# WIRELESS SENSOR NETWORKS:

# ALGORITHM ENGINEERING

## CSE 7350 By Dr. Lee McFearin

RADHIKA DHARULKAR
SMU ID# 46911479

# I. Executive Summary

## 1. Introduction and Summary

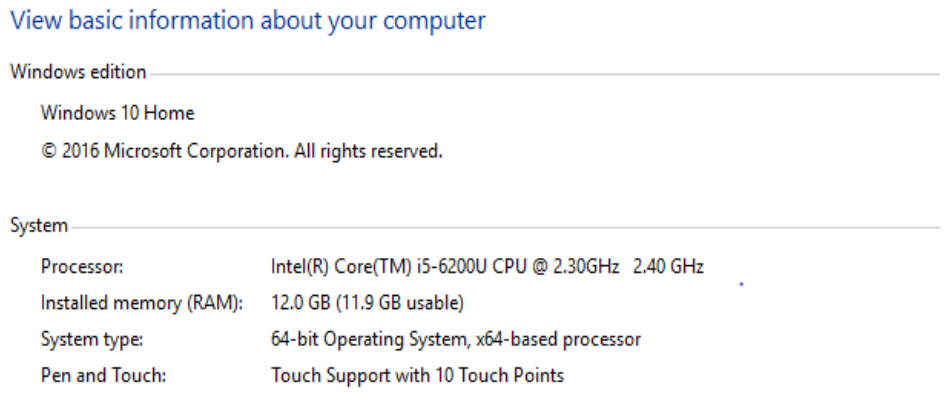
A wireless sensor network (WSN) is a wireless network consisting of spatially distributed nodes to monitor physical or environmental conditions or just establish a connection in an area. The WSN is built of nodes and is usually used for monitoring purposes like process or health monitoring, remote environmental monitoring, etc. Such a network is typically made of thousands of nodes with each node consisting of a transmitter and a receiver to establish a connection between nearby nodes. Each node can only communicate with another node within a certain range and hence to establish a connection over the entire area, careful distribution of nodes is very crucial.

In this project, we implement an algorithm for determining a coloring, terminal clique, and a selection of bipartite subgraphs that are produced by an algorithm for graph coloring in a random geometric Graph (RGG). These results model a wireless sensor network (WSN) with each bipartite subgraph providing a communication backbone.

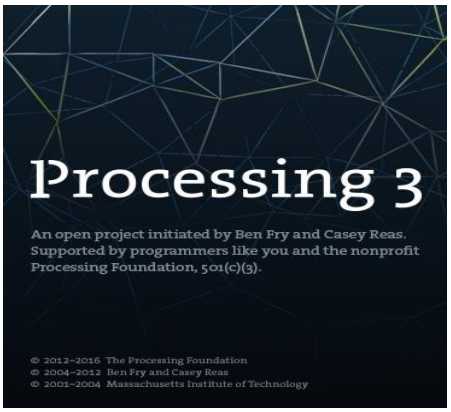
## 2. Programming Environment Description

**System information:**

| | |
|---|---|
| OS Name | Microsoft Windows 10 Home |
| Version | 10.0.14393 Build 14393 |
| OS Manufacturer | Microsoft Corporation |
| System Manufacturer | HP |
| System Model | HP Pavilion Notebook |
| Processor | Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz, 2400 Mhz, 2 Core(s), 4 Logical Processor(s) |
| Installed Physical Memory (RAM) | 12.0 GB |
| Total Physical Memory | 11.9 GB |
| Available Physical Memory | 6.74 GB |
| Total Virtual Memory | 13.7 GB |
| Available Virtual Memory | 7.77 GB |
| Page File Space | 1.81 GB |



**Programming platform information:**




Processing is an open source computer programming language and integrated development environment (IDE) built for the electronic arts, new media art, and visual design communities with the purpose of teaching the fundamentals of computer programming in a visual context, and to serve as the foundation for electronic sketchbooks. The Processing language builds on the Java language, but uses a simplified syntax and a graphics user interface. It was easy to work with processing because of my existing familiarity with java and very easy to learn use of processing commands.


I chose to make use of the Eclipse IDE to write my processing code. To use processing in eclipse, I needed to import the processing libraries into my project in eclipse.

To make it work, the Processing IDE does a lot of stuff for us in the background 'Java' environment to make it easy for us as programmers to create sketches. However, to make full use of all the Java awesomeness at our disposal, it's helpful to use a more robust IDE, which is why we are jumping into Eclipse.

To use processing library in eclipse all the processing core libraries needed to be imported (only core.jar is sufficient to make processing run in eclipse but if you want to use P3D, you must import all the core libraries).

## 3. Summary Table

| # | Nodes | Average Degree | Topology | R (threshold) | Maximum Degree | Minimum Degree | Terminal Clique Size | Color set size |
|---|---|---|---|---|---|---|---|---|
| 1 | 1000 | 32 | square | 0.102 | 46 | 10 | 21 | 22 |
| 2 | 4000 | 64 | square | 0.144 | 92 | 18 | 35 | 37 |
| 3 | 16000 | 64 | square | 0.072 | 95 | 17 | 37 | 39 |
| 4 | 64000 | 64 | square | 0.018 | 107 | 15 | 42 | 45 |
| 5 | 64000 | 128 | square | 0.025 | 178 | 34 | 50 | 59 |
| 6 | 4000 | 64 | disk | 0.127 | 100 | 25 | 35 | 40 |
| 7 | 4000 | 128 | disk | 0.18 | 166 | 47 | 58 | 59 |
| 8 | 4000 | 64 | sphere | 0.127 | 42 | 4 | 27 | 28 |
| 9 | 16000 | 128 | sphere | 0.09 | 66 | 18 | 30 | 31 |
| 10 | 64000 | 128 | sphere | 0.045 | 70 | 18 | 27 | 30 |

# II. Reduction to Practice

## 1. Data Structure Design

### Array

It is a systematic arrangement of similar datatypes in rows. This data structure has been used throughout the programming.

### Multidimensional Array

It is a systematic arrangement of similar datatypes in rows and columns. Multidimensional arrays are used in the programming to store data in rows and columns. I have preferred to make use of multidimensional arrays instead of an arraylist since array is a primitive datatype and easier to work with. ArrayList is a more useful data structure when we want a datatype that grows dynamically.

### Class

It is a Non-primitive datatype. We define some non-primitive, composite datatypes like classes. We can use classes to store information regarding an object.

### Other primitive datatypes

We have also made use of some other primitive datatypes such as int, float, Boolean, String, double, etc.

## 2. Algorithm Analysis

### Smallest-last ordering and coloring algorithm:

Let $v_1$, $v_2$, $v_3$, ……. $v_n$ be the vertices of a graph. If $v_i$ has the minimum degree, it should appear last on the maximal subgraph. In the maximal subgraph $v_i$, $v_{i+1}$ …. $v_{i+n}$, $v_{i+n}$ will have the least degree. This can be achieved by simply sorting the vertices based on their degree in descending order. This can simply be done by creating an empty array of the size of number of vertices, then finding the vertex with least degree and that vertex in the last place of the new array. We repeat this step until all the vertices from the original array are moved to the next available last position in the new array.

### Coloring:

Vertex coloring is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color. Suppose vertex V1 is adjacent to V2 and V3, a different color will be assigned to V1 than V2 and V3. A terminal clique is a connected graph in which all the vertices of the graph are connected to every other vertex of the graph. In which case the number of colors required So, we can assess that the maximum number of colors required will be equal to the clique size of the graph. So, we need to create a pool of clique size number of unique colors ready to assign to a vertex.

The output of coloring algorithm will later be used to find bipartite subgraphs.

### Backbone selection:

The backbones are selected on the bipartite graphs generated of the graph. When the coloring is done, we can think of all the sets of one color as one set. If we consider two such sets and also the connecting edges, we will get a bipartite graph. So, if we have N such sets then we will get ${}^{N}C_2$ bipartite graphs. Backbone selection is the last step in the project.

## 3. Algorithm Engineering

### Smallest-last ordering and coloring algorithm:

The basic steps involved in this algorithm are: Find, store and delete.

Find the node with the least degree and place it in the last empty place of an empty array of size equal to number of vertices. We delete this vertex from our RGG and then we update the adjacency list after deletion. We also keep a track of the originally adjacent vertices of the deleted vertex. By placing the next vertex with least degree and then placing it in the last available positon of the new list, we can obtain a smallest last ordering of the vertices in the new list.

The first step requires us to find the vertex with the least degree. This can be checked for by considering each vertex sequentially, but to avoid increasing our time complexity by considering each and every node every time we will consider the degree list for selecting the next vertex to be deleted. We can simply look at the degree list and select the vertex with the least degree. This will not be a challenge because the way these vertices are stored is their degree is equal to the index at which it is stored.

Secondly, we want to store this vertex at the last available position of our new list. The new list could be simply a list if we wanted to store just the removed vertices. But since we want to keep a track of the vertices that were adjacent to it, we will require an arraylist. This arraylist will be very helpful when we want to do the graph coloring based on our smallest last ordering algorithm.

Next, we will need to delete this vertex that we just moved from degree list. While doing this, we will need to modify the adjacency list along with the degree list. We must delete the entire record of the vertex and also any other occurrences of that vertex (adjacency list of it's adjacent vertices) from the adjacency list. Then we delete the vertex from the degree list. Also, we must decrease the degree of all the adjacent vertices by one in the degree list. So, the degree of all the connected vertices will reduce down by one in the degree list.

The find, store and delete step must be carried out until all the vertices have been deleted and we will get the smallest last ordering of the vertices. This arraylist will then be provided to the coloring algorithm. To do so we will have to visit all the vertices and all the edges of the graph and hence, the complexity of this algorithm will turn out to be $O(|V| + |E|)$.

### Coloring algorithm:

The very basic way to generate a pool of colors is to generate a list of random colors by using a random number between 0 to 255 for RGB values of the color. To begin with the coloring algorithm, we select the first element from the smallest last ordering list and traverse our way down the list. We consider the next vertex from the smallest last list and assign a color to it from our pool of random colors. While assigning a color to a vertex, the color of its adjacent vertices is checked and any color assigned to the adjacent vertices is ignored. The next available color in the pool of colors is assigned to the vertex.

So, the steps involve: consider a vertex, check colors of adjacent vertices and ignore them, assign the next available color from the pool.

To color all the vertices, we will have to visit all the vertices. And to ensure that no two adjacent vertices have the same color, we will have to go through all the edges. Hence the complexity of this algorithm will be $O(|V| + |E|)$.

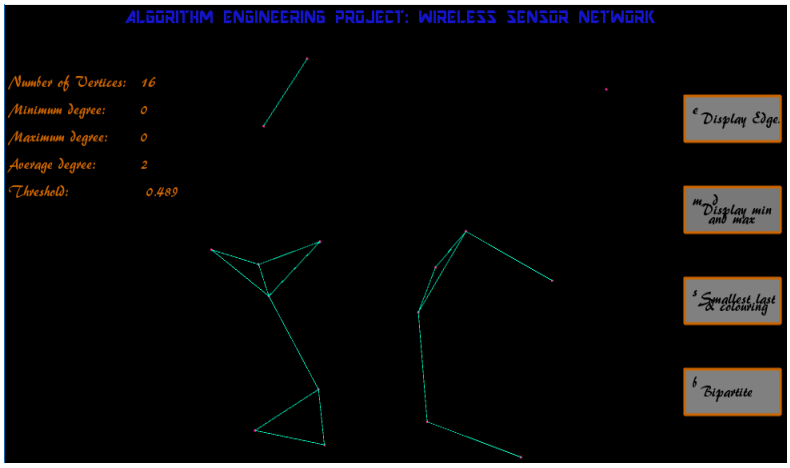
### Bipartite Graph backbone selection:

Since we follow smallest last order for coloring, some of the top colors will contain more number of vertices as opposed to a very few in the later. Out of all the colors possible, we will only consider the first 5 largest colors and we can then get $^5C_2$ = 10 pairs of colors available for selecting our backbone. Out of which we only take forward 2 major backbones.

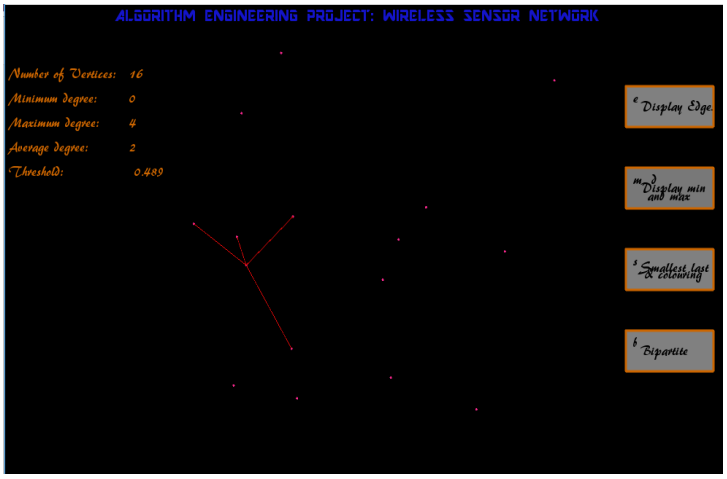
## 4. Verification Walkthrough:

We take 16 vertices in a unit square as an example for walkthrough the entire project step by step. Number of vertices = 16; r= 0.4

We get the following graph with the above inputs:




The maximum degree of the graph is 4 and the minimum degree is 0.




The degree list in this case will be:

| | |
|---|---|
| 0 | 16 |
| 1 | 1,2,10,15 |
| 2 | 13,11,8,9,3,5 |
| 3 | 4,7,12,14 |
| 4 | 6 |

Now, step by step, all the vertices will be deleted and added to a new list in smallest last manner.
First, the vertex with lowest degree will be deleted from the degree list and added to the last available position of the smallest last order list. The degree list and adjacency list will be appropriately altered.
The adjacency list now is as follows:

| | | | |
|---|---|---|---|
| 1 | 2 | 9 | 7,8 |
| 2 | 1 | 10 | 11 |
| 3 | 4,6 | 11 | 10,12 |
| 4 | 3,5,6 | 12 | 11,13,14 |
| 5 | 4,6 | 13 | 12,14 |
| 6 | 3,4,5,7 | 14 | 13,15 |
| 7 | 6,8,9 | 15 | 14 |
| 8 | 7,9 | 16 | Null |

Now, we delete our first node. The node with the smallest degree i.e. at the top of our degree list is node 16. We delete vertex 16 from the degree list and also update adjacency list accordingly:

Degree list

| | |
|---|---|
| 0 | |
| 1 | 1,2,10,15 |
| 2 | 13,11,8,9,3,5 |
| 3 | 4,7,12,14 |
| 4 | 6 |

Adjacency list

| | | | |
|---|---|---|---|
| 1 | 2 | 9 | 7,8 |
| 2 | 1 | 10 | 11 |
| 3 | 4,6 | 11 | 10,12 |
| 4 | 3,5,6 | 12 | 11,13,14 |
| 5 | 4,6 | 13 | 12,14 |
| 6 | 3,4,5,7 | 14 | 13,15 |
| 7 | 6,8,9 | 15 | 14 |
| 8 | 7,9 | - | - |

Since vertex 16 had degree 0, nothing was changed in adjacency list.
The new list of smallest last ordering is as follows:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Node | | | | | | | | | | | | | | | 16 |
| Degree when deleted | | | | | | | | | | | | | | | 0 |

The next vertex to be deleted will be vertex 1.

Degree list

| | |
|---|---|
| 0 | 2 |
| 1 | 10,15 |
| 2 | 13,11,8,9,3,5 |
| 3 | 4,7,12,14 |
| 4 | 6 |

Adjacency list

| | | | |
|---|---|---|---|
| - | - | 9 | 7,8 |
| 2 | null | 10 | 11 |
| 3 | 4,6 | 11 | 10,12 |
| 4 | 3,5,6 | 12 | 11,13,14 |
| 5 | 4,6 | 13 | 12,14 |
| 6 | 3,4,5,7 | 14 | 13,15 |
| 7 | 6,8,9 | 15 | 14 |
| 8 | 7,9 | 16 | Null |

Since vertex 16 had degree 0, nothing was changed in adjacency list.
The new list of smallest last ordering is as follows:

| Node | | | | | | | | | | | | | | 1 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Degree when deleted | | | | | | | | | | | | | | 1 | 0 |

After all the vertices have been deleted from the degree list as per the steps discussed, we get a smallest order list that looks like this:

| Node | 14 | 13 | 12 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 15 | 11 | 10 | 2 | 1 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Degree when deleted | 0 | 1 | 2 | 0 | 1 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 0 | 1 | 0 |

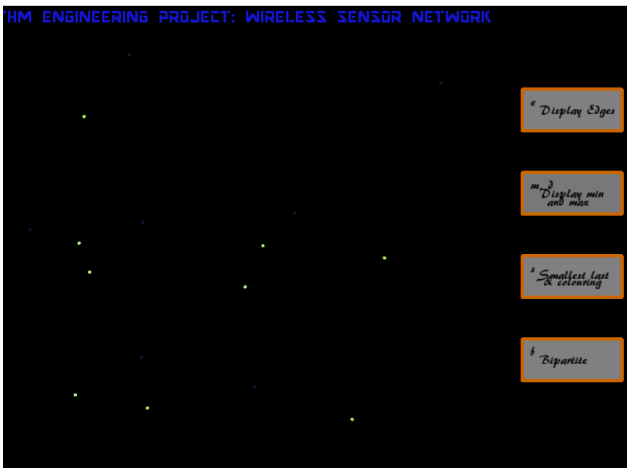
This is the result of smallest last ordering algorithm.
The next step done is coloring. The above list is considered as an input in coloring algorithm. Coloring is started by assigning a color to our first node in the above list. So, vertex 14 will get the color C0. Now for the next vertex, it is checked if it's adjacent vertices have the color C0. The adjacent vertex to 13 is 14 and it has the color C0. So, the next color C1 will be considered for vertex 13. The next vertex to be considered is 12. An adjacent vertex of 12 already has the color C0 assigned (14) and another adjacent vertex (13) has color C1 assigned, so the vertex 12 is assigned color C2.

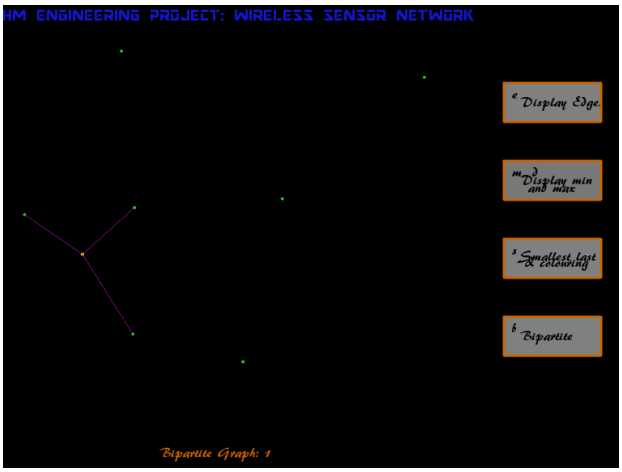
| Node | 14 | 13 | 12 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 15 | 11 | 10 | 2 | 1 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| color | C0 | C1 | C2 | | | | | | | | | | | | | |

Now, moving forward, now of the adjacent vertices of vertex 9 has been assigned the color C0. So, color C0 is assigned to vertex 9. Moving forward in this way, all the vertices are colored:

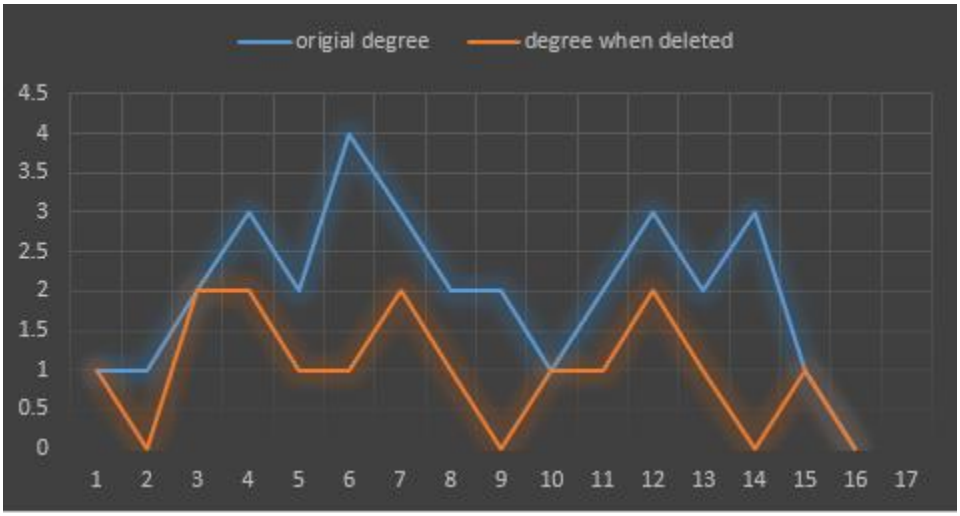
| Node | 14 | 13 | 12 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 15 | 11 | 10 | 2 | 1 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| color | C0 | C1 | C2 | C0 | C1 | C2 | C0 | C1 | C2 | C1 | C1 | C1 | C1 | C0 | C1 | C0 |

The next step is forming bi partite graph. The result of backbone selection is in the image below



Summary:


origial degree
degree when deleted
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

# III. Benchmark Result Summary

## 1. Benchmark1(SQUARE):

N = 1000; average degree = 32

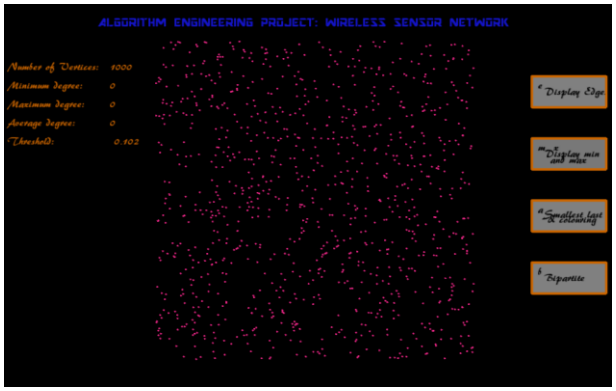



*Figure 1: randomly distributed points on a unit square*

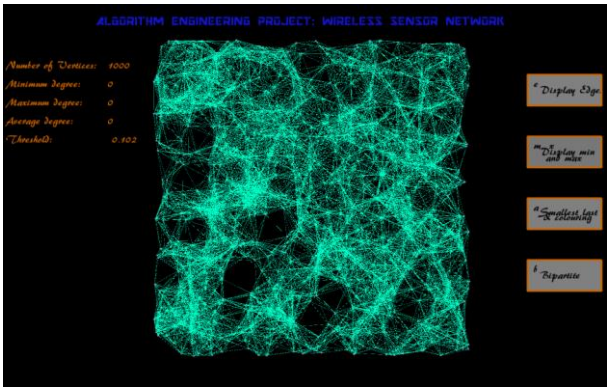



*Figure 2: edges*

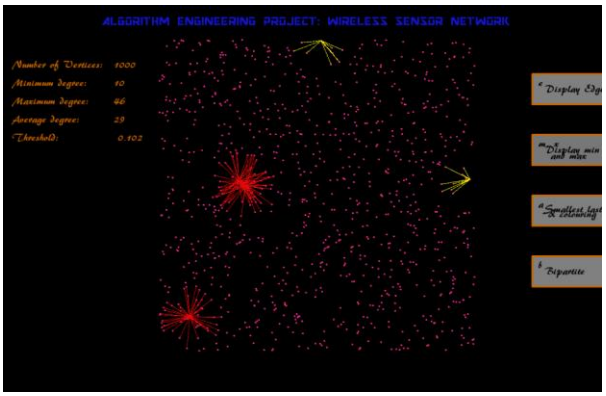



*Figure 3: minimum and maximum degree edges*

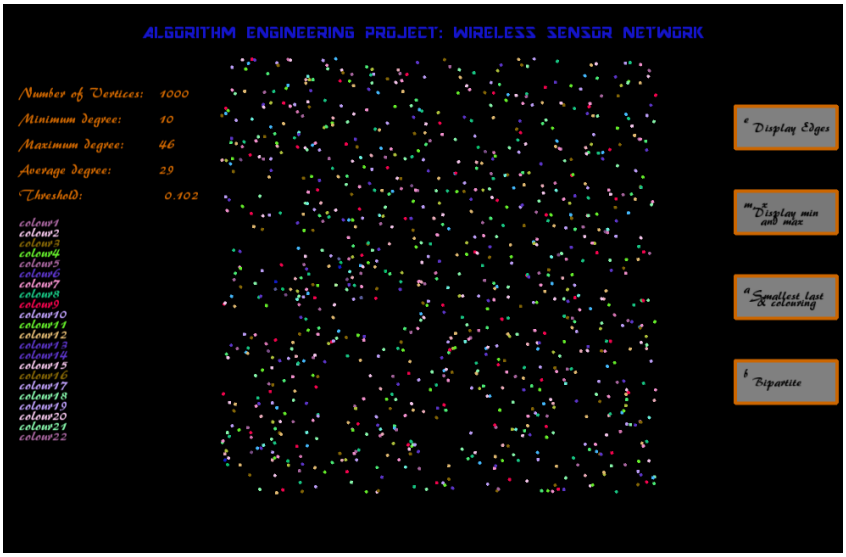



*Figure 4 smallest last ordering and colouring algorithm*

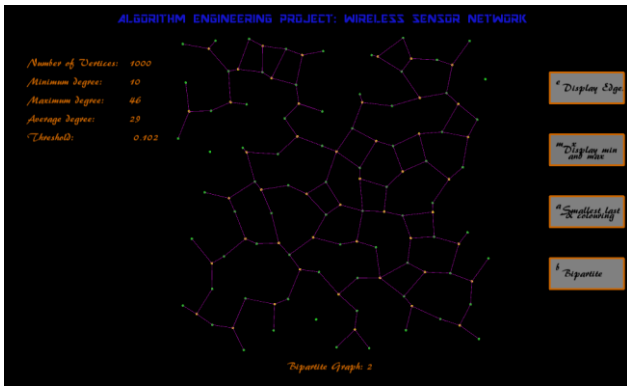



*Figure 5 backbone 2*

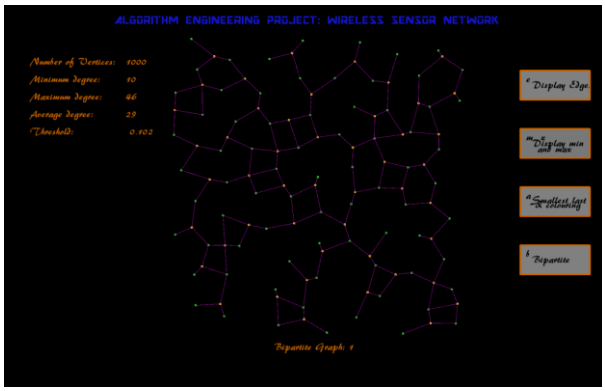
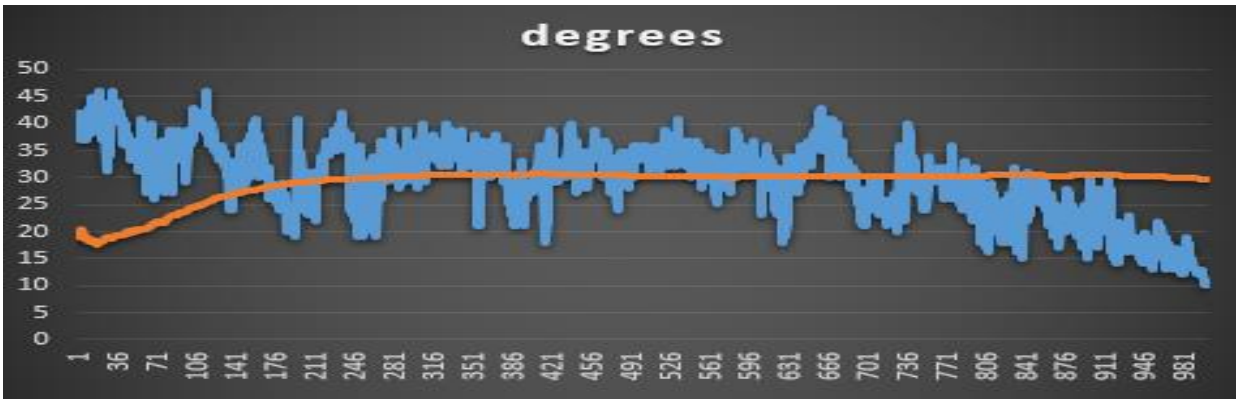
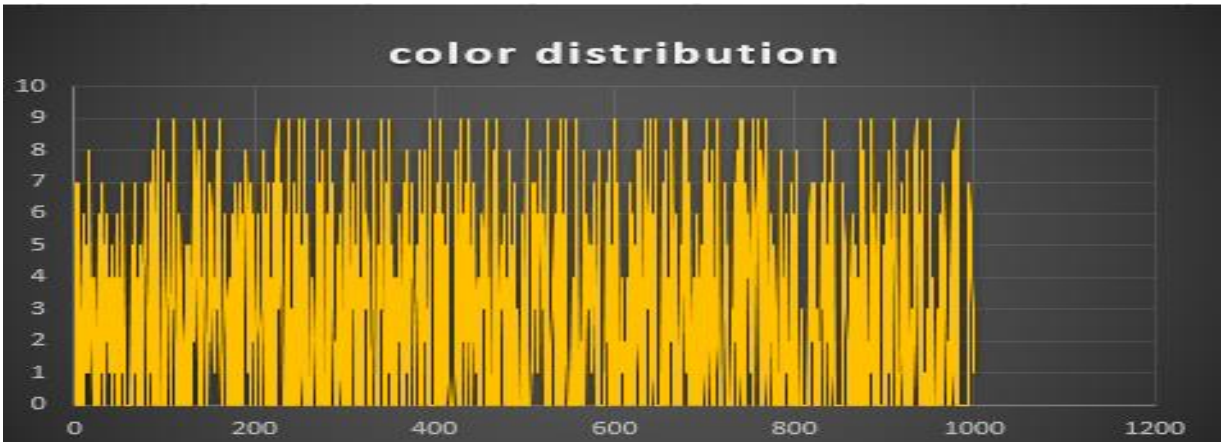



*Figure 6 backbone 1*




*Graph 1: degree distribution*



## 2. Benchmark2(SQUARE):

N = 4000; Average degree = 64

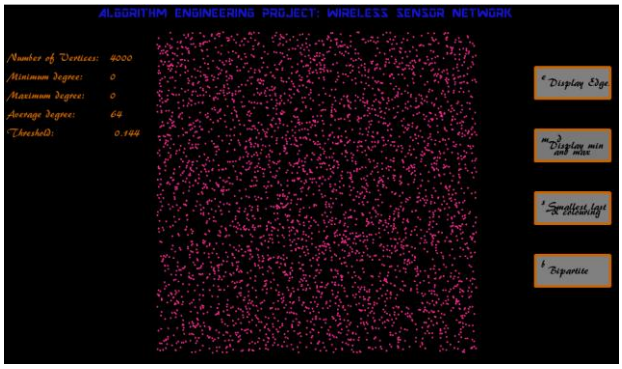



*Figure 7: randomly generated points*

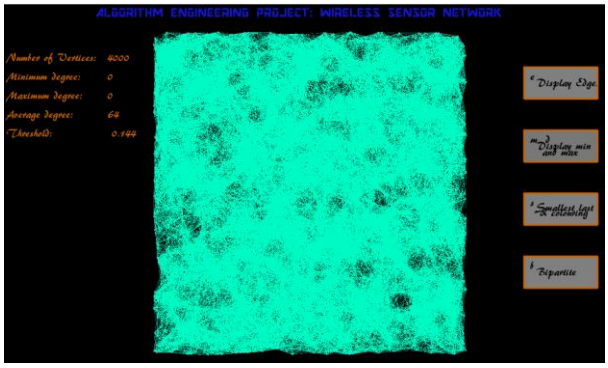



*Figure 8: edges*

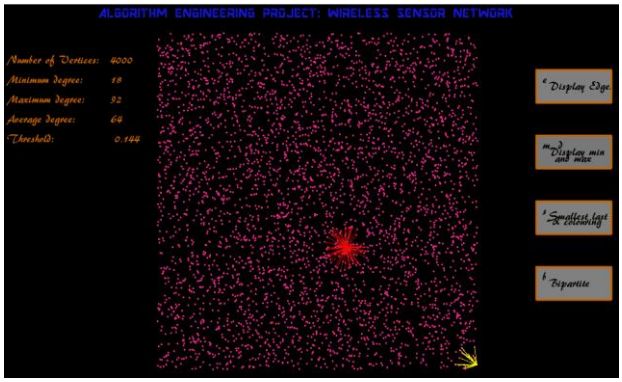



*Figure 9: min and max edges*

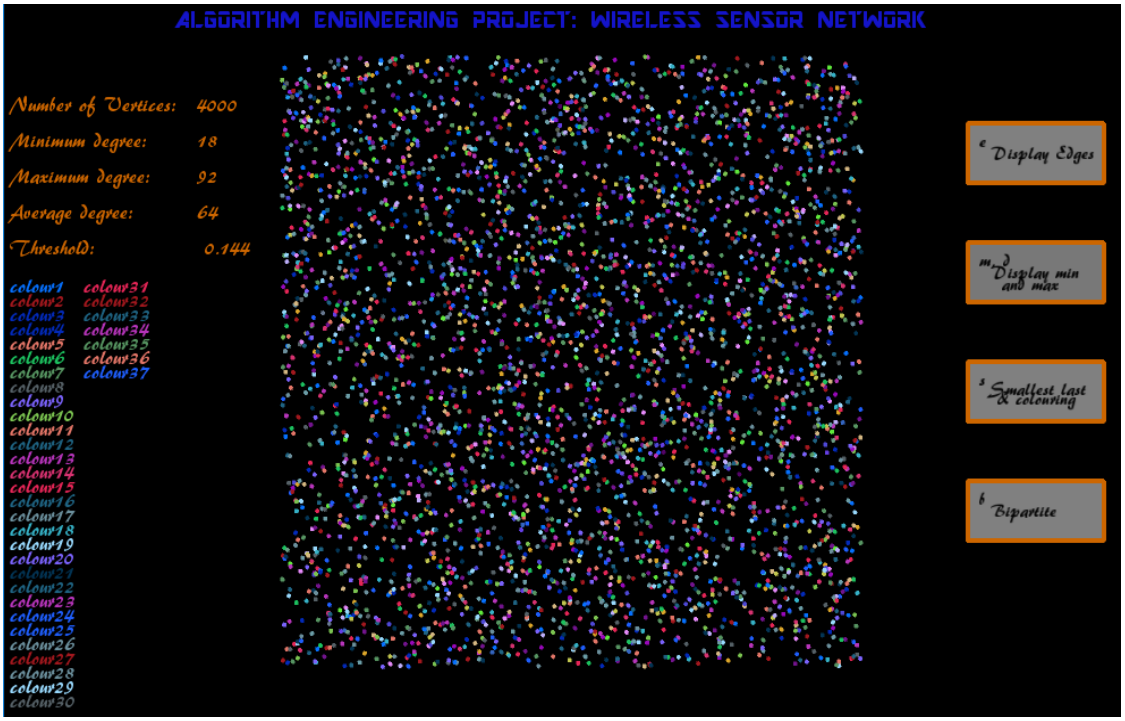



*Figure 10: smallest last ordering and coloring*

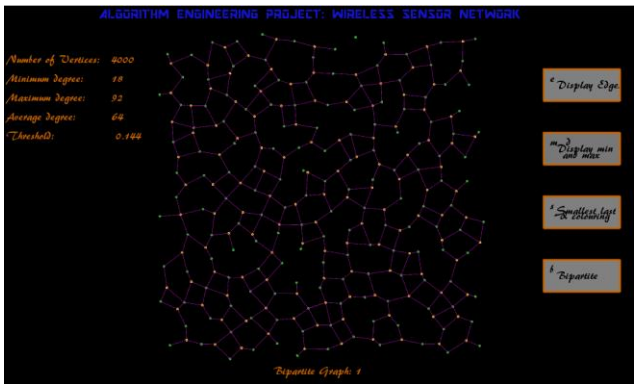



*Figure 11: backbone 1*

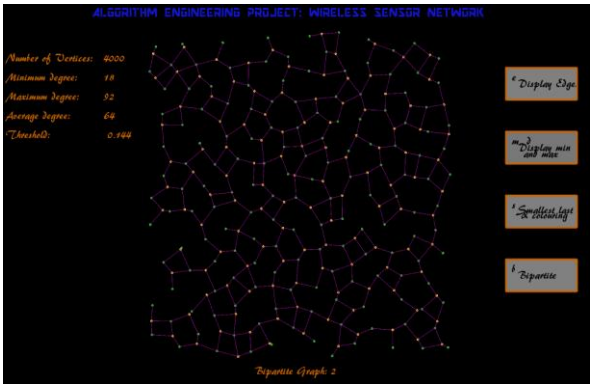
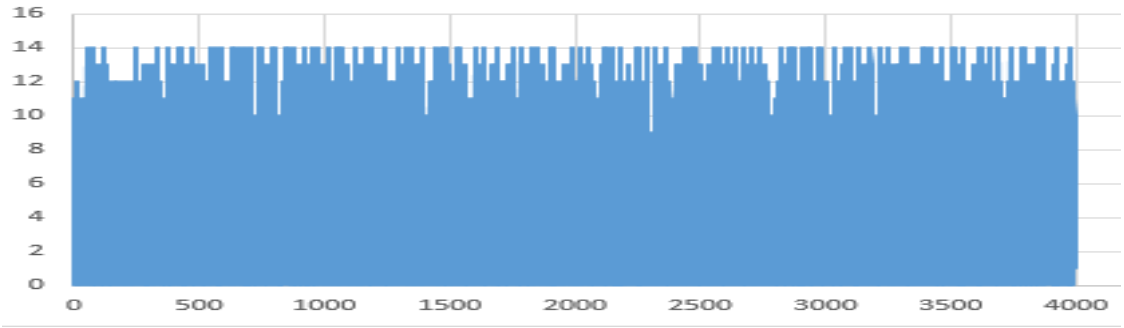
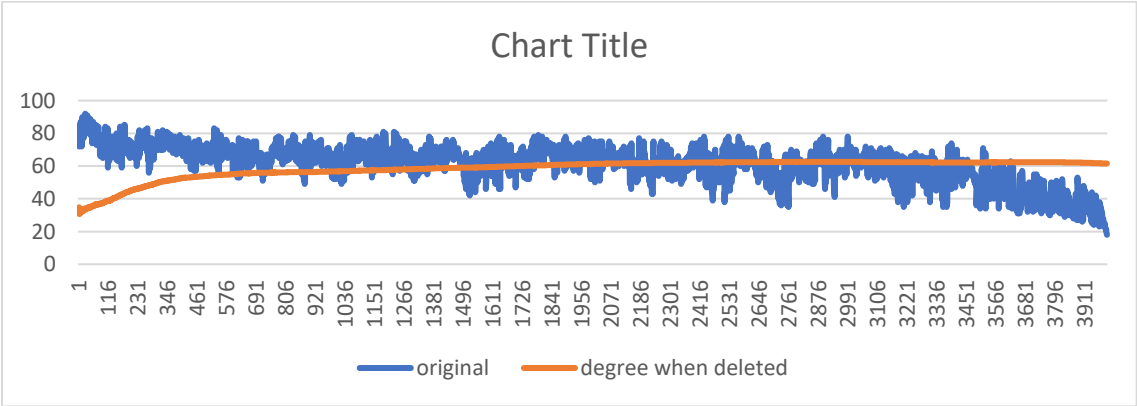



*Figure 12: backbone 2*

Summary:




*Graph 2: color distribution*




*Graph 3: degree distribution*

## 3. Benchmark3(SQUARE):

N= 16000; average degree = 64

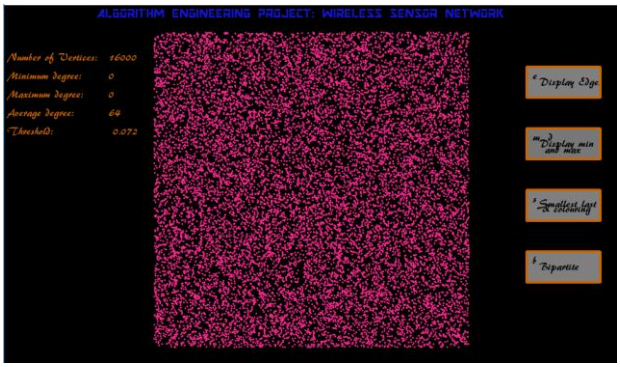



*Figure 13:RGG*

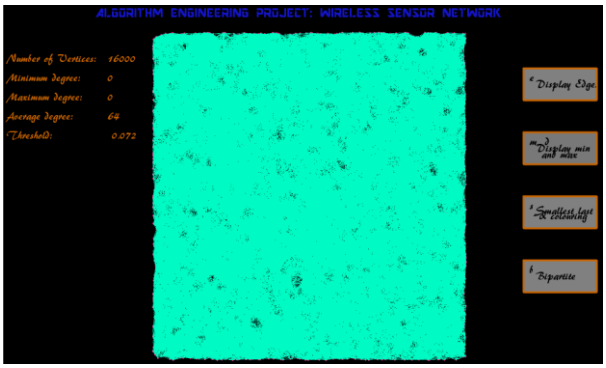



*Figure 14 edges*

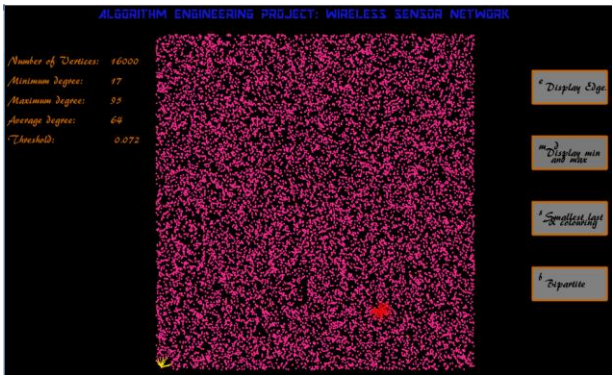



*Figure 15: min and max degree edges*

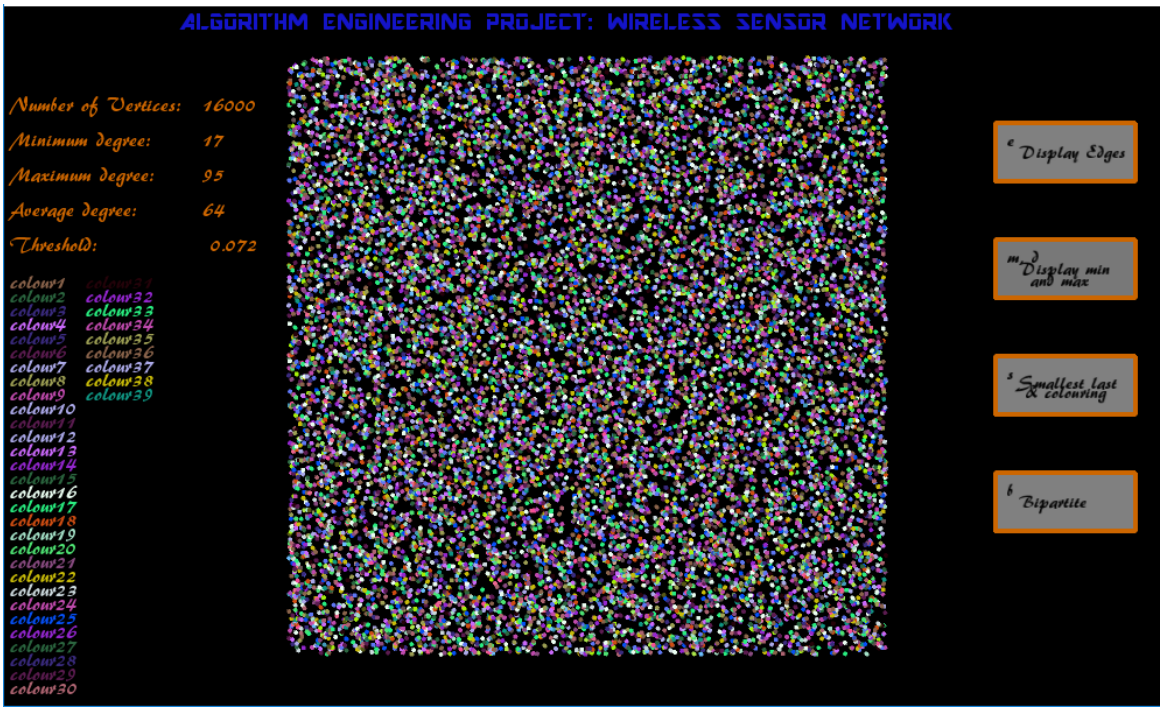



*Figure 16: smallest last ordering and coloring*

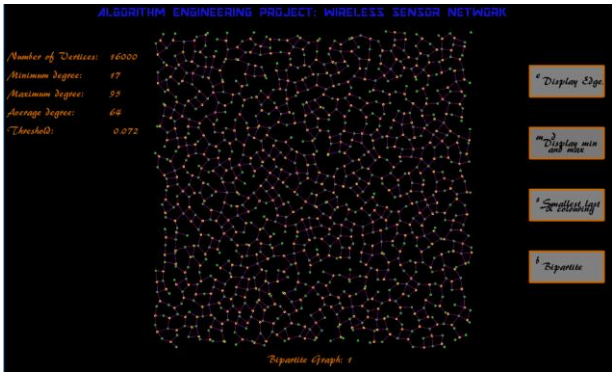



*Figure 17: backbone 1*

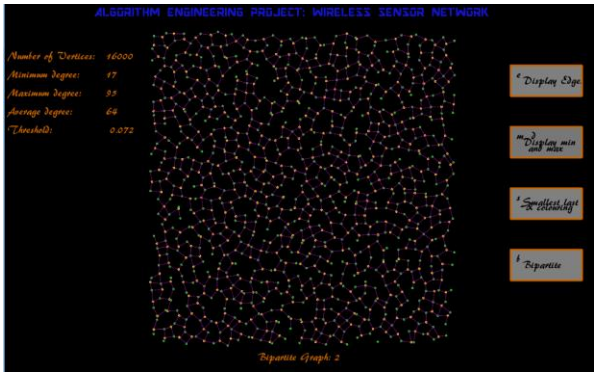
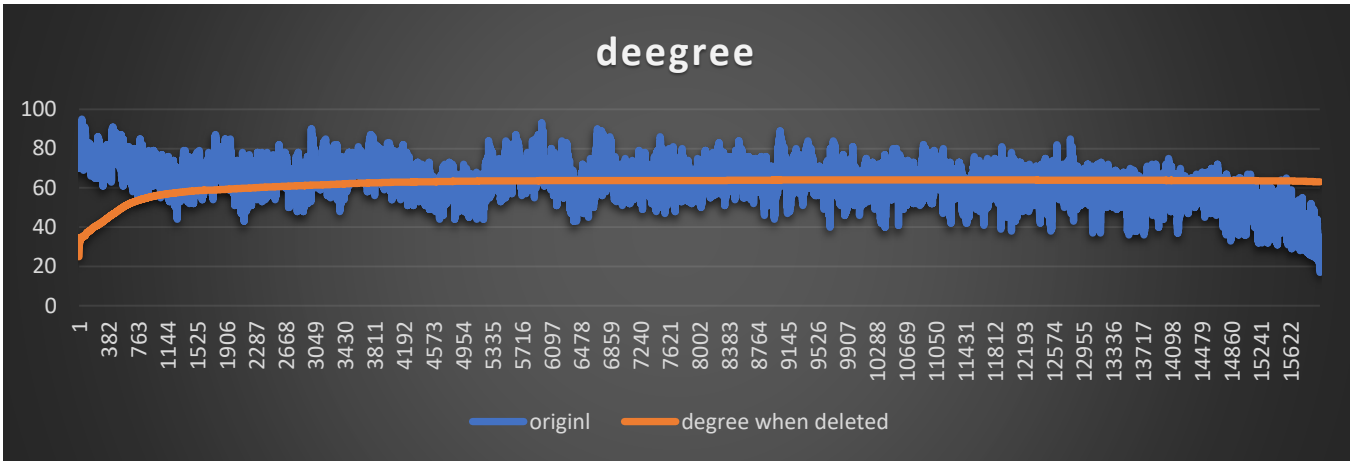
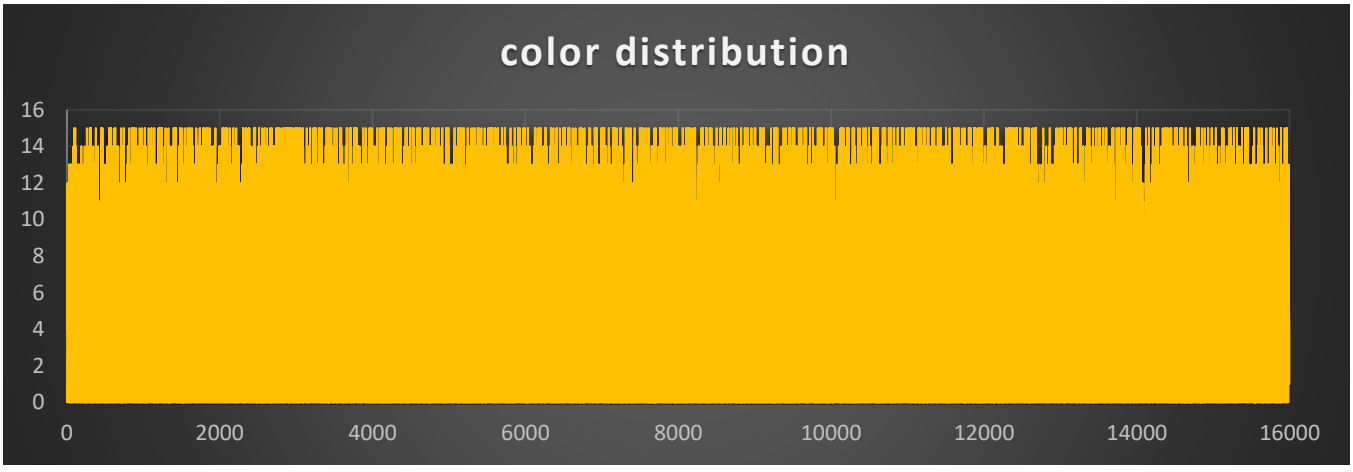



*Figure 18 backbone 2*

Summary:



## 4. Benchmark4(SQUARE):

N = 64000; avg degree = 64

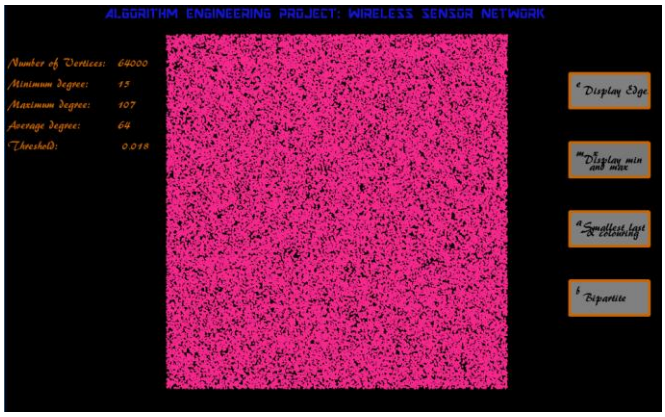



*Figure 19uniformly random points*

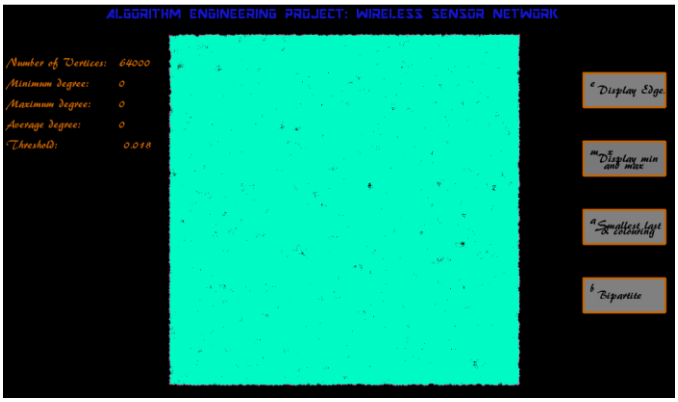



*Figure 20edges*

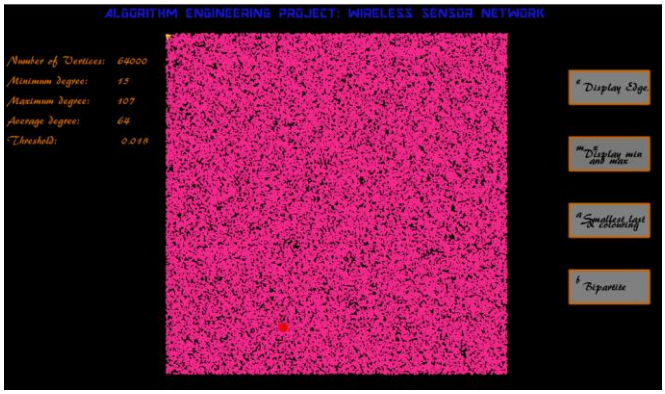



*Figure 21: min and max edges*

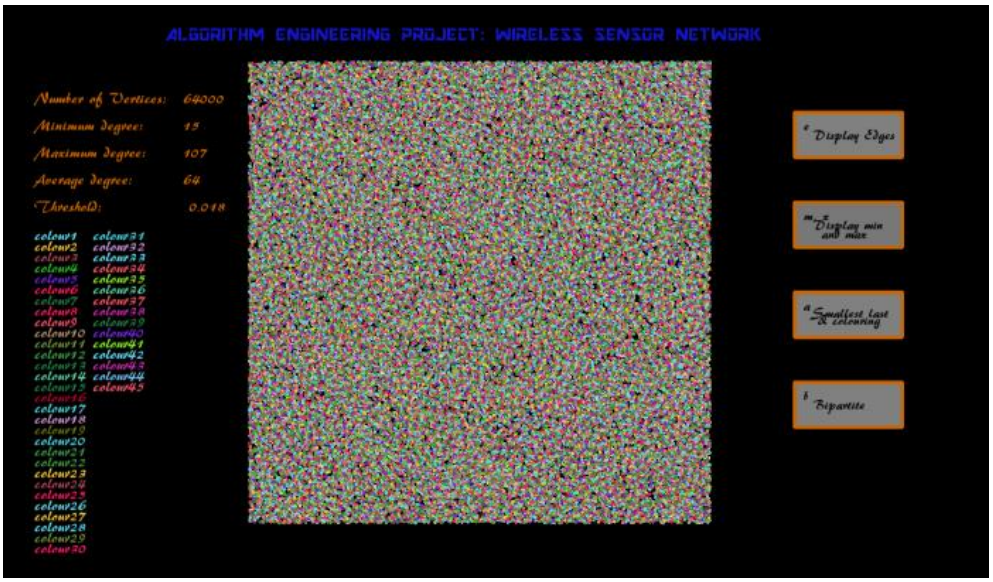



*Figure 22 result of smallest last and coloring*

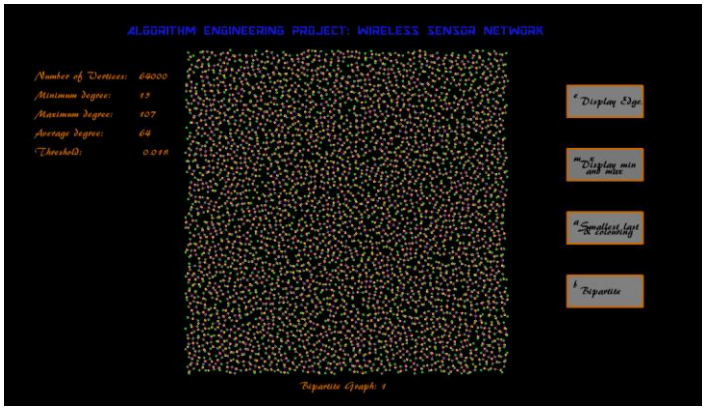



*Figure 23 backbone1*

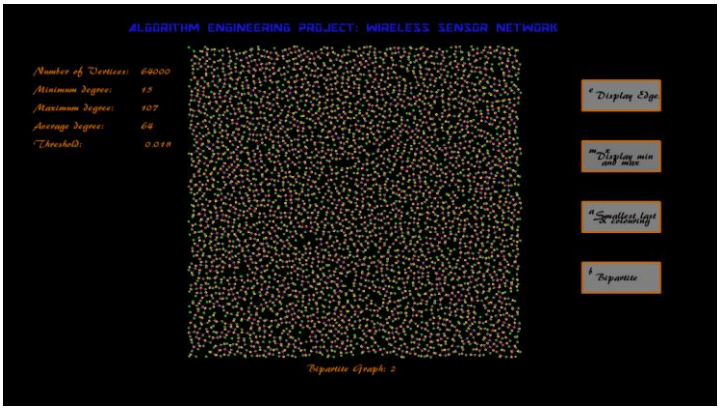
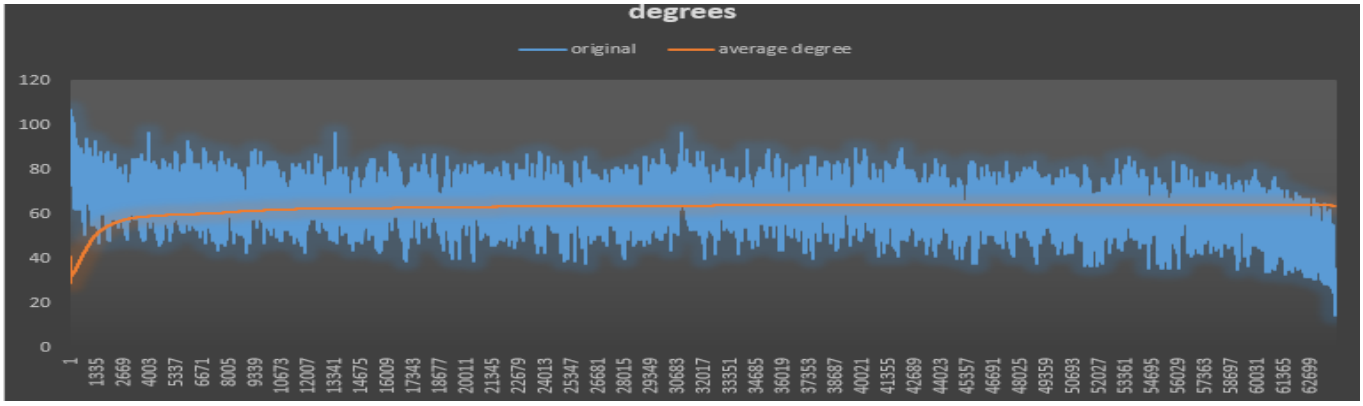
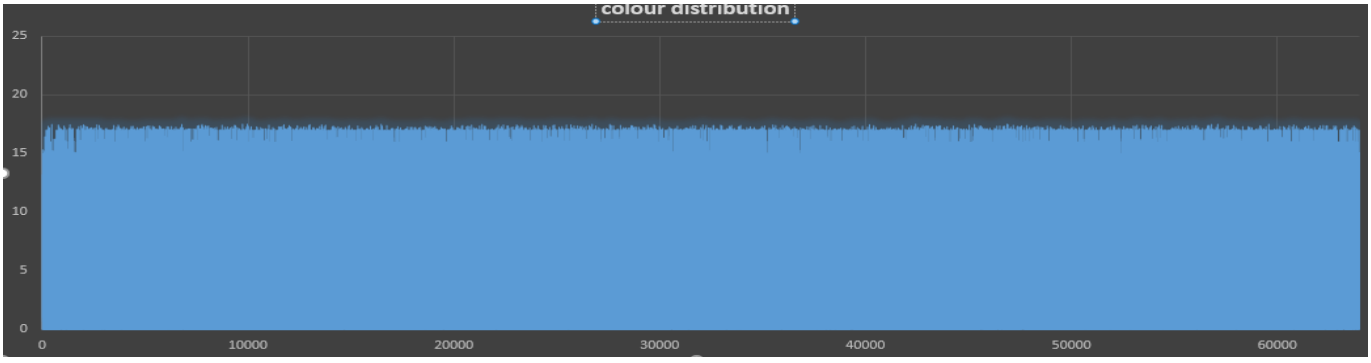



*Figure 24: backbone2*

## Summary:




*Graph 4: degree comparison*




*Graph 5: color distribution*

## 5. Benchmark5(SQUARE):

N = 64000;  average degree = 128

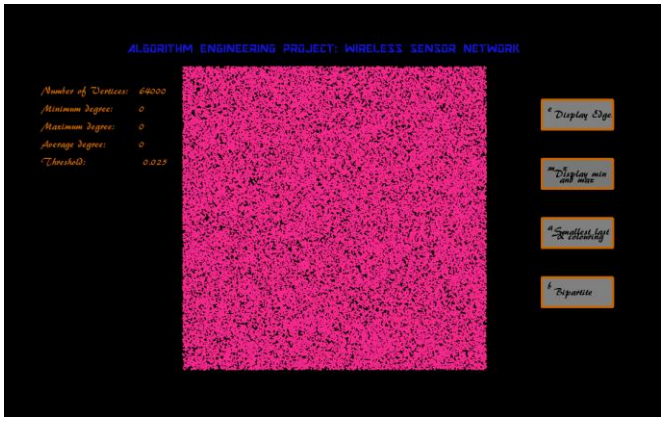



*Figure 25 RGG*

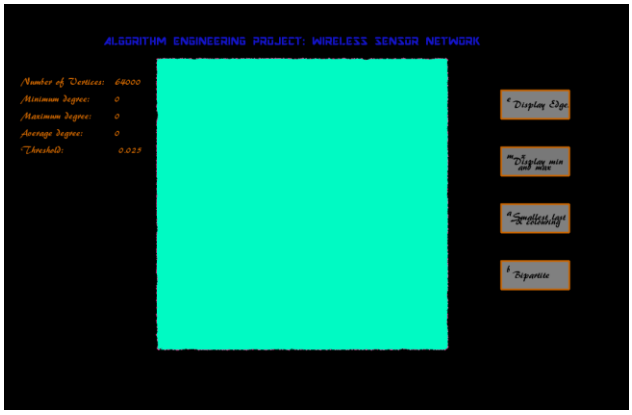



*Figure 261: edges*

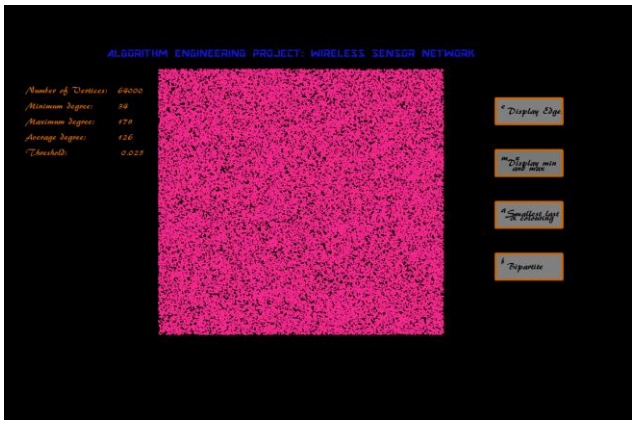



*Figure 27 min max calculation*

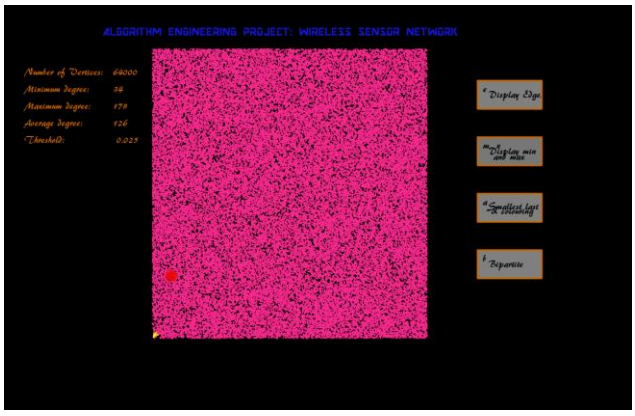



*Figure 28 min max degree edges*

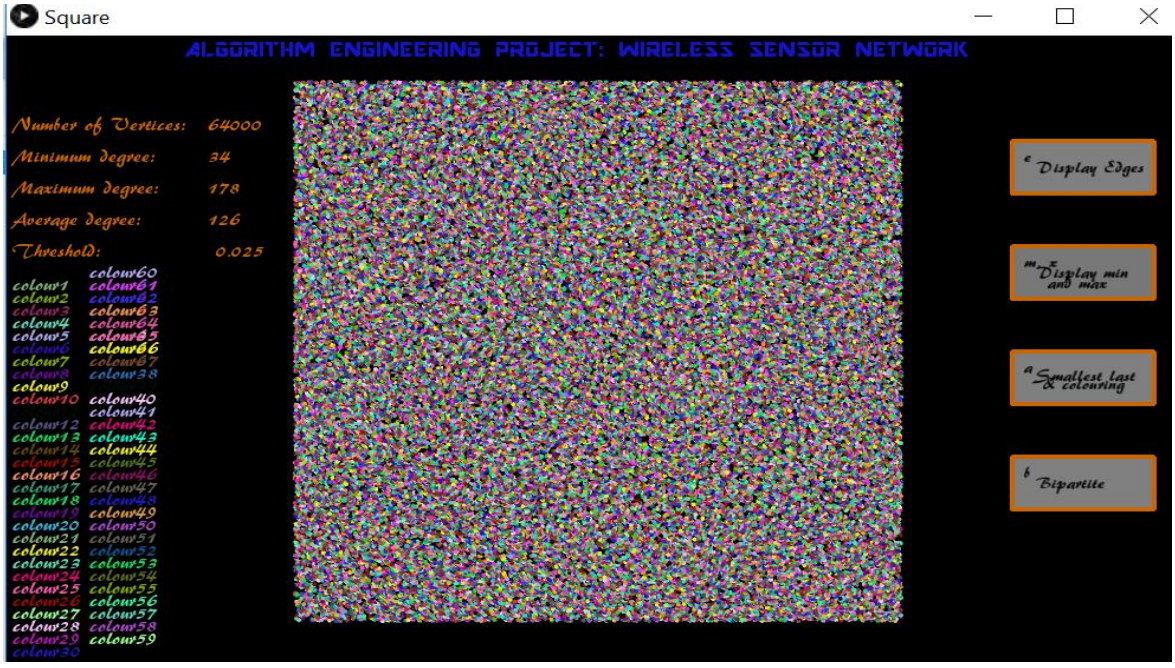



*Figure 29: Smallest last ordering and coloring*

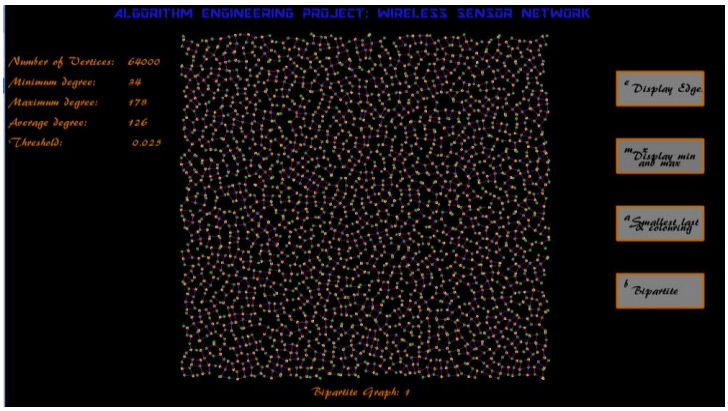



*Figure 30: backbone1*

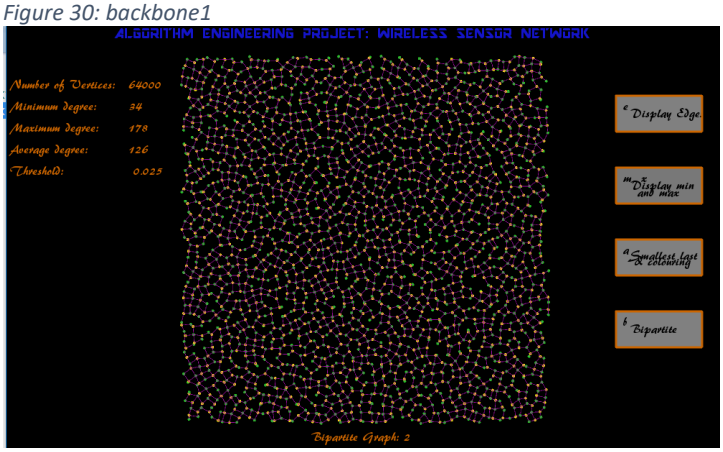
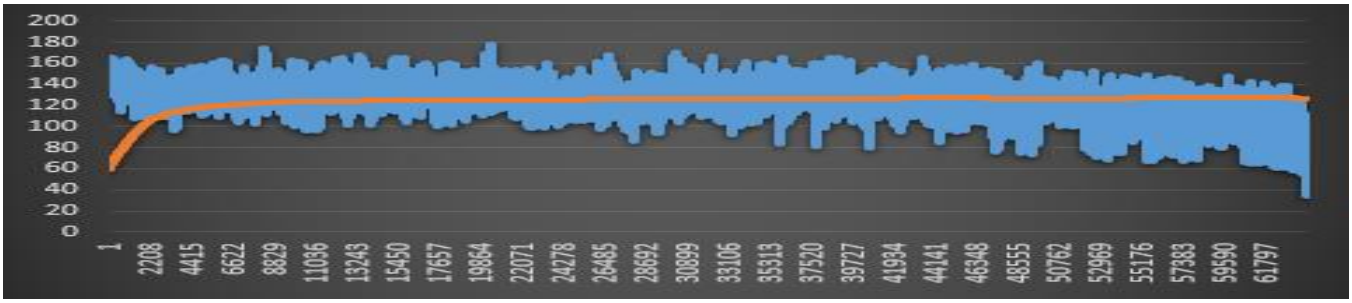
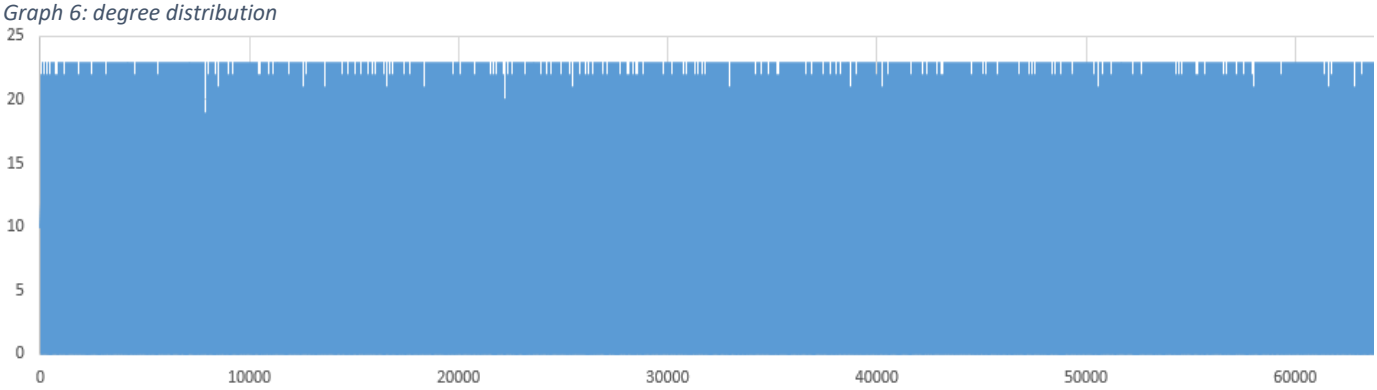



*Figure 31: backbone2*

## Summary:




*Graph 6: degree distribution*




*Graph 7: color distribution*

## 6. Benchmark6(DISK):

N= 4000; Average degree= 64

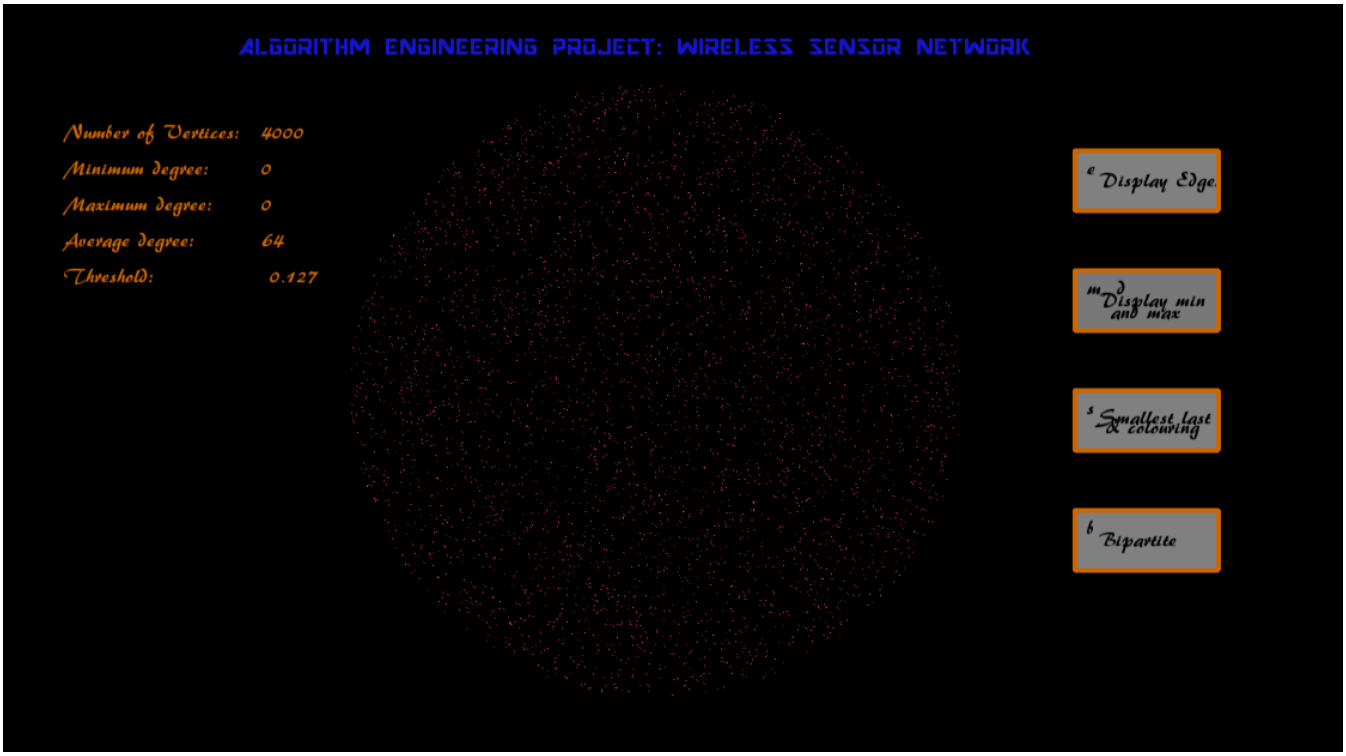



*Figure 32 RGG*

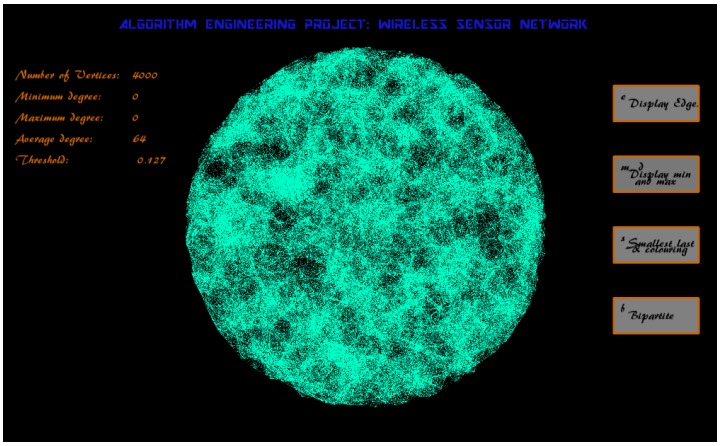



*Figure 33: edges*

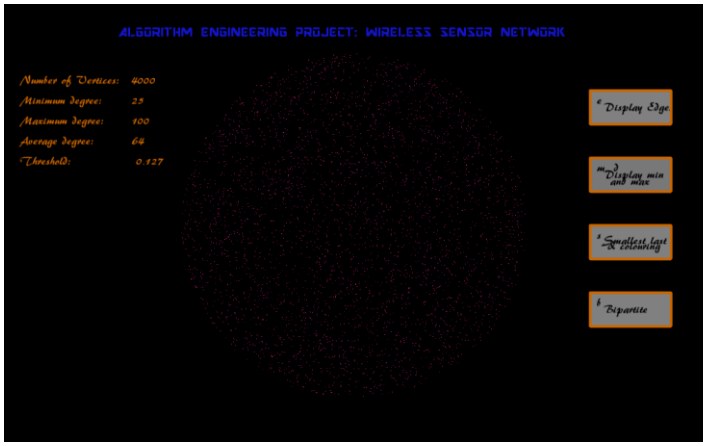



*Figure 34:min and max degree calculation*

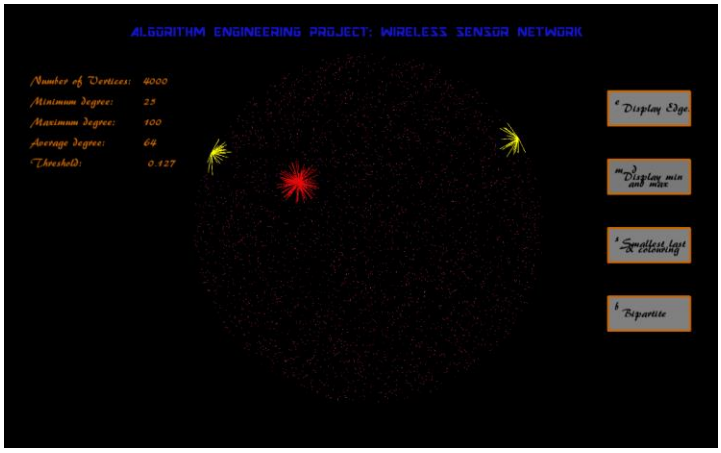



*Figure 35: min and max degree edges*

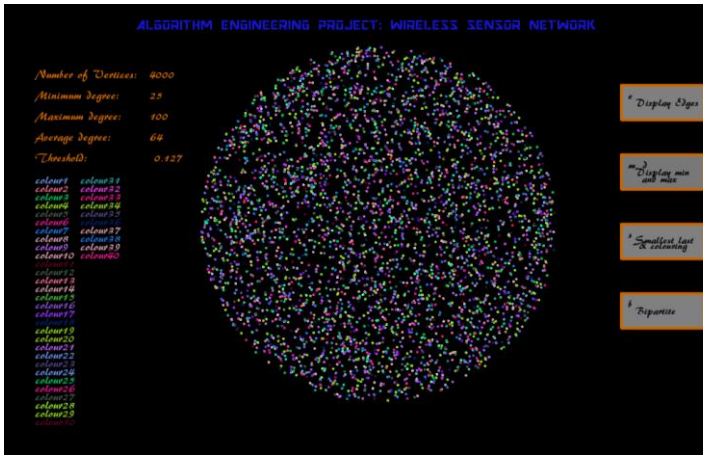
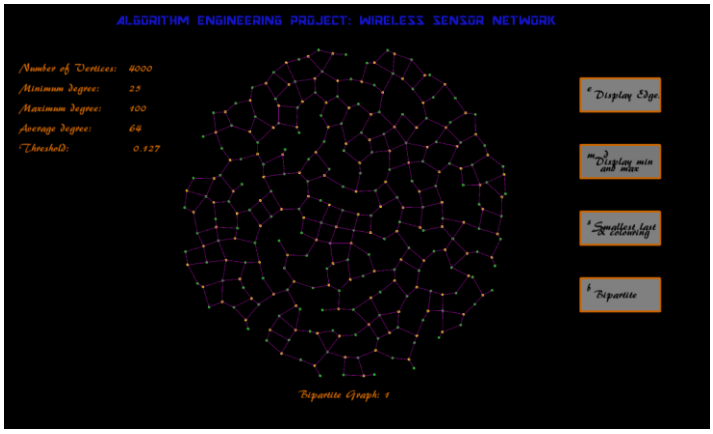



*Figure 36: smallest last ordering and coloring*



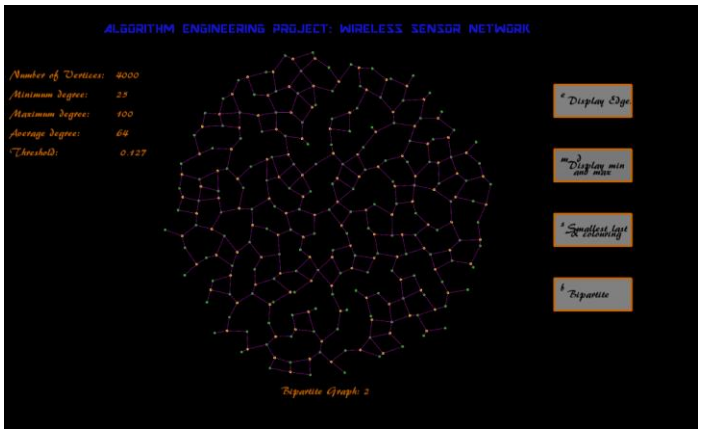
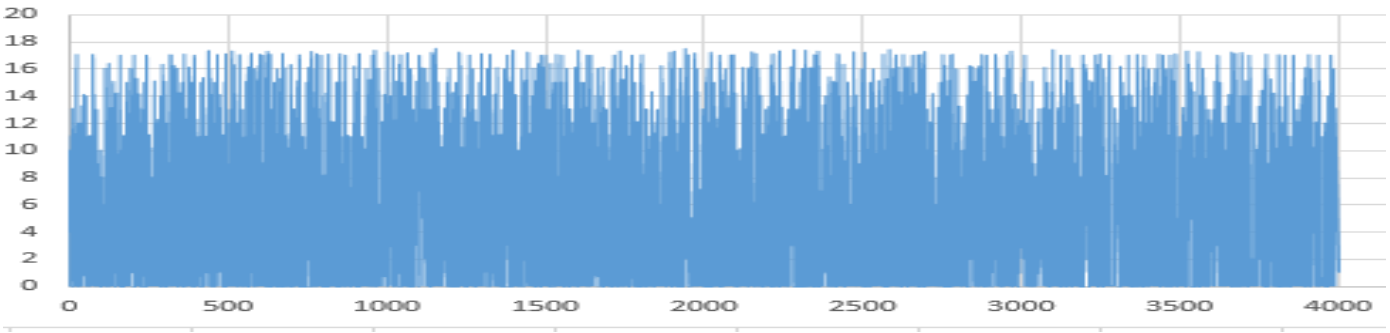
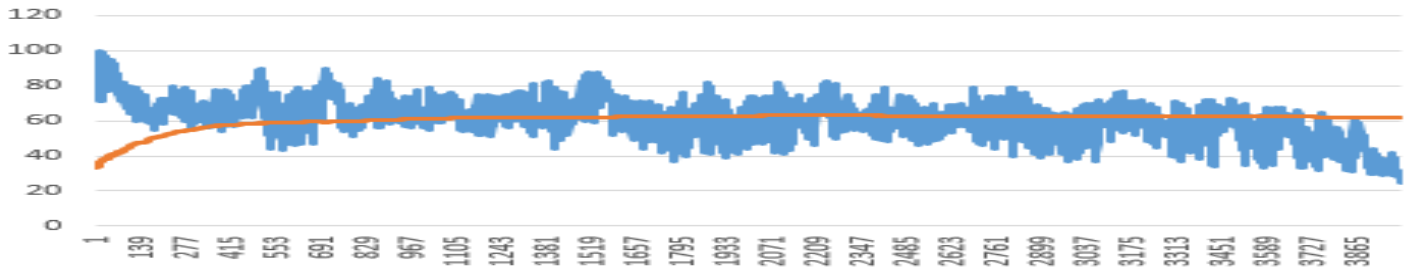




*Figure 37: backbone1*

## Summary:




*Graph 8: color distribution*




*Graph 9 DEGREE DISTRIBUTION*

## 7. Benchmark7(DISK):

N = 4000; Average degree: 128

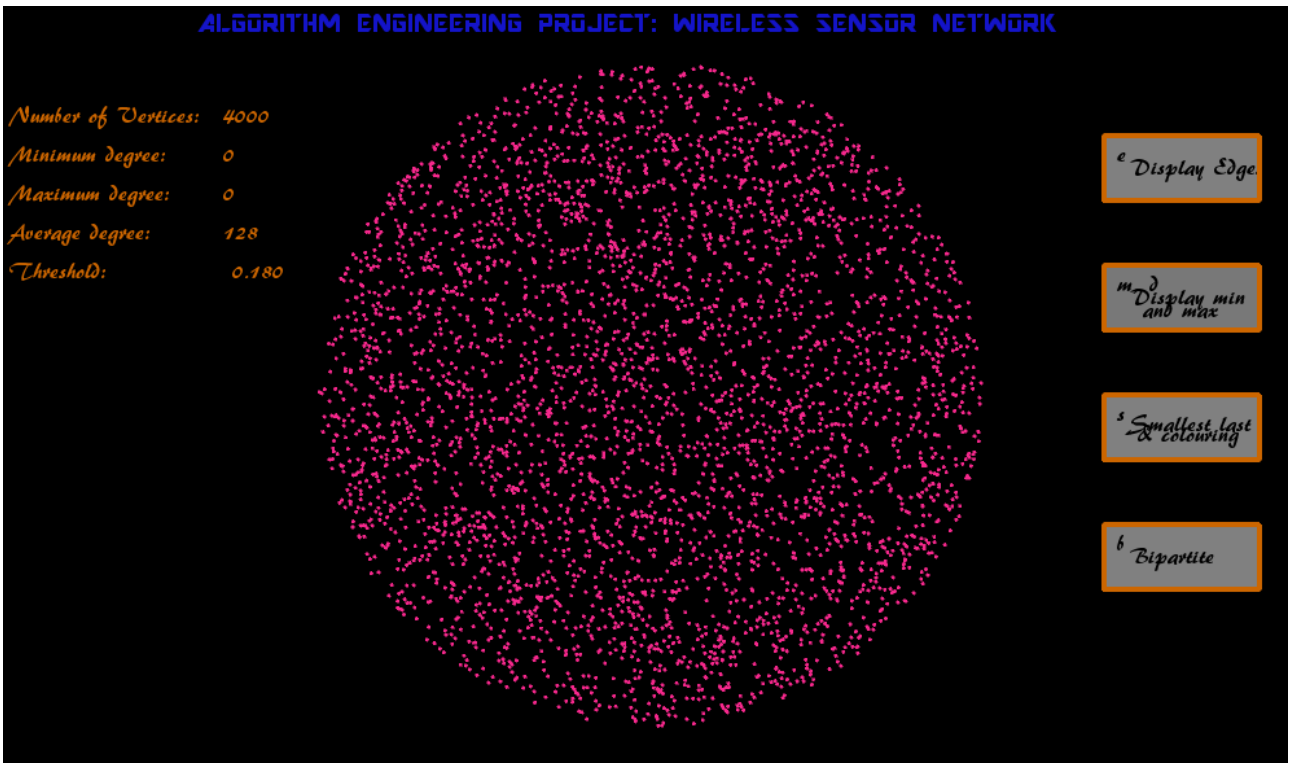



*Figure 39: random distribution of 4000 points over a unit disk*

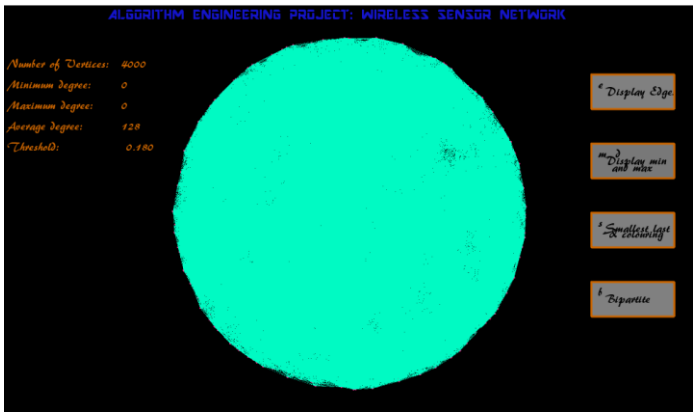



*Figure 40: edges*

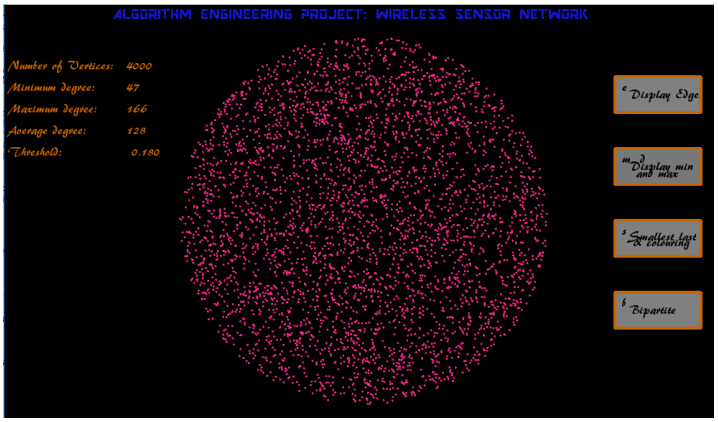



*Figure 41: min and max degree calculation*

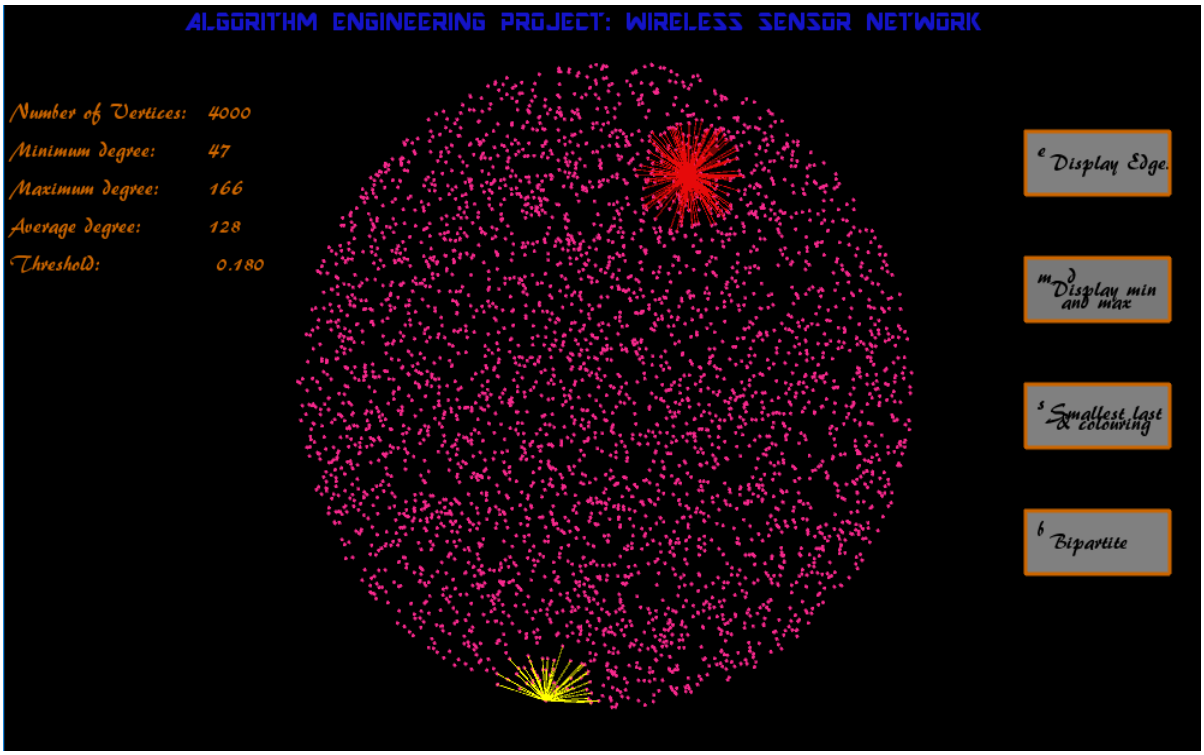



*Figure 42: min and max degree edges*

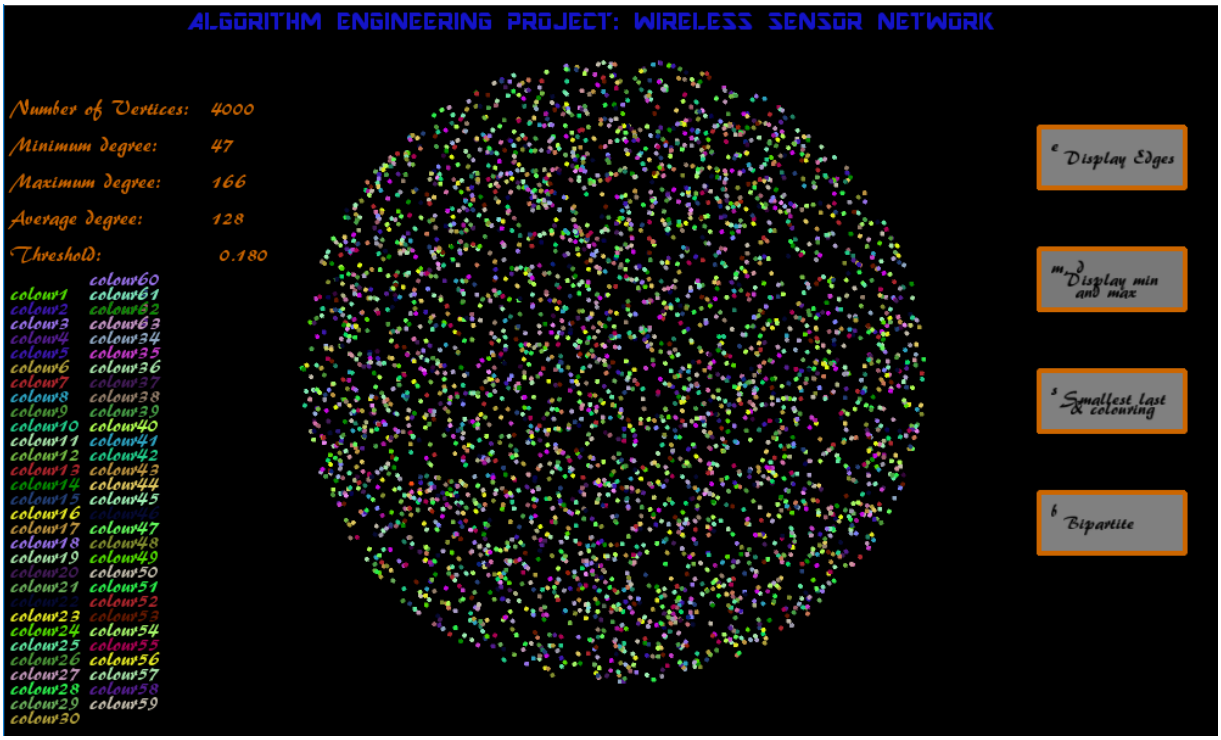



*Figure 43: smallest last ordering and coloring*

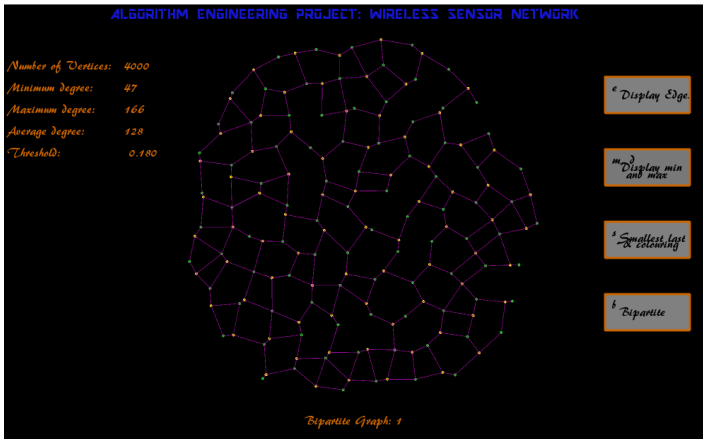



*Figure 44: backbone1*

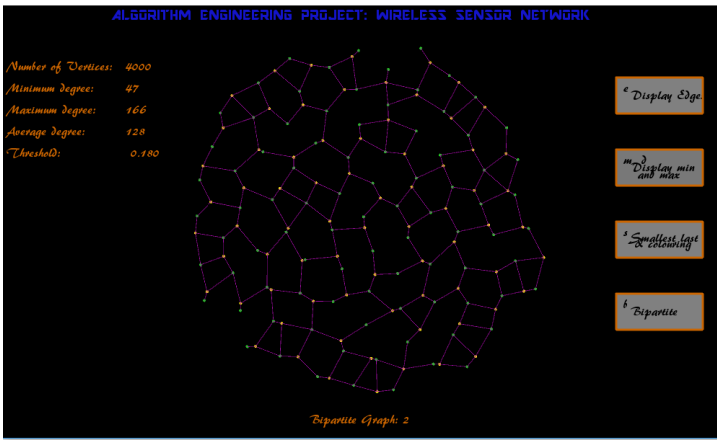
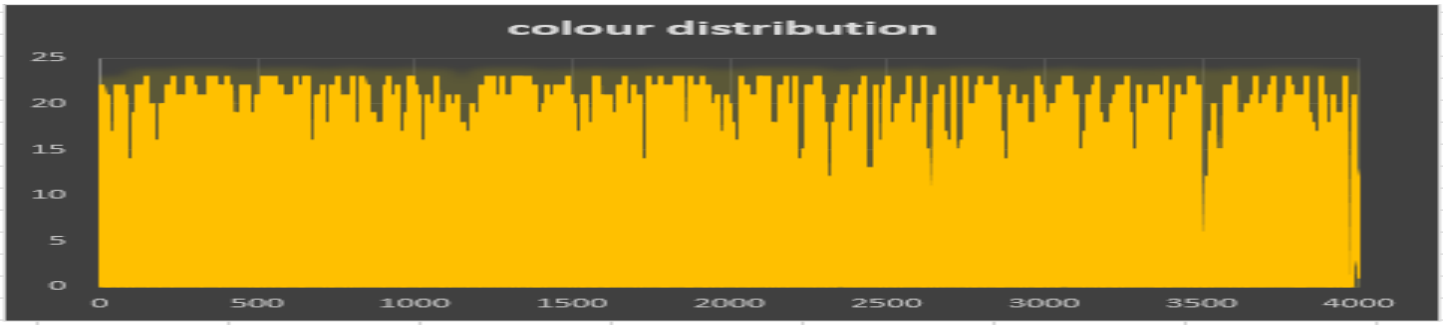
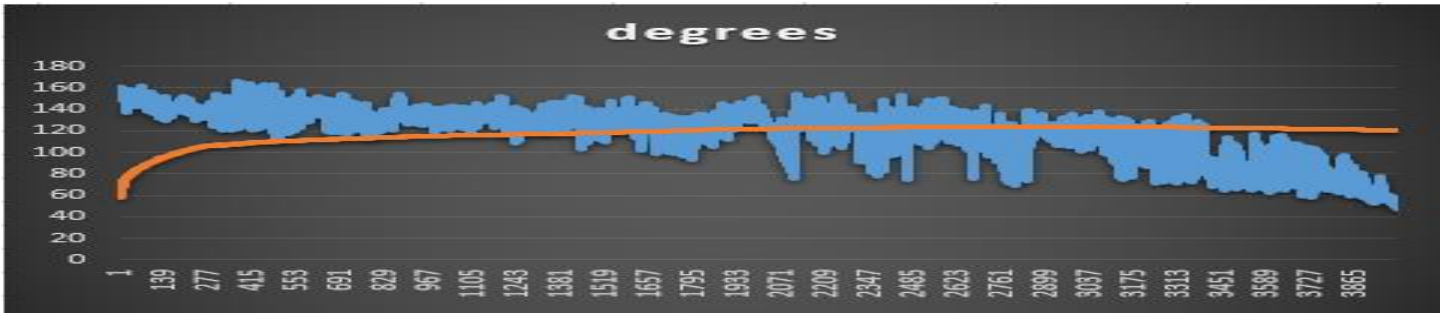



*Figure 45: backbone2*

## Summary:




*Graph 10 color distribution*




*Graph 11 degree distribution*

## 8. Benchmark8 (SPHERE):

N=4000; avg degree = 64

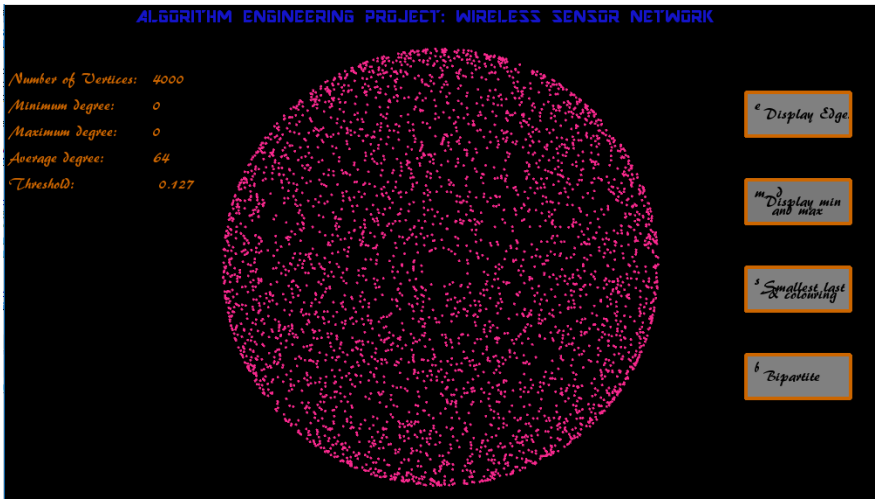



*Figure 46: 4000 random points on a sphere*

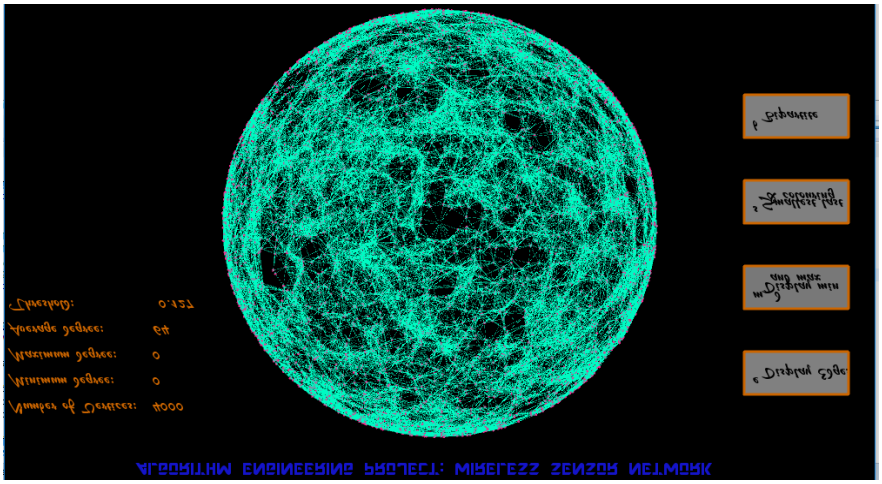


*Figure 47: edges*

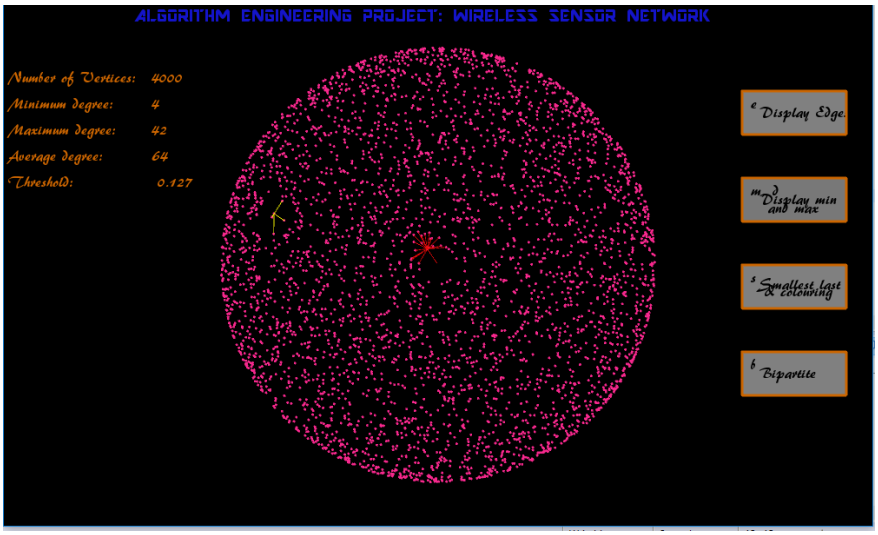



*Figure 48: min and max degree edges*

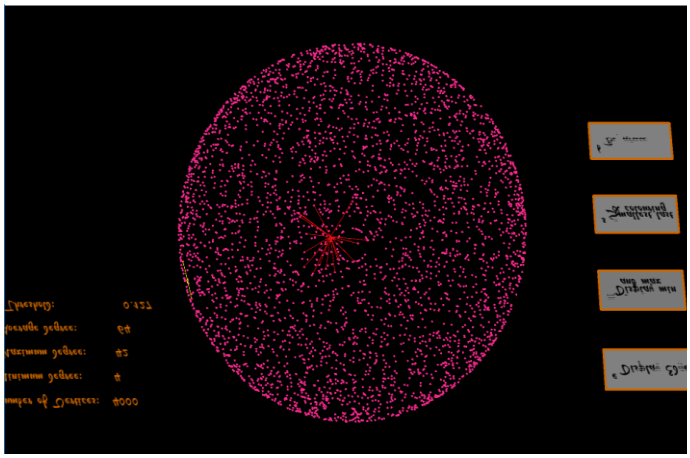


*Figure 49: sphere rotated for better view*

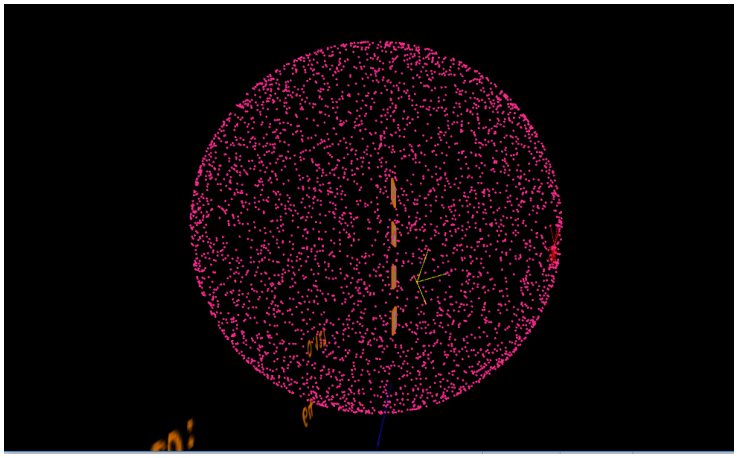


*Figure 50: sphere rotated for better view*

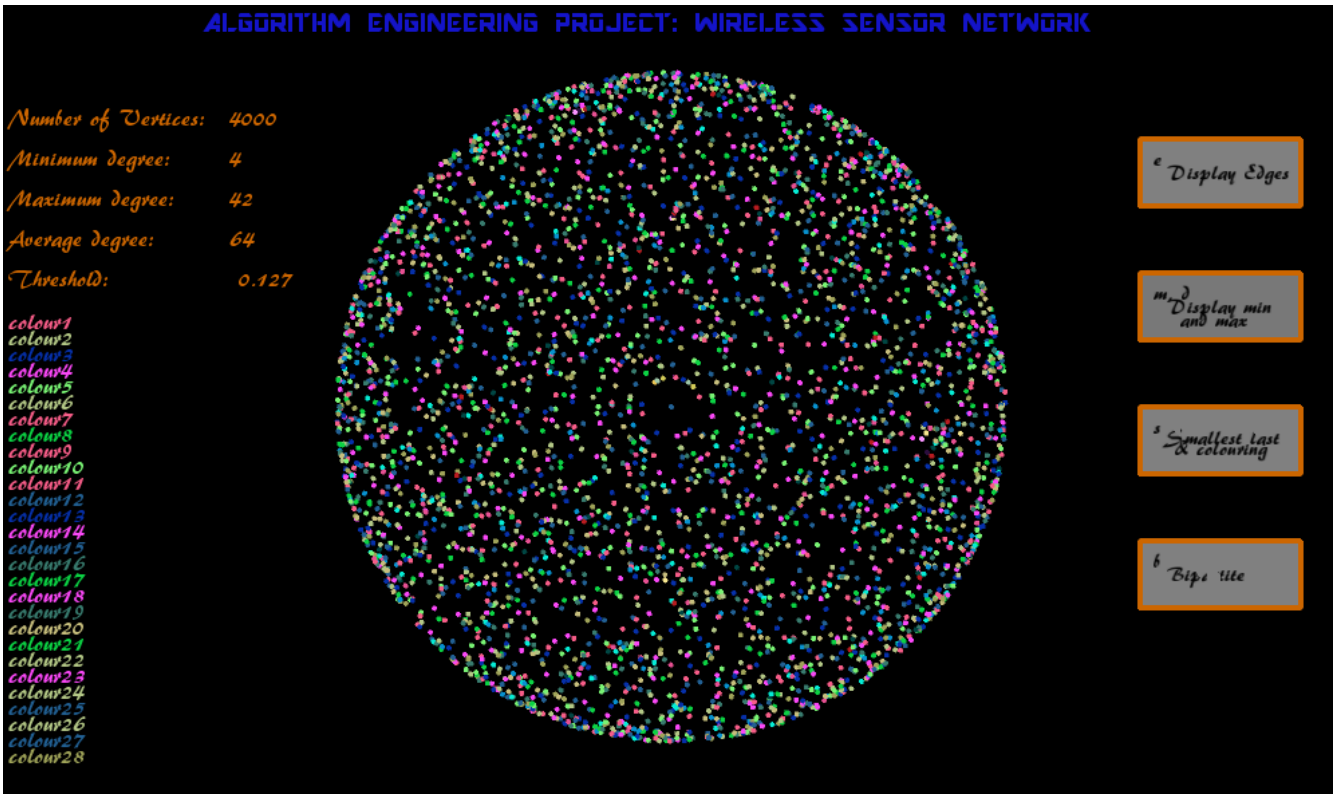



*Figure 51: smallest last ordering and coloring*

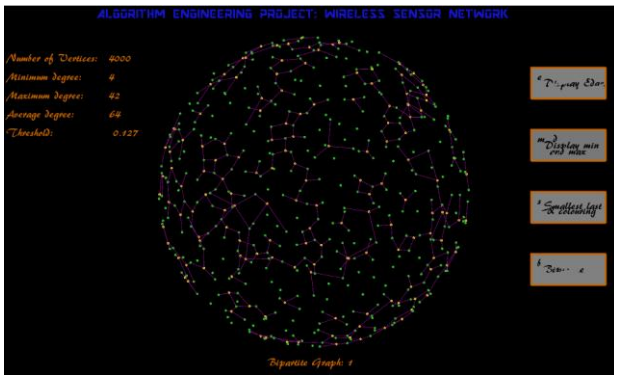



*Figure 52: backbone 1*

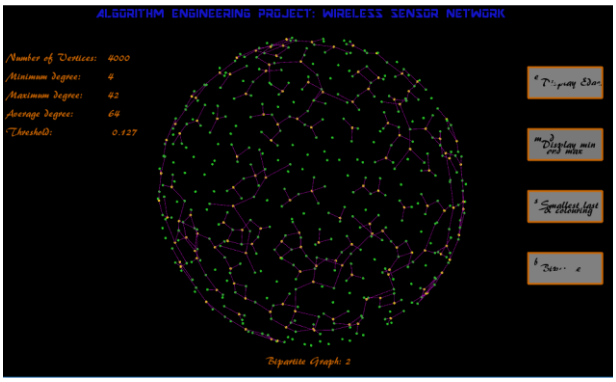
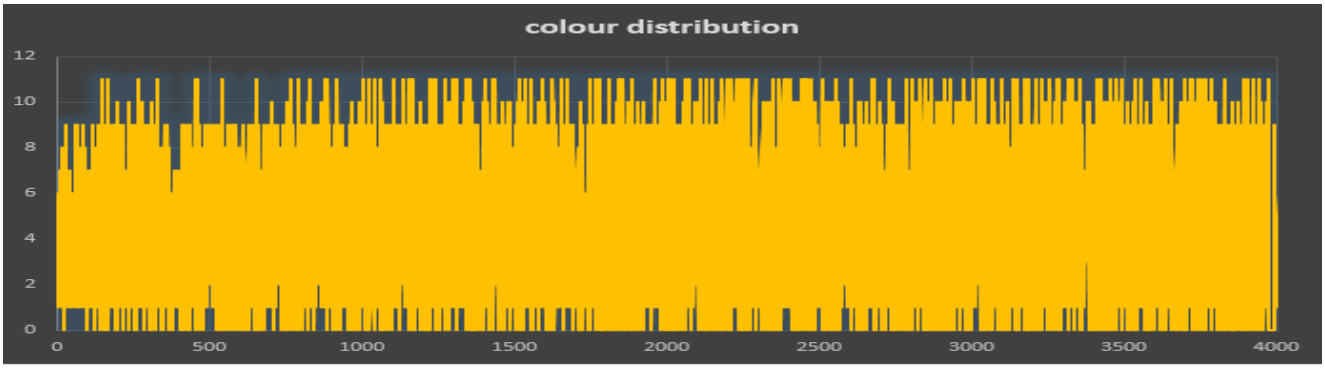
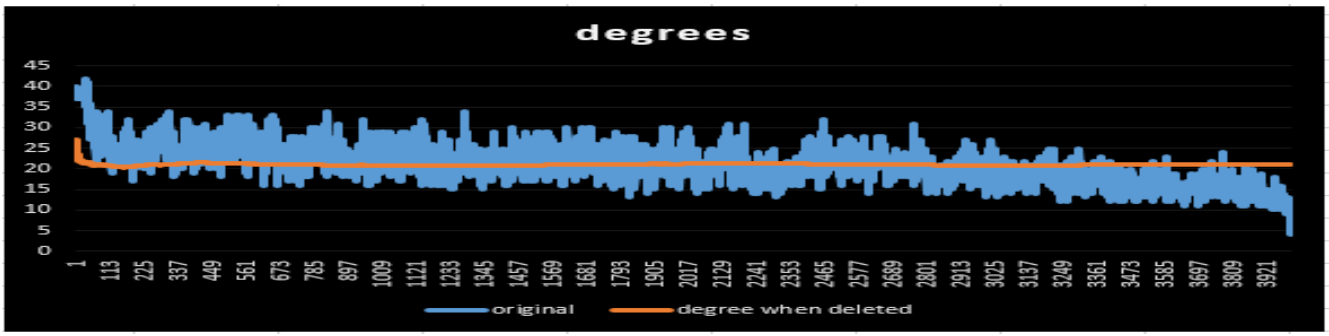



*Figure 53: backbone 2*

## Summary:




*Graph 12: colour distribution*




*Graph 13: degree vs degree when deleted*

## 9. Benchmark9(SPHERE):

Number of vertices = 16000;

Average degree = 128

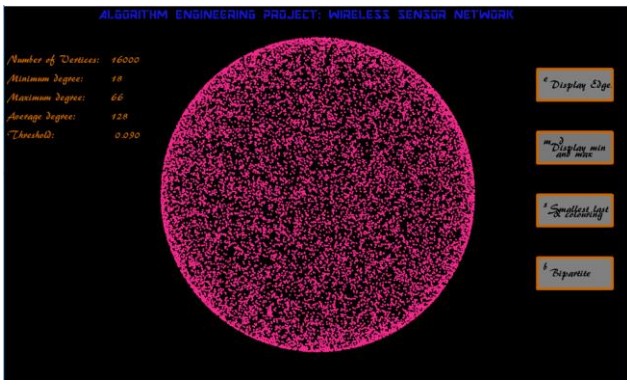



*Figure 54:uniquely random points*

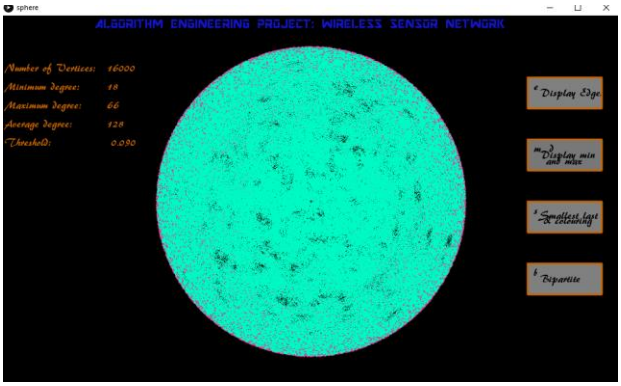



*Figure 55: edges*

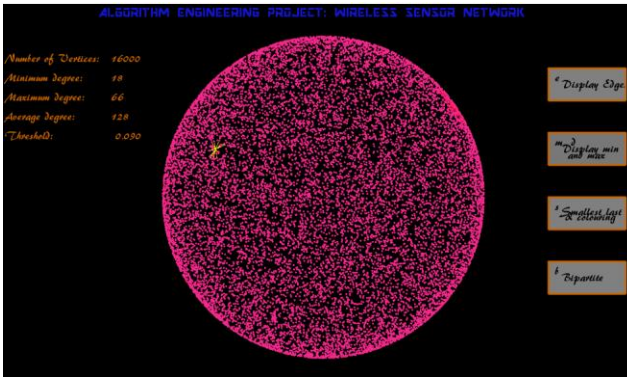



*Figure 56: min and max degree vertices*

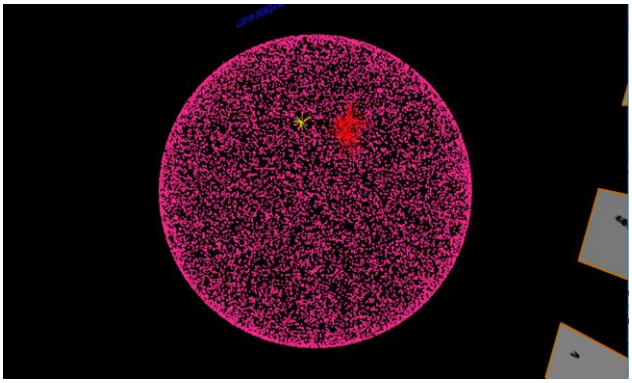


*Figure 57: min and max degree vertices*

Since the minimum and maximum degree vertex was not visible from this angle, I had to rotate it in order to view it better.

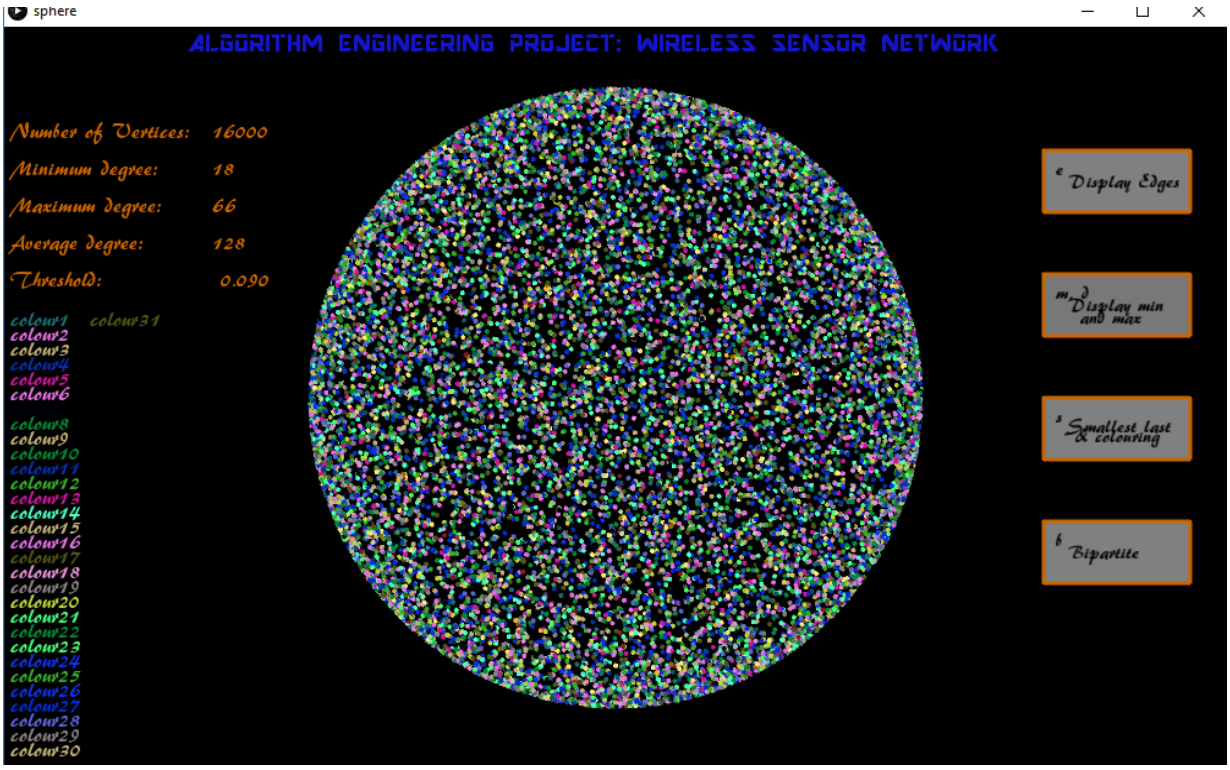



*Figure 58: smallest-last order colouring*

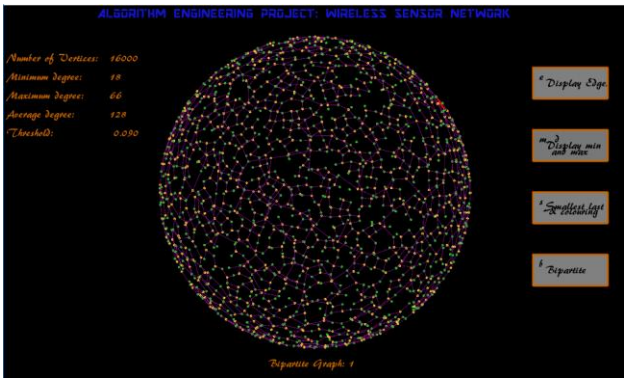



*Figure 59: backbone 1*

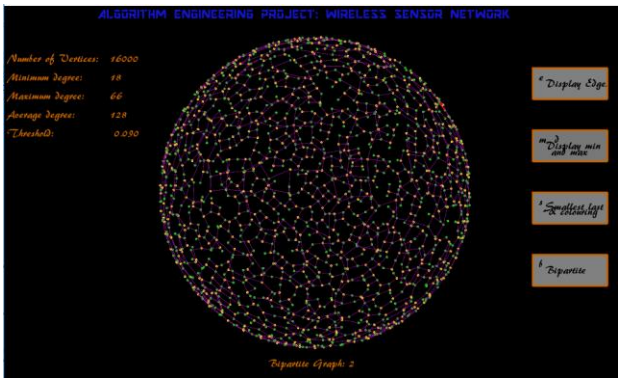
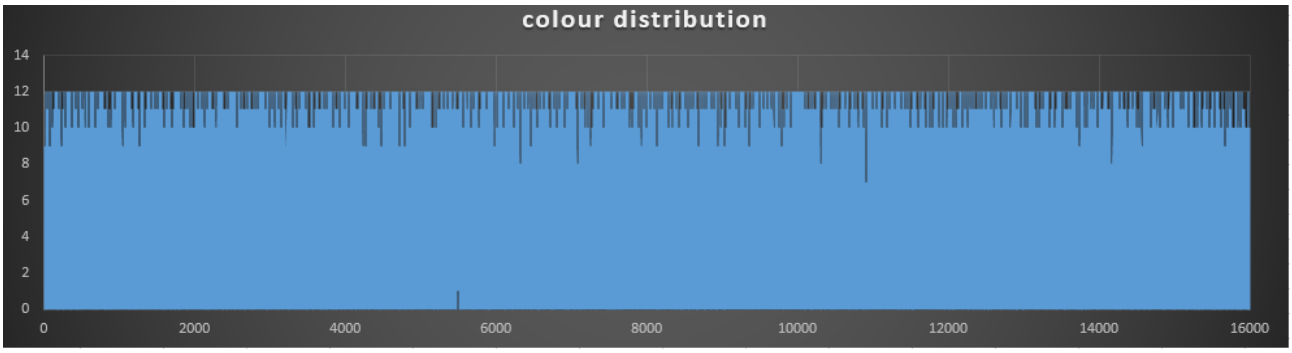
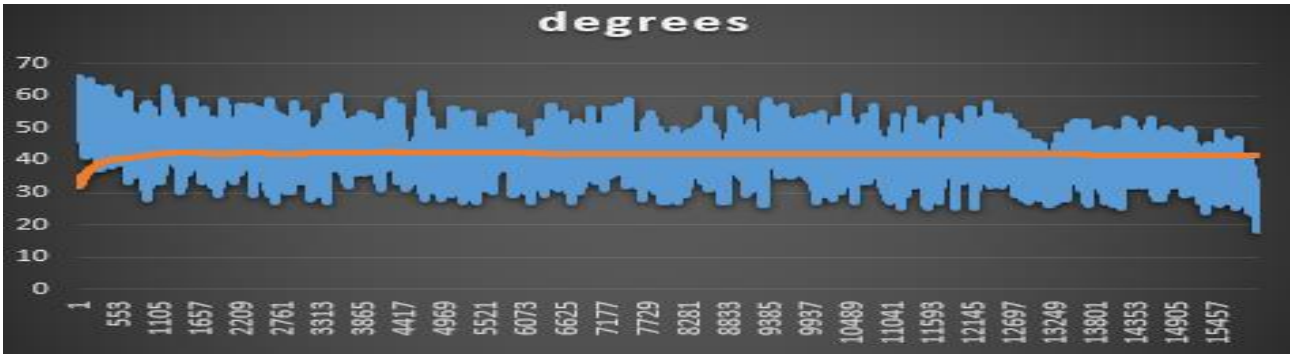



*Figure 60: backbone 2*

Summary:



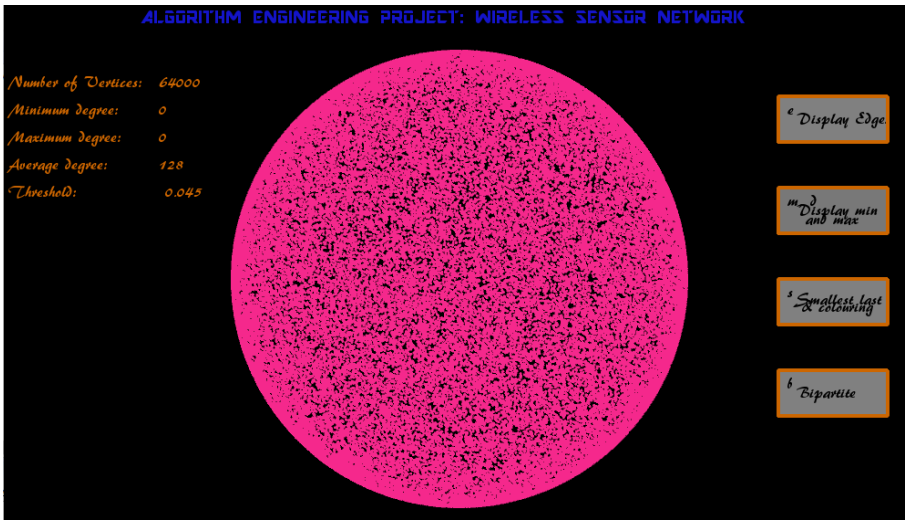

*Graph 14: colour distribution*

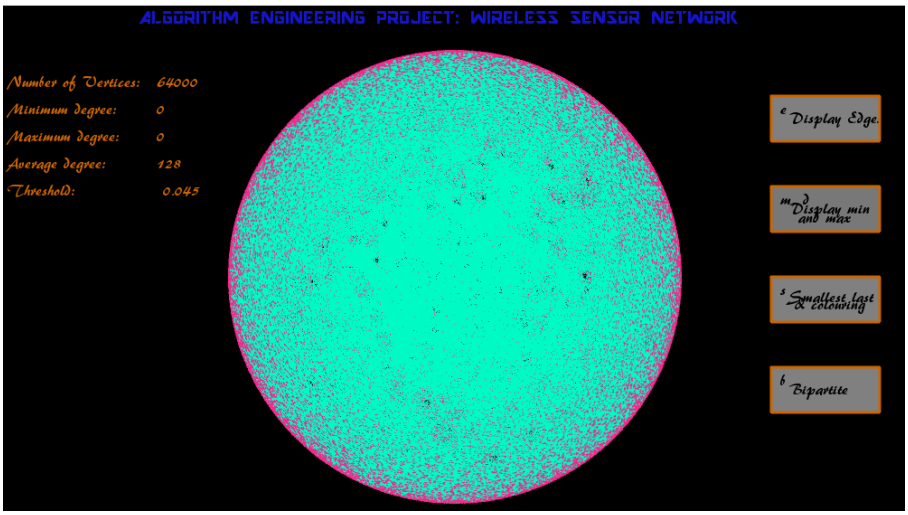


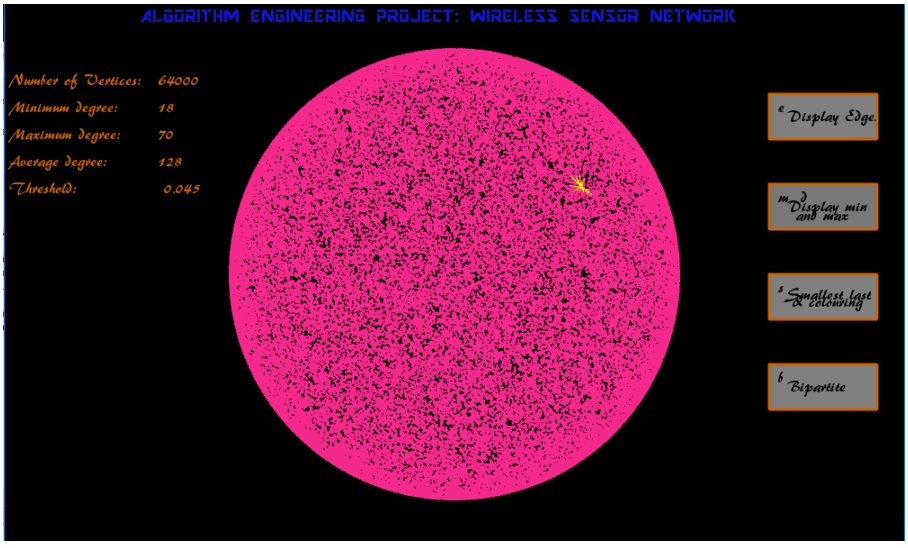

*Graph 15: degree distribution*

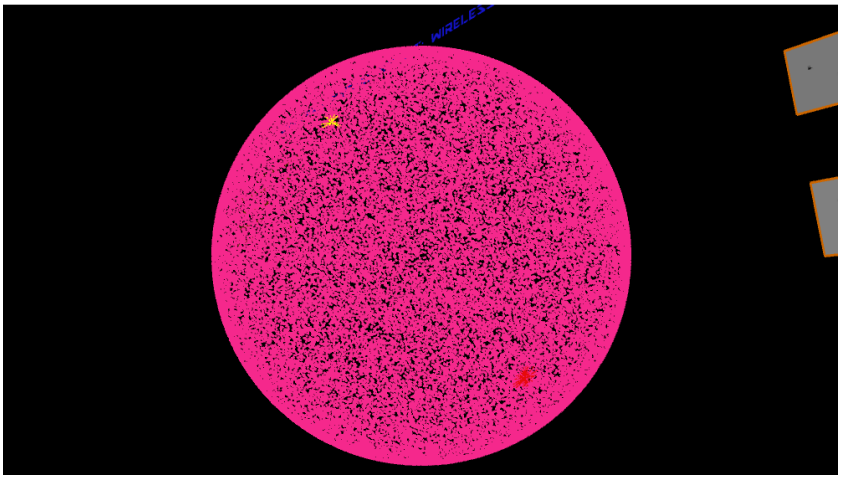
## 10. Benchmark10 (SPHERE):

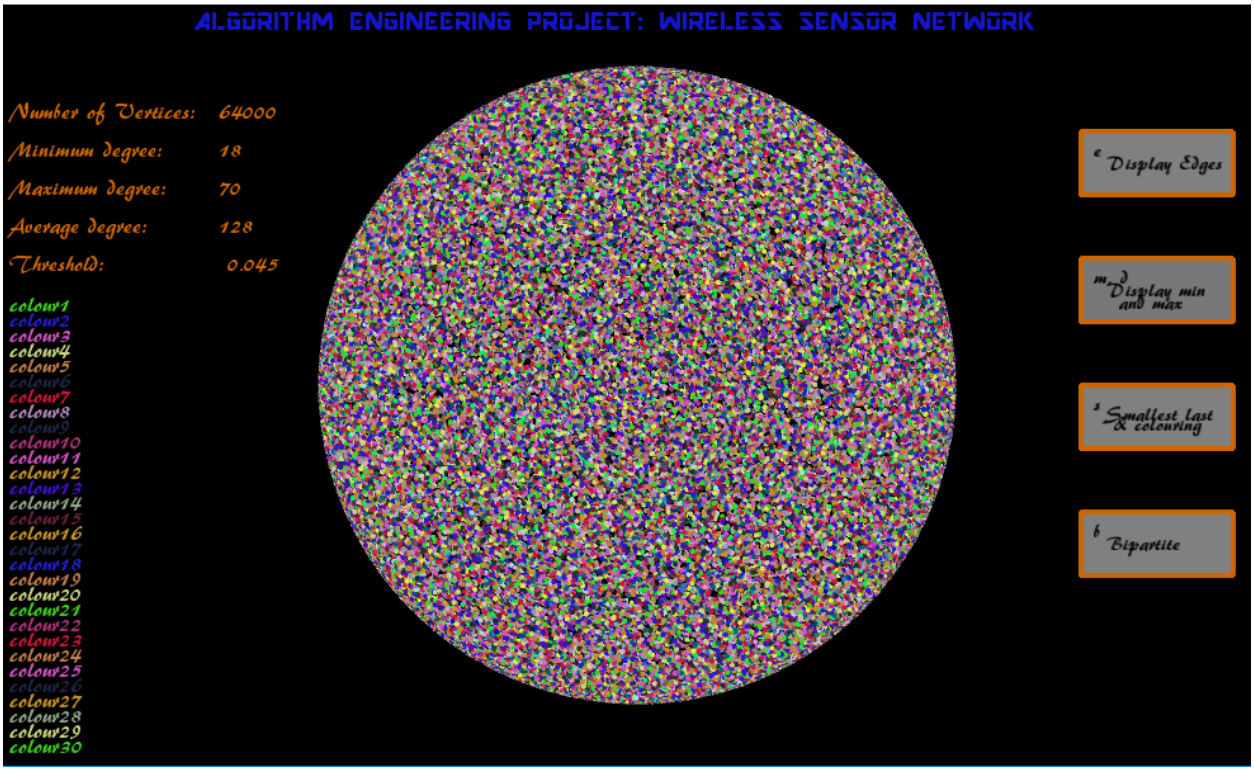
N = 64000; Average degree = 128

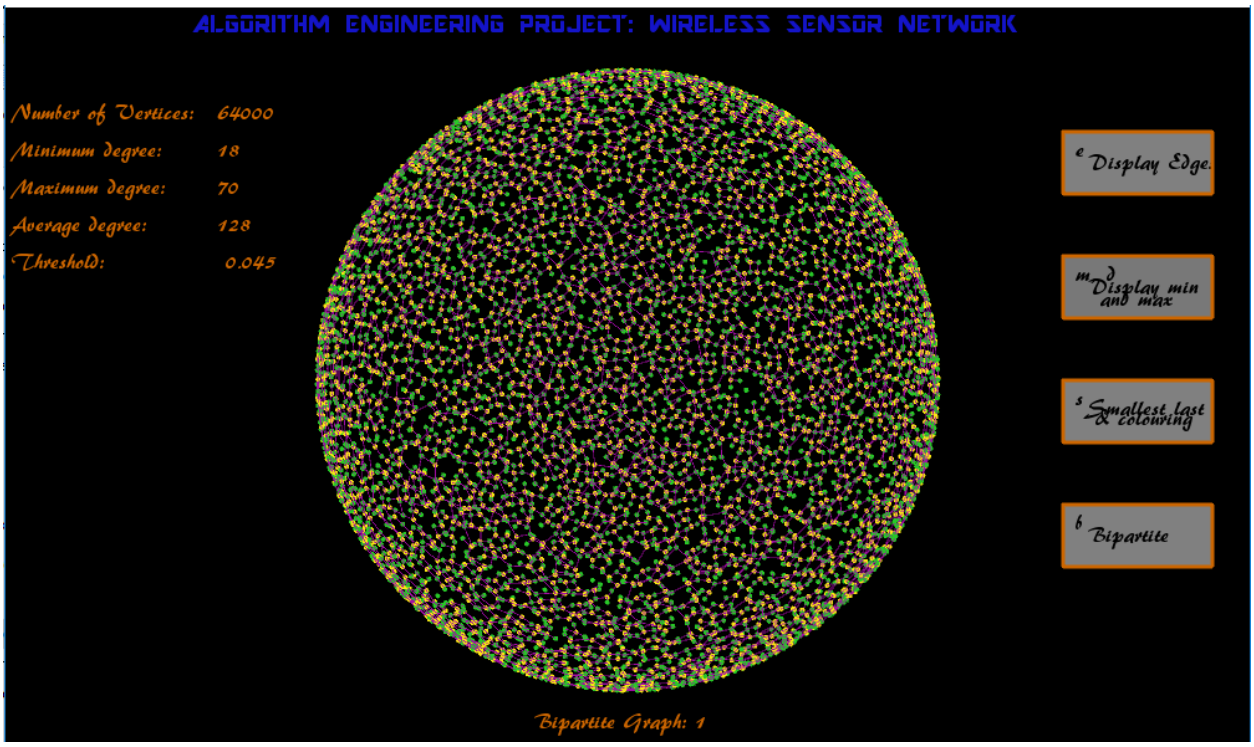


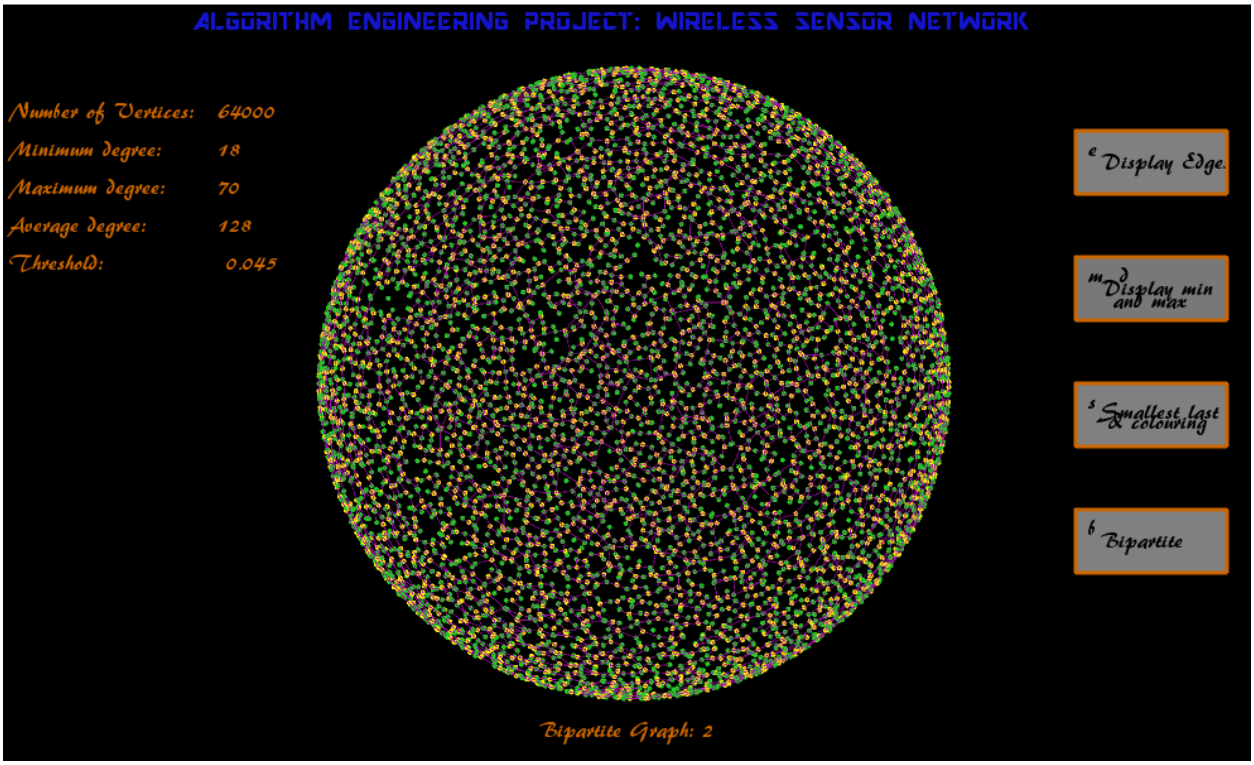



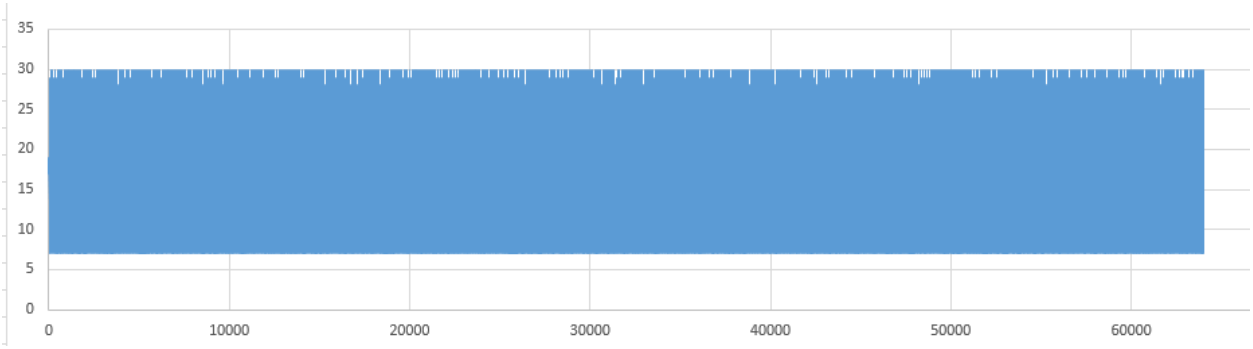



ALGORITHM ENGINEERING PROJECT: WIRELESS SENSOR NETWORK
Number of Vertices: 64000
Minimum degree: 18
Maximum degree: 70
Average degree: 128
Threshold: 0.045
colour1
colour2
colour3
colour4
colour5
colour6
colour7
colour8
colour9
colour10
colour11
colour12
colour13
colour14
colour15
colour16
colour17
colour18
colour19
colour20
colour21
colour22
colour23
colour24
colour25
colour26
colour27
colour28
colour29
colour30
c Display Edges
m d Display min and max
s Smallest last & colouring
b Bipartite


ALGORITHM ENGINEERING PROJECT: WIRELESS SENSOR NETWORK
Number of Vertices: 64000
Minimum degree: 18
Maximum degree: 70
Average degree: 128
Threshold: 0.045
e Display Edge
m Display min and max
s Smallest last & colouring
b Bipartite
Bipartite Graph: 1


ALGORITHM ENGINEERING PROJECT: WIRELESS SENSOR NETWORK
Number of Vertices: 64000
Minimum degree: 18
Maximum degree: 70
Average degree: 128
Threshold: 0.045
e Display Edge
m Display min and max
s Smallest last & colouring
b Bipartite
Bipartite Graph: 2

Summary:




*Graph 16: color distribution*

# IV. Reference

[1] https://en.wikipedia.org/

[2] https://processing.org/

[3] D. W. Matula and L. Beck, “Smallest-last ordering and clustering and graph coloring algorithms,” J. of the ACM, vol. 30, no. 3, pp. 417–427, 1983

[4] http://lyle.smu.edu/~zizhenc/file/Wireless%20Sensor%20Network

[5] [6] D. Mahjoub, D. W. Matula, “Constructing efficient rotating backbones in wireless sensor networks using graph coloring”

[6] D. W. Matula, “A min-max theorem for graphs with application to graph coloring,” SIAM Review, nol. 10, pp. 481–482, 1968

[7] D. Mahjoub and D. W. Matula. "Employing (1-epsilon) Dominating Set Partitions as Backbones in Wireless Sensor Networks."

[8] R. Asgarnezhad, J. Akbari Torkestani. "A New Classification of Backbone Formation Algorithms for Wireless Sensor Networks."