Wireless Sensor Network

ROHIT KURHEKAR
47152070

CSE 7350
Dr. Lee D. McFearin

Table of Contents

1. Executive Summary

1.1 Introduction and Summary

Wireless Sensors can communicate with each other using a finite set of frequencies which can be re-used only if the sensors that use them are at a minimum distance from each other. Wireless sensor networks are spatially distributed sensors to monitor physical or environmental conditions such as sound, temperature, pressure, etc. Their role is to pass the data through the network to a main location. In general network simulators such as OPNET and NetSim are used to simulate a wireless sensor network[1]. In this project, a wireless sensor network is modelled using a random geometric Graph (RGG). A random geometric graph is an undirected graph drawn on a bounded region with vertices randomly, uniformly and independently placed and connected if the distance of any two of them is less than a threshold[2]. The project implements Smallest Vertex last ordering to determine coloring, terminal clique and bipartite sub graph with two independent sets.

The functioning to get the required output can be classified in 3 main parts:

1] **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.

2] **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".

3] **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

The programming environment consisted of a 17- inch Asus ROG. The following information outlines the hardware and software architecture.

Hardware Specifications:

- Operating System: Microsoft Windows 10
- Graphics: Nvidia GTX 1060 (6 GB)
- System Type: x64-based PC
- Processor: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60 GHz, 2.59 GHz
- Installed Physical Memory (RAM): 16.0 GB

Software Specifications:

- Language: Java, Processing
- IDE: Processing, Eclipse
- Text editor: Atom
- Graph Generation: MS Excel ,Originlab

Firstly I chose Eclipse as my IDE as I was familiar with java and was going to use PApplet Java class which implements most of the Processing language's features. But while implementation I realized there were lots of errors due to which I decided to switch to processing completely and ported all of my code to Processing.

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.

1.3 Summary Table

SUMMARY TABLE OF BENCHMARK #1 TO #7							
BM	#1	#2	#3	#4	#5	#6	#7
N	1000	4000	16000	64000	64000	4000	4000
GAD	32	64	64	64	128	64	128
Dist.	Square	Square	Square	Square	Square	Disk	Disk
R	0.1	0.07	0.03	0.018	0.025	0.12	0.179
E	14222	11520	35300	210012	389568	110598	241568
MinD	6	23	18	14	31	20	51
MaxD	66	86	77	95	172	77	155
MDD	20	37	35	42	72	35	68
Colors	19	38	35	40	63	31	62
C1	74	160	640	2498	1360	195	89
C2	70	157	639	2310	1355	191	86

BM: Bechmark **N:** Number of vertices/points **GAD:** Given Average degree
Dist.: Distribution **E:** Number of Edges **MinD:** Minimum Degree
MaxD: Maximum degree **MDD:** Maximum degree to delete **C1:** Max points with color C1
C2: Mac points with color C2 after C1

We have given inputs as Nodes, average degree and topology. However, for generation of RGG we require a connectivity threshold R which is calculated as :

$R = \sqrt{(A/\pi N)}$ if Square **$R = \sqrt{(A/N)}$ if Disk** **$R=\sqrt{(4A/N)}$ if Sphere**
where, A = given average degree; N = number of node.

Backbone Results ::

SUMMARY OF BACKBONE ANALYSIS OF #1 TO #7							
BM	#1	#2	#3	#4	#5	#6	#7
N	1000	4000	16000	64000	64000	4000	4000
GAD	32	64	64	64	128	64	128
Dist.	Square	Square	Square	Square	Square	Disk	Disk
Backbones	6	6	6	6	6	6	6
EIBack	187	413	15090	62556	78812	3432	4941
NIBack	160	328	1385	5182	2698	442	172
FIBack	29	87	13707	57376	76116	2992	4771

Domination	99.15%	99.63	97.99%	99.72%	99.2%	99.2%	99.70%

BM: Bechmark **N:** Number of vertices/points **GAD:** Given Average degree
Dist.: Distribution **FIBack:** Faces in Backbone **EIBack:** Edges in Backbone
VIBack: Vertices in Backbone

1.4 Referrences:

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

[4] Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein, "Introduction to Algorithms", 3rd ed, pp.603-612

[5] Lynda.com, "Processing: Interactive Data Visualization", www.lynda.com.

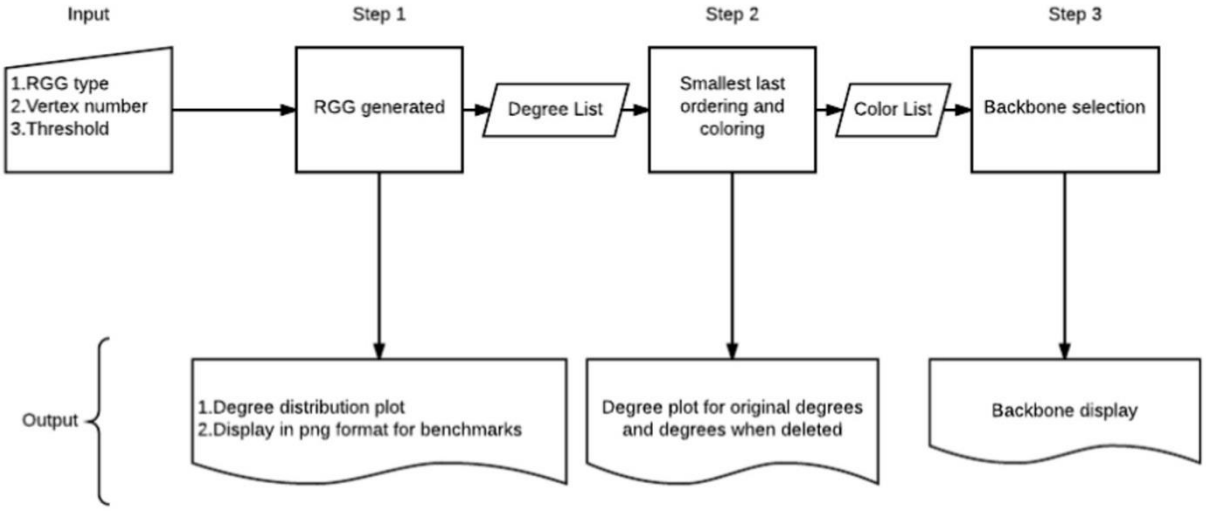
[6] Zizhen Chen, Wireless Sensor Network Project

[7] Tao Zhong, Wireless Sensor Network Project

2.Reduction to practice

2.1 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:



2.2 Data Structure Design

The basic component of RGG is the vertex or nodes or point ; 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	
NAME	INFORMATION
Float x,y	Coordinates of x, y
Int index	Index id for the point
Int component	Component id
ArrayList<Node> adjPoint	Record Adjacent Vertex
Color c	Color id

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 :

I tried implementing DegreeList as an ArrayList of LinkedLists, but was unable to use it properly, so instead I used Hash Map Its index represents how many degrees the nodes have. The Hash Map stores the arraylist of points with the degree being the Index number ,due to which some vertices will be null as there might not be a point for that specific degree, so we store null for that. While deleting (smallest ordering) this causes a **null exception** and error occurs in the program but this can be avoided by running the code again.

2.3 Algorithm Description and Analysis

Smallest last vertex ordering:

Smallest-Last Vertex Ordering algorithm is a greedy algorithm which takes the adjacency list mentioned before as input and generate another list starts from the vertex with largest degree and ends with the vertex with smallest degree. 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. When the algorithm finished, we'll get an array with smallest-last ordering and it's also an adjacency list but with less adjacent vertices for each node.

Graph coloring:

Once we get the smallest last ordering, we need to find the colors for each vertex, or node. To each node, we assign a color randomly to it but it can't be same with any color of vertex that adjacent to this node according to the adjacency list. An example of this process will be that we take ,a variable c=1 and if c==1 we

paint the node with a color and store the color to the list which makes it the first element of the list ,so now if the node j has zero degree color it with the first in the list and if its nonzero ,but it has a adjacent vertex covering all the colors in the list ,give the vertex a new color ,continue this process till c<= no of points. 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.

2.4 Algorithm Engineering:

In this project I am using exhaustive method to traverse. So 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?
For a Circle :

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

$ad = N\pi r^2$

For a sphere:

$\pi r2/ad = 4\pi r2/N$

$ad = Nr^2/4$

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.

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

I managed to sort and use **Cell Method** to improve the linear time, I will be adding the code for the same in the index ,but I wasn't able to implement that in the rest of the structure so I am sticking with Exhaustive method.

For smallest last ordering:

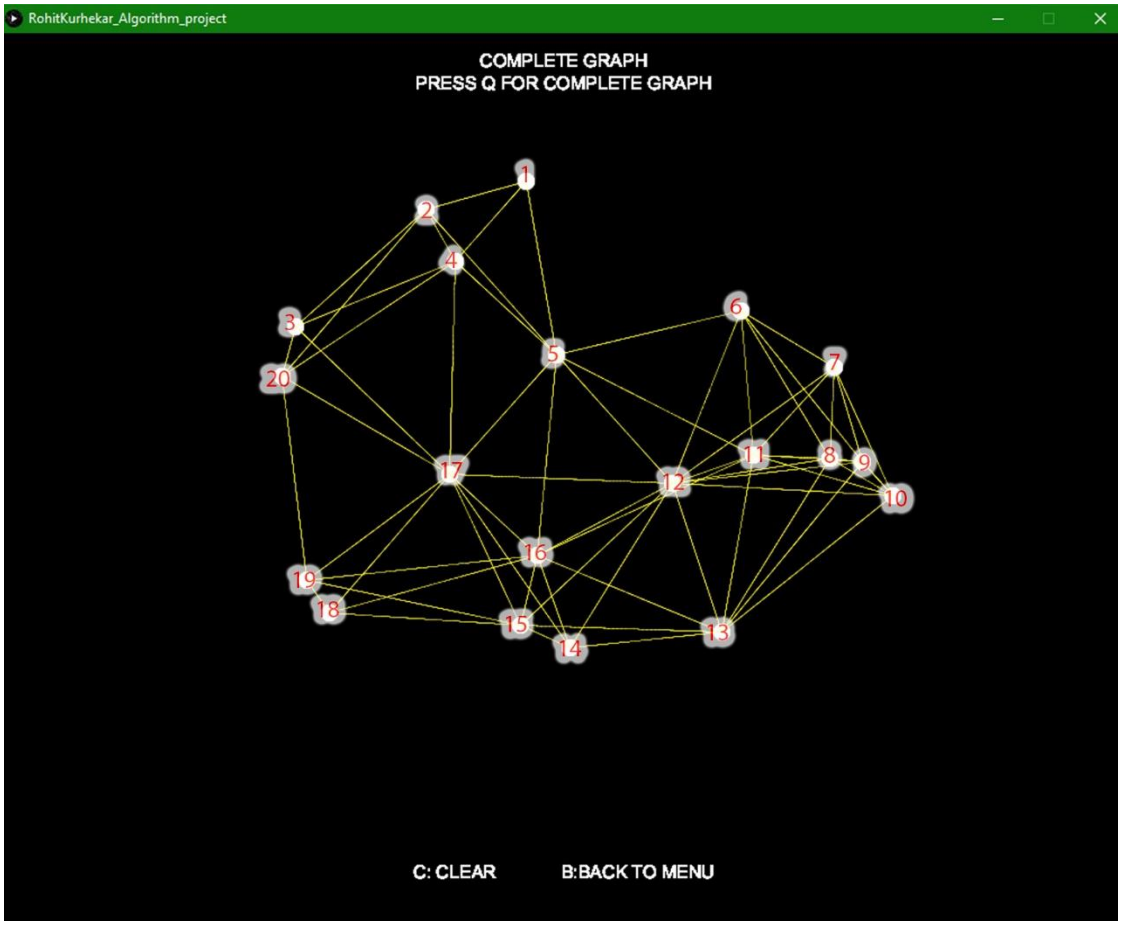
In java, the ArrayList takes O(N) space, and the Array takes O(N) space as well. So the space complexity for smallest last ordering is O(N). In implementation, we find one node from the DegreeList with least index, that gives time O(1); then we walk through the neighbors of this node and remove this 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:

After we get the smallest last ordering of a adjacency list with smallest last ordering and the degree or each vertex is equal to the degree in sub graph when deleting them from previous algorithm. My method visits each neighbor of a node, that gives time O(Nr2), for n nodes O(N2r2). Then assigning the color gives time O(1). Which is also O(N+E).

2.5 Verification Walkthrough:

The first step is to generate 20 vertices within a unit square and connect any two vertices with an edge if they are within the distance of 0.4(threshold) from each other.



GRAPHICAL REPRESENTATION OF THE POINTS

Vertex and its adjacent Nodes

Vertex	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Neighbor	2 4 5	1 3 4 5 20	2 4 20	1 2 3 5 17	1 2 4 6 11 12 16 17	5 7 8 9 11 12	6 8 9 10 11 12	6 7 9 11 13	6 7 8 10 13	7 8 9 11 12 13	5 6 7 8 10 12 13	5 6 7 8 9 10 11 13 14 15 16 17	8 9 10 11 12 14 15 16	12 13 15 16 17	12 13 14 16 17 18 19	5 11 12 13 14 15 16 17 18 19	3 4 5 12 14 15 16 18 19 20	15 16 17 19	15 16 17 18 20	3 2 4 17 19

On basis of the adjacency list above, the degree list will be created as follows:

Degree	0	1	2	3	4	5	6	7	8	9	10	11	12
Vertex				1,3	18	2,4,8,9,14,19,20	6,7,10	15	5,11,13	16	17		12

- **Smallest Last Ordering Algorithm:**

A copy of the Adjacency list is created. The vertices to be removed is 1 with degree 3.

Hence, 1 is added to smallest vertex list and degree of its neighbor's (2,4,5) is decremented by 1.

So now the Smallest vertex order array would contain:

Vertex							1
Original Degree							3
Deleted Degree							3

Now, Node 1 is deleted and the degree list is updated as follows with new degree of Nodes 2,4 and 5:

Vertex	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Neighbor	2 4 5	~~1~~ 3 4 5 20	2 4 20	~~1~~ 2 3 5 17	~~1~~ 2 4 6 11 12 16 17	5 7 8 9 11 12	6 8 9 10 11 12	6 7 9 11 13	6 7 8 10 13	7 8 9 11 12 13	5 6 7 8 10 12 13	5 6 7 8 9 10 11 12 13 14 15 16 17	8 9 10 11 12 14 15 16	12 13 15 16 17	12 13 14 16 17 18 19	5 11 12 13 14 15 16 17 18 19	3 4 5 12 14 15 16 18 19 20	15 16 17 19	15 16 17 18 20	3 2 4 17 19

Degree	0	1	2	3	4	5	6	7	8	9	10	11	12
Vertex				~~1~~,3	18	2,4,8,9,14,19,20	6,7,10	15	5,11,13	16	17		12

Next vertex to be deleted will be node 3 with degree 3

Vertex	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Neighbor	2 4 5	~~1~~ ~~3~~ 4 5 20	2 4 20	~~1~~ 2 ~~3~~ 5 17	~~1~~ 2 4 6 11 12 16	5 7 8 9 11 12	6 8 9 10 11 12	6 7 9 11 13	6 7 8 10 13	7 8 9 11 12 13	5 6 7 8 10 12 13	5 6 7 8 9 10 11 12 13 14 15 16 17	8 9 10 11 12 14 15 16	12 13 15 16 17	12 13 14 16 17 18 19	5 11 12 13 14 15 16 17 18 19	3 4 5 12 14 15 16 18 19 20	15 16 17 19	15 16 17 18 20	~~3~~ 2 4 17 19

					17						16	13 14 15 16 17	16			18 19	18 19 20			

Degree	0	1	2	3	4	5	6	7	8	9	10	11	12
Vertex				~~1~~,~~3~~	18	2,4,8,9,14,19,20	6,7,10	15	5,11,13	16	17		12

Vertex						3	1
Original Degree						3	3
Deleted Degree						3	3

Next vertex to be deleted will be node 18 with degree 4

Vertex	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Neighbor	2 ~~4~~ 5	~~1~~ ~~3~~ ~~4~~ 5 20	2 4 20	~~1~~ 2 ~~3~~ 5 17	~~1~~ 2 4 6 11 12 16 17	5 7 8 9 11 12	6 8 9 10 11 12	6 7 9 11 13	6 7 8 10 13	7 8 9 11 12 13	5 6 7 8 10 12 13	5 6 7 8 9 10 11 12 13 14 15 16 17	8 9 10 11 12 14 15 16	12 13 15 16 17	12 13 14 16 17 ~~18~~ 19	5 11 12 13 14 15 17 ~~18~~ 19	3 ~~4~~ 5 12 ~~14~~ 15 16 ~~18~~ 19 20	15 16 17 19	15 16 17 ~~18~~	~~3~~ 2 4 17 19

Degree	0	1	2	3	4	5	6	7	8	9	10	11	12
Vertex				~~1~~,~~3~~	~~18~~	2,4,8,9,14,19,20	6,7,10	15	5,11,13	16	17		12

Vertex					18	3	1
Original Degree					4	3	3
Deleted Degree					4	3	3

Next vertex to be deleted will be node 2 with degree 5

Vertex	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Neighbor	~~2~~ 4 5	~~1~~ ~~3~~ 4 5 20	~~2~~ 4 20	~~1~~ ~~2~~ ~~3~~ 5 17	~~1~~ ~~2~~ 4 6 11 12 16 17	5 7 8 9 11 12	6 8 9 10 11 12	6 7 9 11 13	6 7 8 10 13	7 8 9 11 12 13	5 6 7 8 10 12 13	5 6 7 8 9 10 11 12 13	8 9 10 11 12 14 15 16	12 13 15 16 17	12 13 14 16 17 ~~18~~ 19	5 11 12 13 14 15 17 ~~18~~ 19	3 ~~4~~ 5 12 14 15 16 ~~18~~ 19 20	15 16 17 19	15 16 17 ~~18~~ 20	~~3~~ ~~2~~ 4 17 19

Degree	0	1	2	3	4	5	6	7	8	9	10	11	12
Vertex				~~1~~,~~3~~	~~18~~	~~2~~,4,8,9,14,19,20	6,7,10	15	5,11,13	16	17		12

.

Vertex				2	18	3	1
Original Degree				5	4	3	3
Deleted Degree				3	4	3	3

Now, next node to be removed will be 4 and so on. This process will continue till all the nodes are deleted.

Thus we get final list as:

1,3,18,2,4,8,9,14,19,20,6,7,10,15,5,11,13,16,17,12

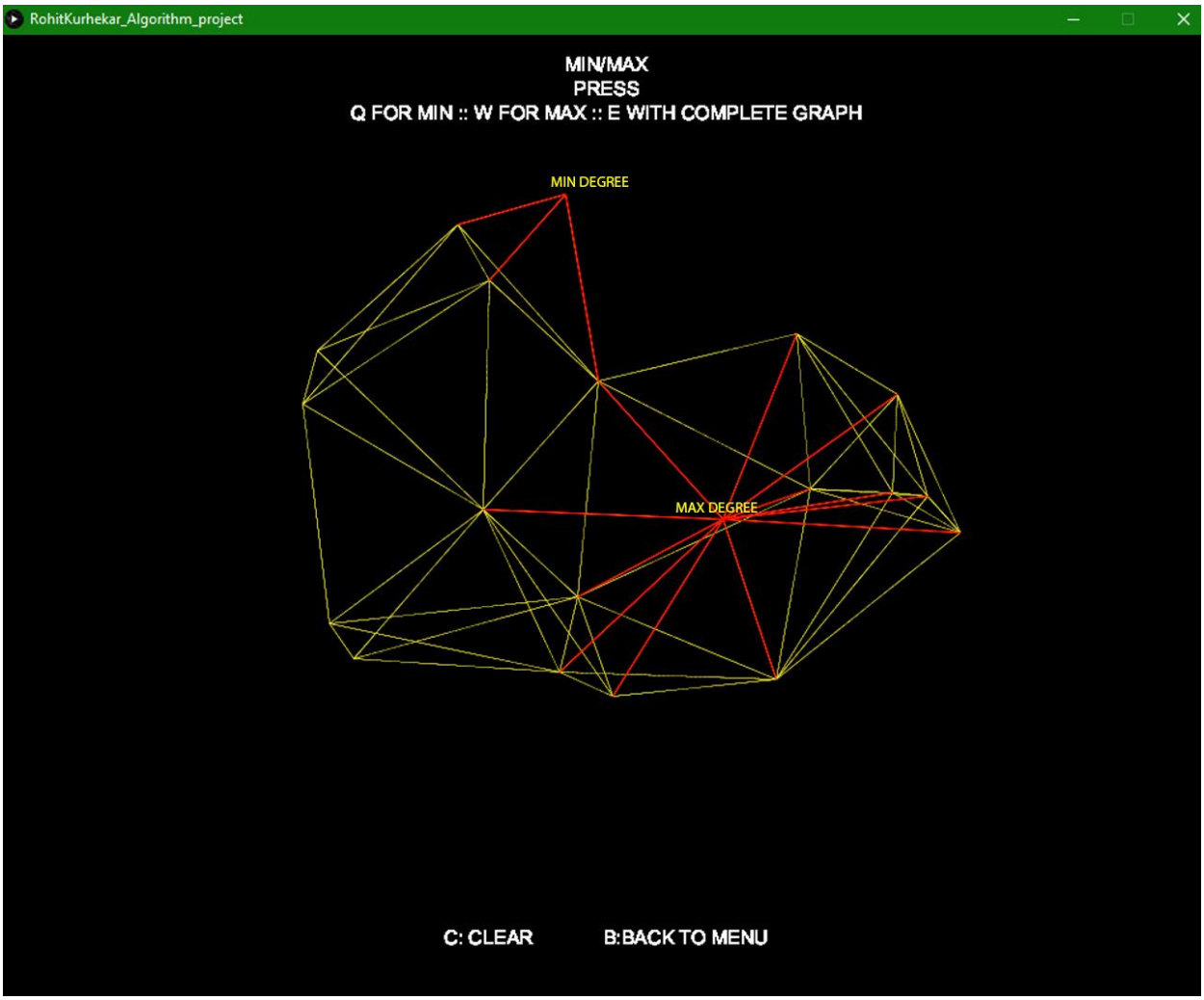
However, as the list is Smallest vertex last, it will be reversed for coloring and will be considered as:

12,17,16,13,11,5,15,10,7,6,20,19,14,9,8,4,2,18,3,1

Smallest last order:

So the table to would end up like this;

Vertex	12	17	16	13	11	5	15	10	7	6	20	19	14	9	8	4	2	18	3	1
Original Degree	12	10	9	8	8	8	7	6	6	6	5	5	5	5	5	5	5	4	3	3
Deleted Degree	0	1	2	2	3	4	4	3	3	4	1	4	5	4	5	2	3	4	3	3



MIN MAX DEGREE

Graph Coloring:

We start with the last vertex that was removed i.e. first vertex of smallest last ordering which is node 12. The first available color is Red (0)

Vertex	12	17	16	13	11	5	15	10	7	6	20	19	14	9	8	4	2	18	3	1
Color	0																			

Vertex 17 is next to be colored and its neighbor's 3,4,5,12,14,15,16,18,19,20 need to be checked so 17 can be assigned next available color. As 12 is its neighbor which has color 0 we cannot assign that to 17.
Therefore, 17 is assigned next color which is Blue (1)

Vertex	12	17	16	13	11	5	15	10	7	6	20	19	14	9	8	4	2	18	3	1
Color	0	1																		

Next vertex to be colored is vertex 16 and its neighbors 5,11,12,13,14,15,17,18,19 are to be checked. As 12 and 17 are colored red and blue respectively 16 cannot be assigned these colors. Hence, 16 is assigned next available color which is green (2)

Vertex	12	17	16	13	11	5	15	10	7	6	20	19	14	9	8	4	2	18	3	1
Color	0	1	2																	

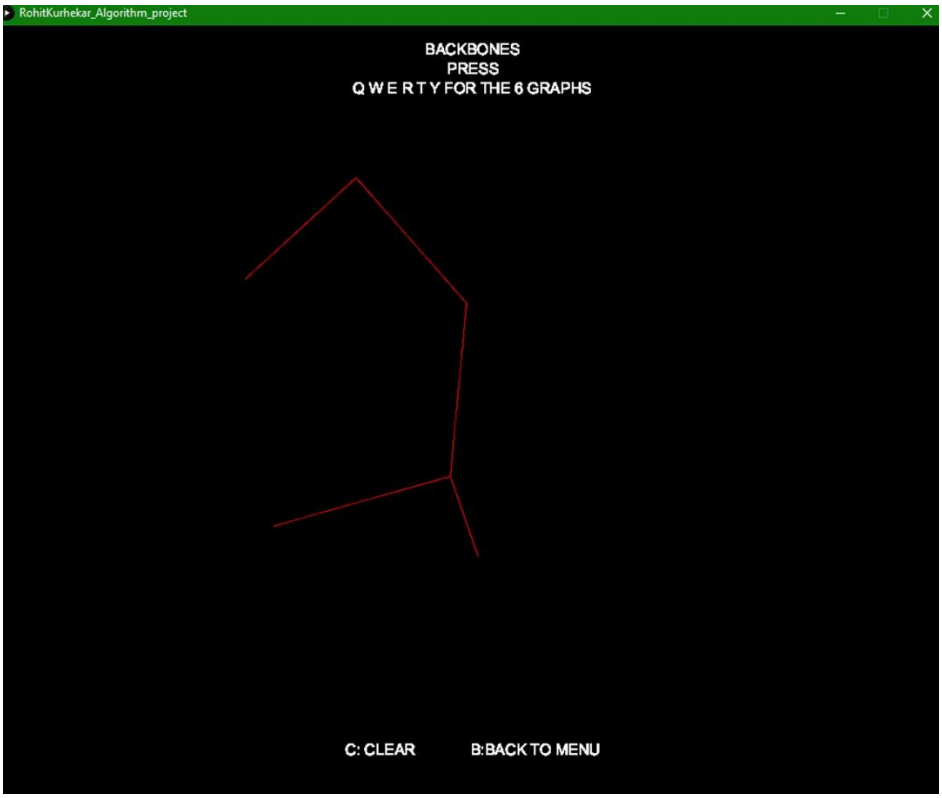
Repeating this process for all the vertices we get the graph coloring as follows:

Vertex	12	17	16	13	11	5	15	10	7	6	20	19	14	9	8	4	2	18	3	1
Color	0	1	2	1	3	4	3	2	1	2	0	4	4	0	4	0	1	0	2	2

Where color 0(Red), 1(Blue),2 (Green),3 (Yellow), 4 (Cyan)

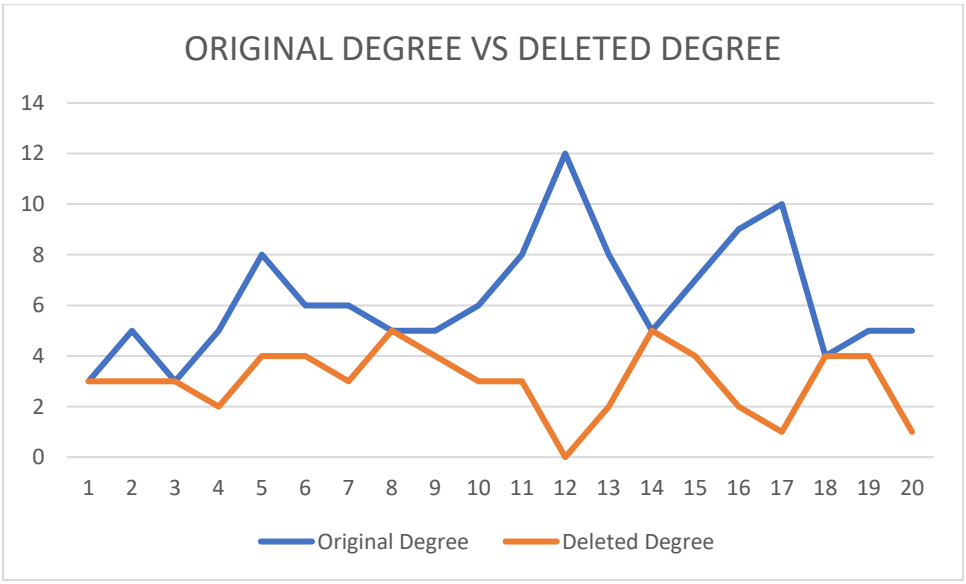
The 4 most frequently used colors are: Red, Blue, Green, Cyan and on basis of 4C2 combinations we get top 2 combinations as Red-Green and Red-Blue/Red-Cyan. These are used to generate the bipartite sub graph. From the sub graph we find the largest component and that will be our backbone.

BACKBONE 1



BACKBONE 2

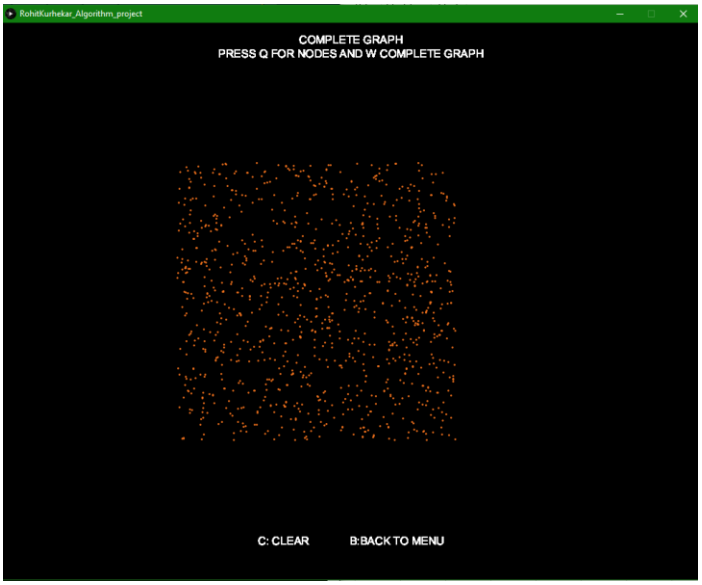
Here is the plot for the degree when deleted values and the original degree values for the vertices ordered by the Smallest Last order.



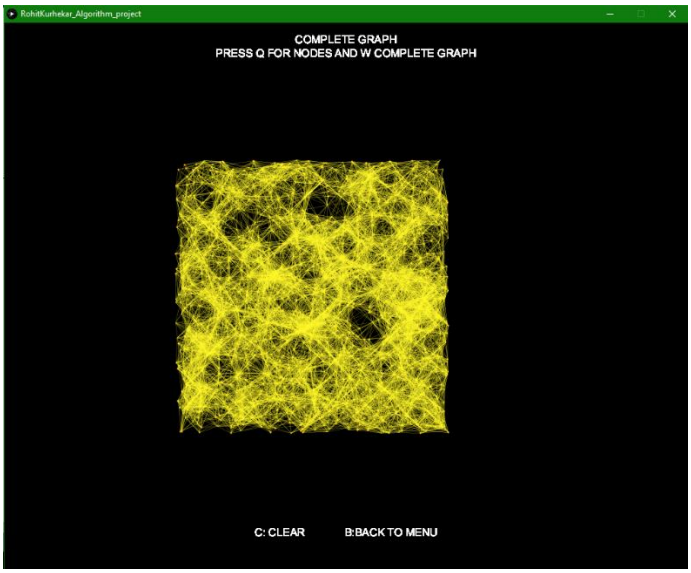
3. Result Summary

Color notations- Green: Maximum degree region, Blue: Minimum degree region.

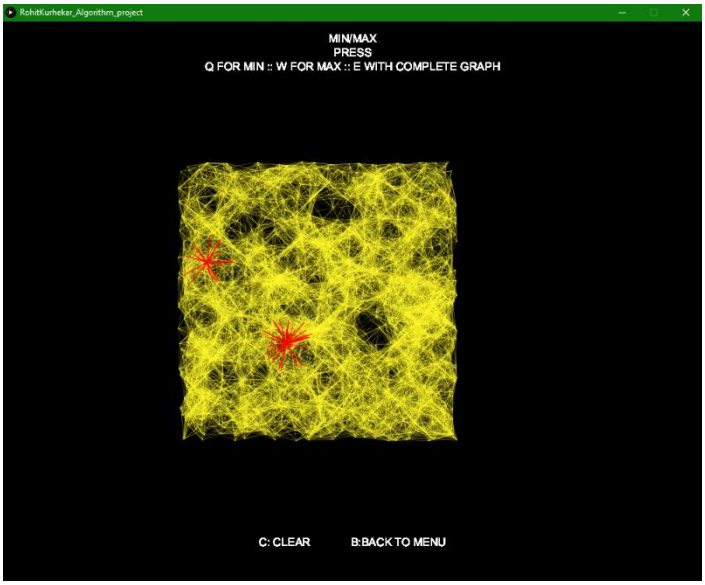
Benchmark 1 (1000 vertices, 30 average degree, Square):



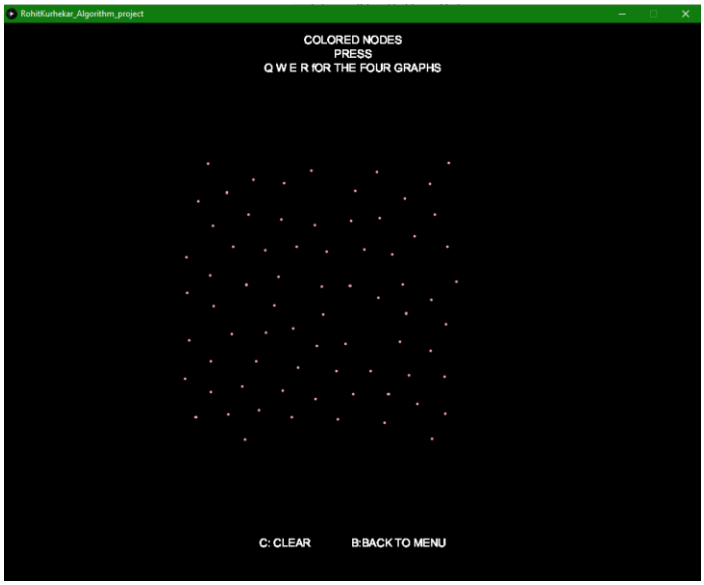
NODES



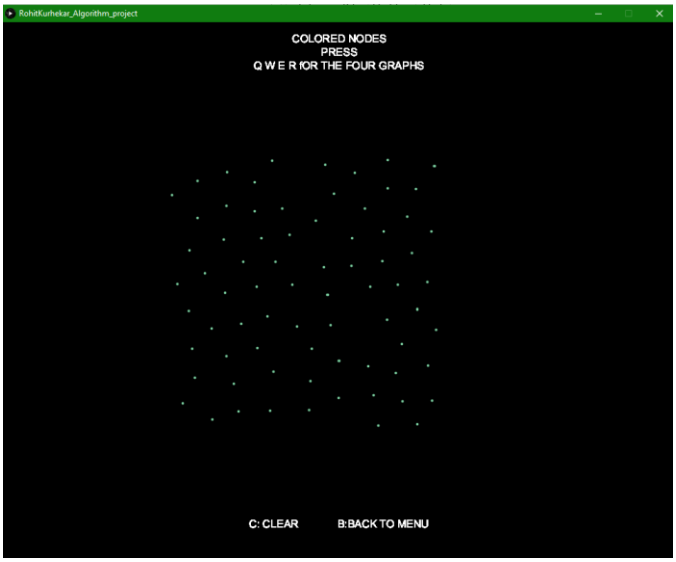
COMPLETE GRAPH



MIN/MAX DEGREE

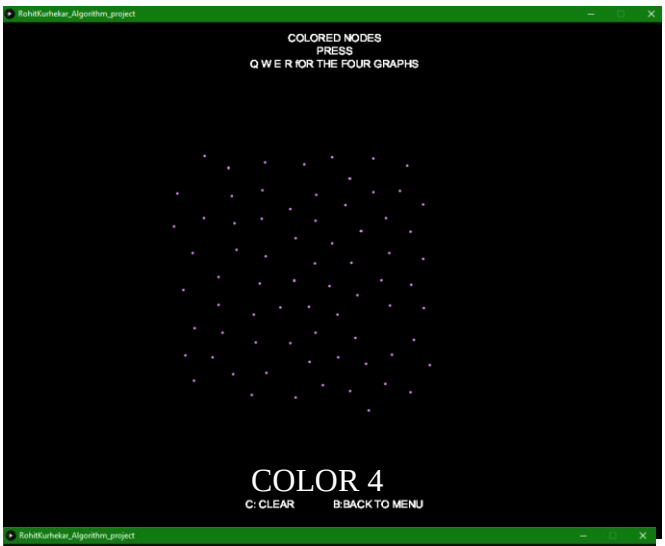


COLOR 1



COLOR 2

COLOR 3



COLOR 4

Bipartite 1-2

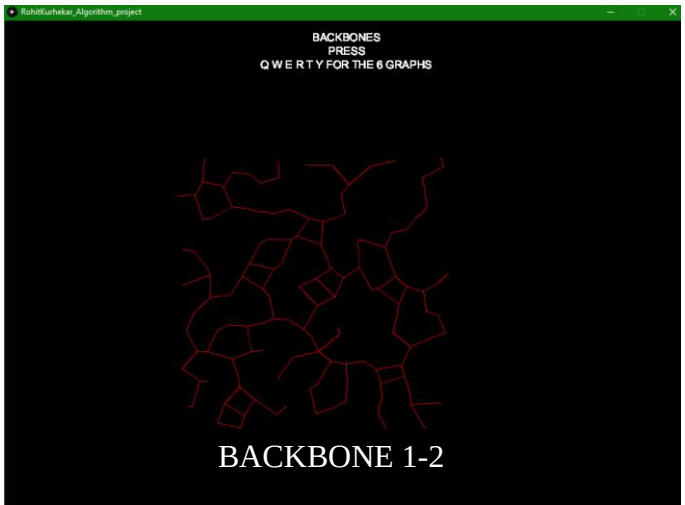
BIPARTITE 1-3

BIPARTITE 1-4

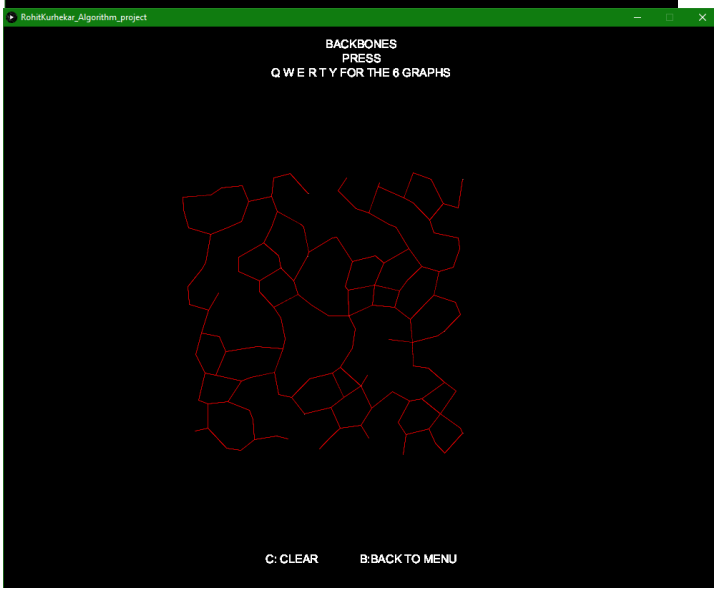
BIPARTITE 2-3

BIPARTITE 2-3

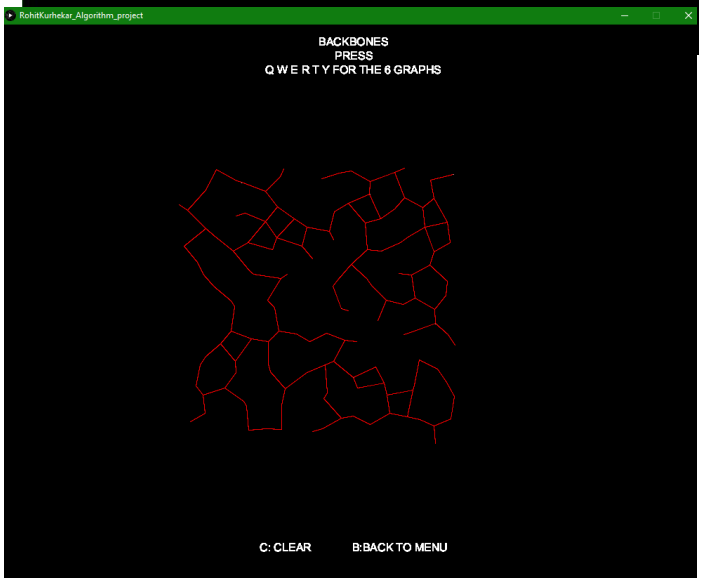
BIPARTITE 3-4



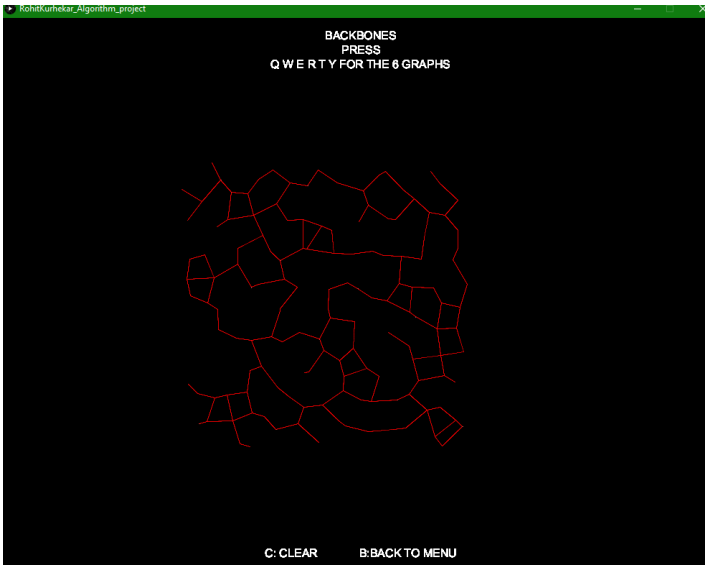
BACKBONE 1-2



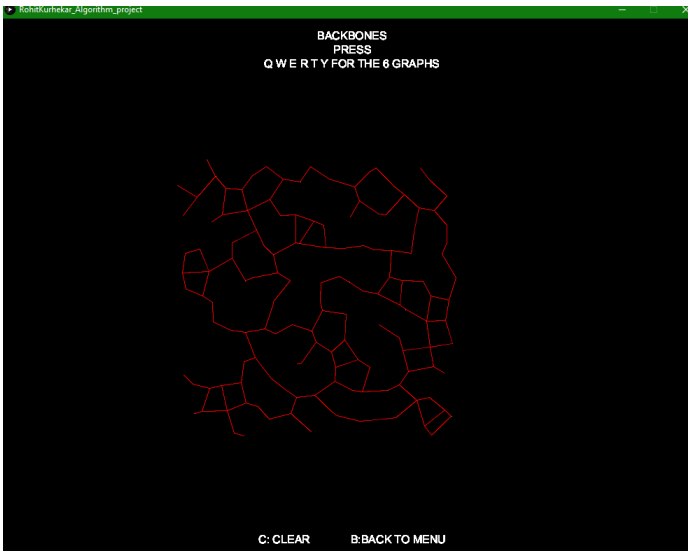
BACKBONE 1-3



BACKBONE 1-4

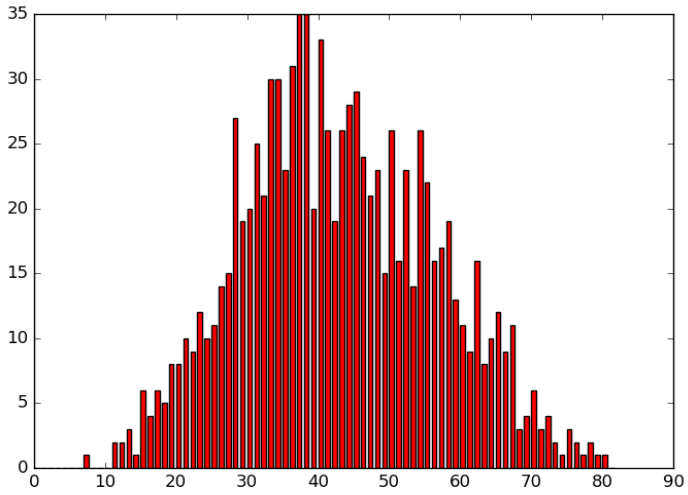


BACKBONE 2-3

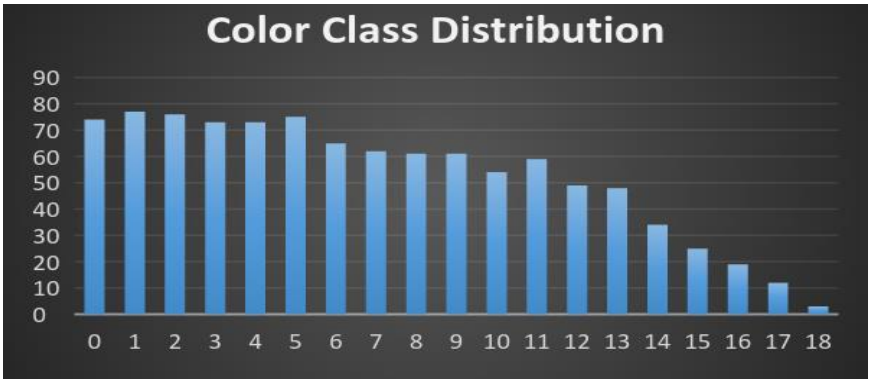


BACKBONE 2-4

BACKBONE 3-4



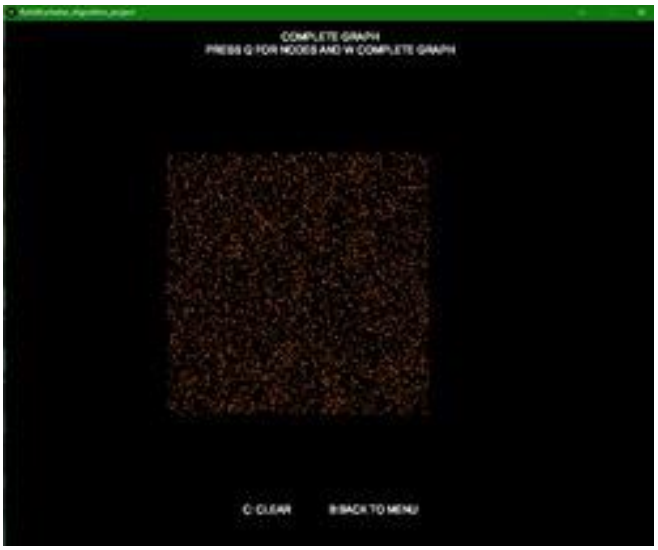
HISTOGRAM OF DEGREE



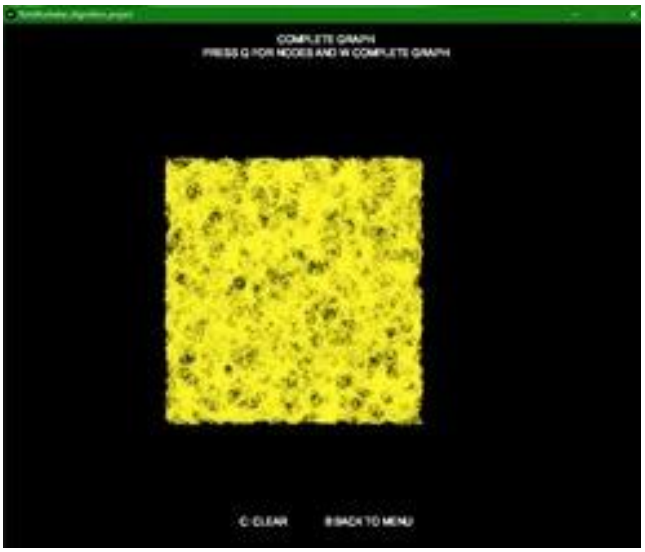
BACKBONE	VERTICES	EDGES	FACES
1. 1-2	17	1254	1239
2. 1-3	18	1314	1298
3. 1-4	18	1314	1298
4. 2-3	17	1254	1239
5. 2-4	17	1254	1239
6. 3-4	18	1314	1298

Similar Values were recorded for the following Benchmarks as well. They will be attacheed in the appendix.

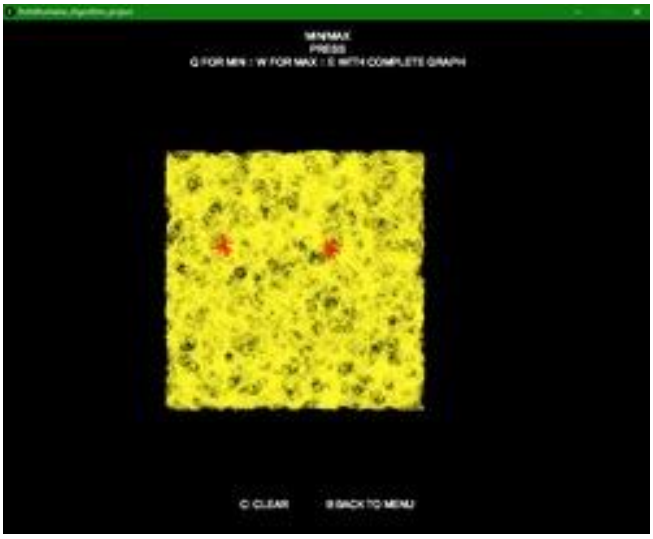
Benchmark 2 (4000 vertices, 64 average degree, Square):



NODES



COMPLETE GRAPH



MIN-MAX

COLOR 1

COLOR 2

COLOR 3

COLOR 4

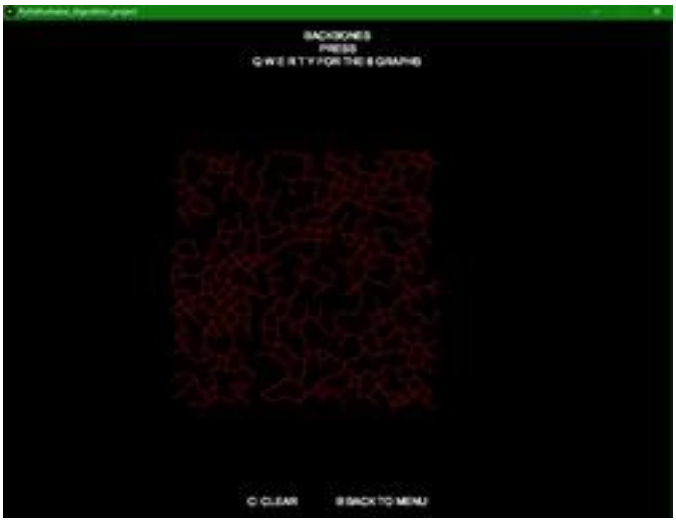
BIPARTITE 1-2

BIPARTITE 1-3

BIPARTITE1-4

BIPARTITE 2-3

BIPARTITE 2-4



BIPARTITE 3-4

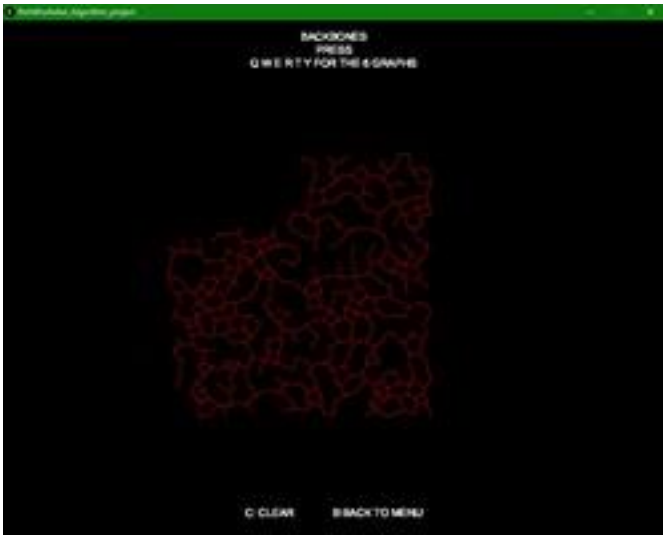
BACKBONE 1-2

BACKBONE 1-3

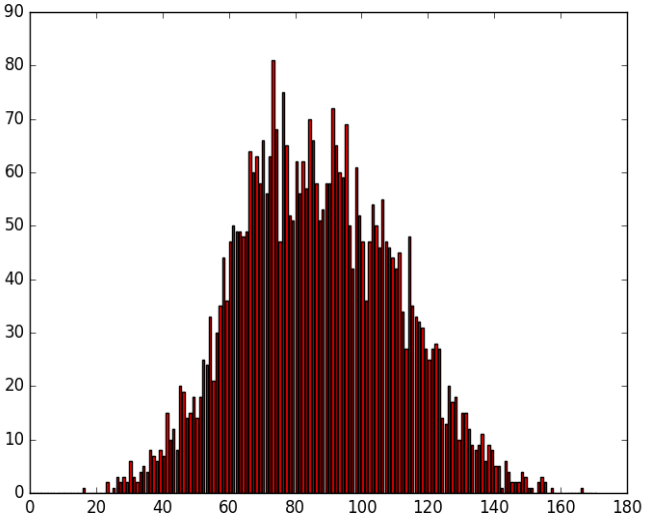
BACKBONE 1-4

BACKBONE 2-3

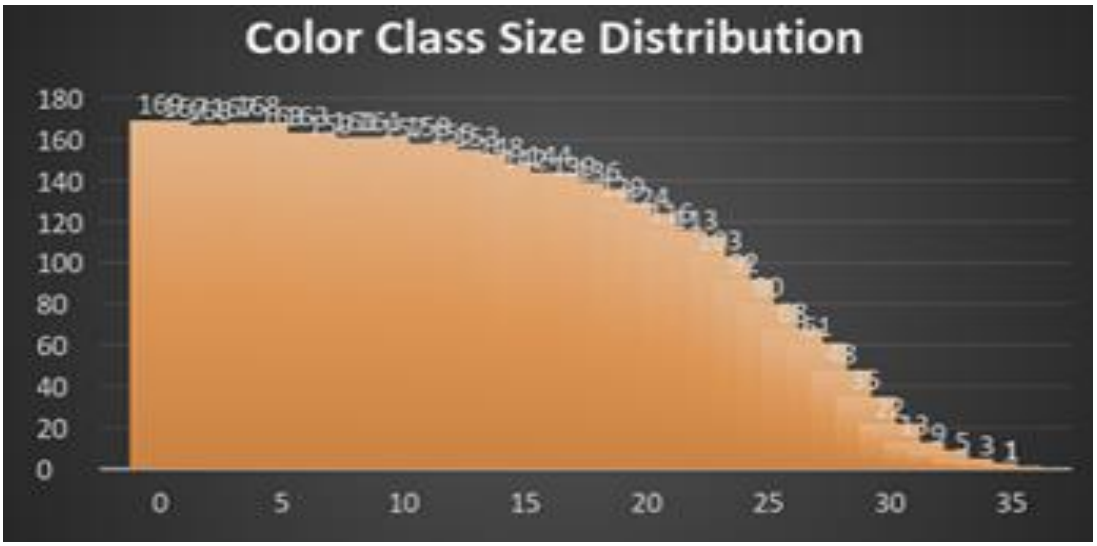
BACKBONE 2-4



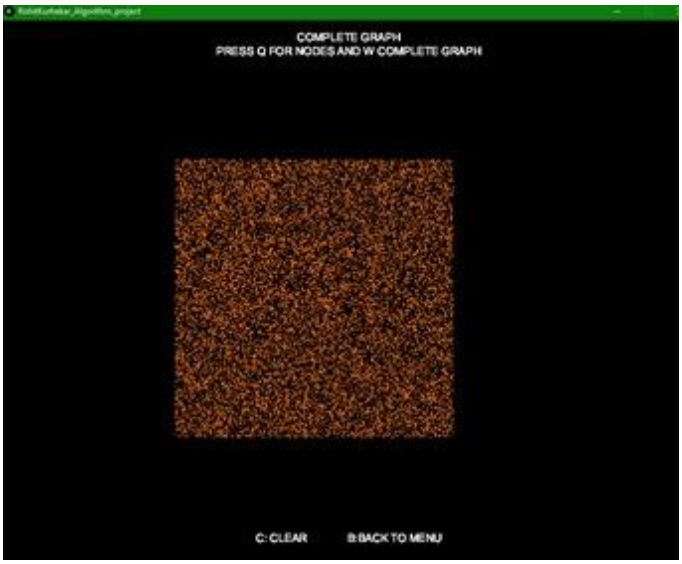
BACKBONE 3-4



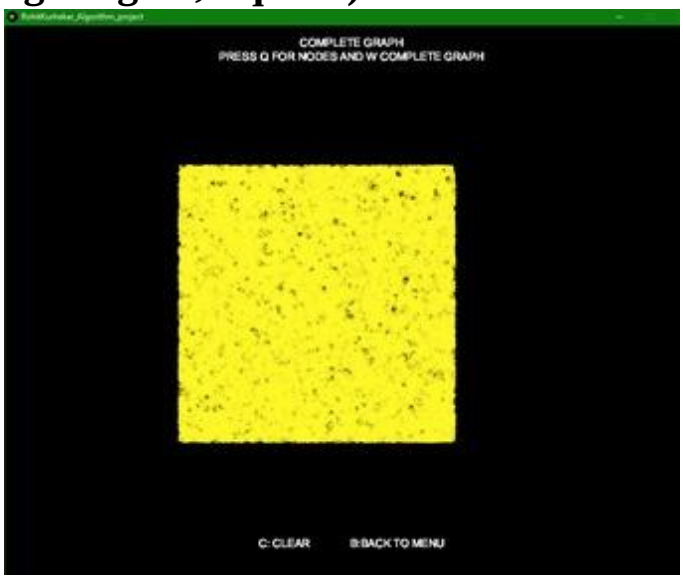
DEGREE DISTRIBUTION



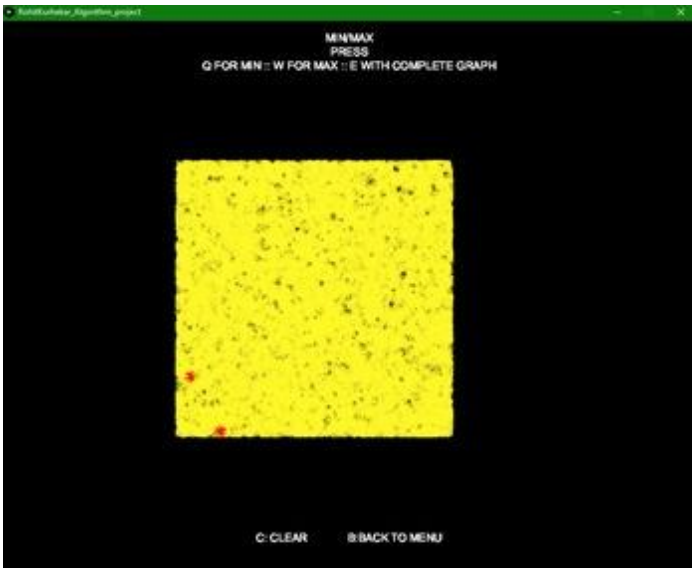
Benchmark 3 (16000 vertices, 64 average degree, Square)



NODES

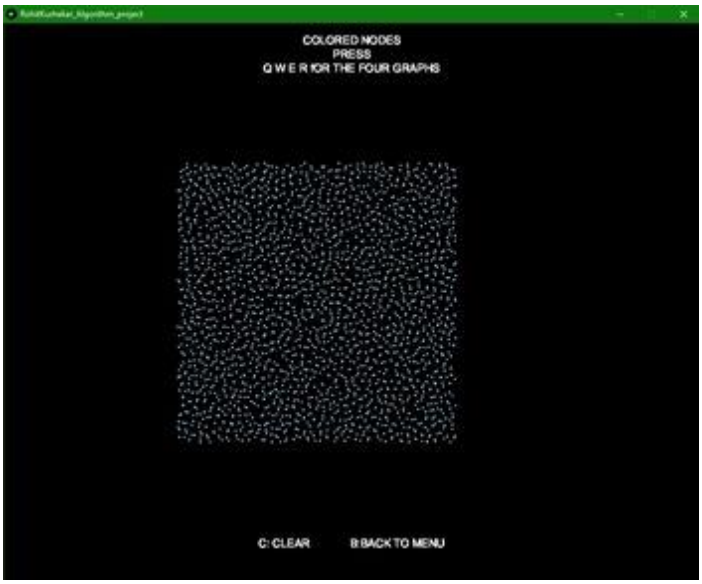


COMPLETE GRAPH



MIN-MAX

COLOR 1 and 2



COLOR 3 and 4

BIPARTITE 1-2

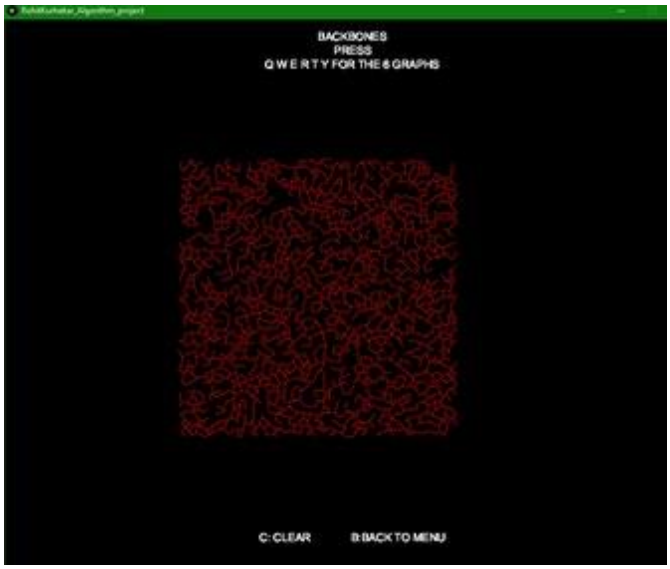
BIPARTITE 1-3

BIPARTITE 1-4

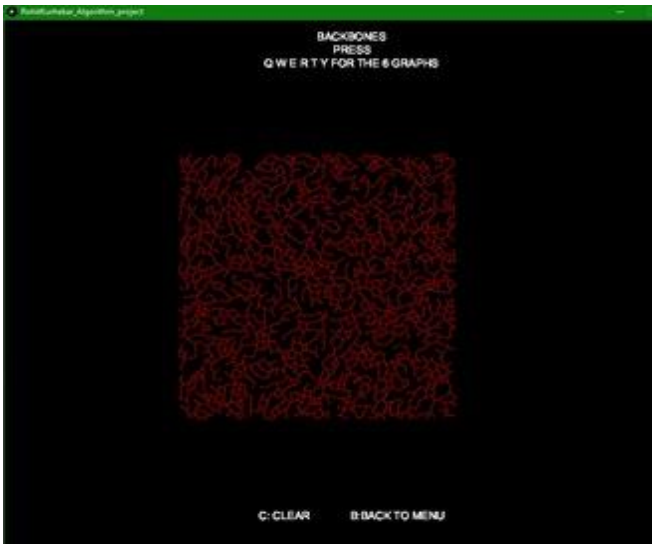
BIPARTITE 1-2

BIPARTITE 2-4

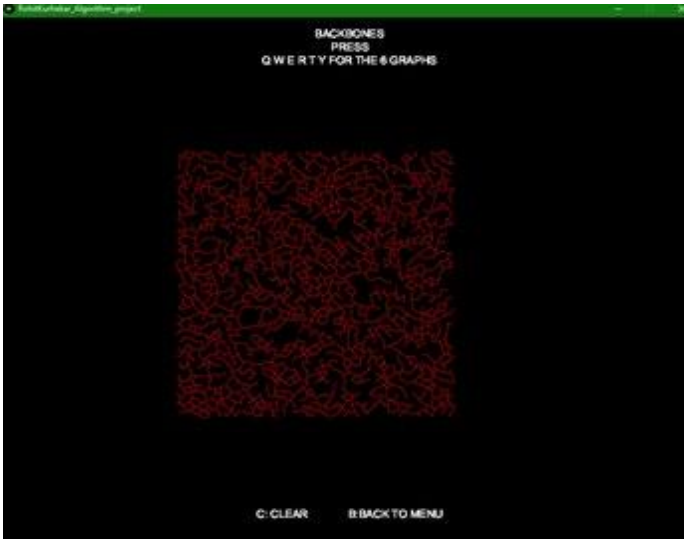
BIPARTITE 3-4



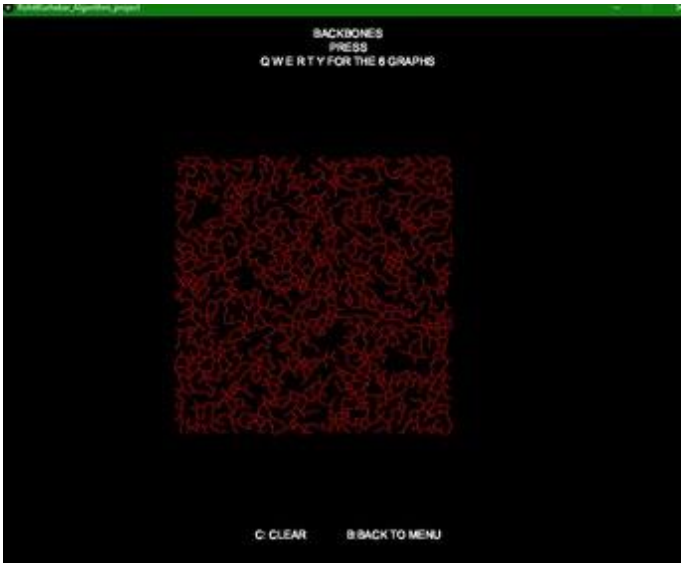
BACKBONE 1-2



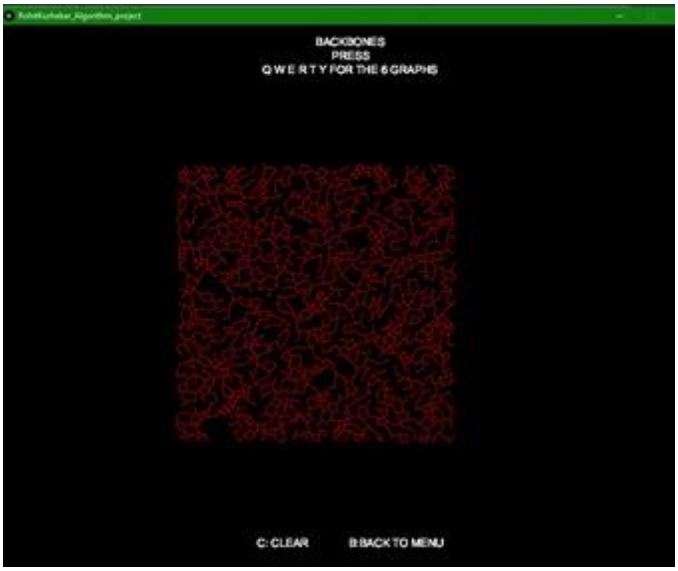
BACKBONE 1-3



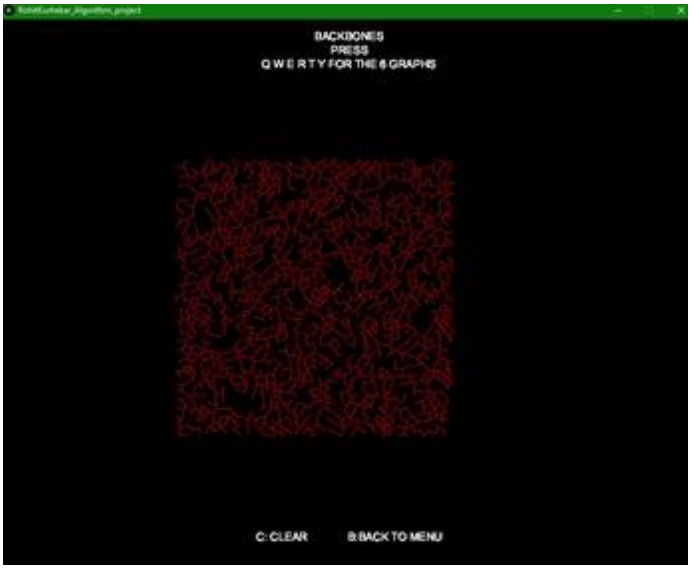
BACKBONE 1-4

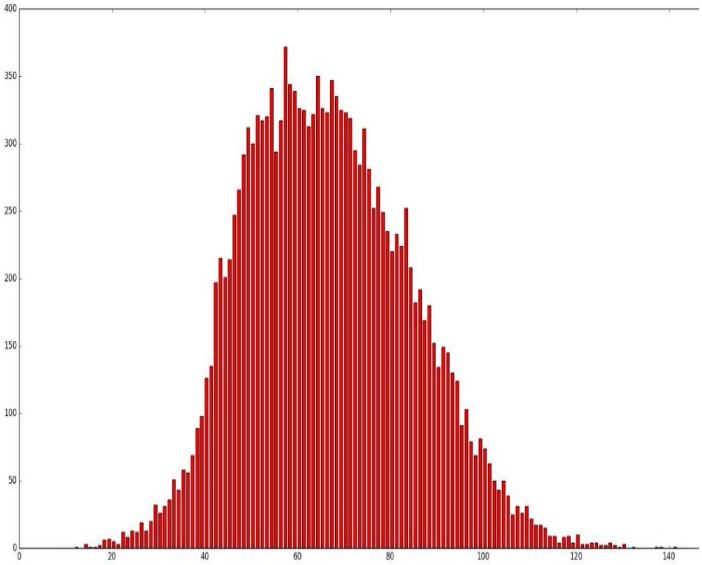


BACKBONE 2-3

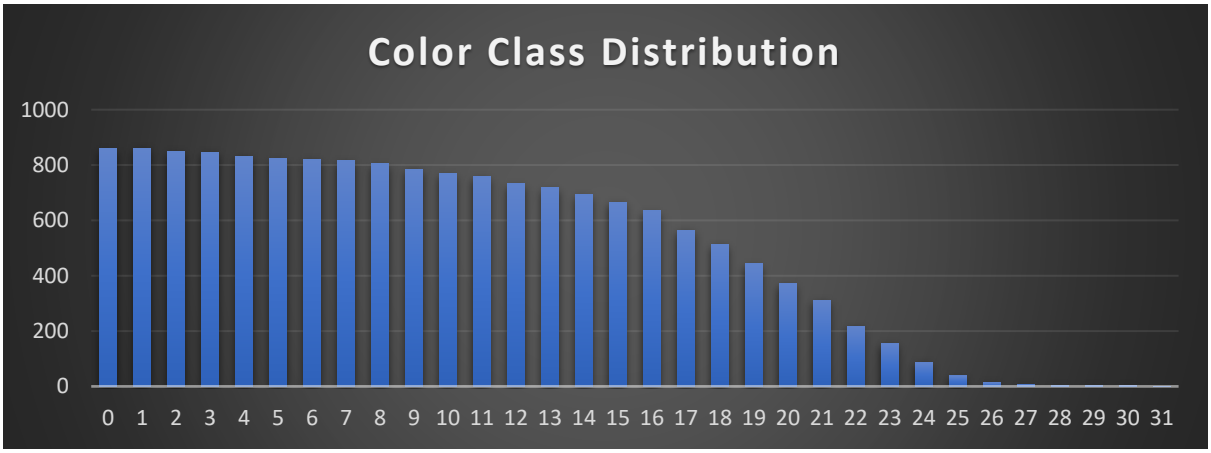


BACKBONE 2-4

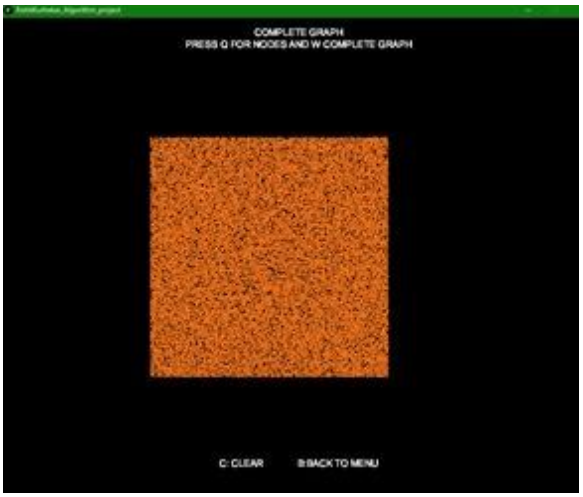




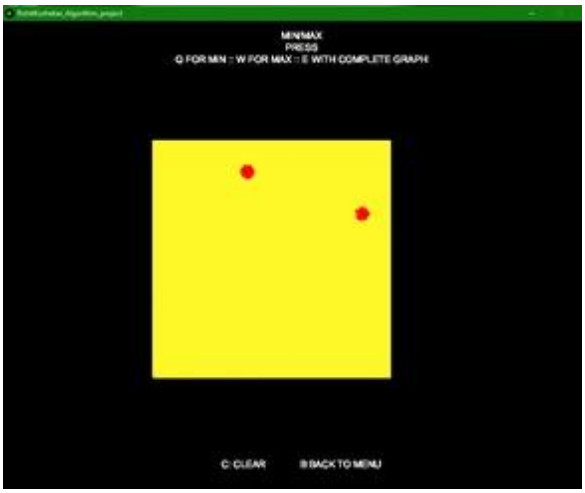
BACKBONE 3-4 HISTOGRAM OF DEGREE



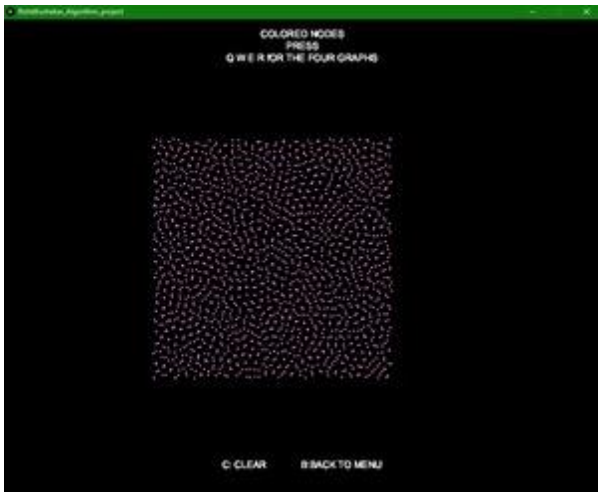
Benchmark 4 (64000 nodes, 64 average degree, Square):



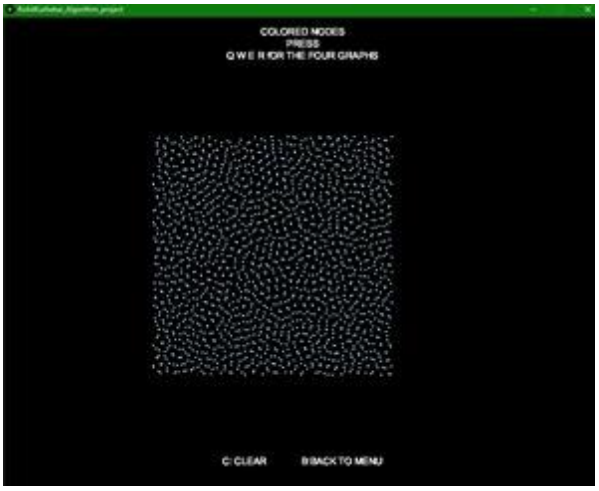
NODES COMPLETE GRAPH



MIN_MAX

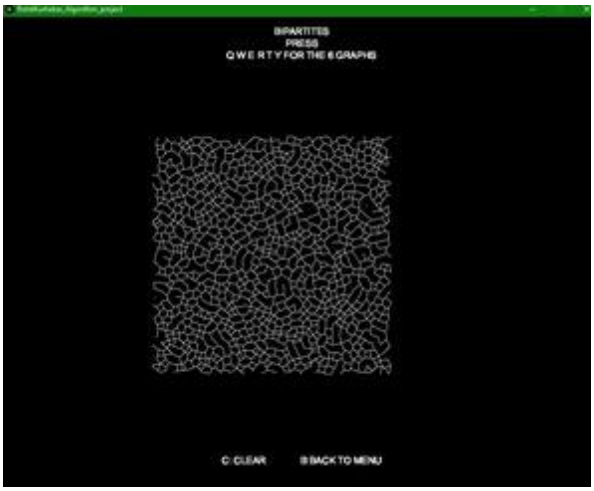


COLOR 1&2



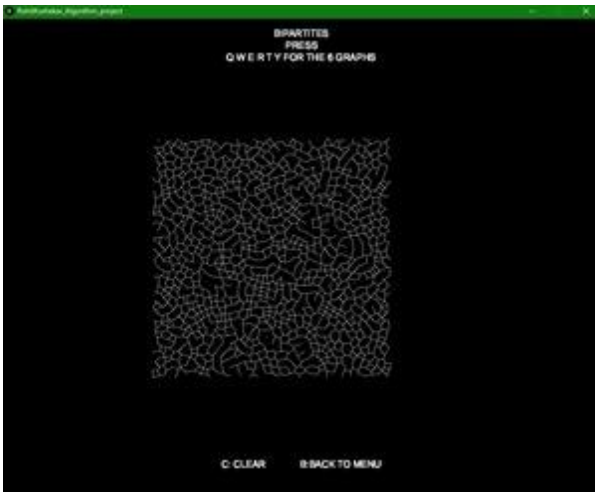
COLOR 3&4

BIPARTITE 1-2



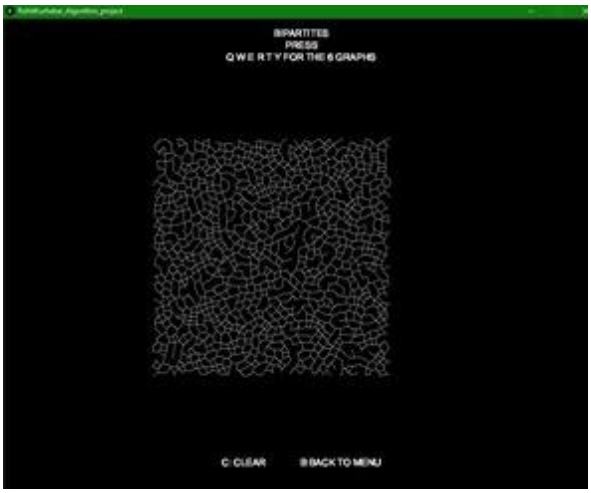
BIPARTITE 1-3

BIPARTITE 1-4

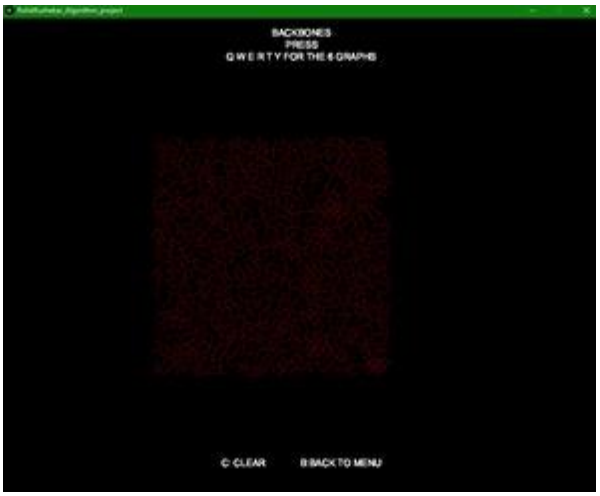


BIPARTITE 2-3

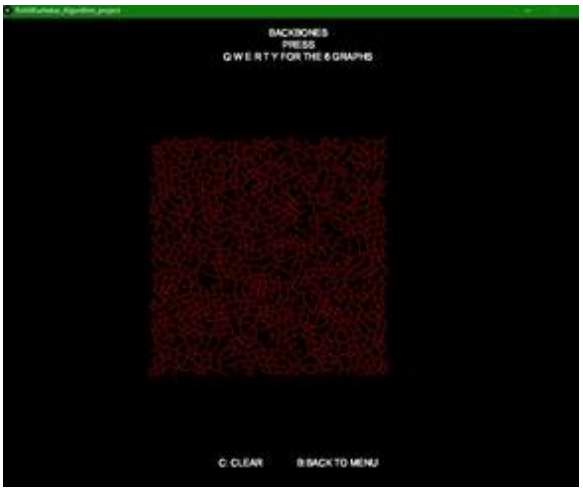
BIPARTITE 2-4



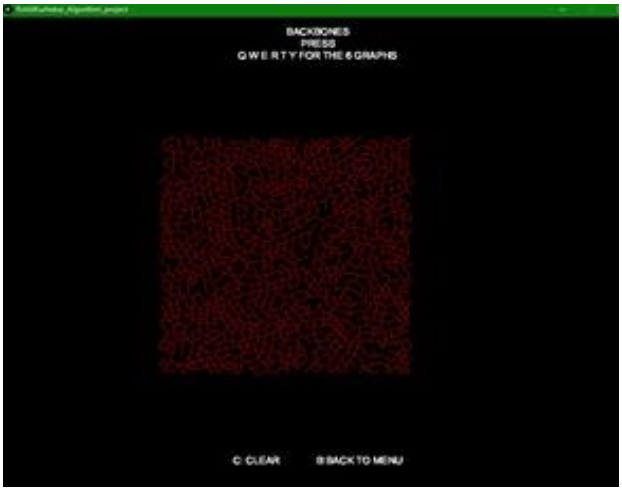
BIPARTITE 3-4



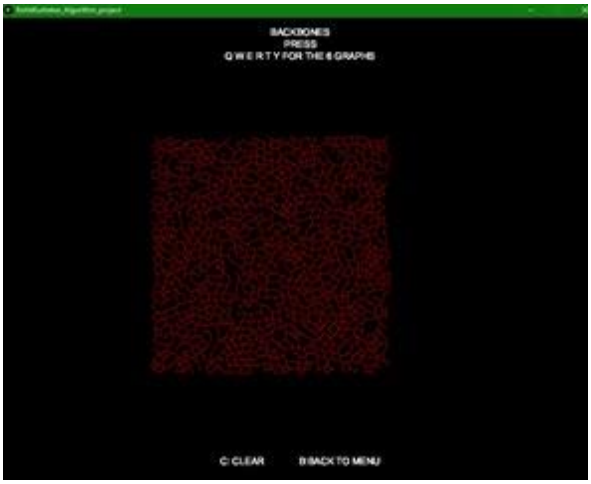
BACKBONE 1-2

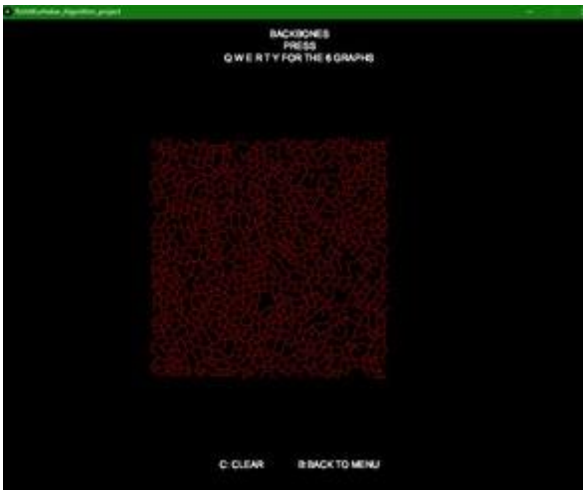


BACKBONE 1-3

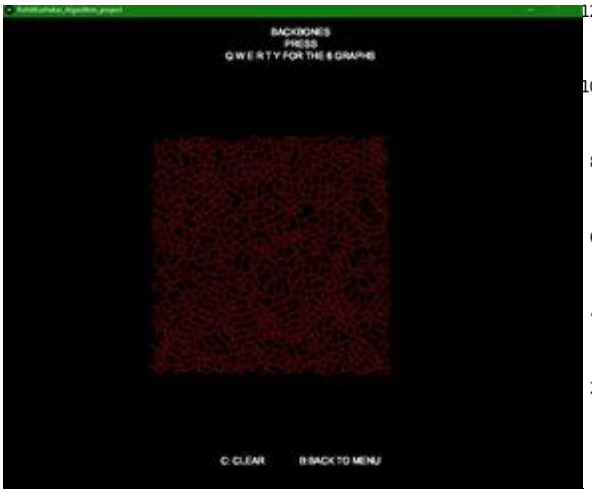


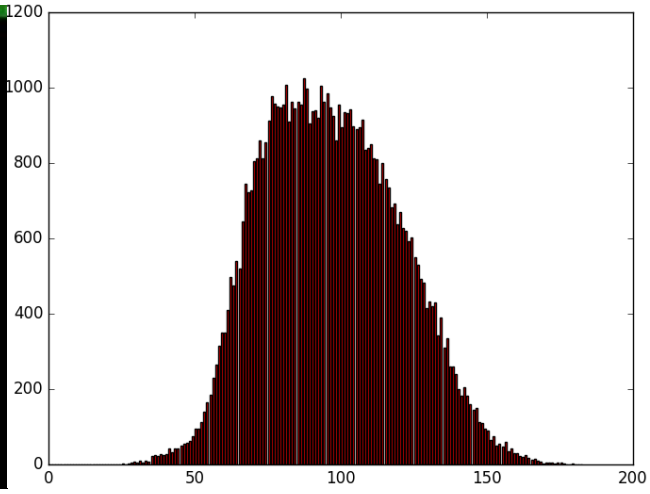
BACKBONE 1-4



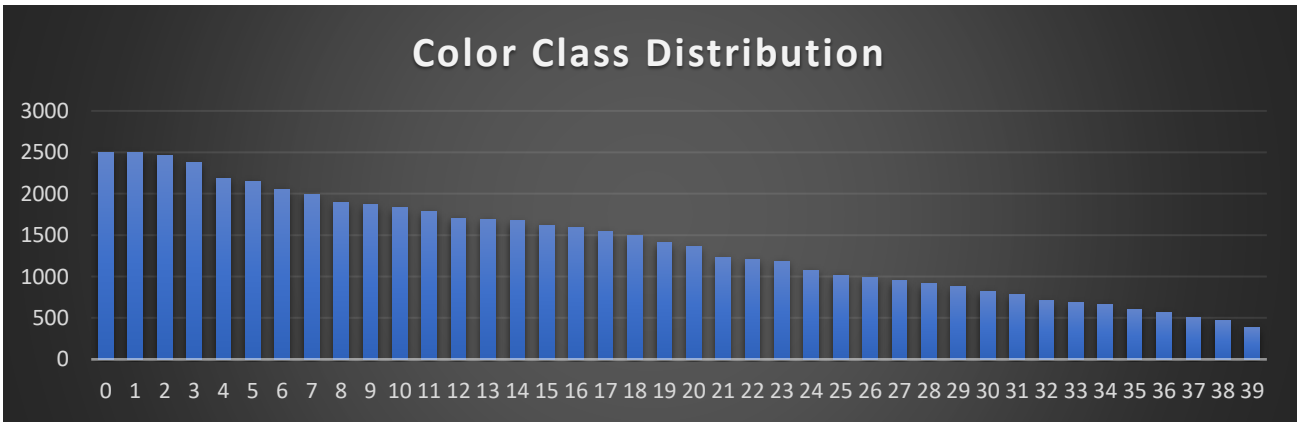


BACKBONE 2-3 and BACKBONE 2-4





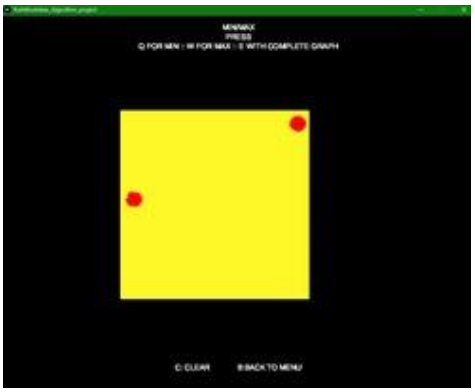
BACKBONE 3-4 DEGREE DISTRIBUTION



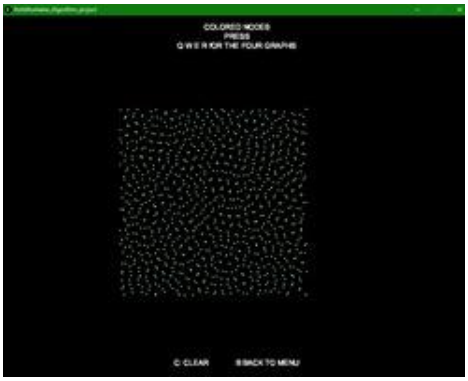
Benchmark 5 (64000 nodes, 128 average degree, Square):

NODES

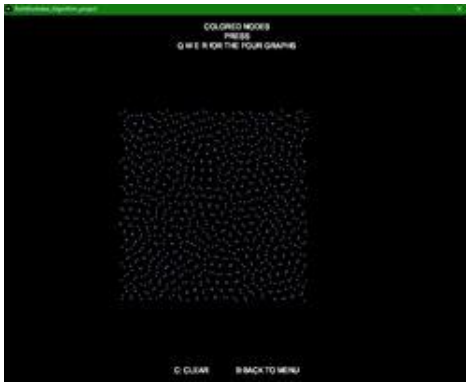
COMPLETE GRAPH



MAX AND MIN D

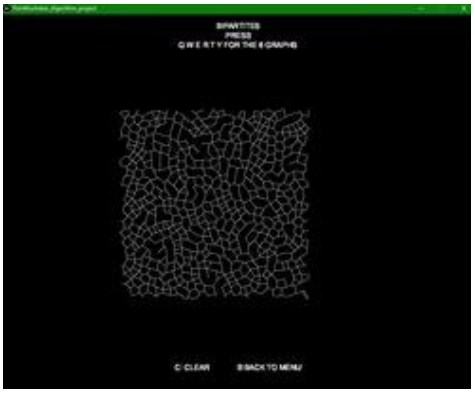


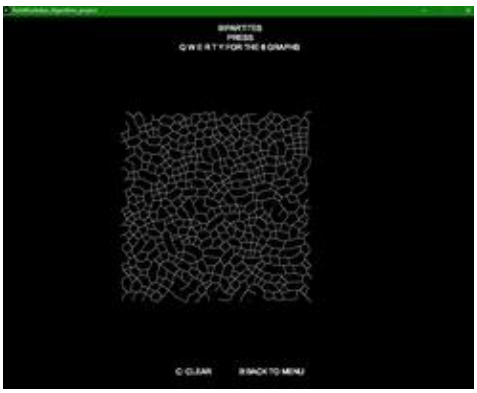
(COLOR 1&2



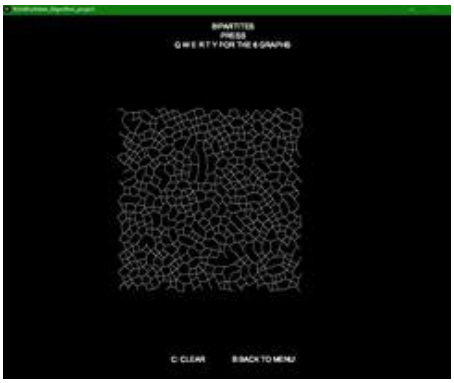
COLOR 3&4

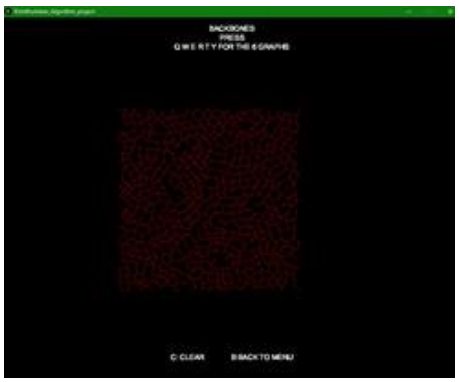
BIPARTITE 1-2

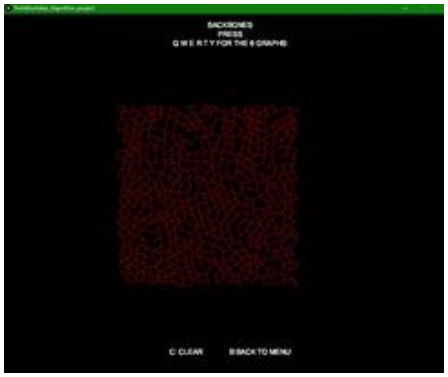




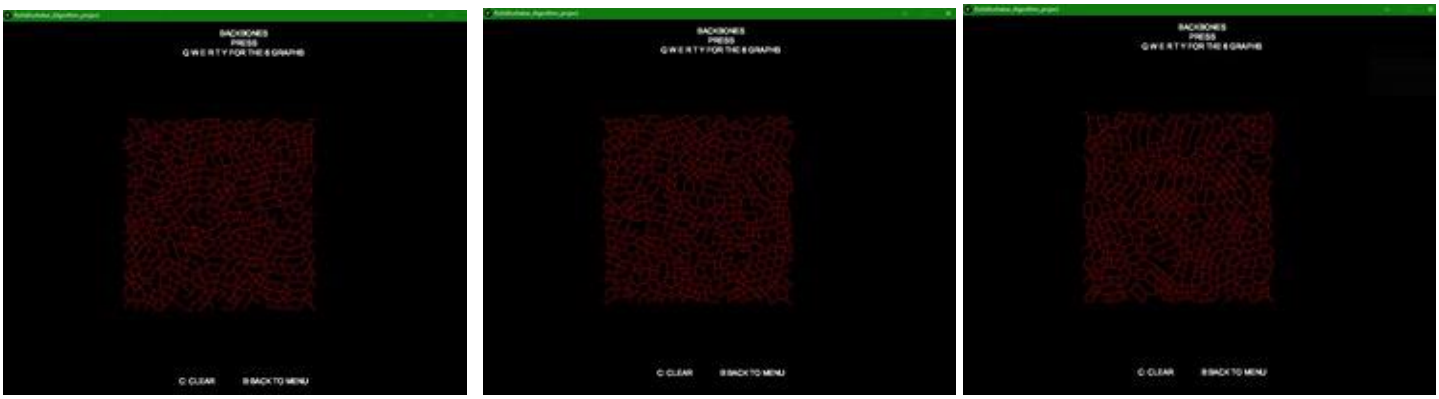
BIPARTITE 1-3 AND BIPARTITE 1-4 AND BIPARTITE 2-3



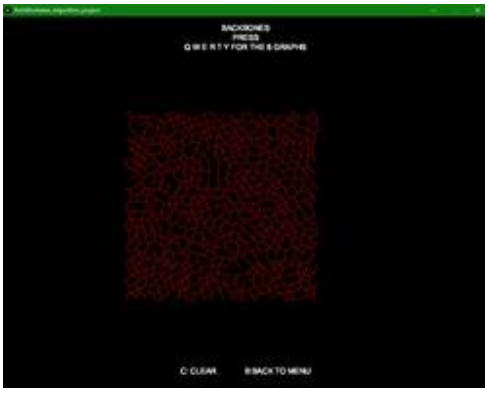


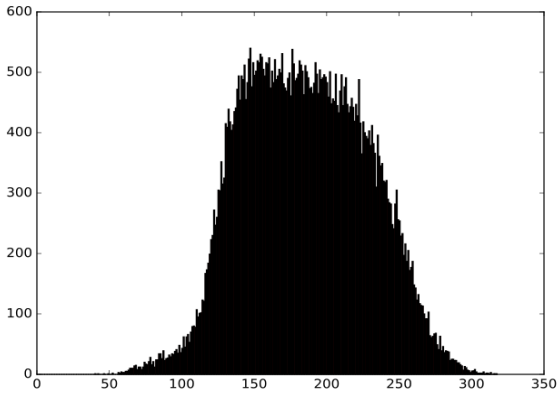


BIPARTITE 3-4 AND BACKBONE 1-2 AND BACKBONE 1-3

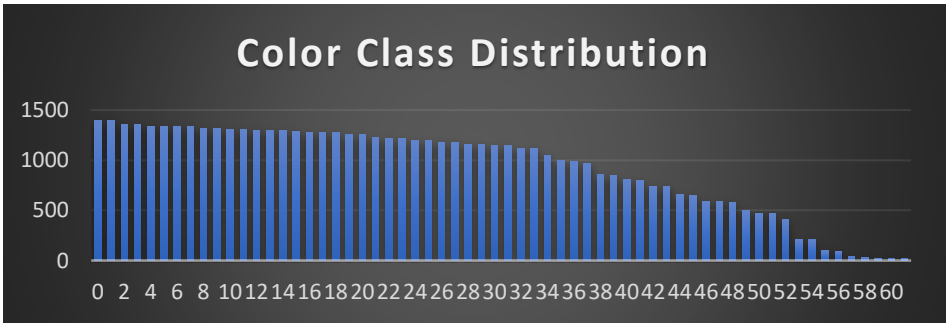


BACKBONE 1-4 AND BACKBONE 2-3 AND BACKBONE 2-4

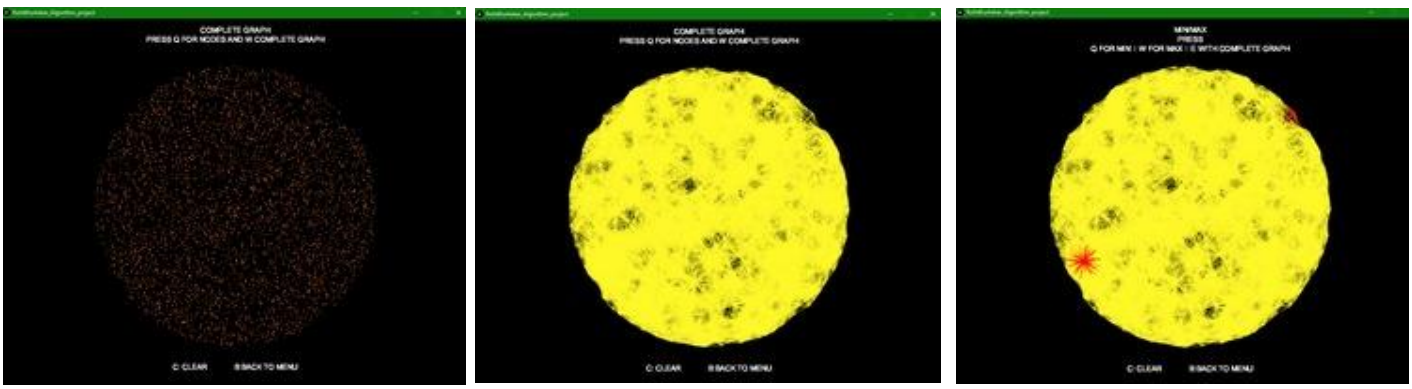




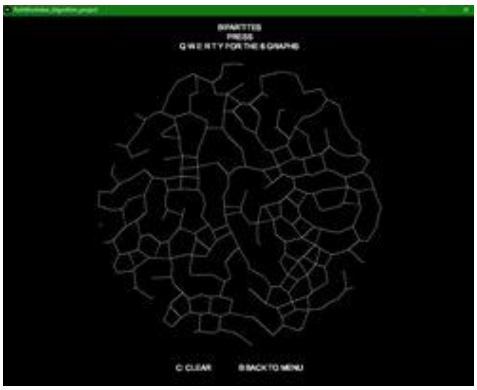
BACKBONE 3-4 DEGREE DISTRIBUTION



Benchmark 6 (4000 vertices, average degree 64, Disk):

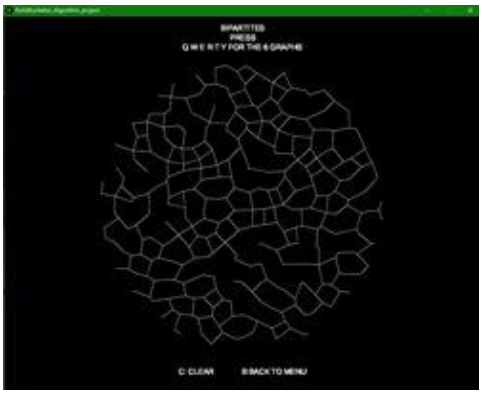


NODES AND COMPLETE GRAPH AND MAX &MIN



COLOR 1&2 AND COLOR 3&4 AND BIPARTITE 1-2

BIPARTITE 1-3

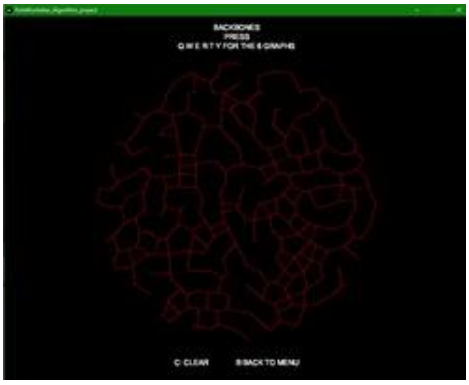


BIPARTITE 1-4

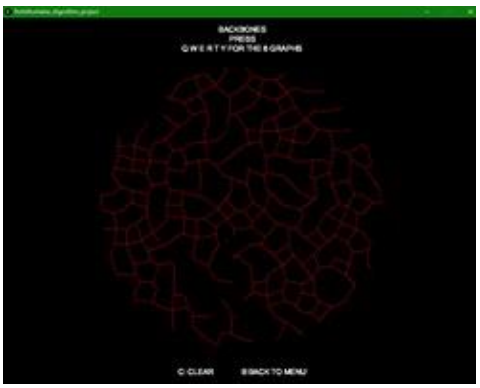
BIPARTITE 2-3

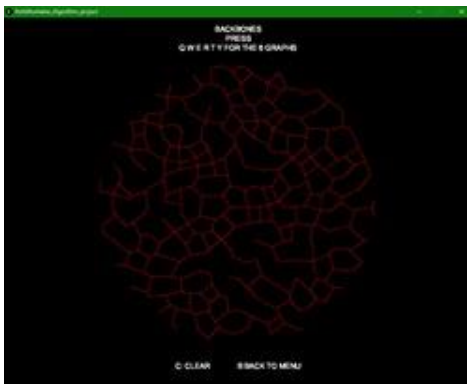
BIPARTITE 2-4

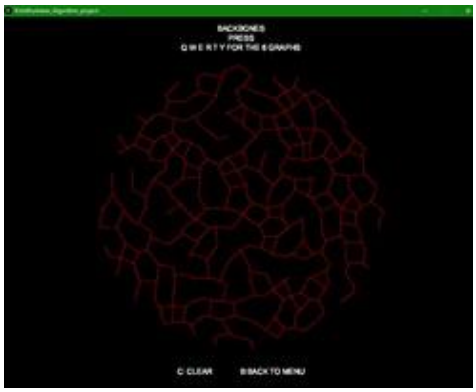
BIPARTITE 3-4



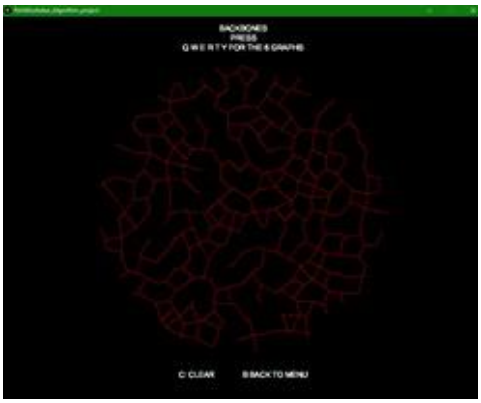
BACKBONE 1-2

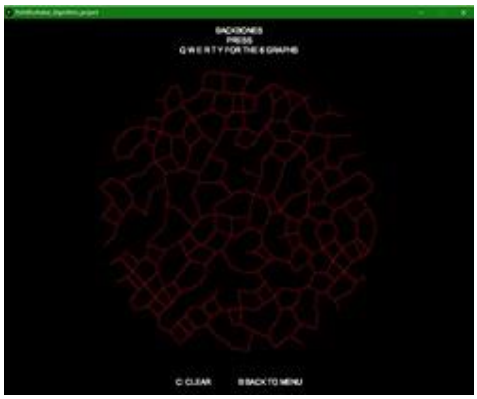


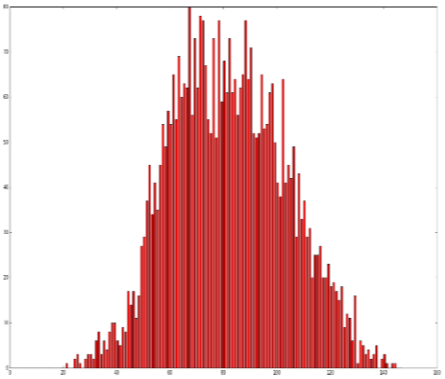




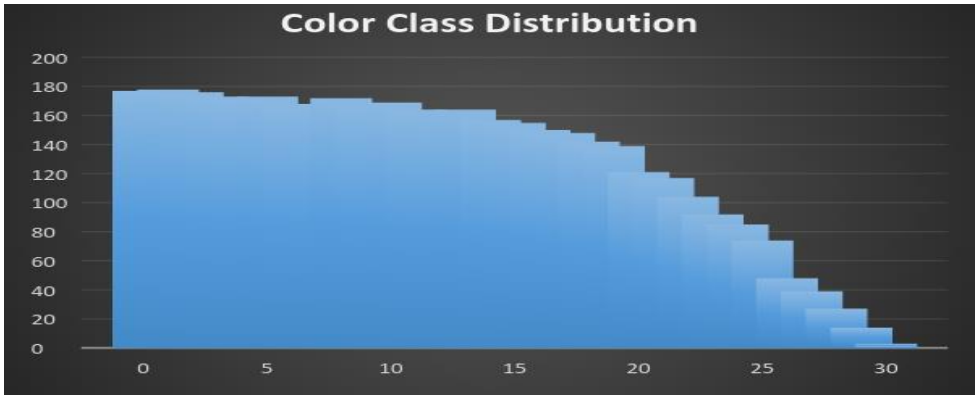
BACKBONE 1-3 BACKBONE 1-4 BACKBONE 2-3



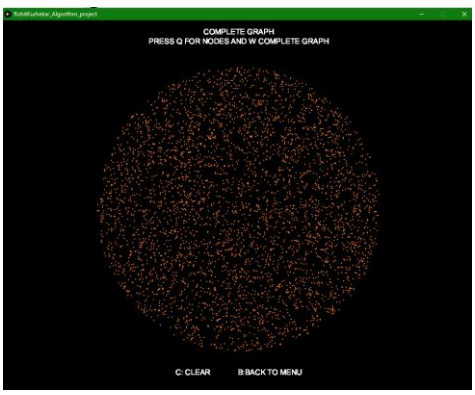


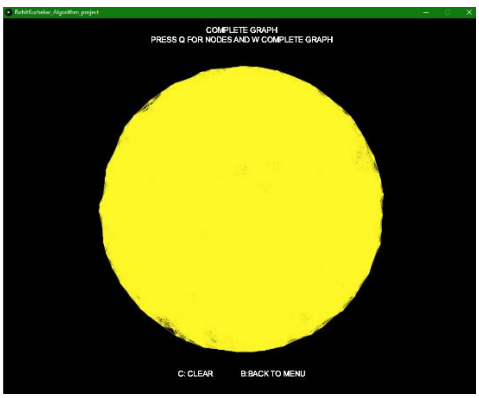


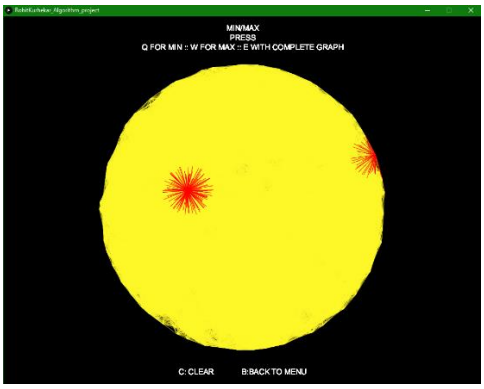
BACKBONE 2-4 BACKBONE 3-4 DEGREE DISTRIBUTION



Benchmark 7 (4000 vertices, 128 average degree, Disk):

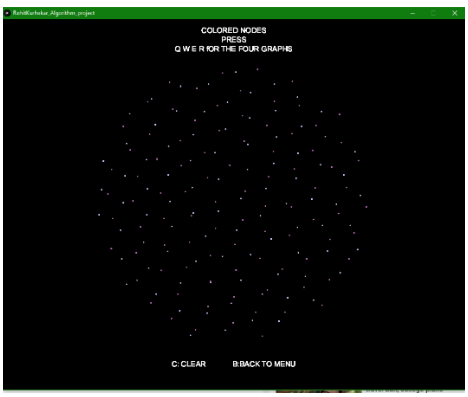


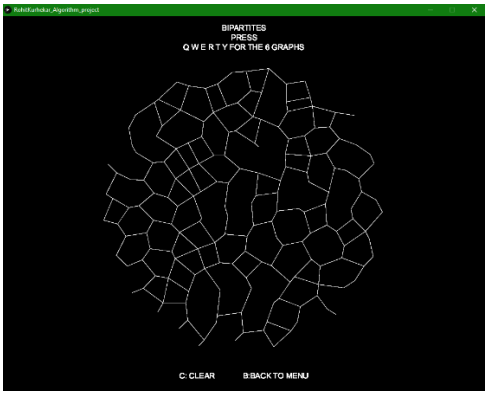




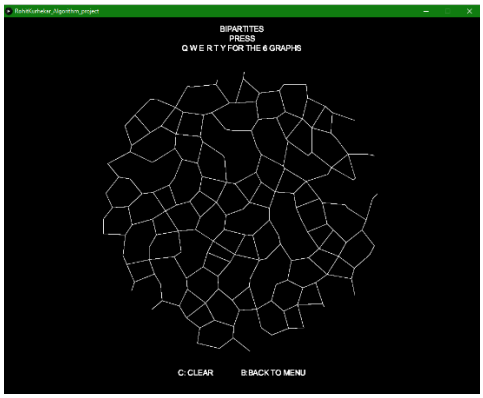
ROHIT KURHEKAR F INAL REPORT 47152070

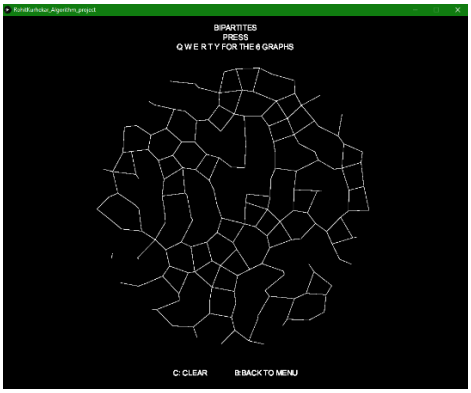
NODES COMPLETE GRAPH MIN-MAX DEGREE



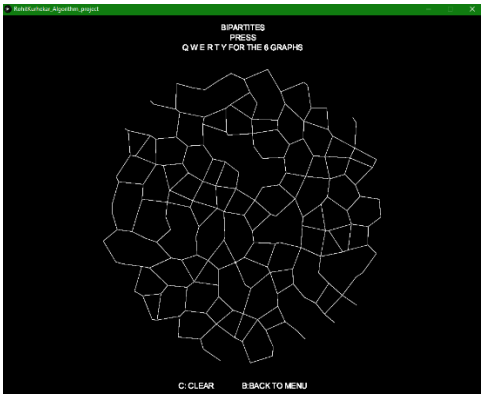


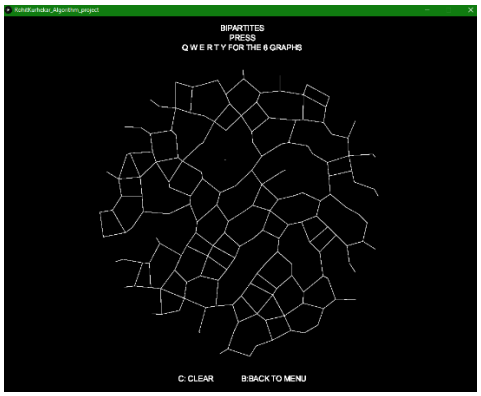
COLOR 1&2 COLOR 3&4 BIPARTITE 1-2

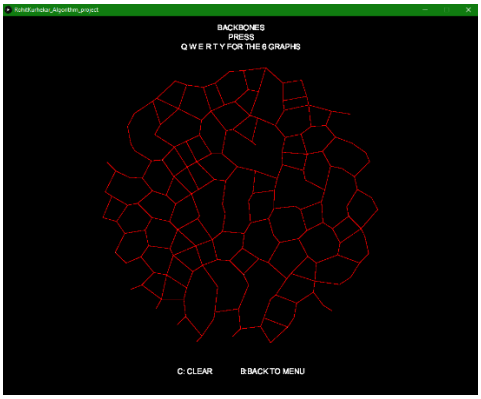




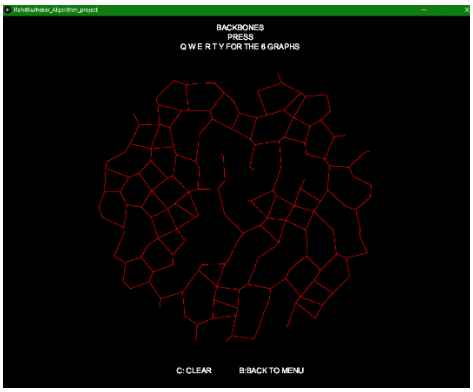
BIPARTITE 1-3 BIPARTITE 1-4 BIPARTITE 2-3



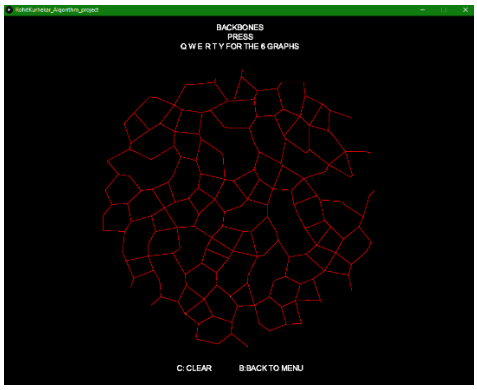




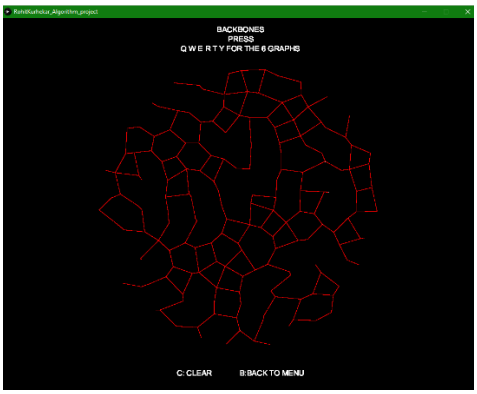
BIPARTITE 3-4 BIPARTITE 3-4 BACKBONE 1-2



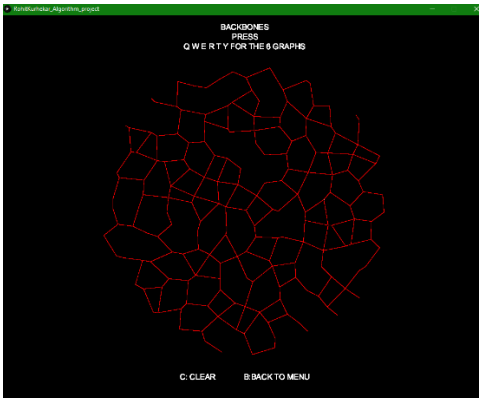
BACKBONE 1-3



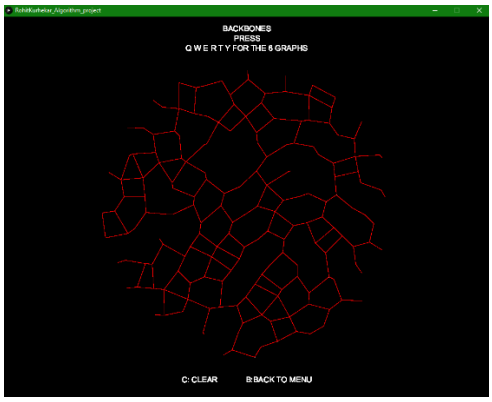
BACKBONE 1-4



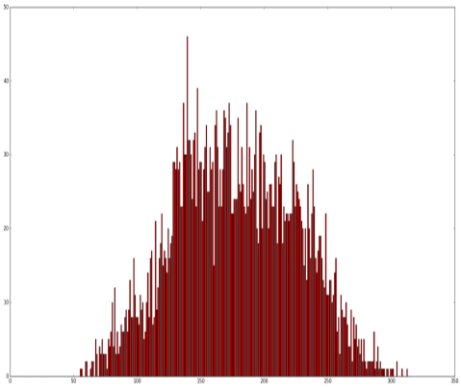
BACKBONE 2-3



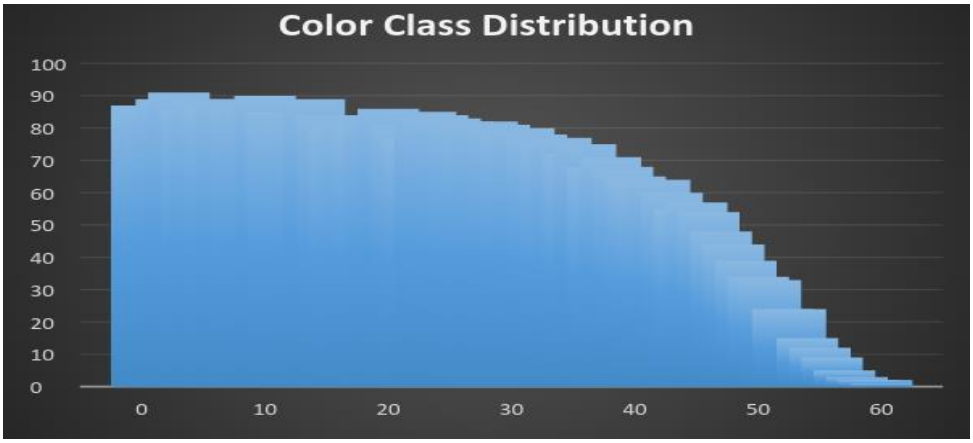
BACKBONE 2-4



BACKBONE 3-4



DEGREE DISTRIBUTION

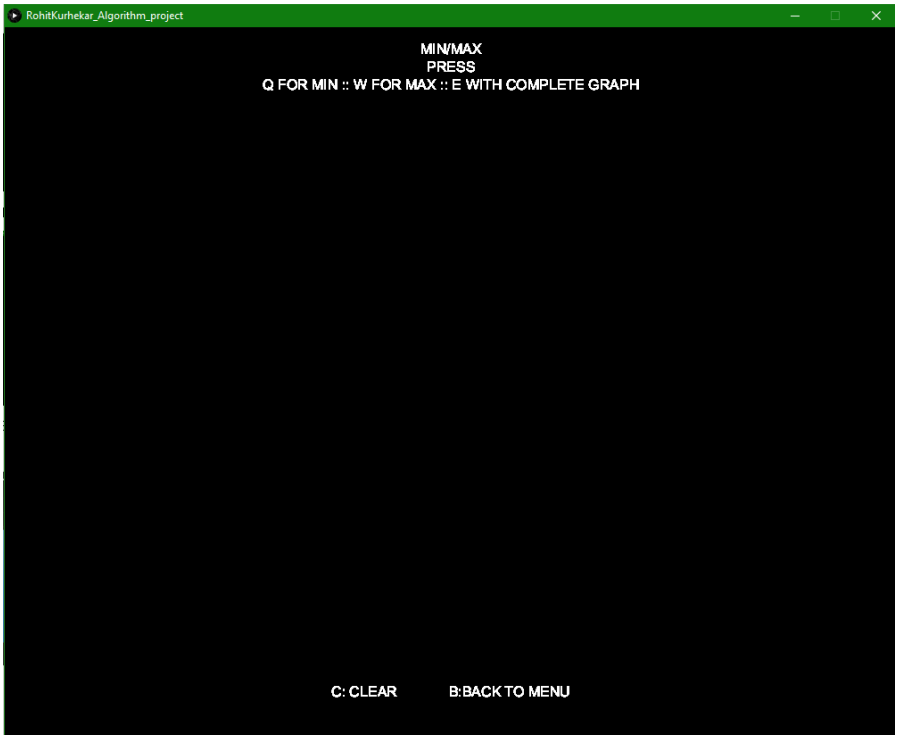


4. APPENDIX::

I have attached my program along with the report

NAVIGATION IS PROVIDED WITHIN THE APPLICATION ITSELF;

ALSO YOU CAN CLEAR THE SCREEN OR COME BACK TO THIS MENU USING C AND B RESPECTIVELY



Some Code Snippets from my program

```
class Node{
  float x , y;
  int index;
  int component = 0;
  ArrayList<Node> nearNode = new ArrayList<Node>();
  Colors c = new Colors(0,0,0);
  Node(float x , float y , int index){
    this.x = x;
    this.y = y;
    this.index = index;
  }
}

void paint(Node []p1){

  for(int i=0 ; i<num ; i++){
    int flag = 0;
    if(p1[i].nearNode.size()==0 || i==0){
       p1[i].c = plt.get(0);
    }else{
        ArrayList<Colors> nearColor = new ArrayList<Colors>();
        Iterator it = p1[i].nearNode.iterator();
        while(it.hasNext()){
          Node nearNode = (Node)it.next();
          nearColor.add(nearNode.c);
        }
        for(int j=0 ; j<plt.size() ; j++){
          boolean b = false;
          Iterator it2 = nearColor.iterator();
```

```
        while(it2.hasNext()){
          Colors nc = (Colors)it2.next();
          b = b || (nc.r==plt.get(j).r&&nc.g==plt.get(j).g&&nc.b==plt.get(j).b);
        }
        if(!b){
          p1[i].c = plt.get(j);
          flag = 1;
          break;
        }
       }
       if(flag == 0){
        Colors temp = insertColor();
        p1[i].c = temp;
        plt.add(temp);
       }
    }
  }
}

Colors insertColor(){
  Colors c;
  while(true){
    c = new Colors(int(random(100,255)),int(random(100,255)),int(random(100,255)));
    boolean test = plt.contains(c);
    if(!test){
      break;
    }
  }
  return c;
```

```
}

Map<Integer , List<Node>> setDegreeList(Node []point , int max){


  Map<Integer , List<Node>> degreeList = new HashMap<Integer , List<Node>>();
  for(int j=0 ; j<=point[max].nearNode.size() ; j++){
     ArrayList<Node> degree = new ArrayList<Node>();
     for(int i=0 ; i<num ; i++){
       if(point[i].nearNode.size() == j){
         degree.add(point[i]);
       }
     }
     degreeList.put(j , degree);
    }
    return degreeList;
}
```