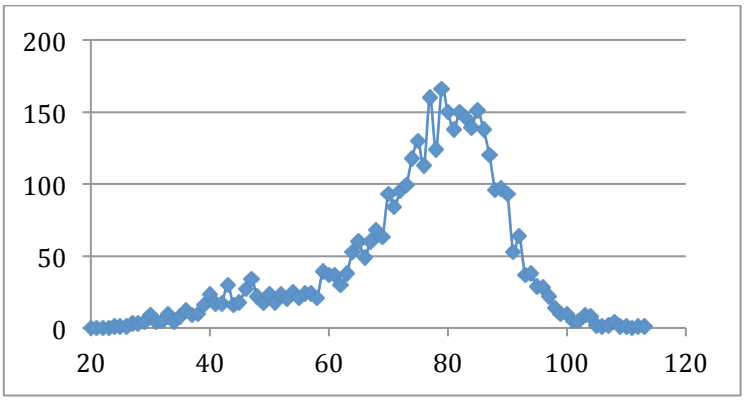


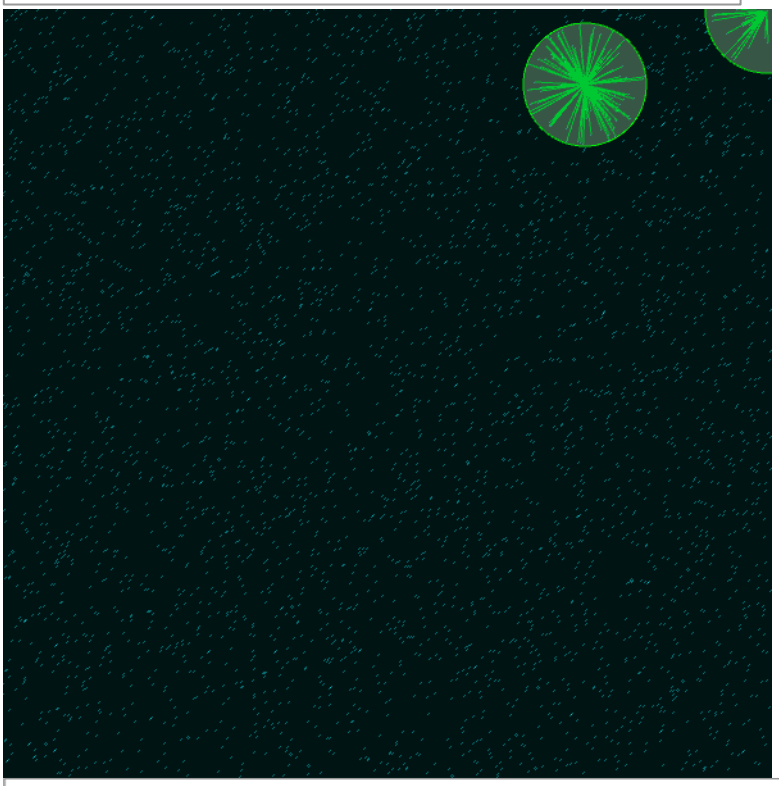


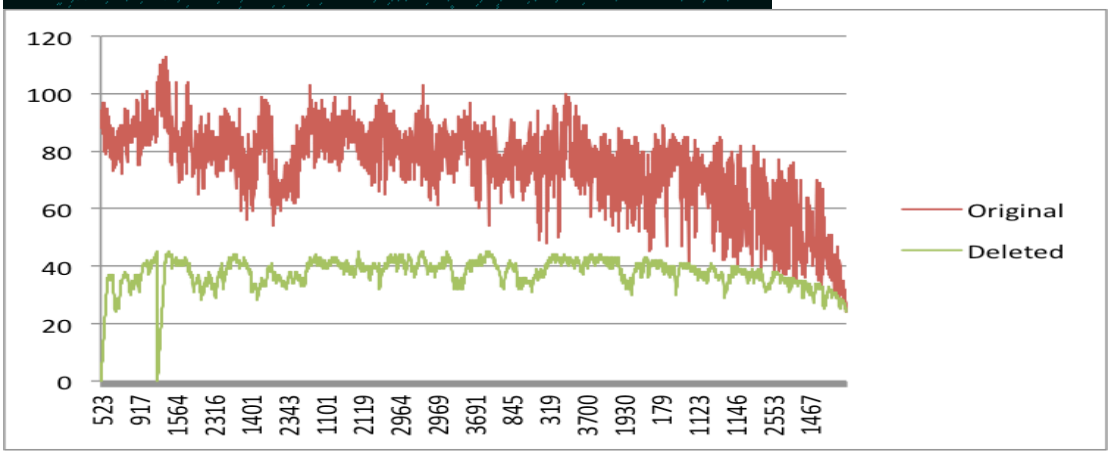


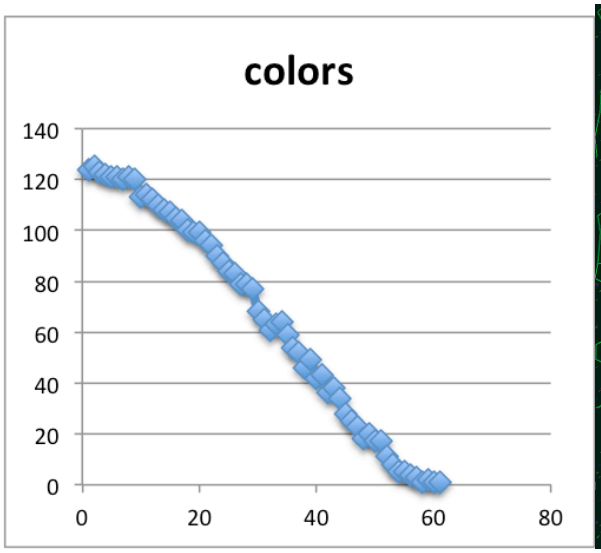


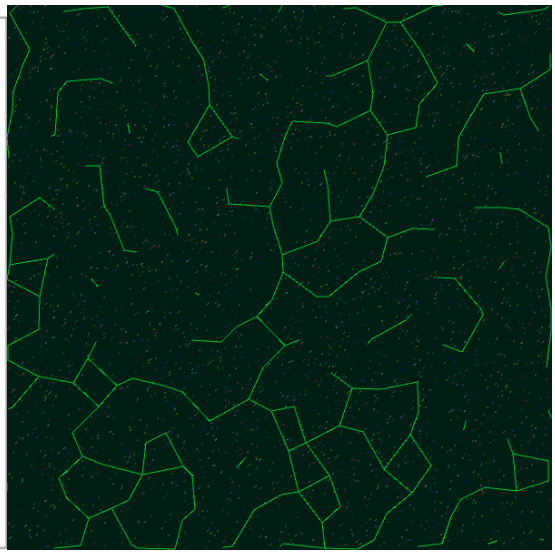
Benchmark 4: (N=4000, R=0.08) ; average degree is 75. Backbone coverage is 1.0.

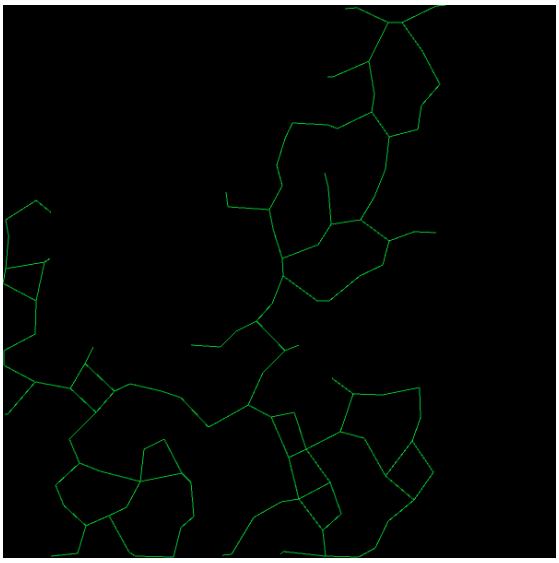


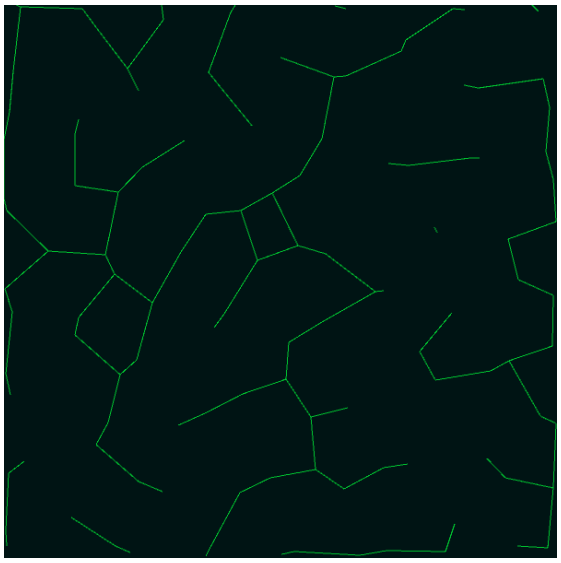


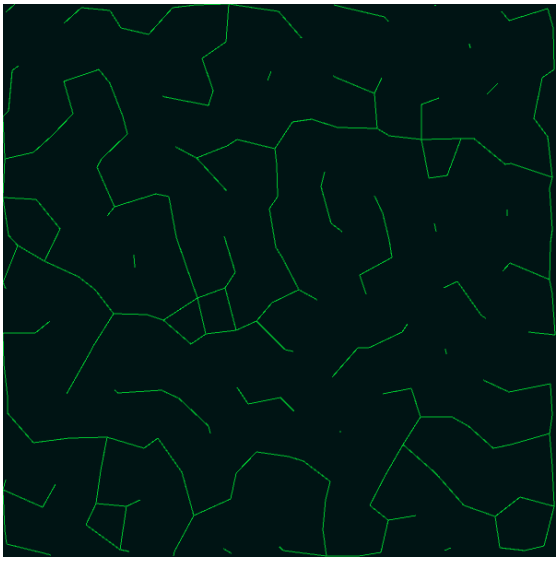


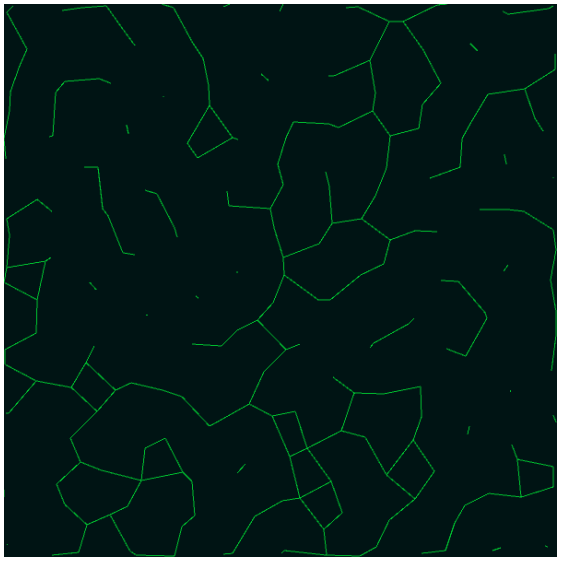

colors
140
120
100
80
60
40
20
0
0
20
40
60
80



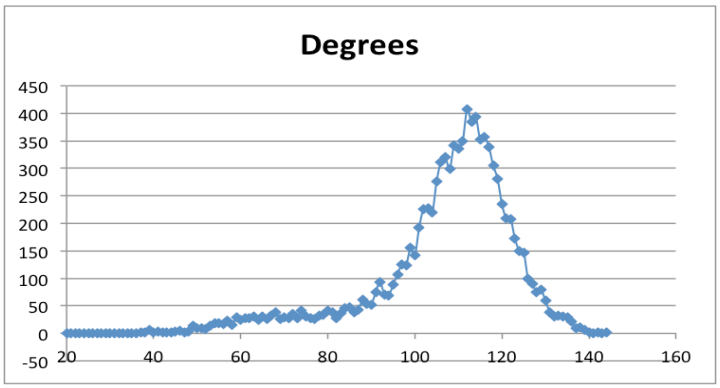


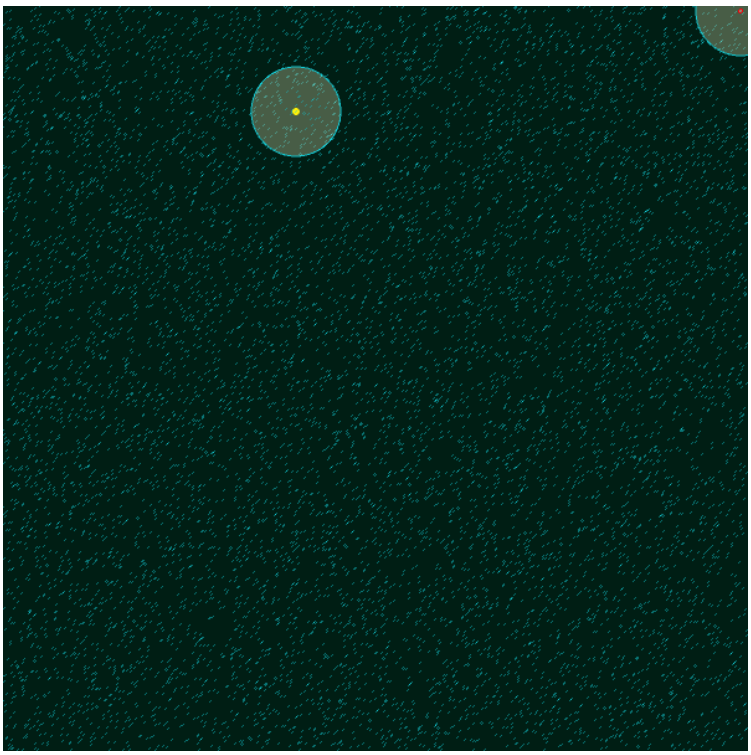


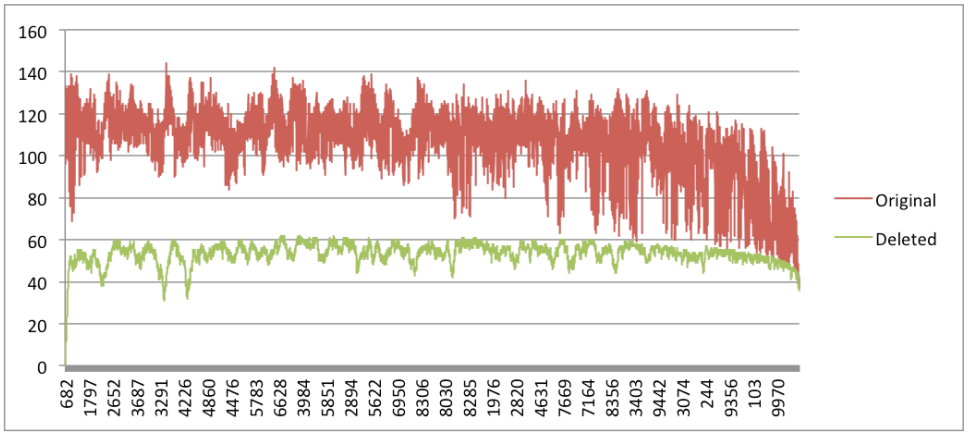


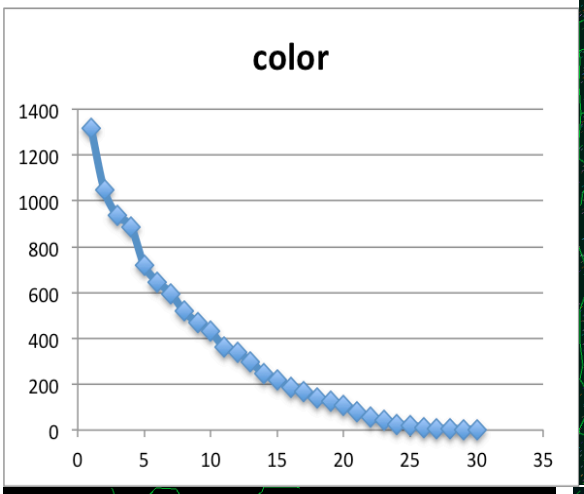


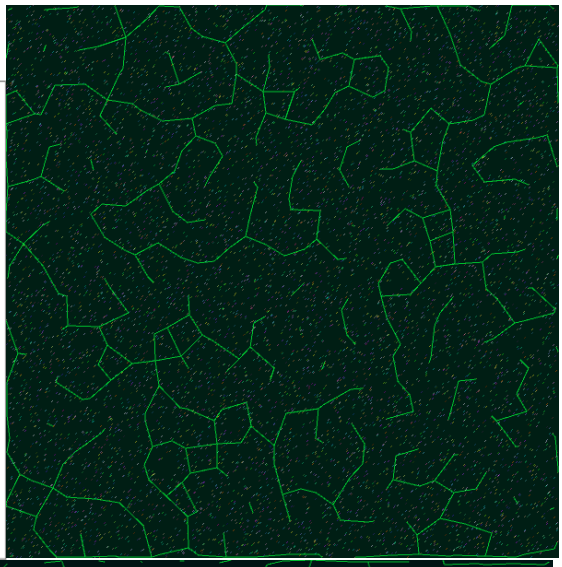
Benchmark 5: (N=10000, R=0.06); Backbone coverage is 1.0

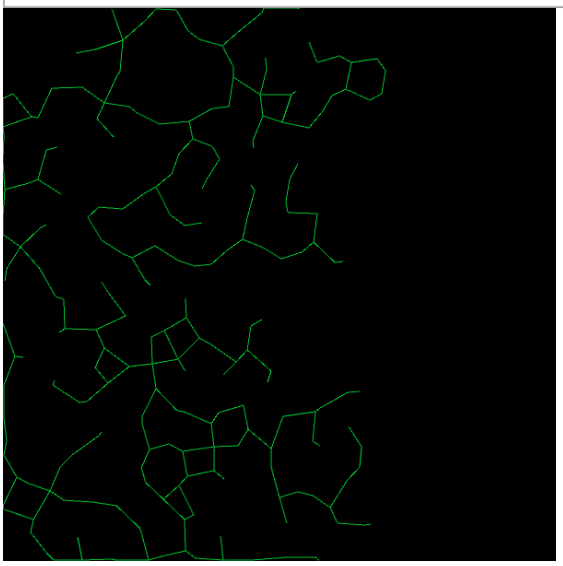


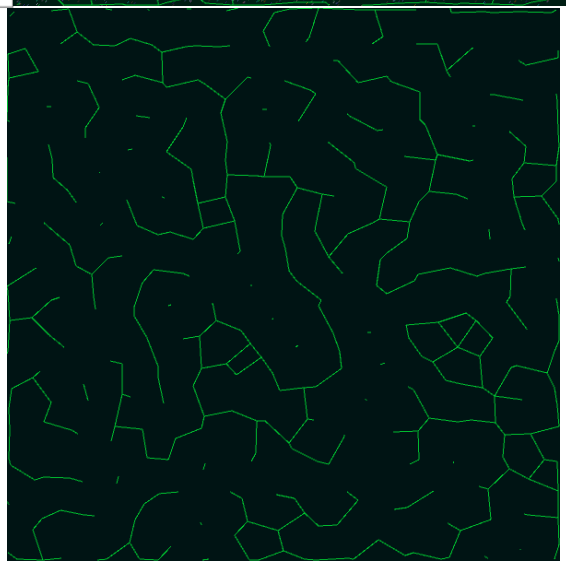


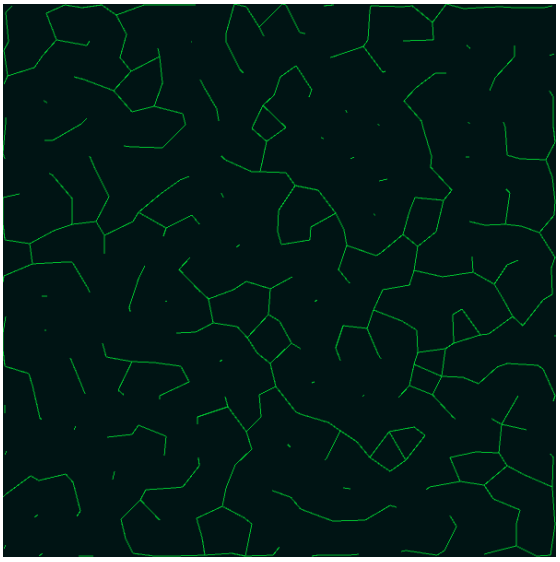


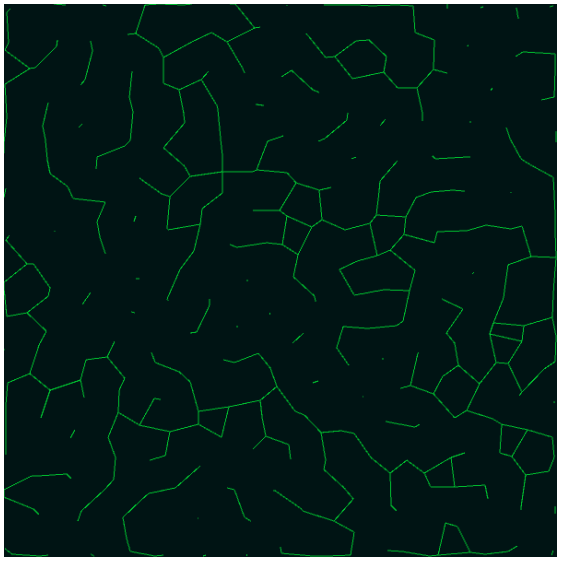

color
1400
1200
1000
800
600
400
200
0
0
5
10
15
20
25
30
35



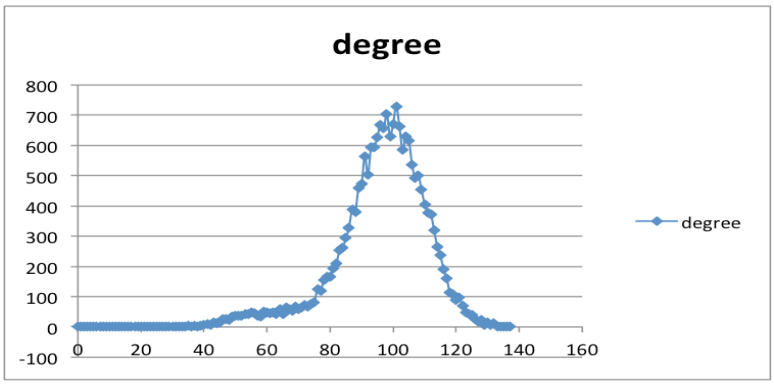


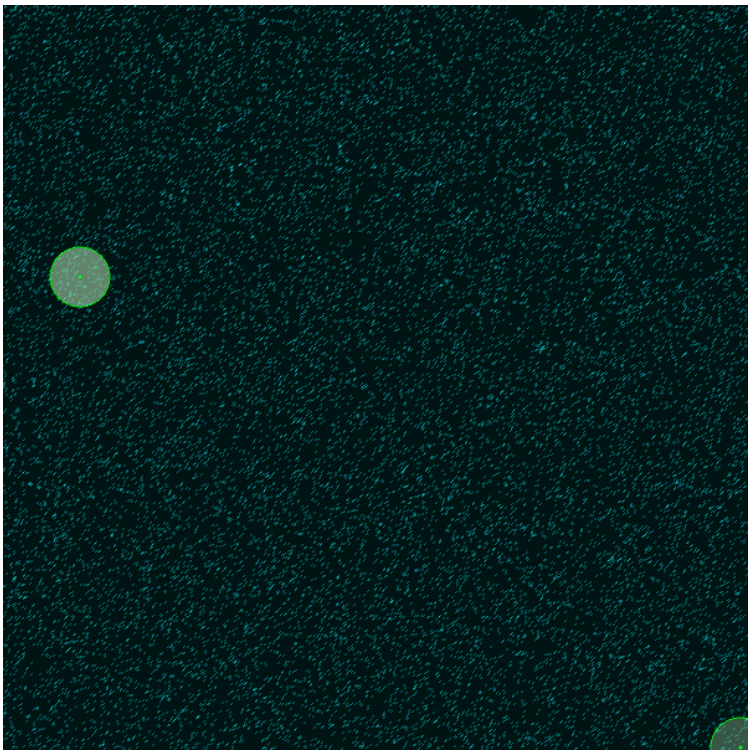


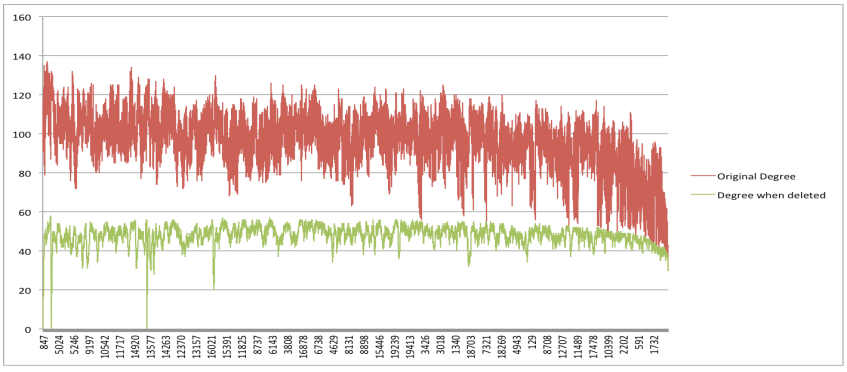


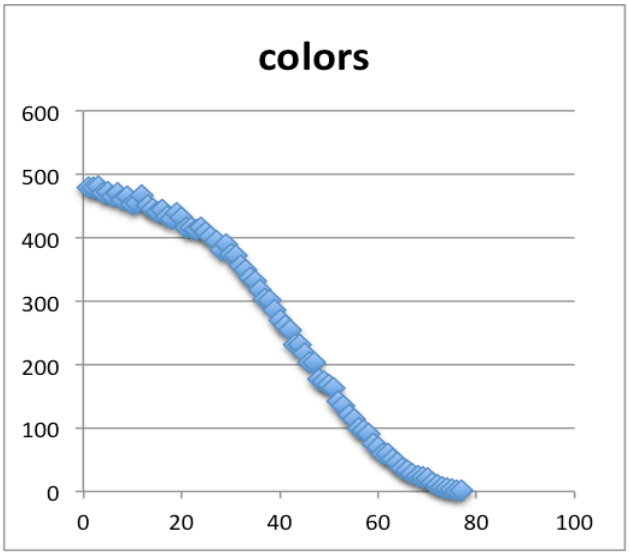


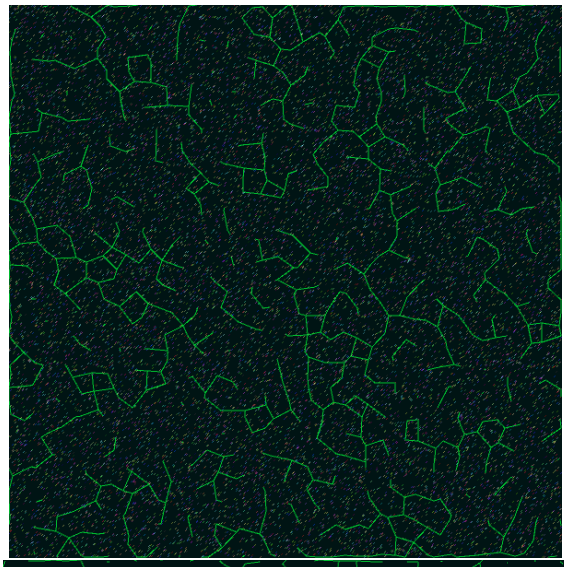
Benchmark 6: (N=20000, R=0.04); Backbone coverage is 1.0

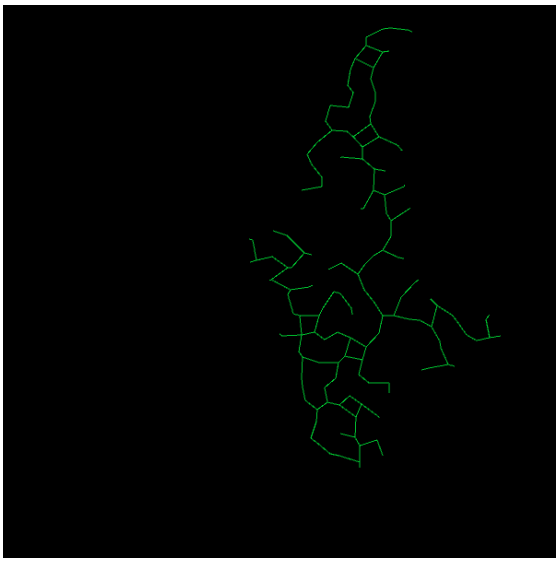


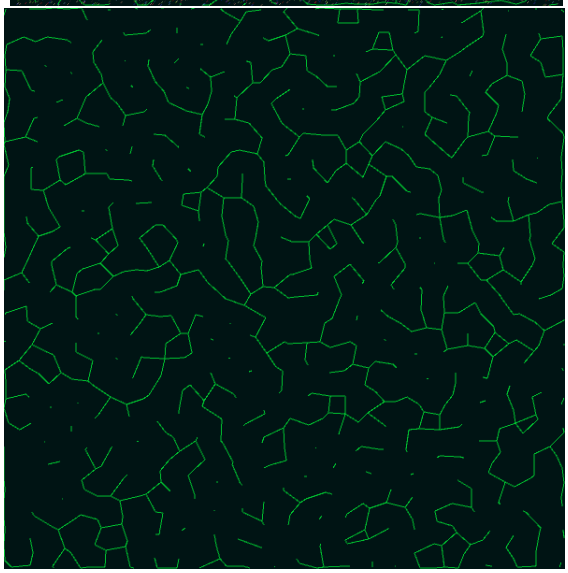


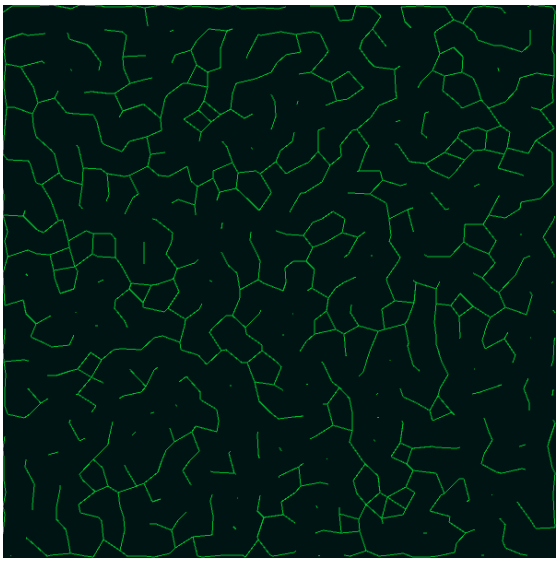


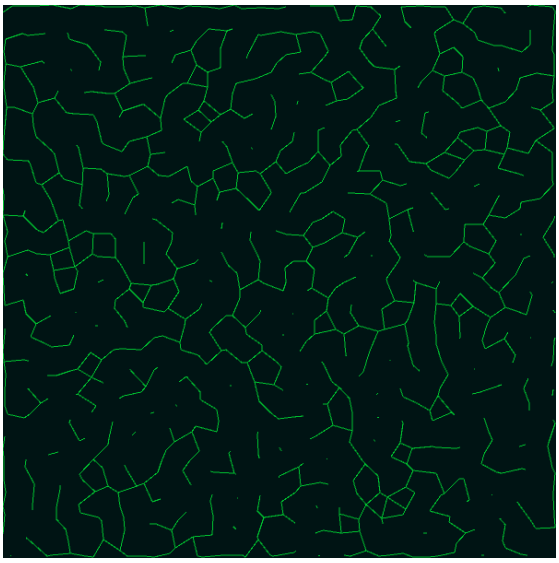

colors
600
500
400
300
200
100
0
0
20
40
60
80
100



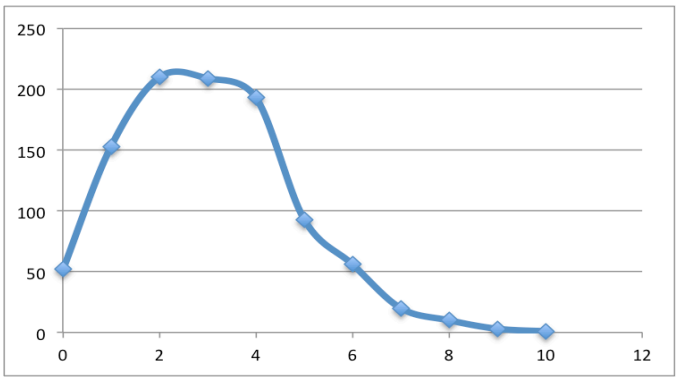


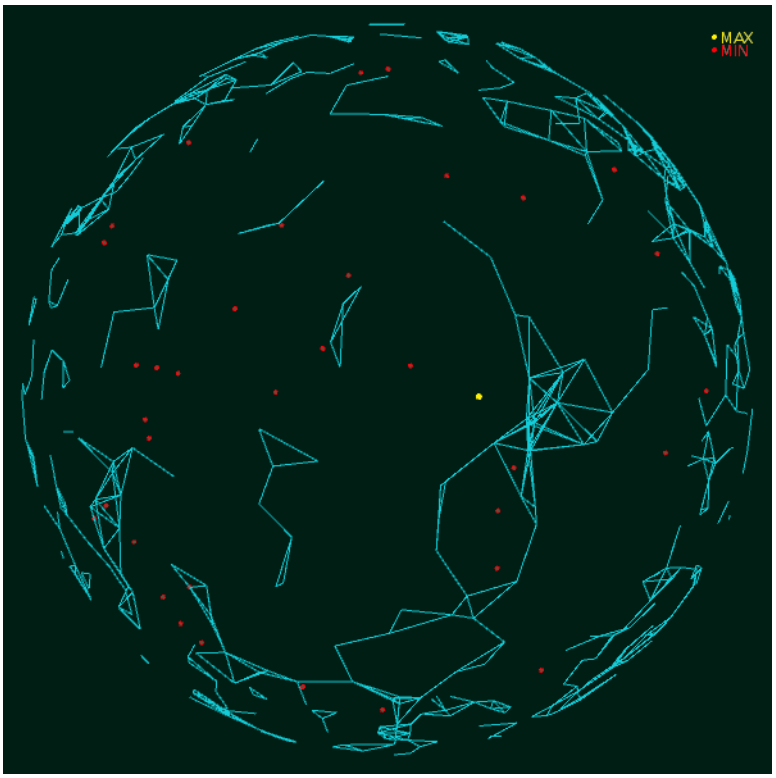


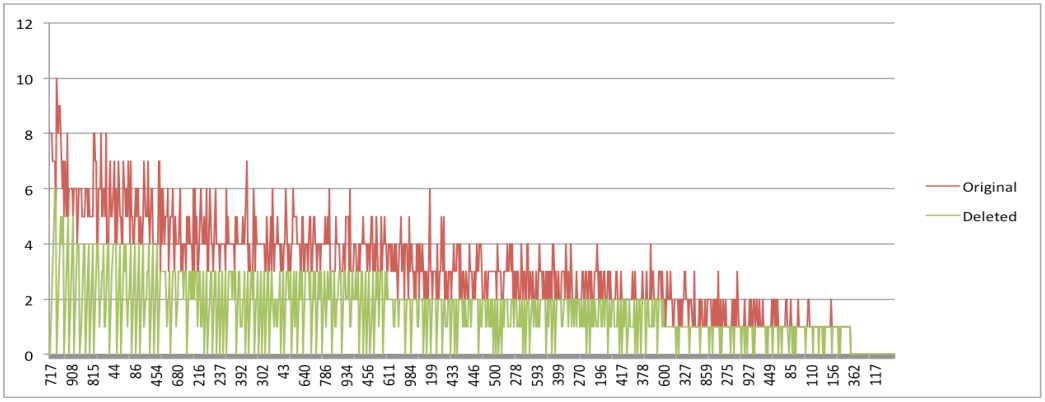


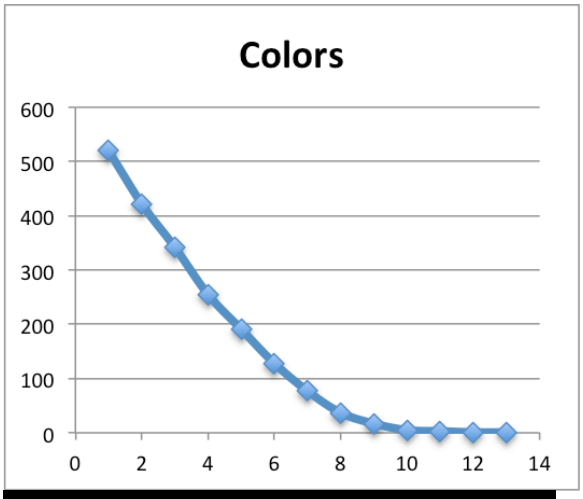


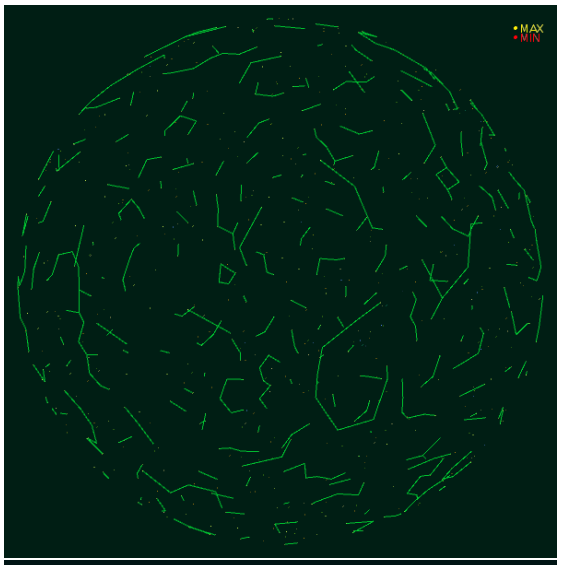
Benchmark 7: (N=1000, R=0.12) Hemisphere backbone coverage is 0.91

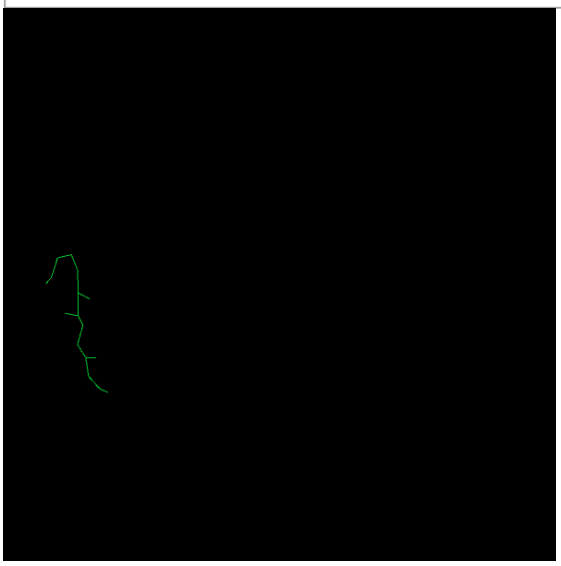


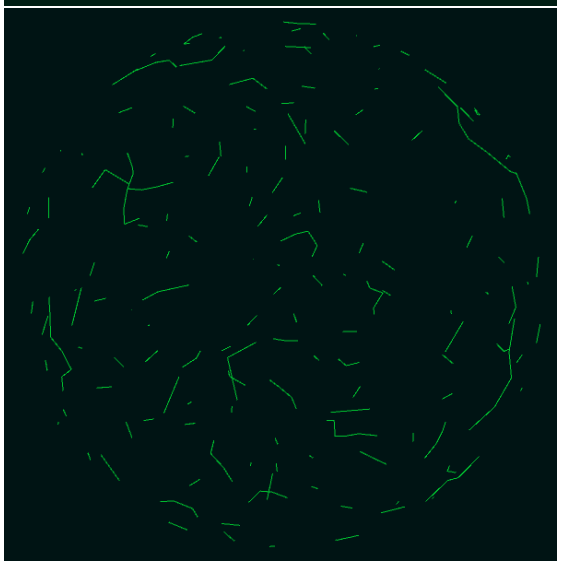


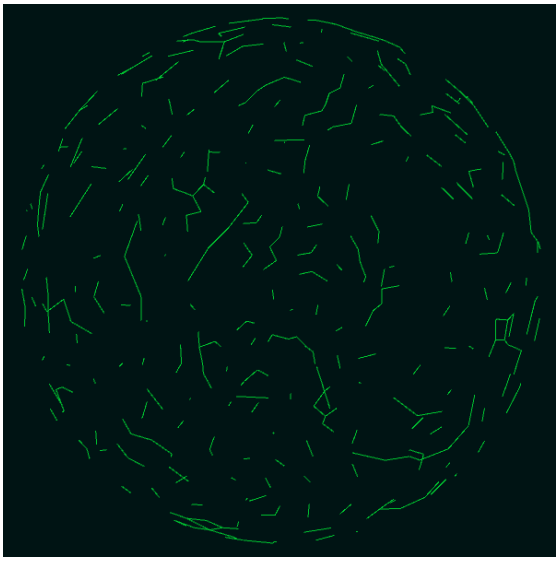


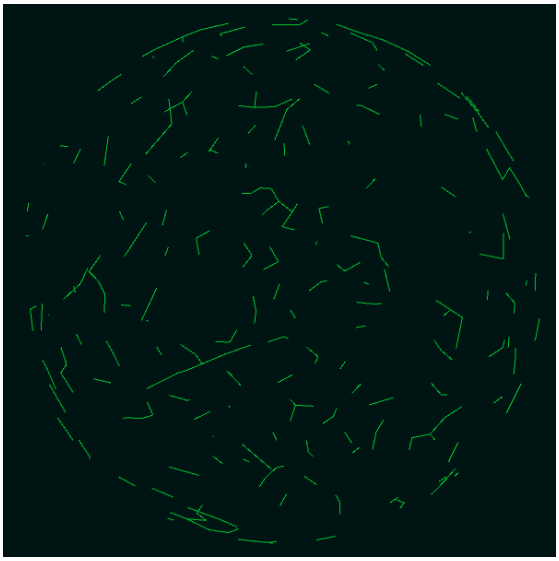

Colors
600
500
400
300
200
100
0
0
2
4
6
8
10
12
14


MAX
MIN

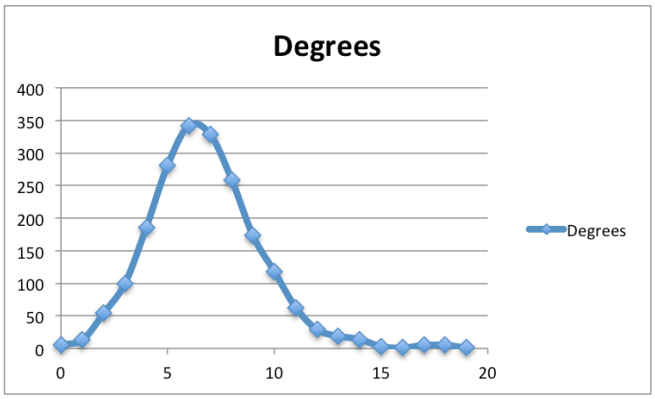


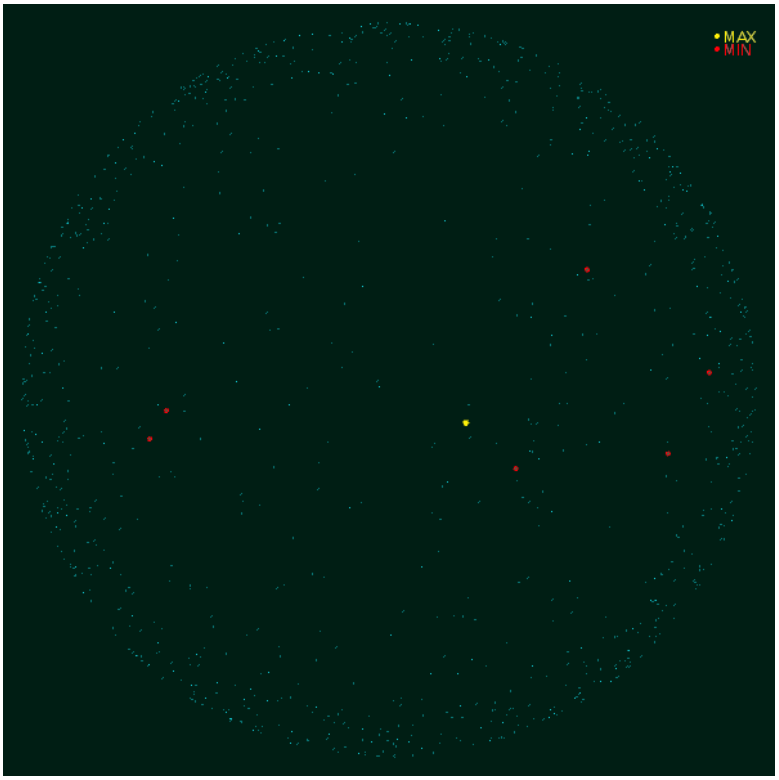


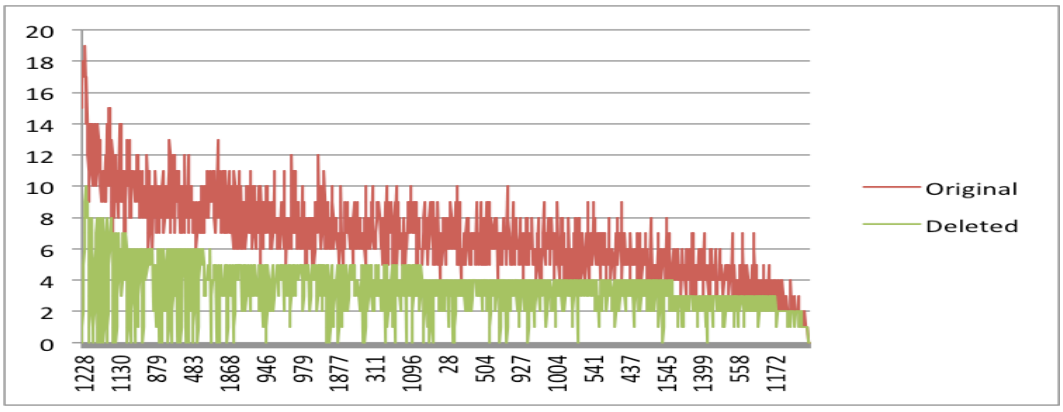


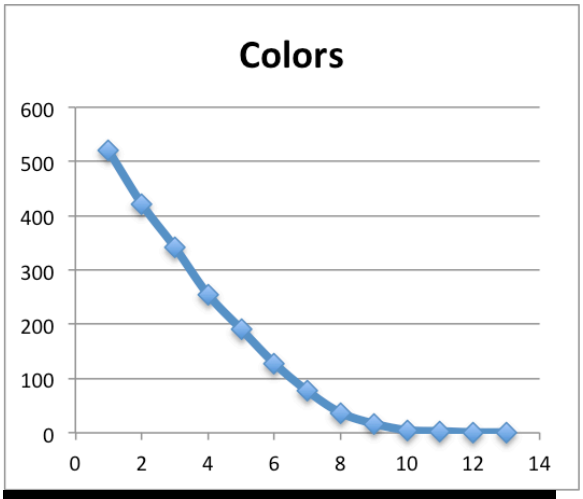


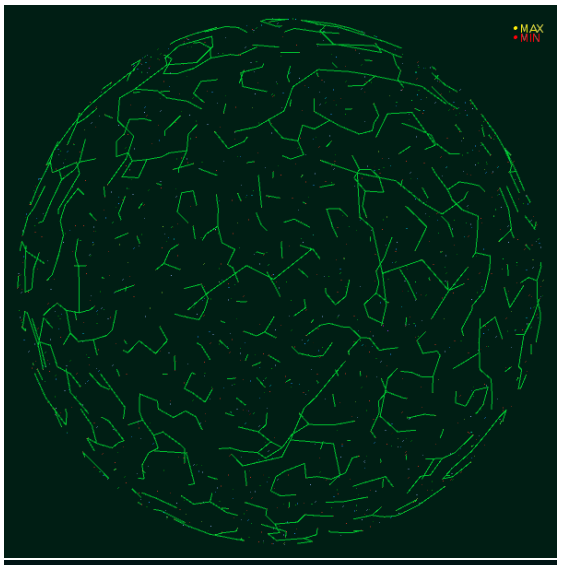
Benchmark 8: (N=2000, R=0.12) Hemisphere backbone coverage is 0.98

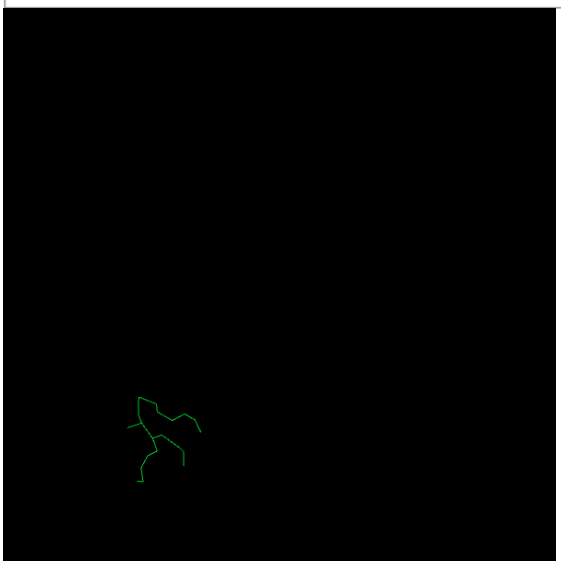


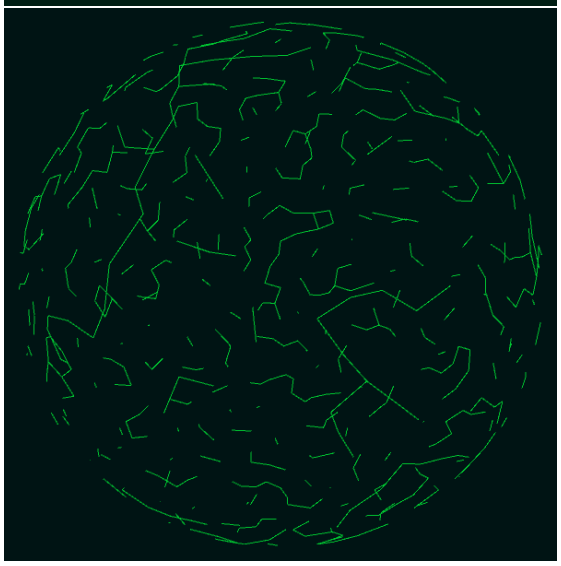


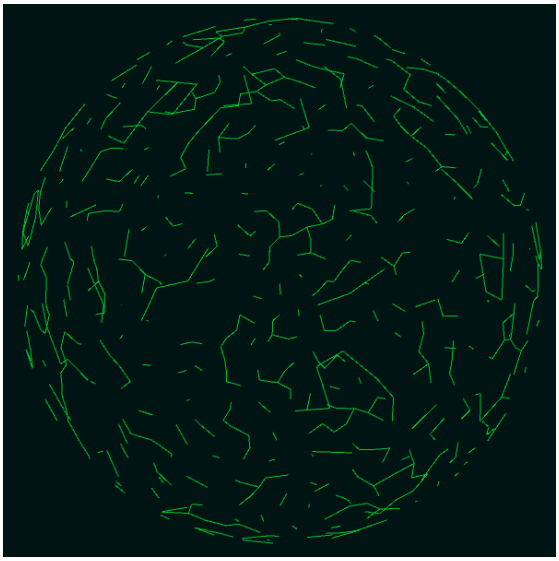


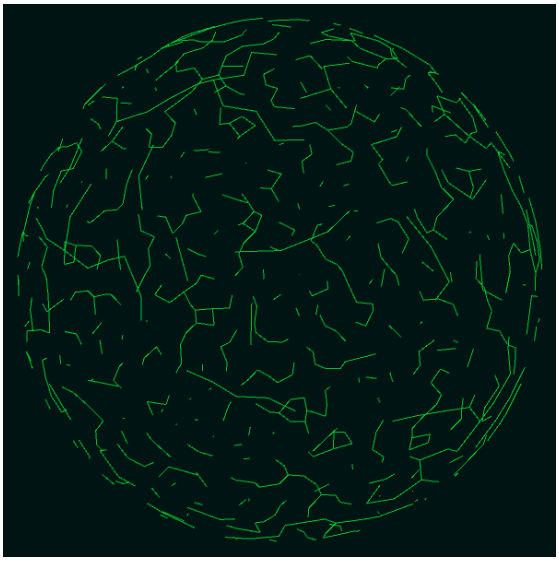

Colors
600
500
400
300
200
100
0
0
2
4
6
8
10
12
14


MAX
MIN

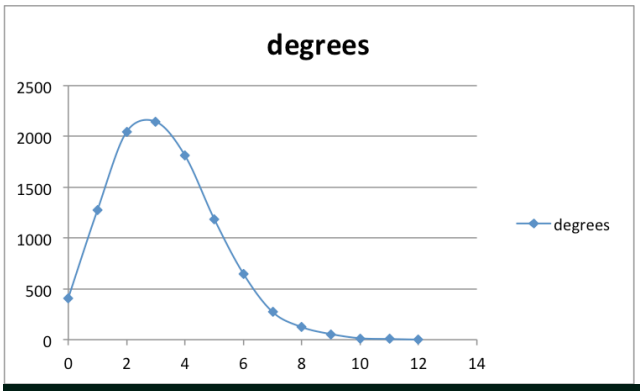


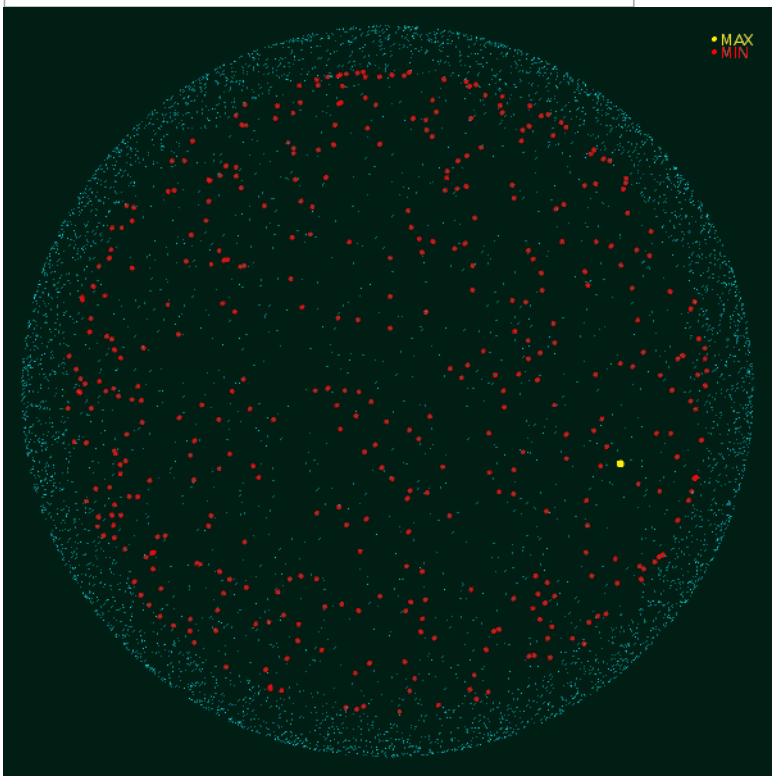


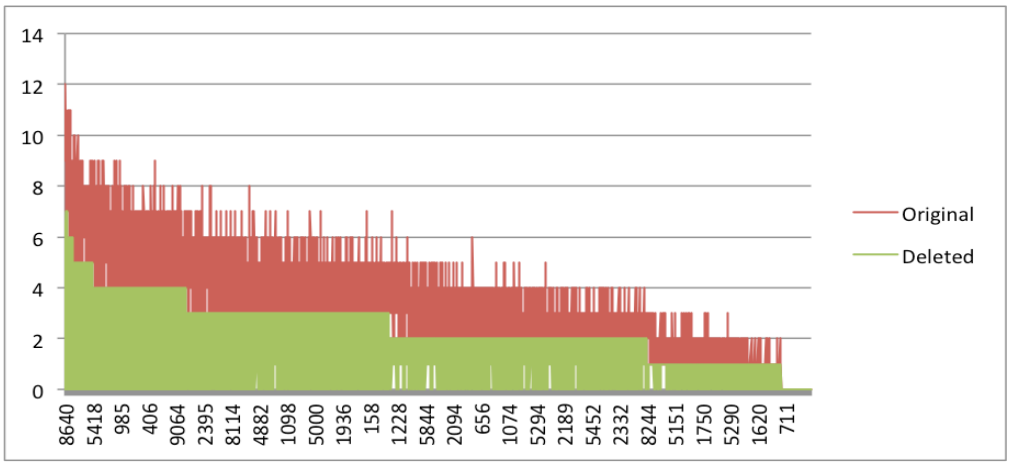


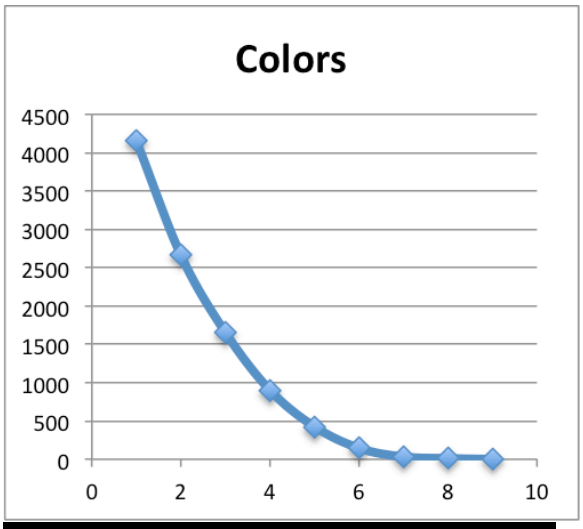


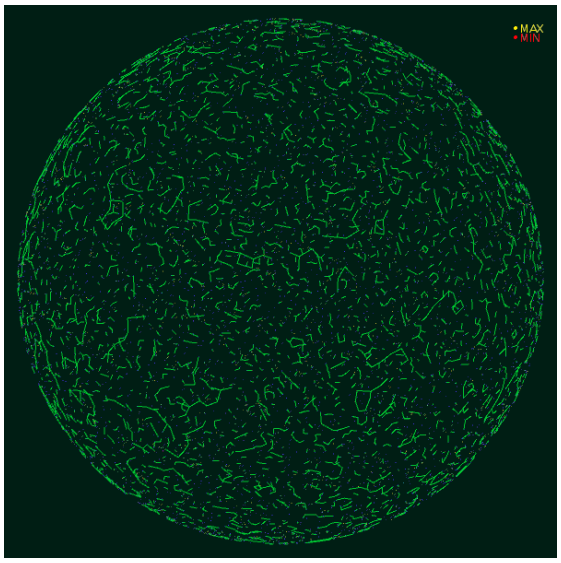
Benchmark 9: (N=10000, R=0.04) Hemisphere backbone coverage is 0.91

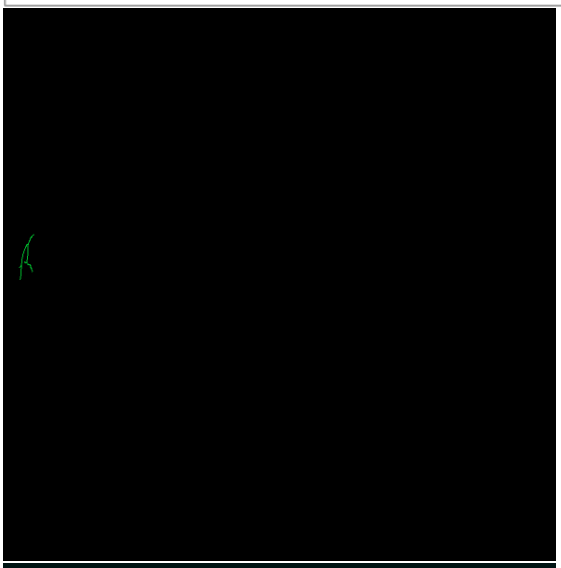


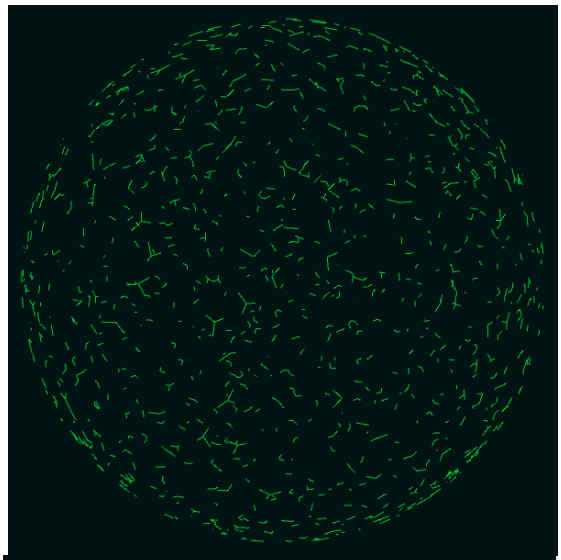


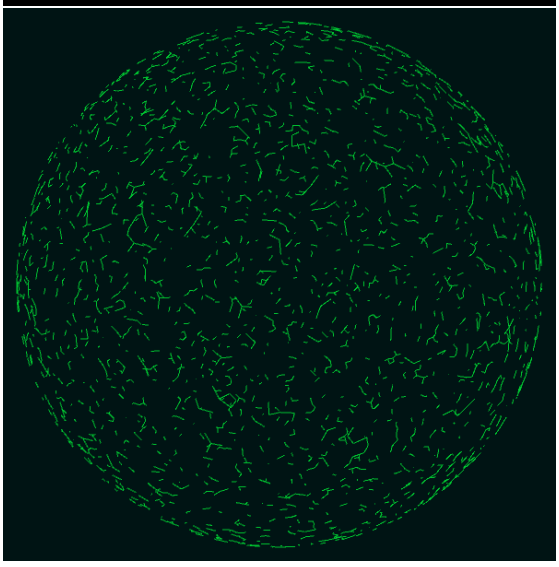


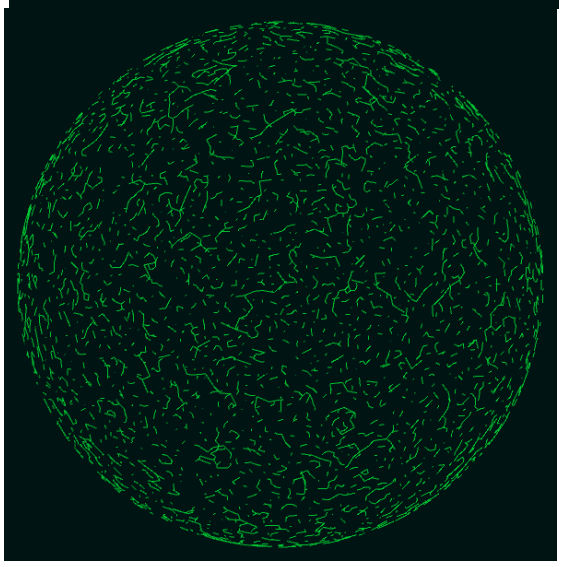

Colors
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
2
4
6
8
10


MAX
MIN

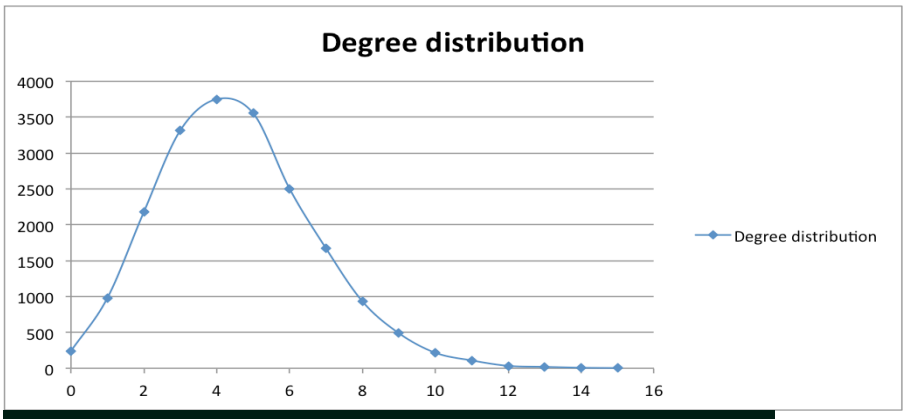


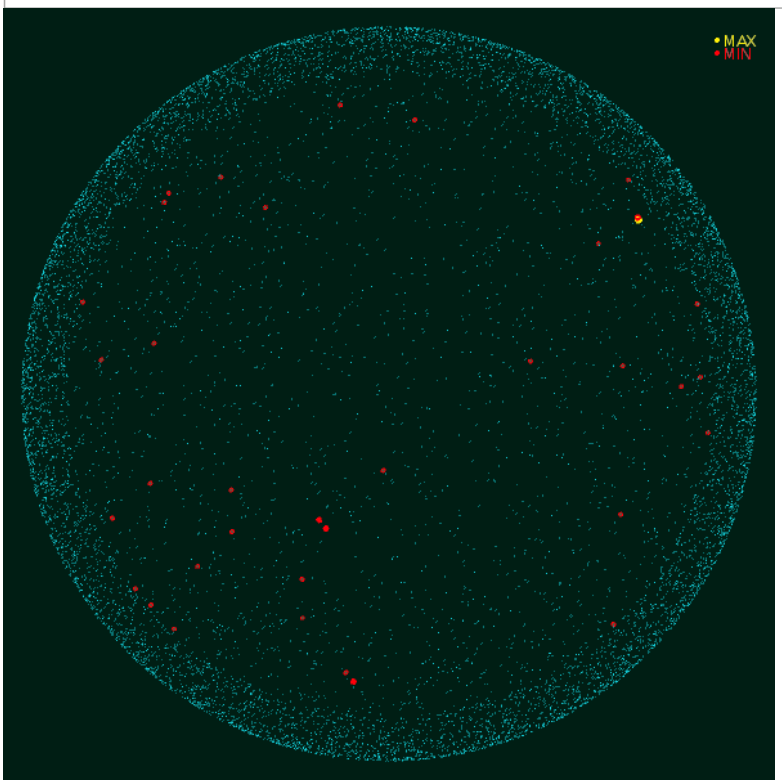


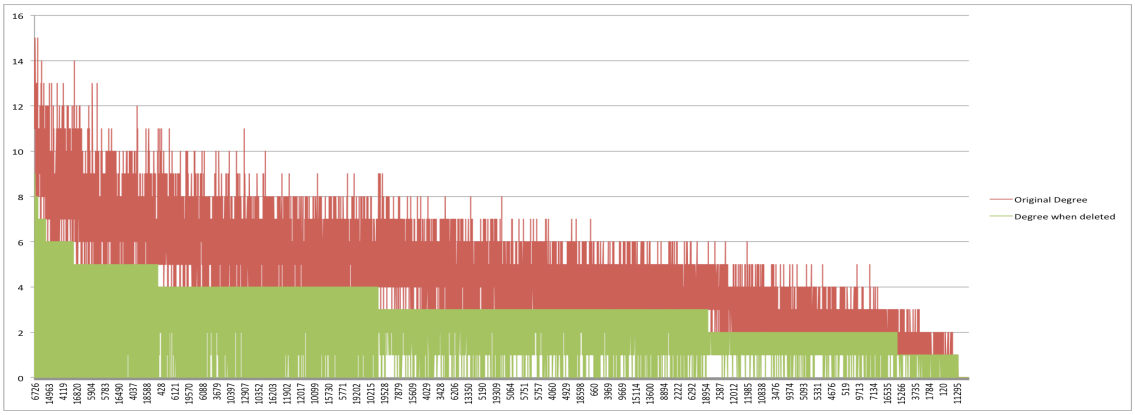


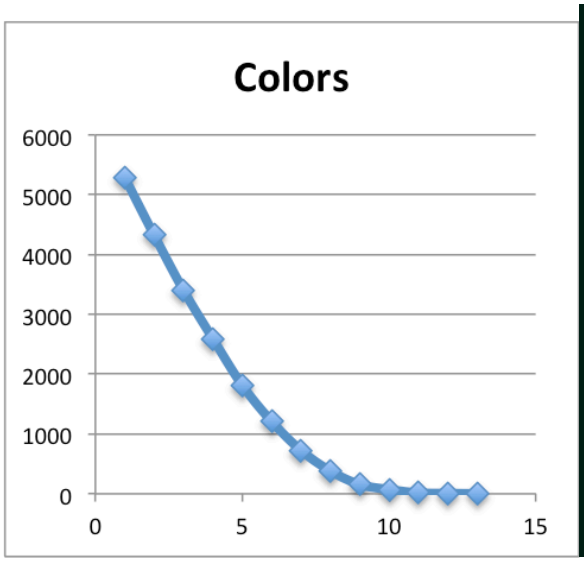


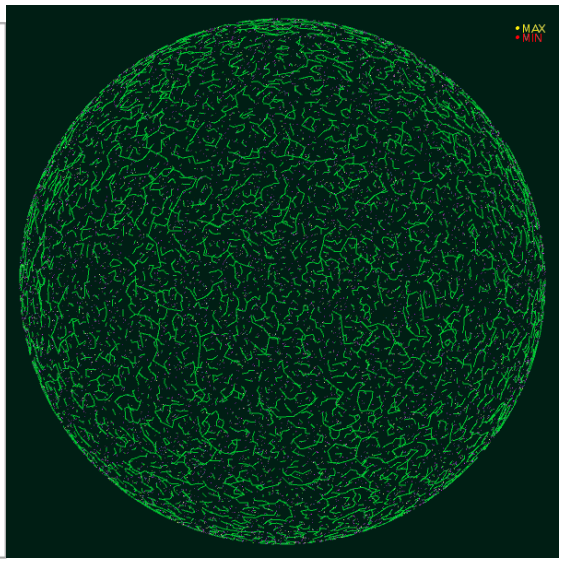
Benchmark 10: (N=20000, R=0.04) Hemisphere backbone coverage is 0.98

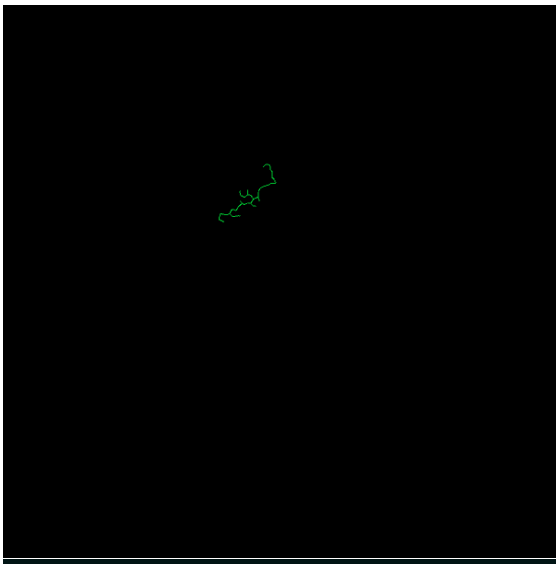


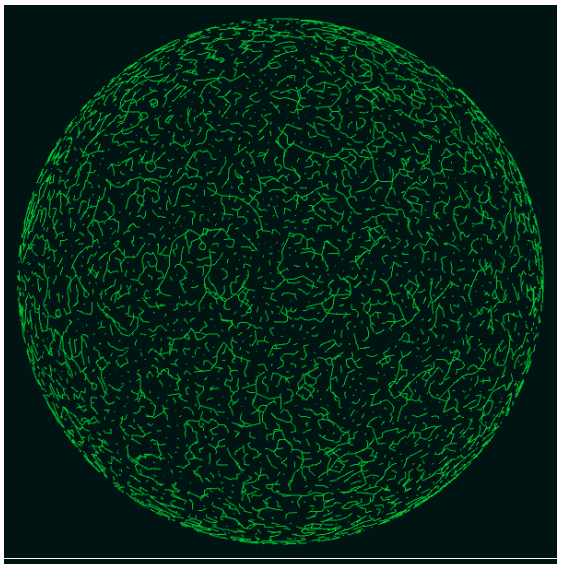


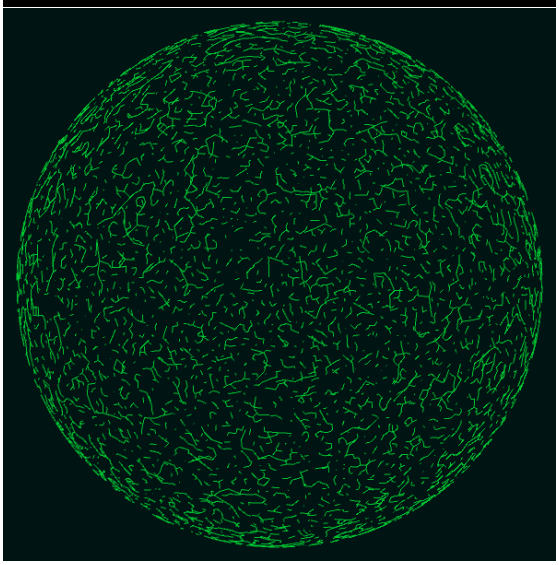


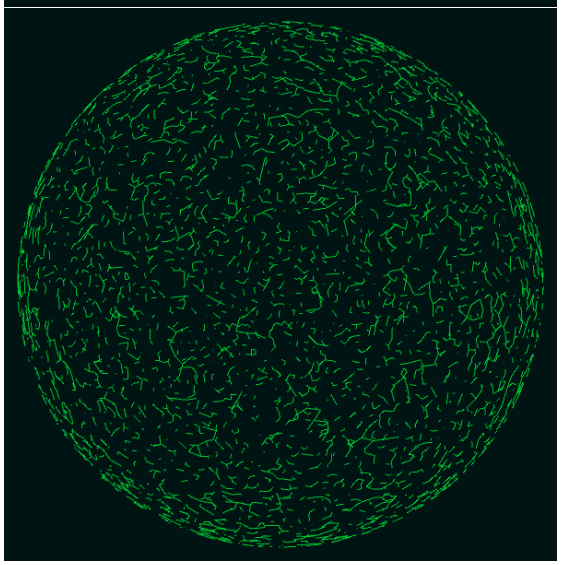

Colors
6000
5000
4000
3000
2000
1000
0
0
5
10
15


MAX
MIN



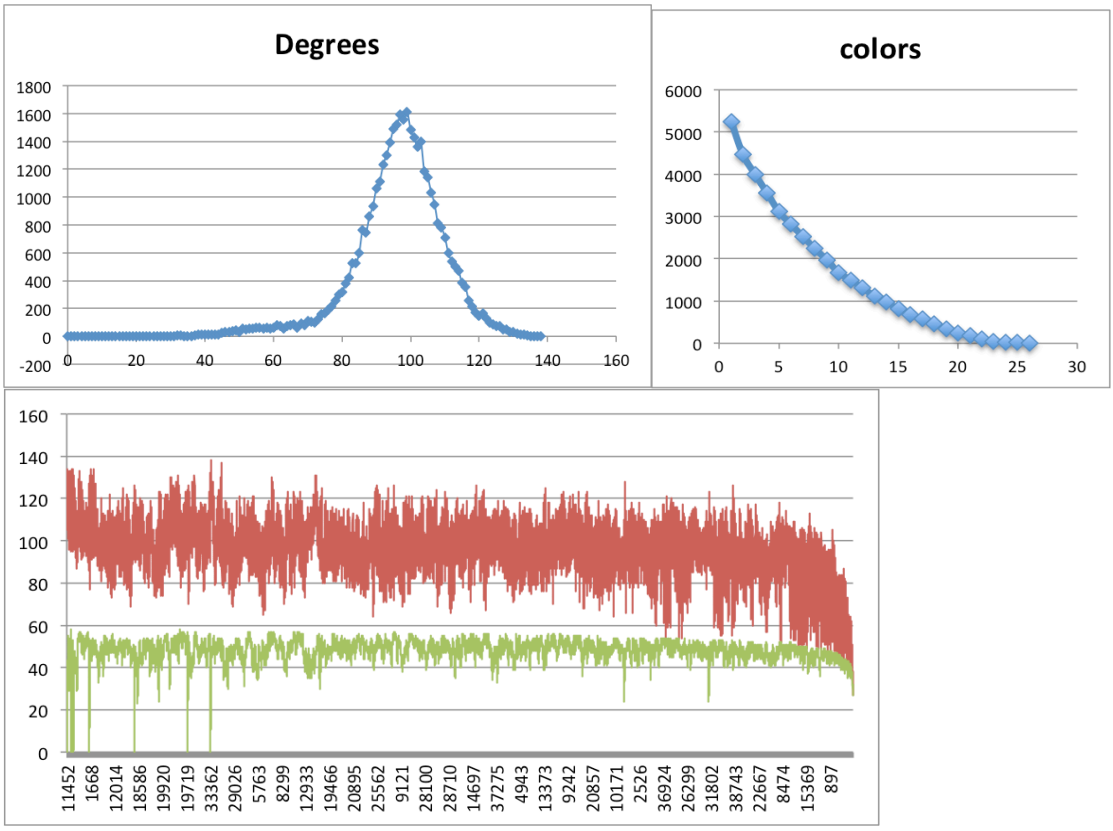


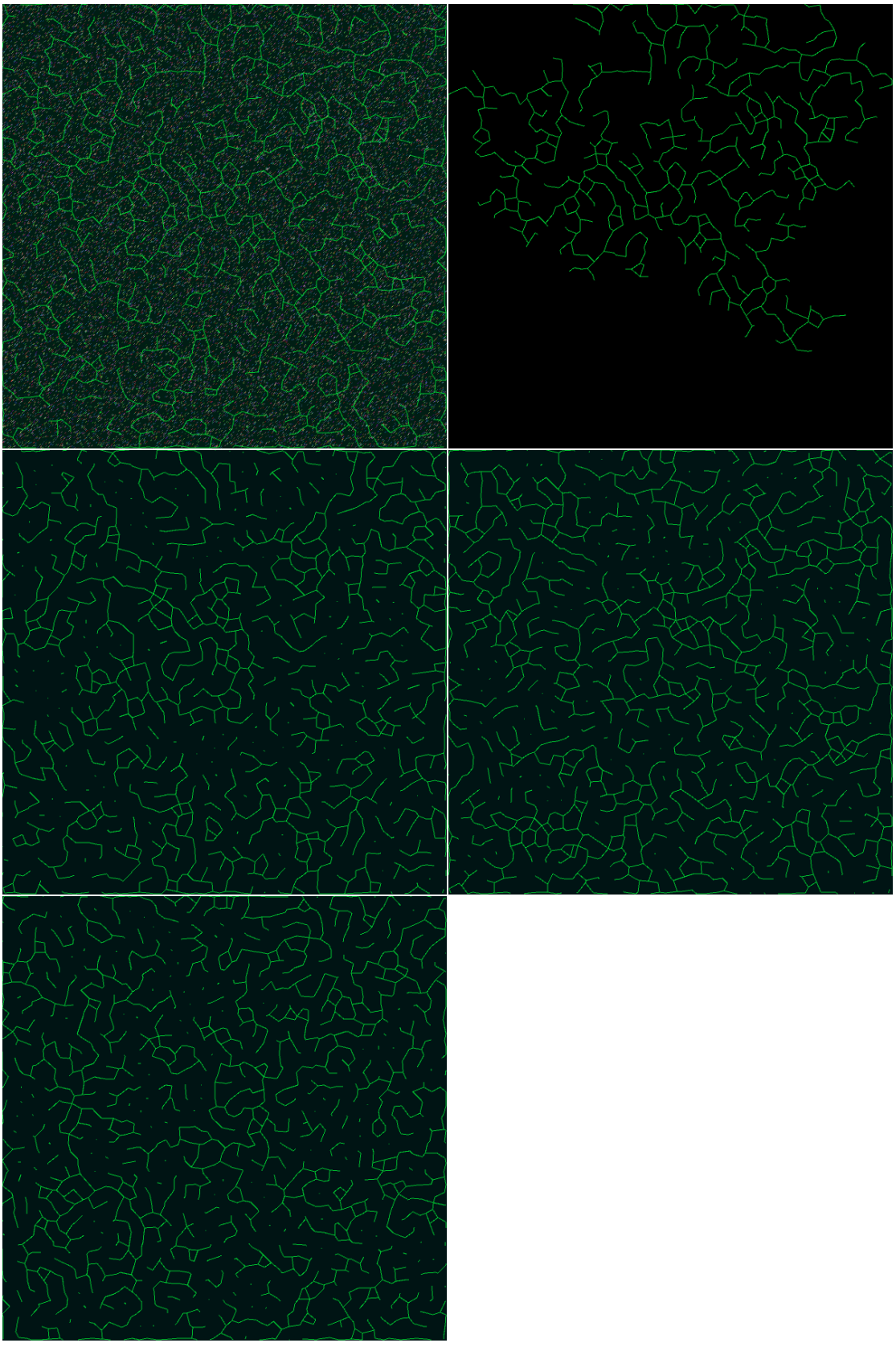




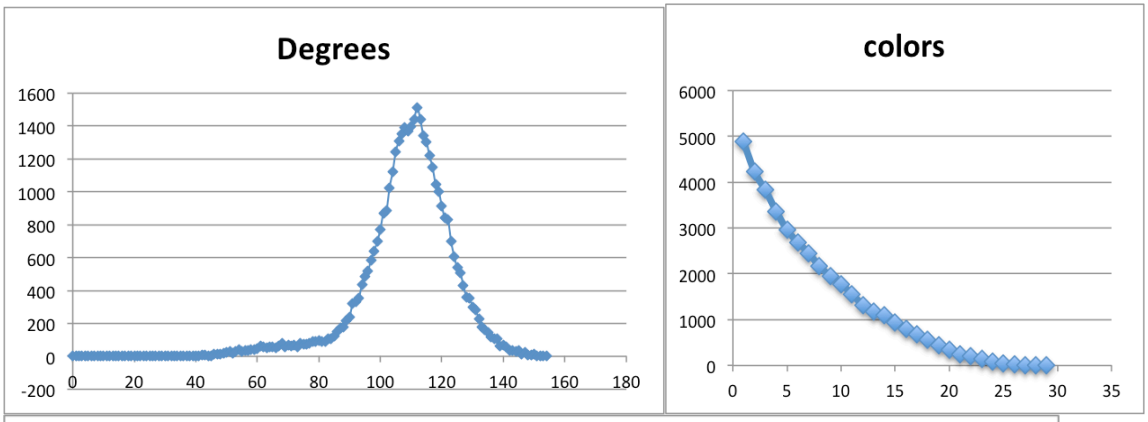
After the display of 10 benchmarks, some of the results are not ideal or nearly ideal. After different trials, I chose more benchmarks and displayed the pictures and plots as following:

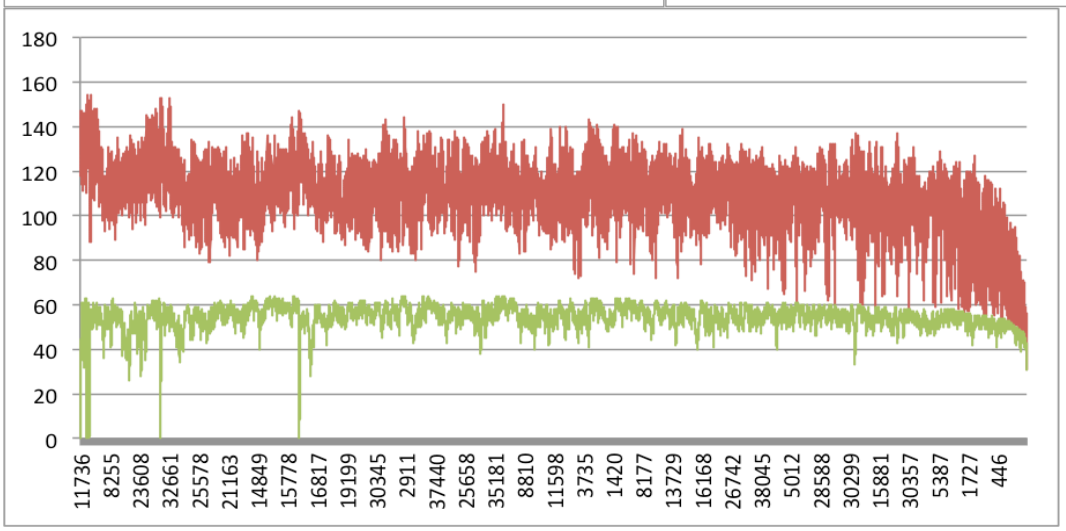
Benchmark 11: (N=40000, R=0.029) square backbone coverage is 0.999975

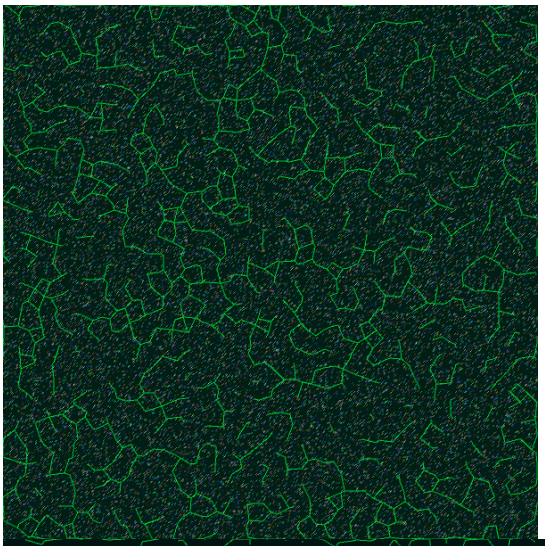
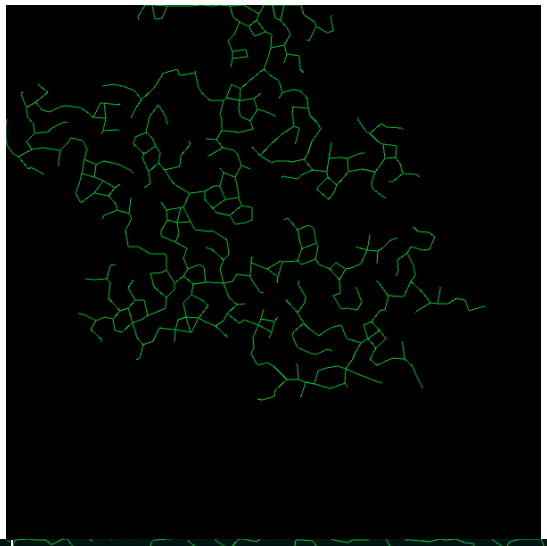
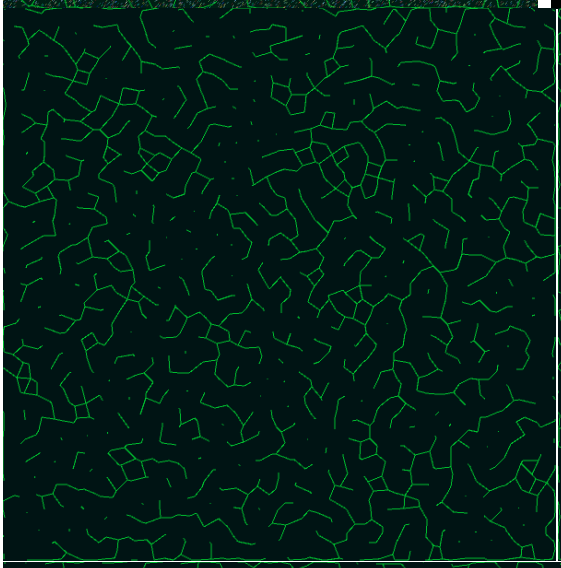
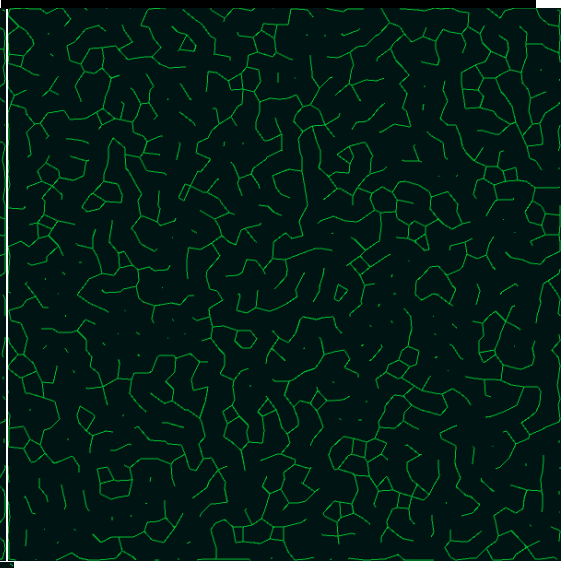
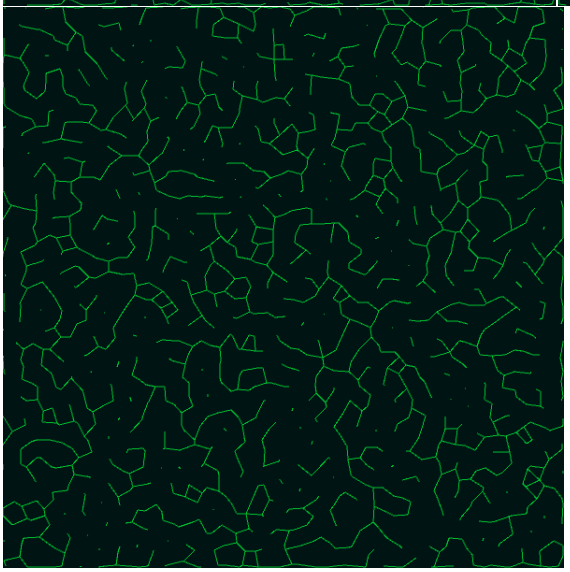




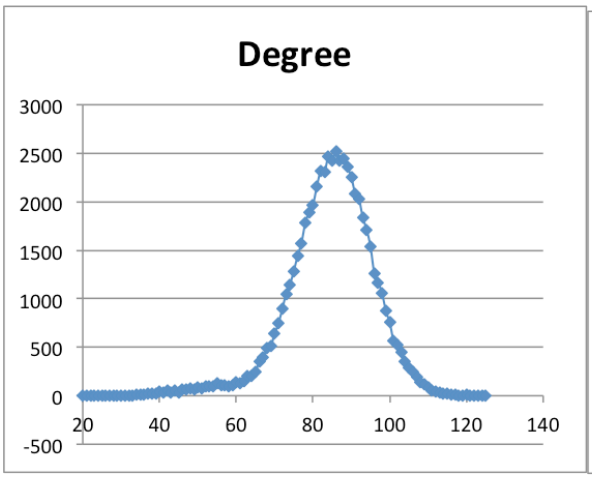
Benchmark 12: (N=40000, R=0.03) square backbone coverage is 1.0

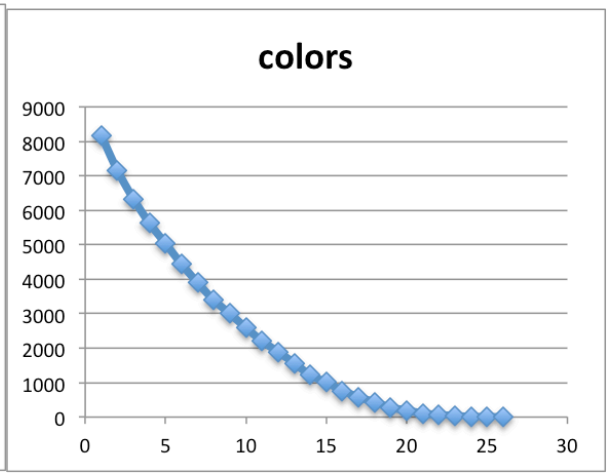


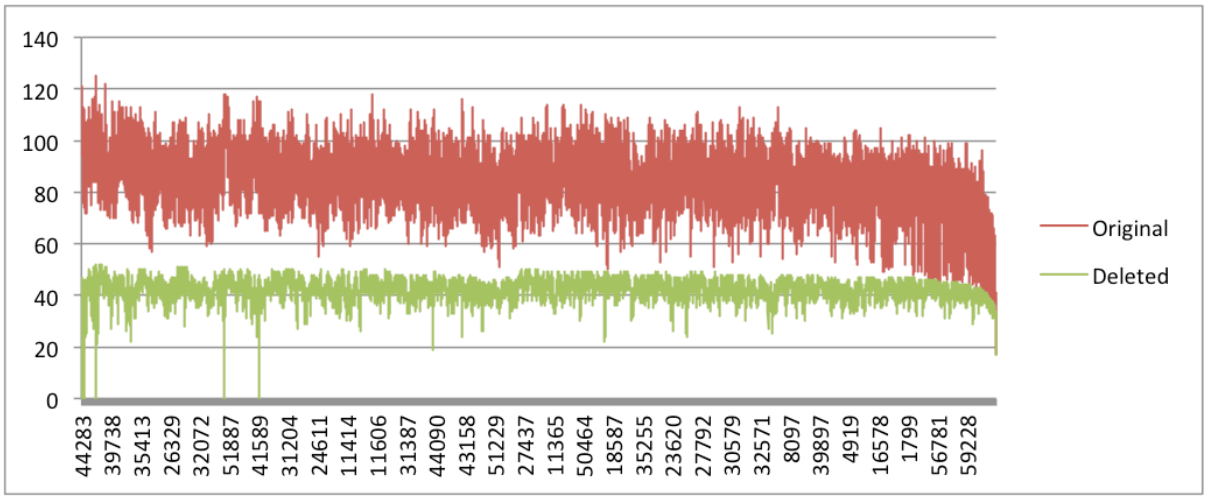


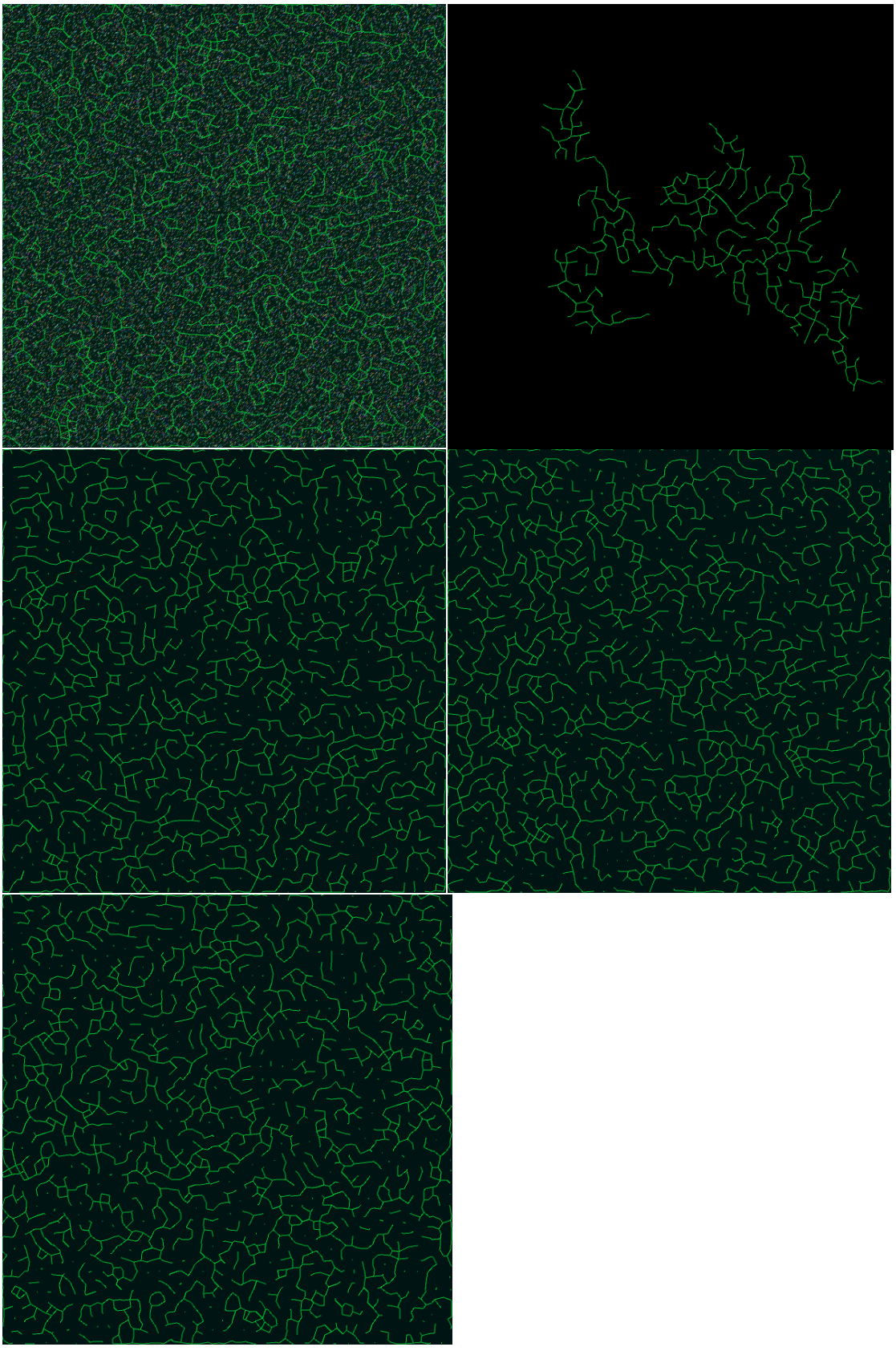






Benchmark 13: (N=60000, R=0.022) sphere backbone coverage is 0.999967

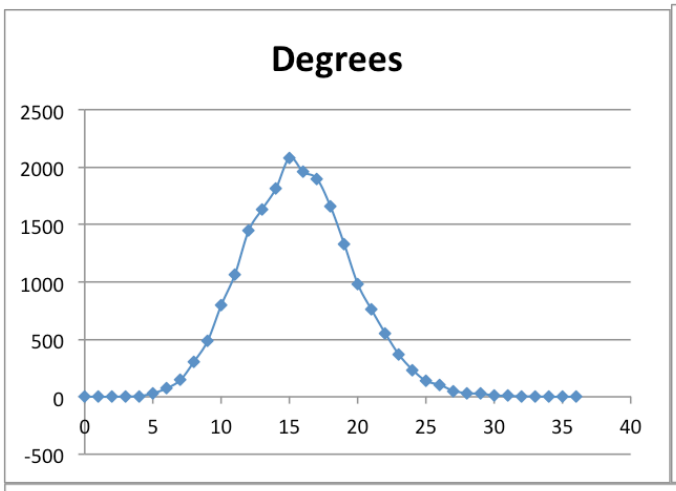


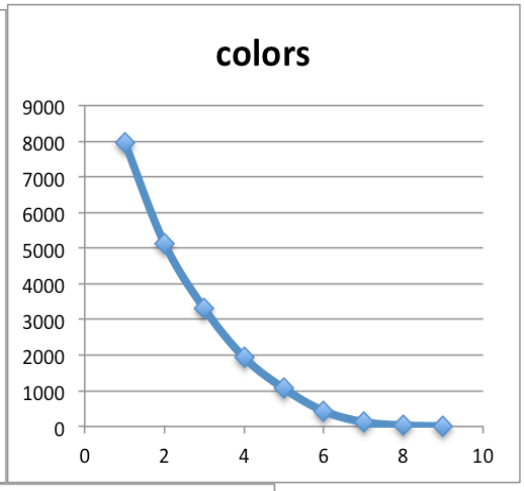


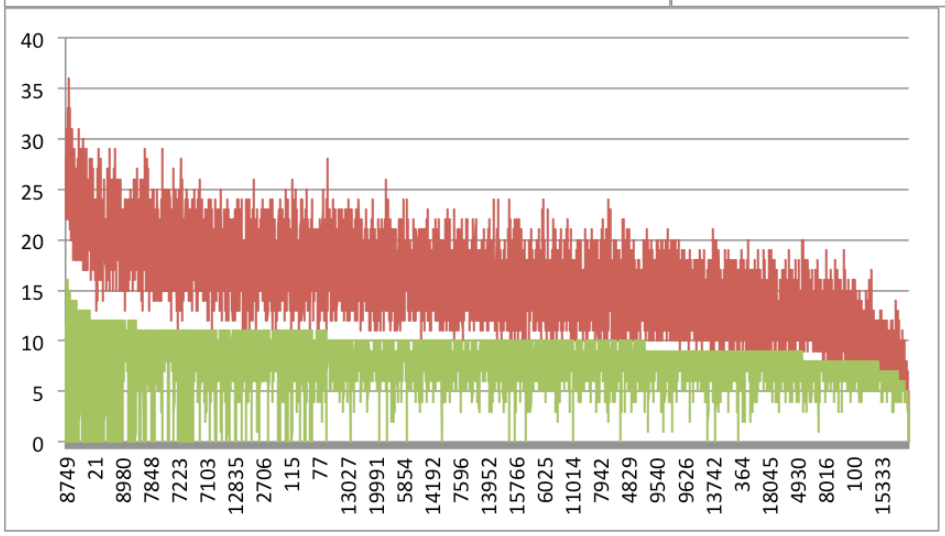


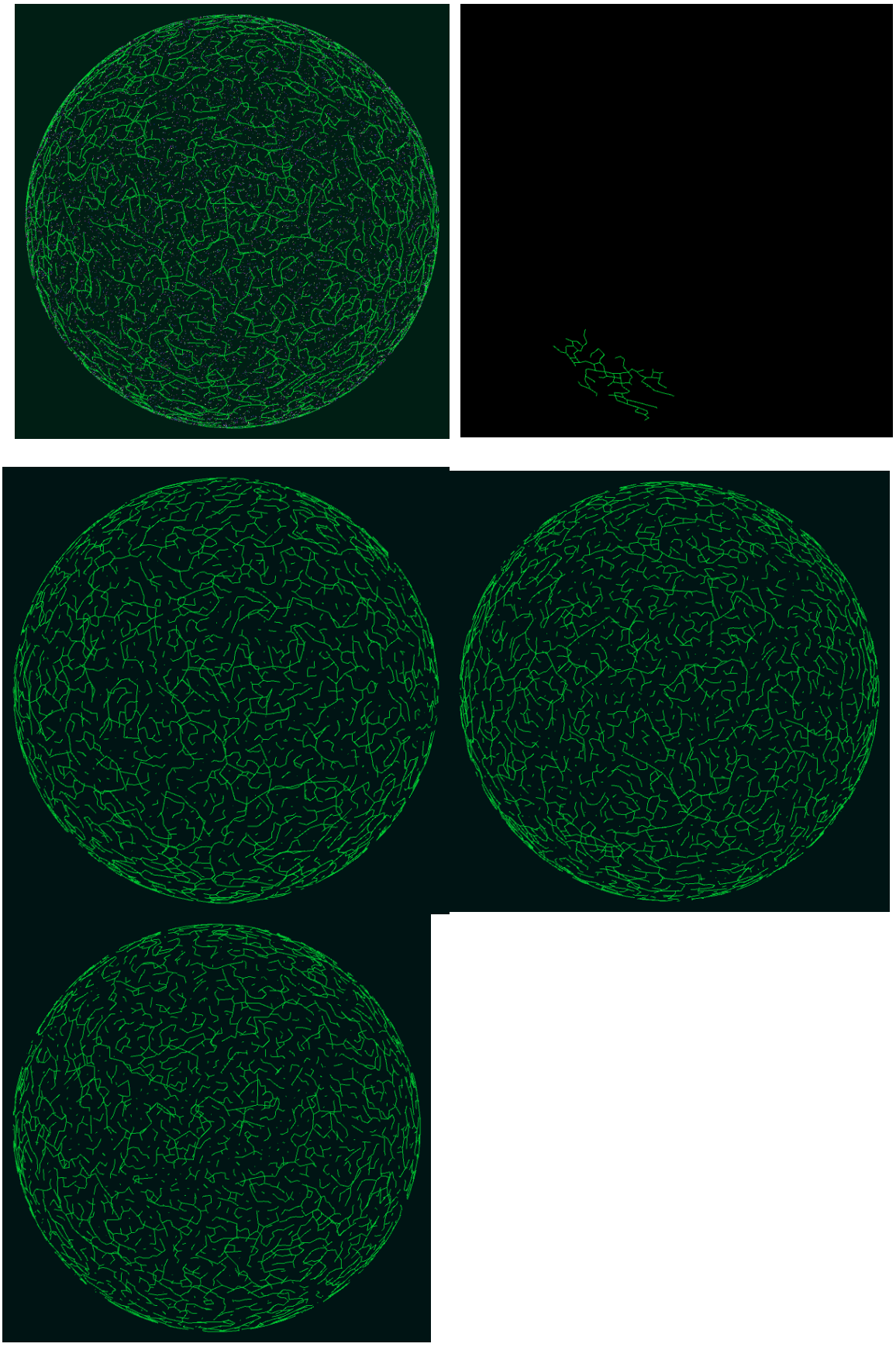


Benchmark 14: (N=20000, R=0.06) sphere backbone coverage is 0.99815

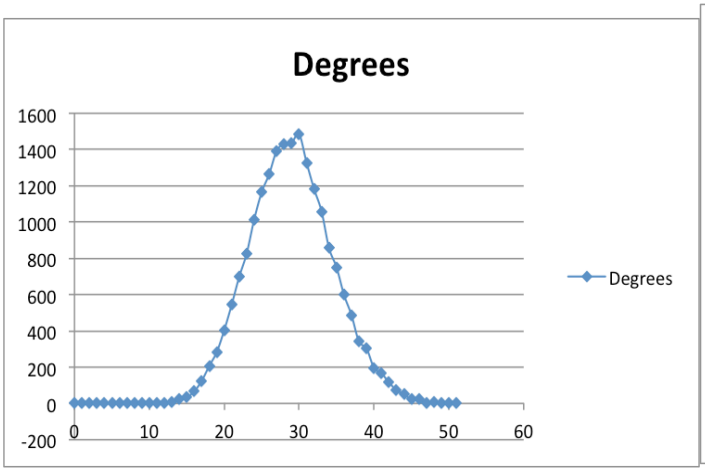


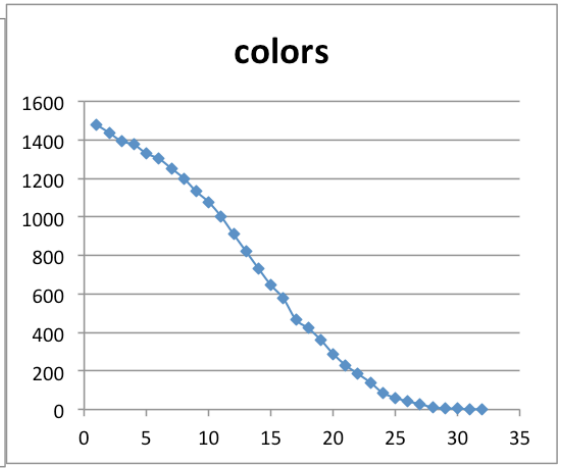


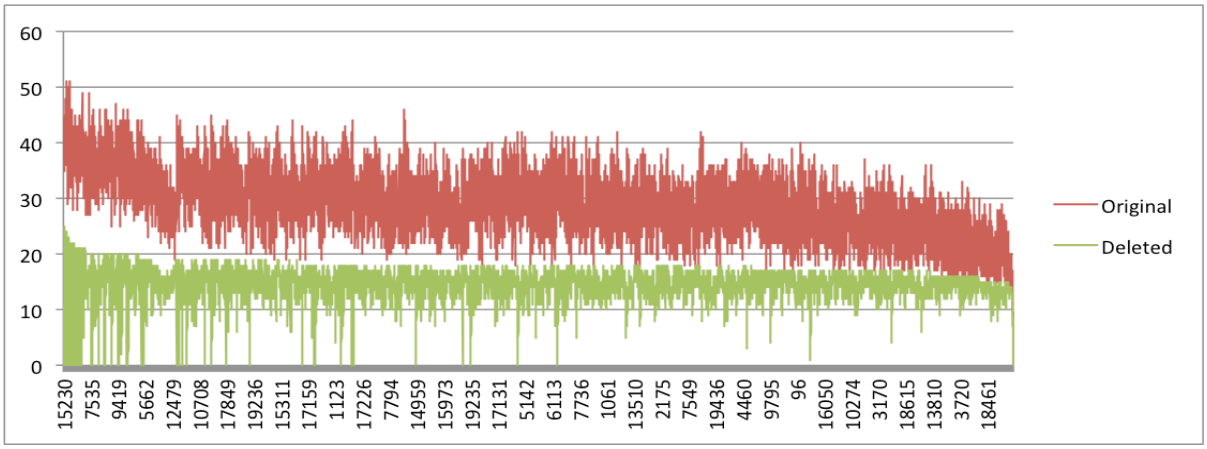


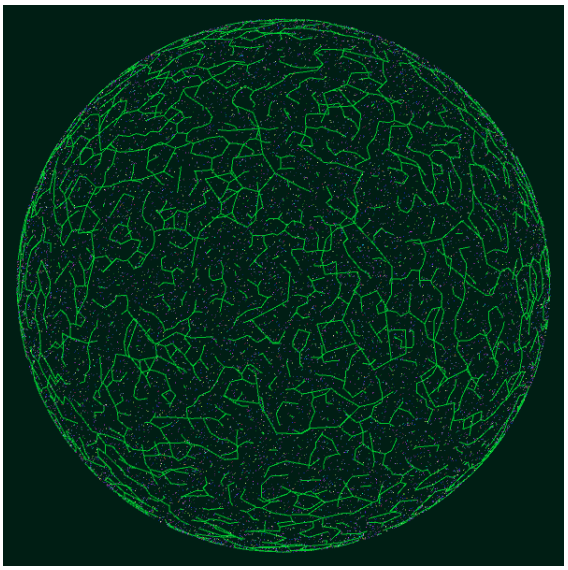


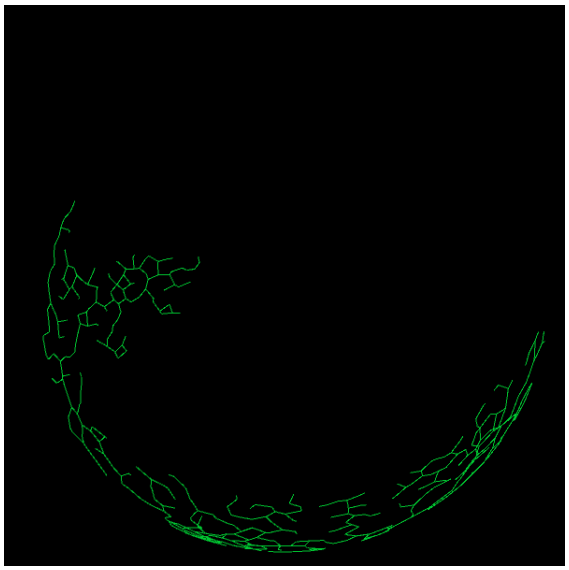
Benchmark 15: (N=20000, R=0.08) sphere backbone coverage is 0.99975

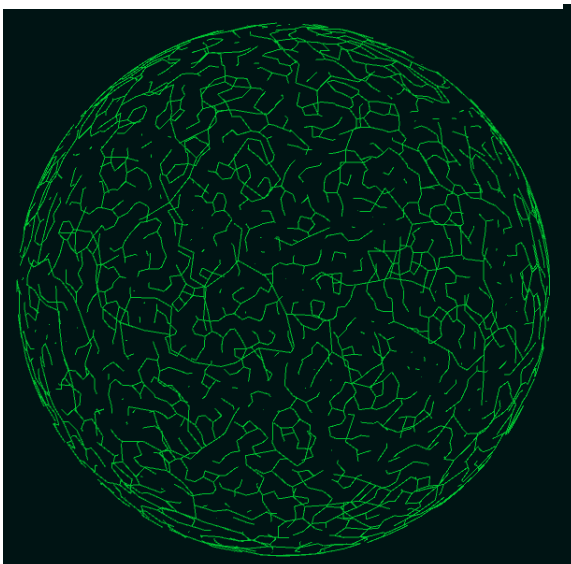


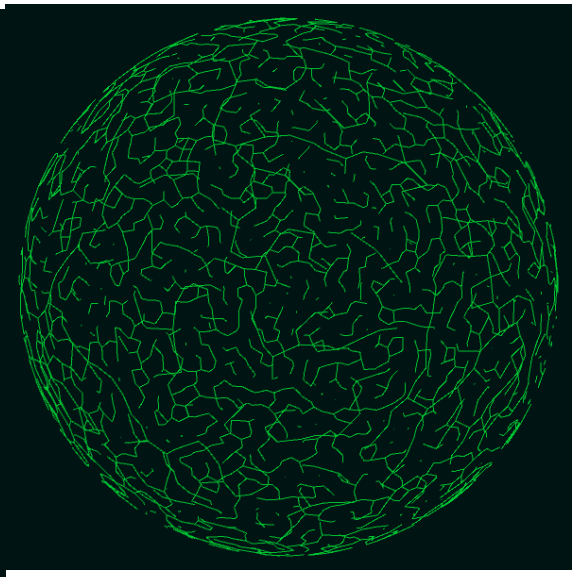


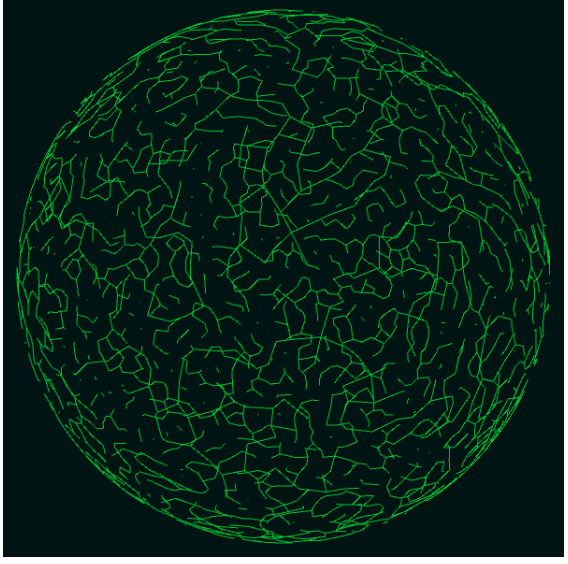




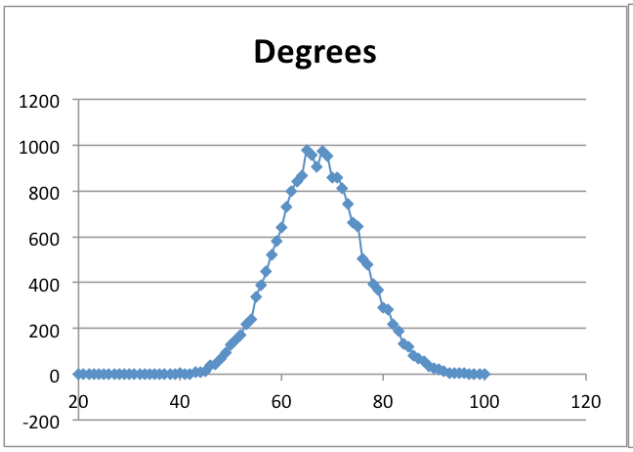


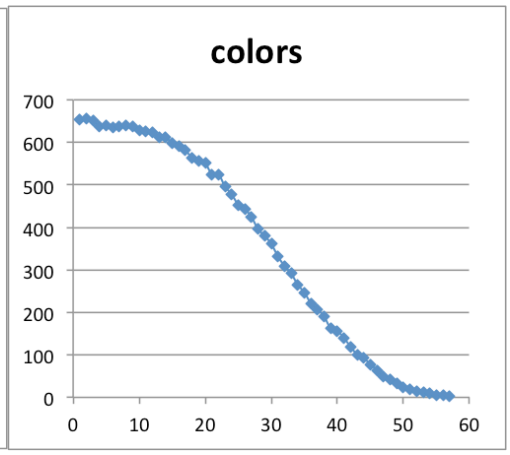


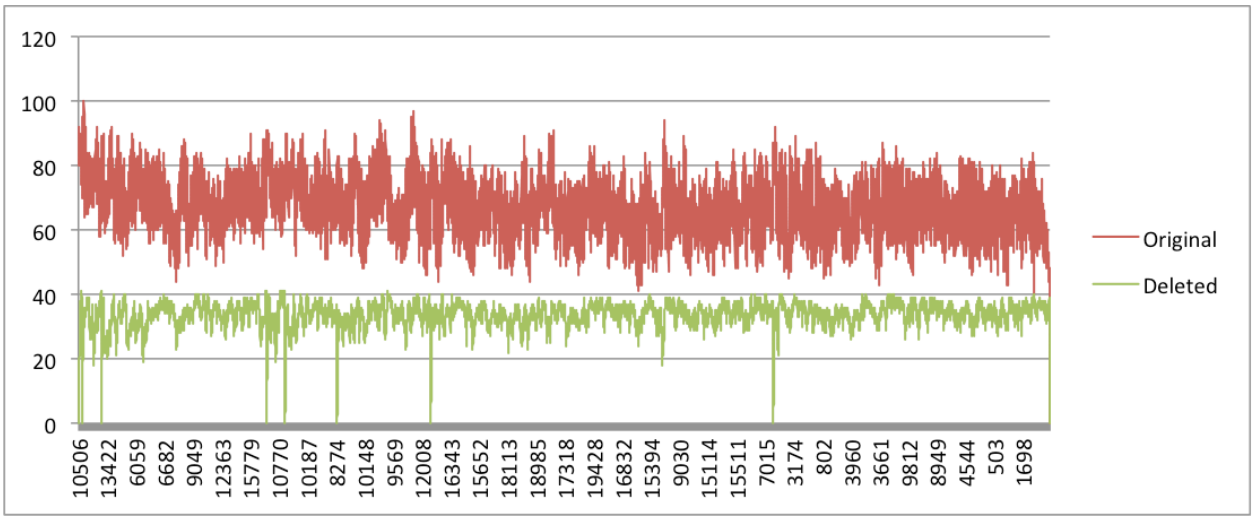


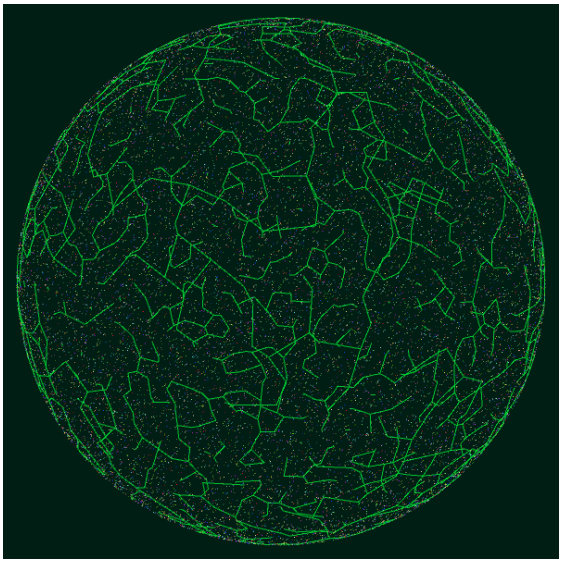


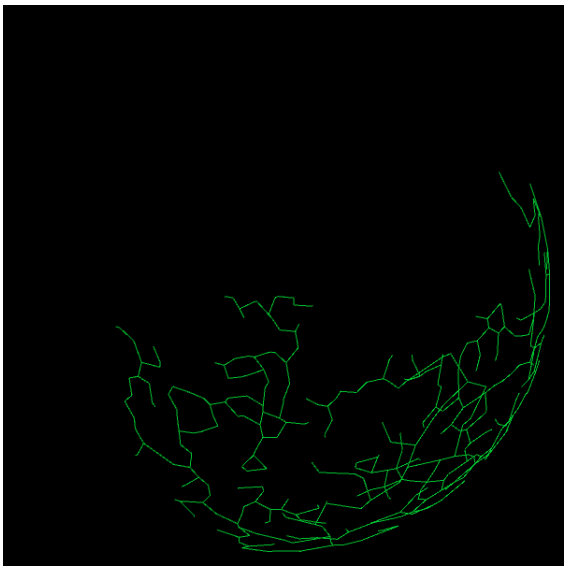
Benchmark 16: (N=20000, R=0.12) sphere backbone coverage is 0.99975

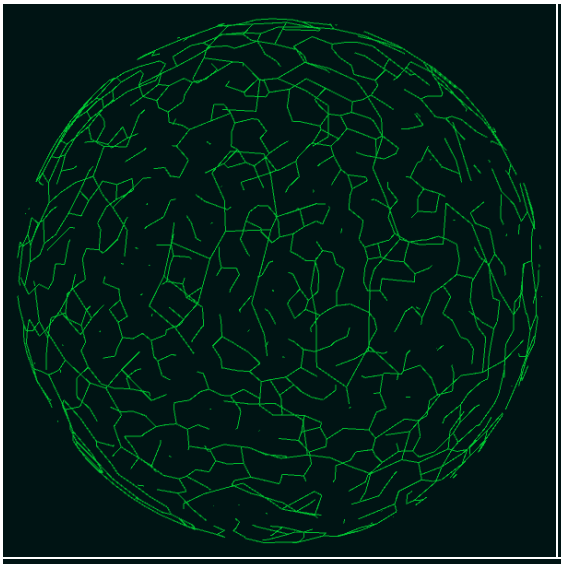


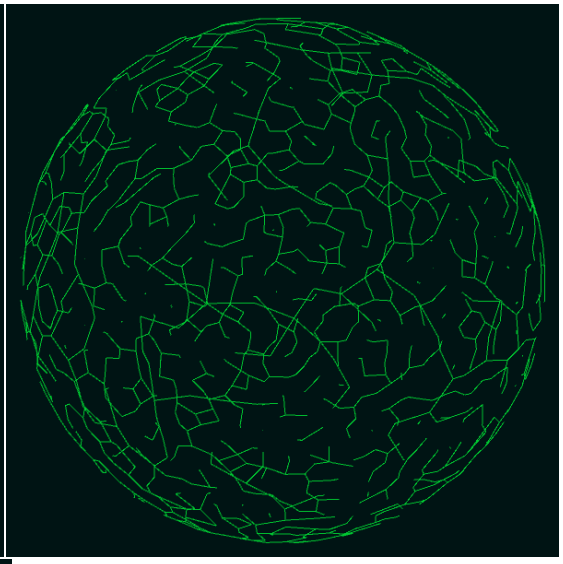


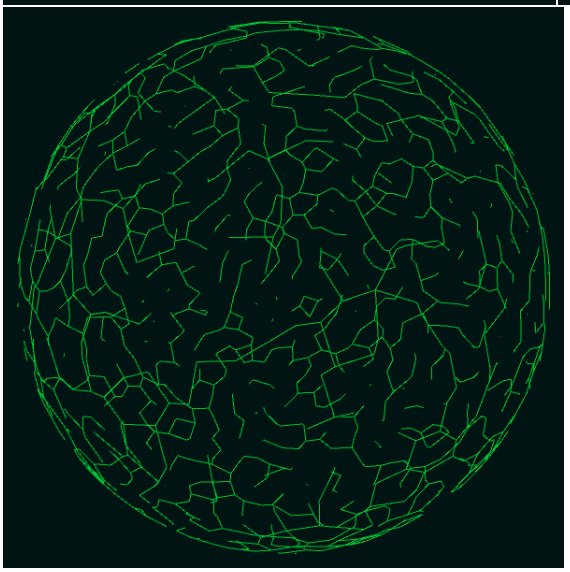




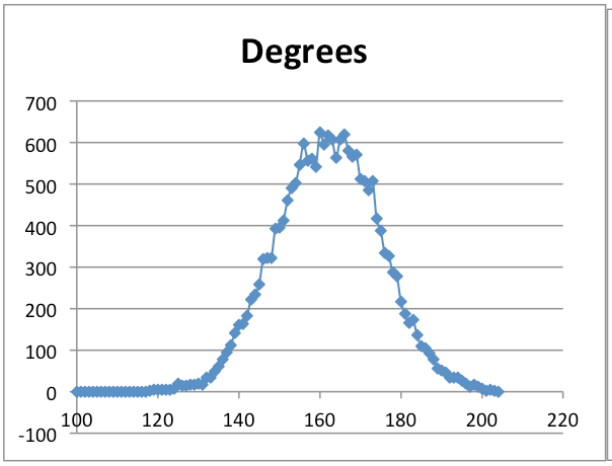


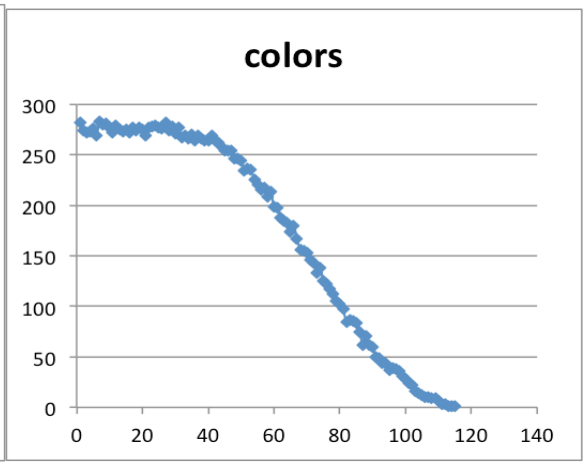


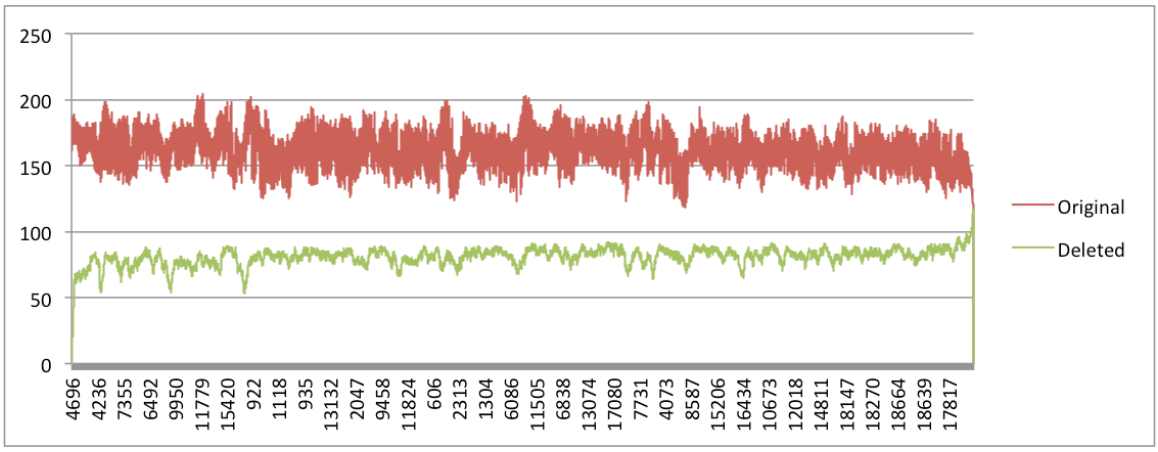


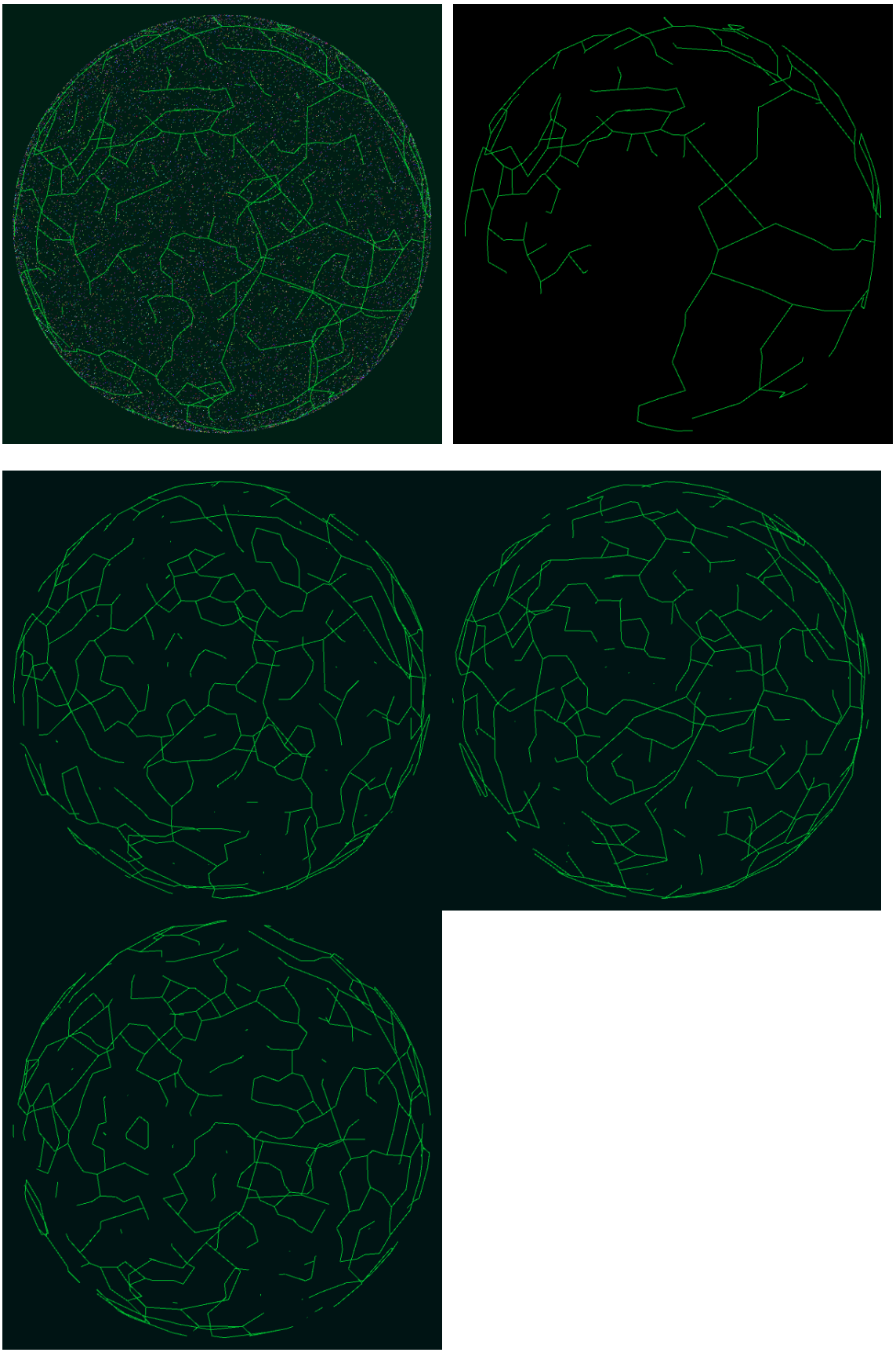


Benchmark 17: (N=20000, R=0.2) sphere backbone coverage is 0.99975

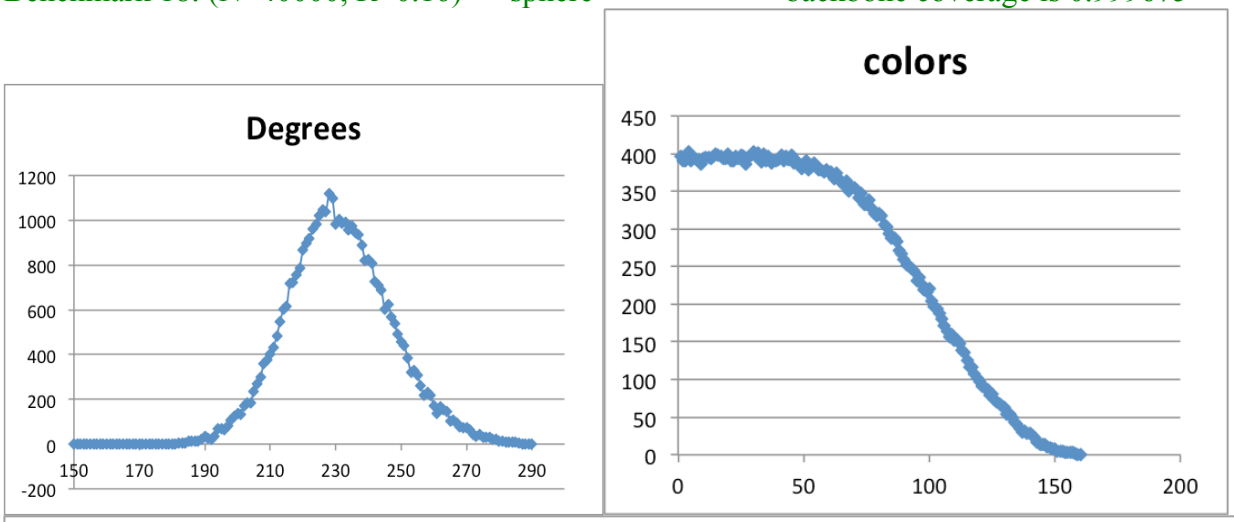


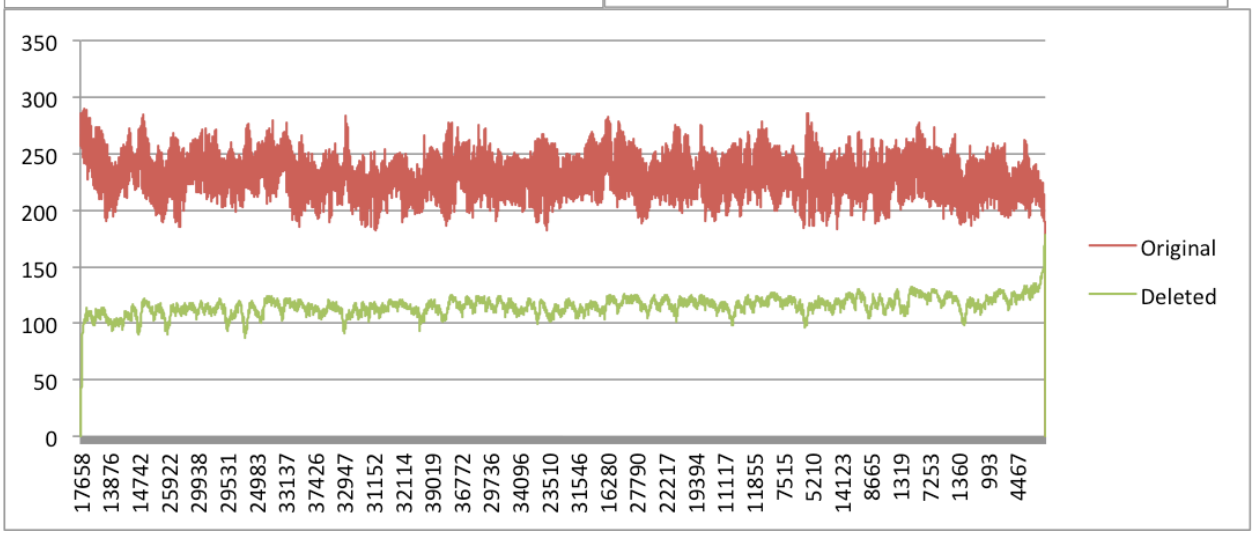


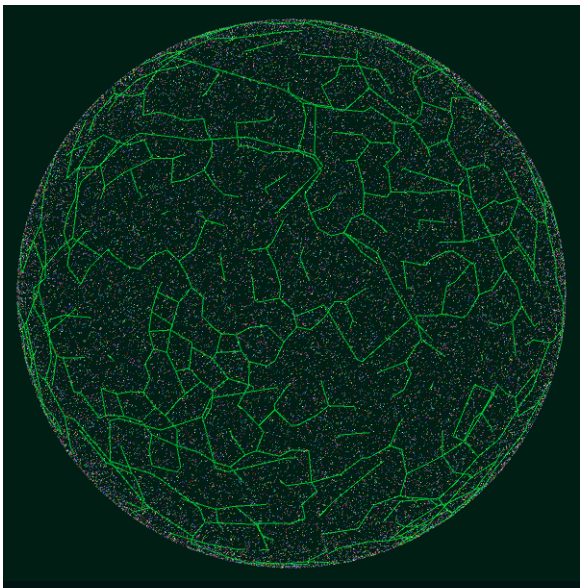


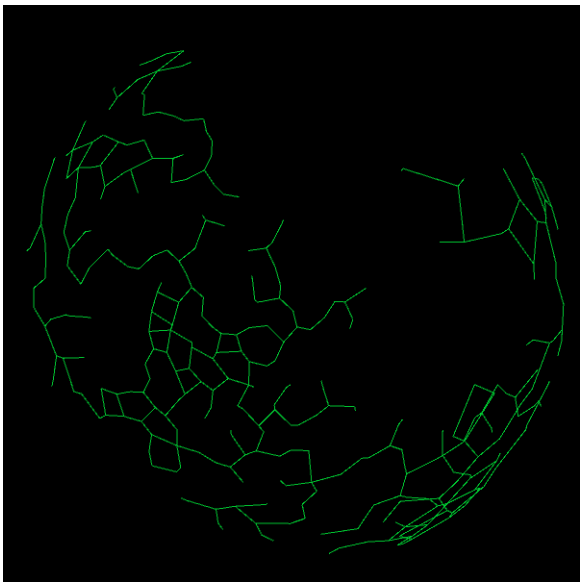


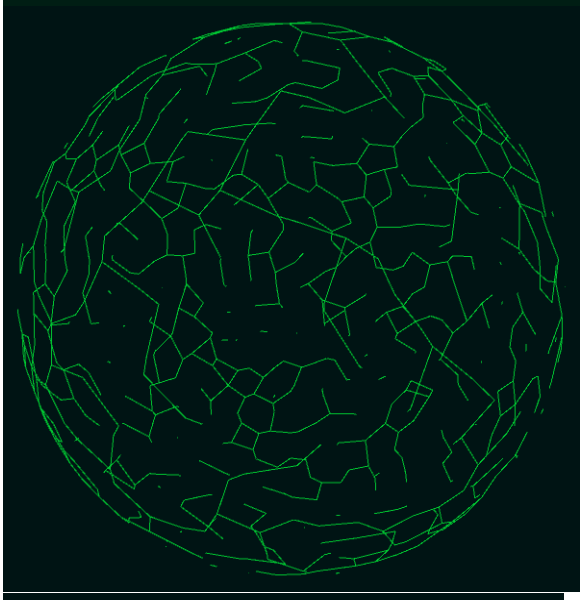
Benchmark 18: (N=40000, R=0.16) sphere backbone coverage is 0.999675

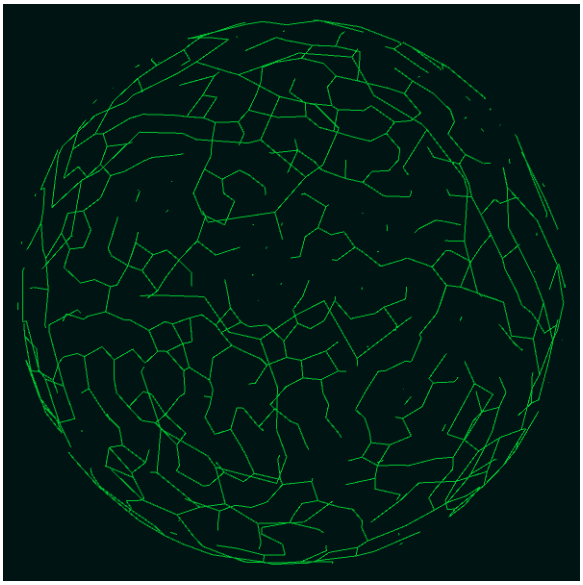


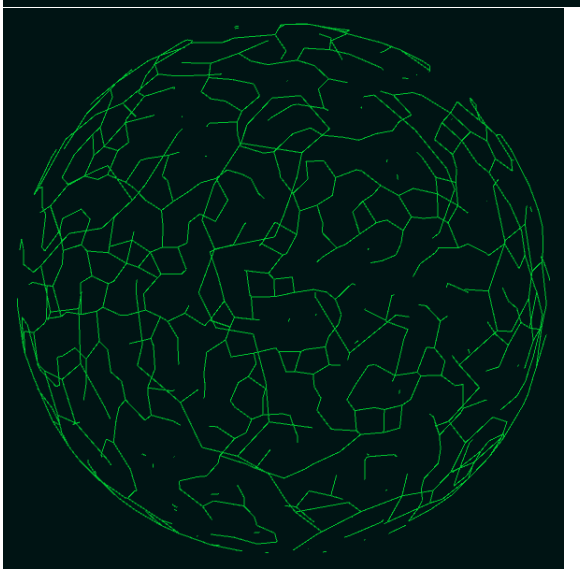




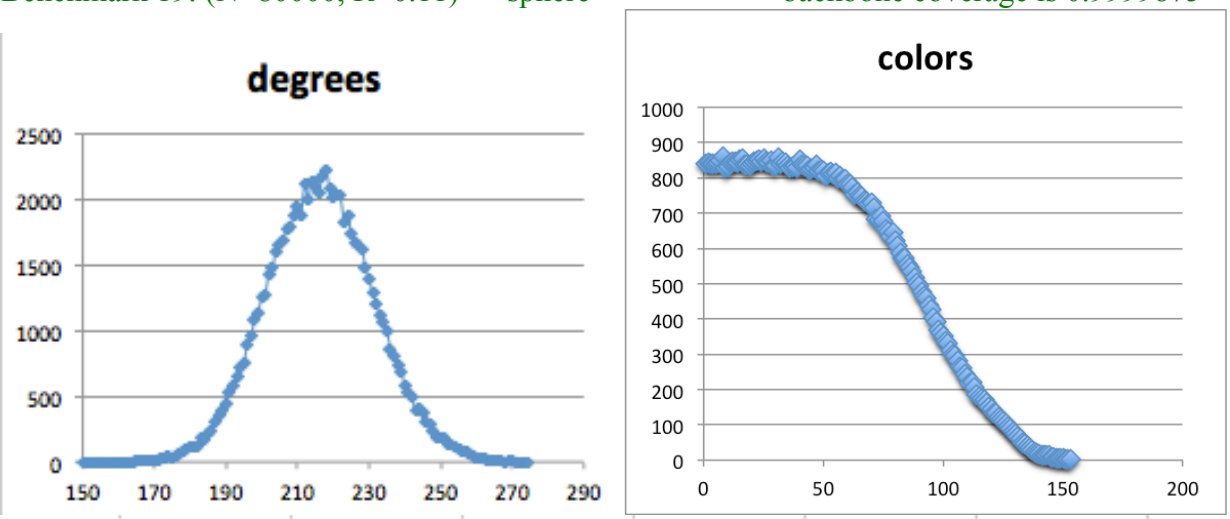


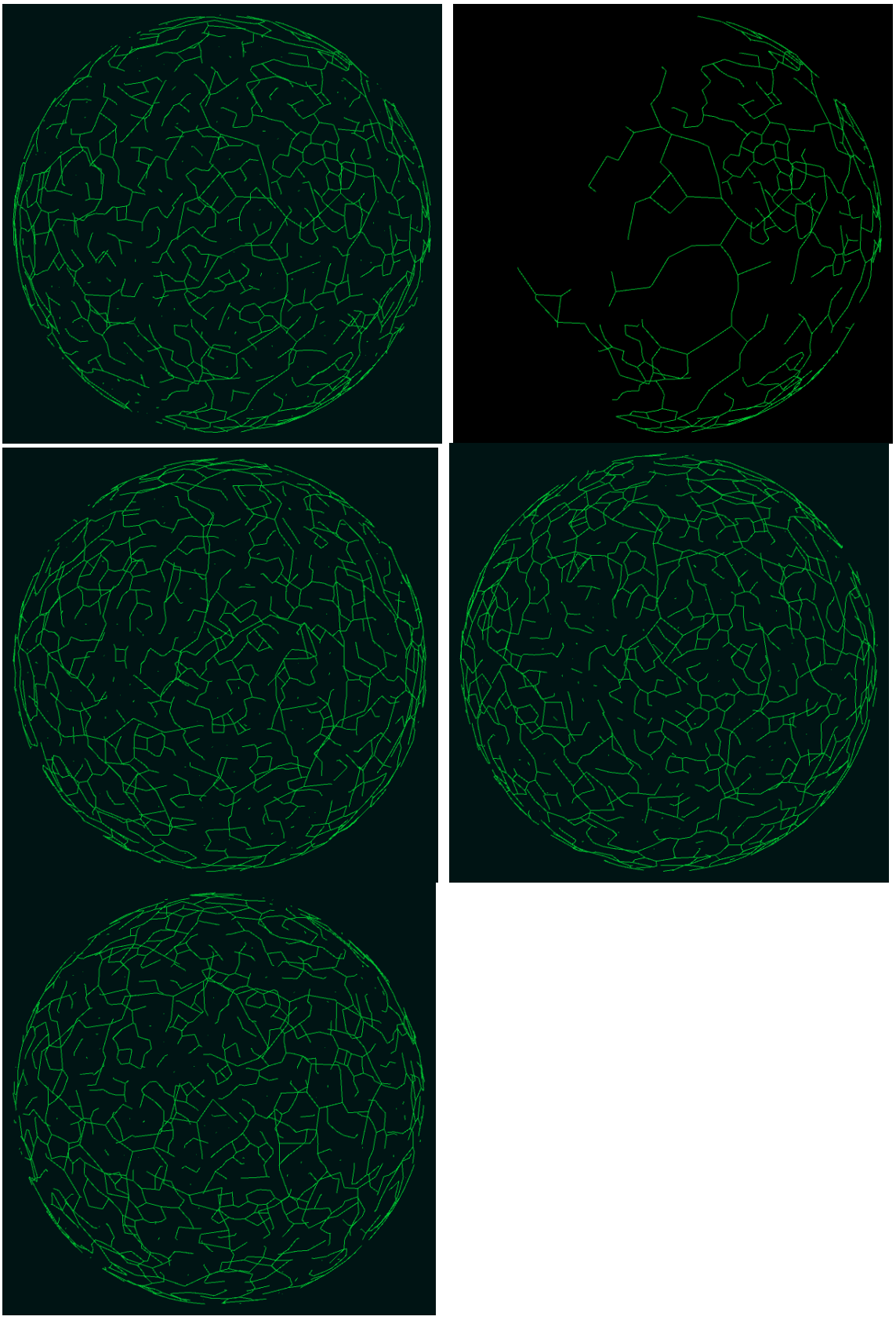






Benchmark 19: (N=80000, R=0.11) sphere backbone coverage is 0.9999875





Benchmark 20: (N=100000, R=0.075) sphere backbone coverage is 0.99998