WIRELESS SENSOR NETWORKS

A Project Report

by

Chandana Datta Uppalapati

SMU ID #44520129

COURSE: ALGORITHM ENGINEERING (CSE 7350)

PROFESSOR: Dr. David W. Matula

SOUTHERN METHODIST UNIVERSITY

DALLAS, TEXAS

USA

Very good project.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	LIST OF FIGURES	i
	LIST OF TABLES	vi
1.	**Executive Summary**	**1**
	1.1 Introduction and summary	1
	1.2 Programming Environment Description	2
2.	**Wireless Sensor Network Backbone Report**	**4**
	2.1 Reduction to practice	4
	2.1.1 Data structure Design	4
	2.1.2 Algorithm Descriptions	4
	2.1.3 Algorithm Engineering	8
	2.1.4 Algorithm Verification	9
	2.1.5 Algorithm Effectiveness	17
	2.1.6 Importance of these algorithms on the Wireless Sensor Networks	19
	2.2 Benchmark Result Summary Table	22
	2.3 Display and plots	23
	2.3.1 Benchmark 1	23
	2.3.2 Benchmark 2	25
	2.3.3 Benchmark 3	28
	2.3.4 Benchmark 4	30
	2.3.5 Benchmark 5	32
	2.3.6 Benchmark 6	34
	2.3.7 Benchmark 7	36
	2.3.8 Benchmark 8	38
	2.3.9 Benchmark 9	40
	2.3.10 Benchmark 10	42
	2.3.11 Benchmark 11	44
	2.3.12 Benchmark 12	46
3.	**References**	**48**

FIGURE NO	NAME	PAGE NO
26.	Relay Independent set size distribution for Primary color sets	25
27.	RGG Representing Maximum degree Vertex and minimum degree vertex (Square, N = 4000, d=40)	26
28.	RGG with all the edges	26
29.	1st Bipartite subgraph	26
30.	2nd Bipartite subgraph	26
31.	3rd Bipartite subgraph	26
32.	Degree Distribution line plot	27
33.	Degree Distribution bar graph plot	27
34.	Vertex degree distribution plot	27
35.	Independent color set sizes distribution	27
36.	Relay Independent set size distribution for Primary color sets	27
37.	RGG Representing Maximum degree Vertex and minimum degree vertex (Square, N = 4000, d=60)	28
38.	1st Bipartite subgraph	28
39.	2nd Bipartite subgraph	28
40.	3rd Bipartite subgraph	28
41.	Degree Distribution line plot	29
42.	Degree Distribution bar graph plot	29
43.	Vertex degree distribution plot	29
44.	Independent color set sizes distribution	29
45.	Relay Independent set size distribution for Primary color sets	29
46.	RGG Representing Maximum degree Vertex and minimum degree vertex (Square, N = 16000, d=100)	30
47.	1st Bipartite subgraph	30
48.	2nd Bipartite subgraph	30
49.	3rd Bipartite subgraph	30
50.	Degree Distribution line plot	31
51.	Degree Distribution bar graph plot	31

LIST OF FIGURES

FIGURE NO	NAME	PAGE NO
1.	Random points for square geometry for N = 20	9
2.	Random geometric graph with vertices and edges.	10
3.	Updated Random geometric graph after deleting First element in the degree list	11
4.	Clique subgraph obtained	12
5.	after deleting all elements from degree list	13
6.	Degree plot	13
7.	Colored vertices	15
8.	Color plot	16
9.	First Bipartite subgraph	16
10.	Boundary effect in disc	17
11.	Boundary effect in Square	17
12.	Example degree plot for square N= 4000 and d =60	18
13.	Example color plot for Square N = 4000 and d = 60	19
14.	Radio around the random points in a circular geometry	20
15.	Part of a network from coloring Algorithm	21
16.	Part of a network with relays	21
17.	RGG Representing Maximum degree Vertex and minimum degree vertex (Square, N = 1000, d=40)	23
18.	RGG with all the edges	23
19.	1st Bipartite subgraph	23
20.	2nd Bipartite subgraph	23
21.	3rd Bipartite subgraph	23
22.	Degree Distribution line plot	24
23.	Degree Distribution bar graph plot	24
24.	Vertex degree distribution plot	24
25.	Independent color set sizes distribution	25

FIGURE NO	NAME	PAGE NO
52.	Vertex degree distribution plot	31
53.	Independent color set sizes distribution	31
54.	Relay Independent set size distribution for Primary color sets	31
55.	RGG Representing Maximum degree Vertex and minimum degree vertex (Square, N = 32000, d=150)	32
56.	1st Bipartite subgraph	32
57.	2nd Bipartite subgraph	32
58.	3rd Bipartite subgraph	32
59.	Degree Distribution line plot	33
60.	Degree Distribution bar graph plot	33
61.	Vertex degree distribution plot	33
62.	Independent color set sizes distribution	33
63.	Relay Independent set size distribution for Primary color sets	33
64.	RGG Representing Maximum degree Vertex and minimum degree vertex (Disk, N = 4000, d=40)	34
65.	RGG with all the edges	34
66.	1st Bipartite subgraph	34
67.	2nd Bipartite subgraph	34
68.	3rd Bipartite subgraph	34
69.	Degree Distribution line plot	35
70.	Degree Distribution bar graph plot	35
71.	Vertex degree distribution plot	35
72.	Independent color set sizes distribution	35
73.	Relay Independent set size distribution for Primary color sets	35
74.	RGG Representing Maximum degree Vertex and minimum degree vertex (Disk, N = 8000, d=100)	36
75.	1st Bipartite subgraph	36
76.	2nd Bipartite subgraph	36
77.	3rd Bipartite subgraph	36

FIGURE NO	NAME	PAGE NO
78.	Degree Distribution line plot	37
79.	Degree Distribution bar graph plot	37
80.	Vertex degree distribution plot	37
81.	Independent color set sizes distribution	37
82.	Relay Independent set size distribution for Primary color sets	37
83.	RGG Representing Maximum degree Vertex and minimum degree vertex (Sphere, N = 4000, d=40)	38
84.	RGG on a semi sphere	38
85.	1st Bipartite subgraph	38
86.	2nd Bipartite subgraph	38
87.	3rd Bipartite subgraph	38
88.	Degree Distribution line plot	39
89.	Degree Distribution bar graph plot	39
90.	Vertex degree distribution plot	39
91.	Independent color set sizes distribution	39
92.	Relay Independent set size distribution for Primary color sets	39
93.	RGG Representing Maximum degree Vertex and minimum degree vertex (Sphere, N = 16000, d=100)	40
94.	RGG on a semi sphere	40
95.	1st Bipartite subgraph	40
96.	2nd Bipartite subgraph	40
97.	3rd Bipartite subgraph	40
98.	Degree Distribution line plot	41
99.	Degree Distribution bar graph plot	41
100.	Vertex degree distribution plot	41
101.	Independent color set sizes distribution	41
102.	Relay Independent set size distribution for Primary color sets	41

FIGURE NO	NAME	PAGE NO
103.	RGG Representing Maximum degree Vertex and minimum degree vertex (Sphere, N = 64000, d=150)	42
104.	RGG on a semi sphere	42
105.	1st Bipartite subgraph	42
106.	2nd Bipartite subgraph	42
107.	3rd Bipartite subgraph	42
108.	Degree Distribution line plot	43
109.	Degree Distribution bar graph plot	43
110.	Vertex degree distribution plot	43
111.	Independent color set sizes distribution	43
112.	Relay Independent set size distribution for Primary color sets	43
113.	RGG Representing Maximum degree Vertex and minimum degree vertex (Disk, N = 80000, d=250)	44
114.	1st Bipartite subgraph	44
115.	Degree Distribution line plot	45
116.	Degree Distribution bar graph plot	45
117.	Vertex degree distribution plot	45
118.	Independent color set sizes distribution	45
119.	RGG Representing Maximum degree Vertex and minimum degree vertex (Disk, N = 500, d=40)	46
120.	1st Bipartite subgraph	46
121.	2nd Bipartite subgraph	46
122.	3rd Bipartite subgraph	46
123.	Degree Distribution line plot	47
124.	Degree Distribution bar graph plot	47
125.	Vertex degree distribution plot	47
126.	Independent color set sizes distribution	47
127.	Relay Independent set size distribution for Primary color sets	47

LIST OF TABLES

TABLE NO	NAME	PAGE NO
1.	Benchmark Result Summary Table	22

CHAPTER – 1
EXECUTIVE SUMMARY

1.1 Introduction and Summary

A sensor network is a combination of large number of small stations which are known as the sensor nodes. These sensor nodes are low-power, low-cost and multi-functional sensor nodes which can be deployed in the Region of Interest. Generally these sensor nodes are made up of sensors, radio transceivers, embedded microprocessors [2]. Sensors communicate with each other in a wireless medium and hence entire sensors forming a network are known as the wireless sensor networks. Therefore we can tell that number of sensor nodes in a wireless sensor network will directly affect the cost of the entire wireless sensor network. There are many applications of wireless sensor networks [1].One of the most important aspect and metric of the wireless sensor network is the coverage level in region that it is deployed. Each sensor has the ability to transmit and receive radio signals and therefore each sensor will have a Radio around them. This combination of radios of all the sensors will form the coverage for the entire wireless sensor network. Therefore it is very important to decide how many sensors are needed and where they need to be placed [7] in the network keeping reliability and robustness in mind. This work simulates a series of wireless sensor network backbones on three different geometries with different number of sensors and provides a best way of deploying of the sensors so that the cost can be reduced by reducing number of sensors, provides good network coverage, efficient routing and also a reliable and robust wireless sensor network system using efficient and effective algorithms. Single and simple one liner importance of this work can be for minimizing the investment cost and maximize the reliability.

Each sensor is considered to be a point in one of the three geometries that are employed in this work square, disc, and sphere. The smallest last vertex ordering algorithm and the graph coloring algorithm [5] [6] are applied in order to get series wireless sensor network backbones from the random set of points along with the plane sweep algorithm which generates the random geometric graph. The interesting part about these algorithms is their effective time and space complexities. The results of this work include graphs and a table along with the obtained sensor network obtained which is also known as a bi-partite subgraph or the backbone subgraph. 5 graph plots for each set of different number of random points on each geometry are shown, with each

graph plot having its own importance. The graph plot 1 represents the number of vertices for the degree ranging from minimum degree to maximum degree and this usually looks like a cliff because of the boundary effect in the square and circular geometries, whereas, for a sphere we will get a cliff but a wider one than the one we get for a square and sphere. The graph 2 is a bar graph style plot of the graph plot 1 which shows the cliff effect clearly. Graph plot 3 shows the degree plot of the vertices initially and when they are being removed. This graph plot also shows an important property of the random geometric graph which is the 'clique'. A clique of a graph is a subset of its vertices such that every two vertices of the subset are connected by an edge forming a complete subgraph [3]. The importance of clique in the random geometric graphs given in [4]. Graph plot 4 shows the number of vertices for each of the color set obtained highlighting the primary color sets. Graph plot 5 shows the number of vertices in the relay independent set for each of the primary color set obtained from the maximum independent set algorithm. A table is presented in this work which gives the summary of results where rows 1 – 5 of the table gives the results for different number of sensors being deployed in square as region of interest. For the circle or disk as the region of interest rows 6 and 7 and if sphere is the region of interest rows 8 – 10 give the results.

1.2 Programming Environment Description

Hardware

PC Model	:	HP Envy
Operating System	:	Windows 8.1
Physical Memory (RAM)	:	8GB
Processor	:	Intel Core i7-4702MQ
Clock Rate	:	2.2 GHz
Processor cores	:	4

Simulation Tools

Language	:	Python 2.7
IDE	:	IPython Notebook
Shell	:	IPython
Graphic Tool	:	OpenGL
Libraries	:	numpy, PyOpenGL, Vispy, Plotly

The number of random points in the program code is dependent on the physical memory available (RAM). Processor cores be advantageous only when the code is designed for parallel programming or running different independent programs on different cores.

Now the programming language used for the simulation is the python. The reason behind the selection of python language is that it is portable so that I can run my code on any machine without any overhead. Even though its runtime is slow when compared to other languages because it interprets during runtime, because of integration with other languages made it more attractive. But instead of regularly writing the code in IDLE editor and executing the code from the command line, an interactive computing command shell named IPython is used in this work. This IPython provides a rich toolkit than compared to the python interpreter, some of them are web-based interactive notebook environment to write, edit the code with shell features, animations, figures etc. IPython also provides powerful interactive shells like the Qt-based or the terminal based. This supports interactive data visualization. OpenGL is one of the best and easy way to do low level graphics using python. OpenGL is a cross platform programming language interface that is capable of creating 2D and 3D graphics with points, lines etc. PyOpenGl is a python binding, that is, the python code acts like a wrapper kind of thing around the native code so that the OpenGL can be controlled from within the context of the Python. As OpenGL is cross platform programming language we can use it with the python. Vispy [8] is a visualization library which offers a Pythonic Object-oriented interface to OpenGL, that is, for the commonly used features of the OpenGL a simplified API is provided by the Vispy. The pipeline in OpenGl defines the way in which the data is processed on the GPU for rendering, Data buffers[9] is one of the major elements for rendering the pipeline and is used to store the numerical data on the GPU. Vispy provides an effective and simple way to store these buffers. In this work the Numpy is used because, operations on the n-dimensional arrays are provided by fast and easy routines. Numpy is one of the most important package for doing scientific computing. Now to hold the vertices or nodes in this work a structured numpy array is used [9]. Plotly is a data visualization tool that I have used to generate the graph plots. The main reason why I used this and the advantage of using plotly is that, it is easy to graph and also each graph we created gets a unique HTML link so that it can be shared with others and the graph is very interactive such that the any part of the plot can analyzed to any extent. In this work under each graph plot I have given a HTML link for that graph plot so that they can be analyzed online.

CHAPTER - 2

WIRELESS SENSOR NETWORK BACKBONE REPORT

2.1 Reduction to Practice

2.1.1 Data Structure Design

There are some data structures used that effect the performance of the code. The position of points or vertices or nodes is stored as the numpy array, so the operations on the arrays and the memory management of the arrays is maintained by the numpy library. Other Data structure is the python 2-D list which is used to store the adjacency list of each vertex. The cost of the operation of inserting an element at the end of the list is O (1). Cost of the delete operation from a specific location of the list is also O (1). But deleting a specific element from the list takes O (n) where n is the number of elements in the list. While performing the smallest last vertex ordering algorithm a 2-D list is used to store the vertices corresponding to each degree ranging from minimum degree to maximum degree and a 1-D list is used to store degrees of the removed vertices in a reverse fashion. Now in the coloring algorithm a 2-D list is used of length (1/3rd size of color set + 1 Relay set) to append the vertices in the corresponding sub list based on the color that is assigned to them. Now for a relay independent set selection a new adjacency 2-D list for the relays that are covered more than three times for each primary color set is developed and then again based on this 2-D list a new 2-D list is used that stores the relay vertices corresponding their degree. These two 2-D lists are used by the maximum independent set algorithm to get the relay independent set.

2.1.2 Algorithm Descriptions

In this work there are four main algorithms that are used to get the series of backbone bipartite graphs.

Plane sweep algorithm is used to generate the adjacency list. We have n points or vertices from which the adjacency list is generated, that is, if the distance between two points is less than a specific threshold distance then the two points are said to be adjacent to each other. If we compare the distance between each point with every other point to the threshold then it takes a lot of time complexity which is O (n^2). This takes a long time whenever we are dealing with large number of points. So, to make the time complexity even better plane sweep method is used. It is used in this work as follows, first all the points are sorted in the x-direction. Now a plane of specific width is

defined and then this plane sweeps all over x-axis. Now each point in that sweep will only need to be compared with the every point within that sweep instead of all the points in that geometry. The degree of the vertex is the number of vertices to which it is adjacent.

Smallest last vertex ordering algorithm is used to order the vertices based on their degree from highest degree to lowest degree. This smallest last order is used in the graph coloring algorithm. From the adjacency list we form a degree list which is a 2-D list where the vertices are sorted based on their degree. To do this we use a **bucket sort** which can be time efficient, that is, just one pass through the adjacency list is sufficient to get the degree list. Here bucket sort takes the advantage because the number of buckets are less when compared to the number of elements or vertices we are dealing with. The sub lists of the new 2-D degree list represent the degree. These sub lists representing each degree can be viewed as the **buckets**. That is sub list 0 represent the bucket of degree 0 so the vertices having degree 0 will placed into this sub list, vertices with degree 1 are placed in the sub list 1 and so on. Now the inputs to this smallest last vertex ordering algorithm are 2-D list of vertices divided based on their degree called degree list and the copy of the original adjacency list. The output for this algorithm is two 1-D lists where one 1-D list is used to store the vertex numbers in the smallest last order. The other output list that we obtain from this algorithm is the degree of vertices at the time they are removed in the smallest last order as the first list. Working of this algorithm is as follows, first element (vertex) of the first sub list of the degree list, that is first element or vertex of the degree 0 list, is selected and appended to the output list 1 from the last to first and the element is removed from the degreelist but before deleting it, adjacency list of this element is checked and the element is also deleted from the adjacency list of the vertices to which the element is adjacent and now the vertices from which the element is deleted are mapped or found in the degree list and their degree is reduced by 1 and so these vertices adjacent to the element are moved to the sub list which is above the sub list in which they are currently present. We also add the current degree of this element to the 2nd output list from last to first. Whatever we append to the output list1 or output list2 we append from last to first so that we get the smallest last vertex ordering. Now after the first element of degree 0 list is deleted from the degree list and after modifications as mentioned above to the degree list we check if there are any more elements in degree 0 list if there are we go and pick the next element in degree 0 list and delete and make necessary modifications to the degree list. If there are no elements in degree 0 list we go to the degree 1 list and pick the first element from there and do the modifications to the

degree list. In this way we do this process by always picking the smallest degree element first and delete it from degree list by making necessary modifications to the degree list. In this process of deleting, the last subgraph that we are going to delete is a fully connected subgraph, that is, each vertex has a connection to every other vertex. This is an important feature for the graph which is known as the Clique.

Now the **smallest last vertex coloring or the graph coloring** algorithm is applied to the random graph. Input for this graph are output list 1 from the smallest last vertex ordering algorithm called smallestlastvertextorder list, that is, list of vertices in the smallest last order. Now the other input for this algorithm is the adjacency list. We create a new output list called vertexcolor list which is of the length equal number of vertices. We also make use of a list called the color list which has list of all the color IDs in it, this list always gets the initial state of having the color IDs each time loop is running. Now first we traverse the smallestlastvertextorder list and we assign the first element with first color, that is, we add the specific color ID to that vertex index in the vertexcolor list. Now we go to the next element of the smallestlastvertextorder list and check the color assigned to the adjacent vertices using adjacency list and vertexcolor list, if the adjacent vertices are assigned with some colors then the assigned colors are deleted from the color list, now this element is assigned with the first color ID that is available in the color list by updating the vertexcolor list at the index equal to the element number. Now the loop again runs by initializing the color list to original state and picking the next element from the smallestlastvertextorder list. In this way all the vertices will be assigned with a specific color and are stored in the vertexcolor list.

Relay Independent set selection or the Maximum Independent set algorithm is used to find the different backbone or the bipartite graphs. Based on the vertexcolor list we got from the graph coloring algorithm we use the bucket sort to divide the vertices based on their color assigned to them. As we are only interested in the first $1/3^{rd}$ colors since number of vertices assigned to each of these $1/3^{rd}$ colors is almost the same. Therefore we create $1/3^{rd}$ empty buckets or lists in this case for each of the different $1/3^{rd}$ color in the color set plus one extra bucket called rely bucket to store all the other vertices called the relay pool of vertices. Now we traverse through the vertexcolor list once and in that traversal the vertices that belong to one of the $1/3^{rd}$ colors are placed in the corresponding buckets that are created and all the other vertices are placed in the relay pool bucket. Now after completely traversing through the vertexcolor list we will get the

buckets and the relay pool filled. All these 1/3rd buckets are just the sub lists of a list called primarycolorset list. The primarycolorset list and the relay pool will be one of the inputs to the Relay Independent set selection algorithm and the other input will be original adjacency list. This Relay Independent set selection algorithm first selects the different list of relays from the relay pool that are connected to 3 or more vertices for each of the primarycolorset lists. Now Maximum Independent set algorithm is applied on each of the selected relay lists. First we pick first primarycolorset from the primarycolorset list and now we traverse the relay pool. We pick the first element from the relay pool and check it with the first primarycolorset that is selected, are there 3 or more vertices to which this selected first relay element is connected, if there are then we will add this first relay element to a list called selectedrelays, delete that element from the relay pool and select the next element from the relay pool do the same process. If it is not connected to 3 or more vertices then we will leave that relay element in the relay pool and pick the next element. Like this the complete relay pool is traversed and we get a list selectedrelays which have the relays that are connected to 3 or more vertices with respect to the first primarycolorset. Now we apply Maximum Independent set algorithm on this selectedrelays list. Now based on the original adjacency list, new adjacency list for the relays in selectedrelays list formed with only the vertices present in this selectedrelays list. Again the bucket sort is used to place the relays into buckets according to their degree forming a list called relaydegree list. After this we traverse the relaydegreelist from the element with minimum degree to highest degree. After picking the first element with minimum degree from the relaydegree list we add it to the output list called maxindependentrelays now we go to each of the vertices to which this element is adjacent based on the new adjacency list and delete all these vertices but before deleting them adjacency list of each of the vertex is checked and each of the vertex is also deleted from the adjacency list of the vertices to which the each of the vertex is adjacent and now the vertices from which each of the vertex is deleted are mapped or found in the relaydegree list and their degree is reduced by 1 and so these vertices adjacent to the each of the vertex that is deleted are moved to the sub list which is above the sub list in which they are currently present. The vertices which are deleted like this are added to a list called temporaryrelays. Now as we have added the first element that we have picked from the relay degree list to the maxindependentrelays list and done the necessary modifications we delete it from the adjacency list and also from the relay degree list. We do not add this element to the temporaryrelays list. Likewise we traverse through the complete

relaydegree list. We get final lists for temporaryrelays and maxindependentrelays, vertices in the temporaryrelays are added back to the relay pool since they can be used for other primary sets. Vertices in the maxindependentrelays list and the vertices from the first primarycolorset are used to form the first backbone bipartite graph. After all these operations we will pick the second primarycolorset sub list from the primarycolorset list but before that we will empty the list called maxindependentrelays, temporaryrelays, relaydegree. Likewise the process will be looped for all of the 1/3rd primary color sets which gives equal number bipartite backbone graph networks.

2.1.3 Algorithm Engineering

Smallest last vertex ordering algorithm, as to delete a specific element which is at specific element in the python 2-D list it takes O (1) and to append to a list it takes O (1) time complexity. Constructing a degree list from the adjacency list requires O (|E| + |V|). Since using bucket sort in one pass through adjacency list we can construct the degree list. We need an extra space of just O (|V|) because we are just storing all the vertices but in a sorted order. Where |E| is the total number of edges in the Random Geometric Graph and |V| is the total number of vertices. As per the implementation the algorithm amount of time needed for each vertex is the equal to the degree of that vertex, since we go to each adjacent vertex of that vertex and update them in the degree list deleting and inserting which is a constant operation. So the total operation for that vertex is the degree of that vertex. Now for all the vertices time required will be the sum of all the degrees which is nothing but the total number edges that is |E| plus the total number of vertices as we are traversing through all the vertices |V|, therefore the time complexity is O (|E| + |V|). Extra storage of 2 |E| is required for copy of adjacency list as we need to successively update the adjacency list upon deletion of a vertex from the degree list and the total storage required for the two adjacency list and the degree will be 4 |E| + O (|V|) which also includes the space required for the output array which stores the smallest last vertex order. A more space efficient smallest last vertex ordering algorithm is proposed in [10].

Graph coloring algorithm, we traverse the list of vertex in the smallest last order, for each vertex we need to look at the adjacent vertices and check the color based on that we delete the colors from the color list, add the color to the vertex color list at the index specified by the vertex, which are constant operations and thus the total time complexity for all the vertices will be sum of the degrees which is the |E| plus |V| time for traversing through all vertices. Therefore the time

complexity is O (|E| + |V|). Space complexity, extra space required for color list and the output vertex color list. For the output color list it is of length O (|V|) and for the color list we do not need any space greater than the number of vertices |V|, we actually need only space equal to the maximum degree obtained from the smallest last vertex ordering algorithm + 1 since we only need those many colors in order to color the graph efficiently so that no two adjacent vertices get the same color, therefore the space for this algorithm we need is O (|V|).

2.1.4 Algorithm Verification

For the verification of the algorithms we consider a comparatively random geometric graph on a square geometry with 20 points and the radius of the circle formed by the radio of each sensor or point is 0.40.

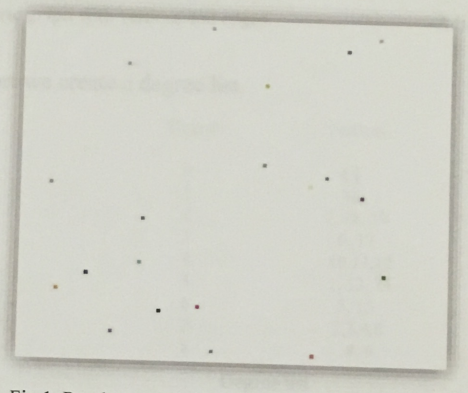

Fig 1. Random points for square geometry for N = 20

Now the plane sweep method is used to find the adjacency list, where a plane sweeps across the square geometry along the x-axis. Now based on the adjacency list edges between vertices are plotted as shown in the below figure.

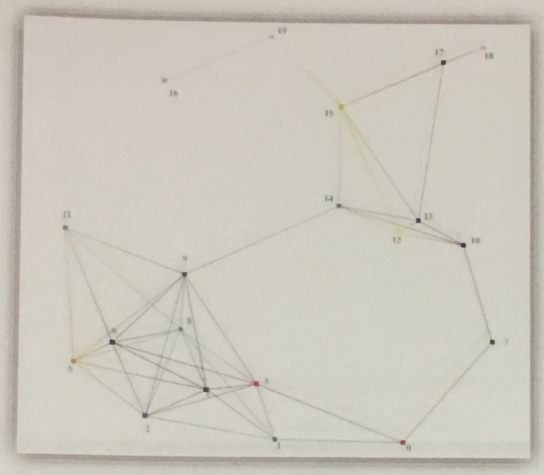

Fig 2. Random geometric graph with vertices and edges.

0	→	[1, 4, 7]
1	→	[0, 2, 3, 4, 8]
2	→	[1, 3, 4, 5, 6, 8, 9]
3	→	[1, 2, 4, 5, 6, 8, 9]
4	→	[0, 1, 2, 3, 6, 8, 9]
5	→	[2, 3, 6, 8, 9, 11]
6	→	[2, 3, 4, 5, 8, 9, 11]
7	→	[0, 10]
8	→	[1, 2, 3, 4, 5, 6, 9, 11]
9	→	[2, 3, 4, 5, 6, 8, 11, 14]
10	→	[7, 12, 13, 14]
11	→	[5, 6, 8, 9]
12	→	[10, 13, 14, 15]
13	→	[10, 12, 14, 15, 17]
14	→	[9, 10, 12, 13, 15]
15	→	[12, 13, 14, 17, 18,19]
16	→	[19]
17	→	[13, 15, 18]
18	→	[15, 17]
19	→	[15, 16]

Adjacency List

Now using the bucket sort we create a degree list.

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

Degree list

Now we apply the **smallest last vertex ordering algorithm** and pick the vertex with lowest degree as, that is, 16 in this case as show in degree list as there no vertices with degree 0. Now we go to the adjacent vertices of 16 using adjacency list and delete 16 from their adjacency lists and correspondingly we update the position of vertices adjacent to 16 in the degree list by moving them to the sub list which is above the one in which they are currently present since we are deleting 16 from them, after that we delete all the elements in the adjacency list of 16, after all these operations we delete 16 from the degree list. Now we add the vertex 16 to the smallestlastvertextorder list at the last. We add the degree of 16 before it is removed from degree list which is 1 in this case at the end of the degreewhenremove list.

Fig 3. Updated Random geometric graph after deleting First element in the degree list

0	→	[1, 4, 7]
1	→	[0, 2, 3, 4, 8]
2	→	[1, 3, 4, 5, 6, 8, 9]
3	→	[1, 2, 4, 5, 6, 8, 9]
4	→	[0, 1, 2, 3, 6, 8, 9]
5	→	[2, 3, 6, 8, 9, 11]
6	→	[2, 3, 4, 5, 8, 9, 11]
7	→	[0, 10]
8	→	[1, 2, 3, 4, 5, 6, 9, 11]
9	→	[2, 3, 4, 5, 6, 8, 11, 14]
10	→	[7, 12, 13, 14]
11	→	[5, 6, 8, 9]
12	→	[10, 13, 14, 15]
13	→	[10, 12, 14, 15, 17]
14	→	[9, 10, 12, 13, 15]
15	→	[12, 13, 14, 17, 18,19]
16	→	[ ]
17	→	[13, 15, 18]
18	→	[15, 17]
19	→	[15]

Updated Adjacency list

Degree	Vertices
0	[ ]
1	19
2	7, 18
3	0, 17
4	10,11,12
5	1, 13, 14
6	5, 15
7	2,3,4,6
8	8, 9

Updated Degree List

*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	16

Smallestlastvertextorder list

Fill from

*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	1

Degreewhenremove List

Now in the same way we go and pick and delete next minimum degree element from the updated degree list which is 19 and the process is repeated, after that we pick and delete 19, 7, 18, 0, 17, 10, 15, 12, 13, 14, 11, 1, 5. Now at this point we find a clique because all the remaining vertices in the degree list present only at a specific degree as shown below.

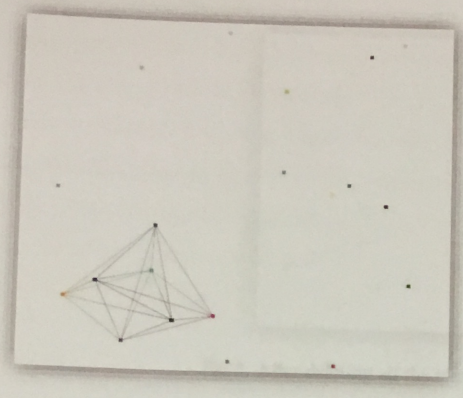

Fig 4. Clique subgraph obtained

0	→	[]
1	→	[]
2	→	[3, 4, 6, 8, 9]
3	→	[2, 4, 6, 8, 9]
4	→	[2, 3, 6, 8, 9]
5	→	[]
6	→	[2, 3, 4, 8, 9]
7	→	[]
8	→	[2, 3, 4,6, 9]
9	→	[2, 3, 4, 6, 8]
10	→	[]
11	→	[]
12	→	[]
13	→	[]
14	→	[]
15	→	[]
16	→	[ ]
17	→	[]
18	→	[]
19	→	[]

Adjacency list

Degree	Vertices
0	[]
1	[]
2	[]
3	[]
4	[]
5	4,2,3,6,8,9
6	[]
7	[]
8	[]

Degree list with Clique subset

*	*	*	*	*	*	5	1	11	14	12	12	15	10	17	0	18	7	19	16

Smallestlastvertextorder list

*	*	*	*	*	*	5	4	4	1	1	2	3	3	2	2	2	2	1	1

Degreewhenremove List

Now we can tell that the length of clique size is 6, that is, the number of vertices at this point forming a complete subgraph. Now in the same way we go and pick and delete next minimum degree element from the updated degree list which is 4 and the process is repeated, after that we pick and delete 2, 3, 6, 8, 9. With this all the elements of the degree list are completed so the output at this stage will be

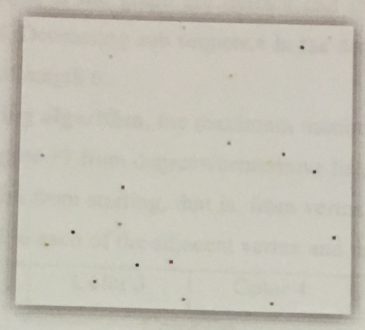

Fig 5. After deleting all the elements from degree list

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

Complete Smallestlastvertextorder list

Access from

0	1	2	3	4	5	5	4	4	1	1	2	3	3	2	2	2	2	1	1

Complete Degreewhenremove List

Now we will plot this degree variation when they are removed from the degree list along with original degree,

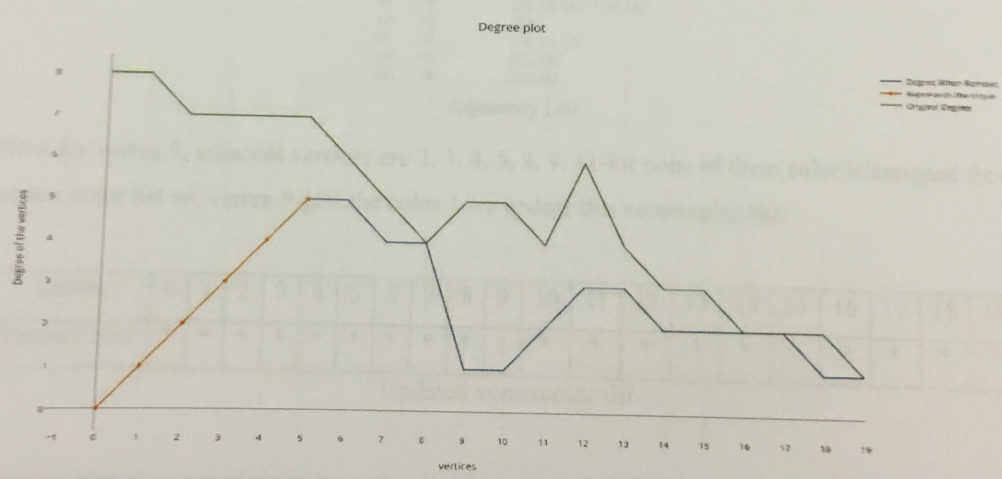

Fig 6. Degree plot

Now at this point all the edges in the graph are deleted and the Smallestlastvertextorder list, degreewhenremove list filled. Decreasing sub sequence in the degreewhenremove list represents the clique and therefore it is of length 6.

For the **Graph Coloring algorithm**, the maximum number of different colors we require is no more than maximum degree +1 from degreewhenremove list, here it is 6 colors. We traverse the Smallestlastvertextorder list from starting, that is, from vertex 9, we check the adjacency list of vertex 9 and color assigned to each of the adjacent vertex and then assign the color to vertex 9.

Color 1	Color 2	Color 3	Color 4	Color 5	Color 6

Color list

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

Output list from this algorithm

0	→	[1, 4, 7]
1	→	[0, 2, 3, 4, 8]
2	→	[1, 3, 4, 5, 6, 8, 9]
3	→	[1, 2, 4, 5, 6, 8, 9]
4	→	[0, 1, 2, 3, 6, 8, 9]
5	→	[2, 3, 6, 8, 9, 11]
6	→	[2, 3, 4, 5, 8, 9, 11]
7	→	[0, 10]
8	→	[1, 2, 3, 4, 5, 6, 9, 11]
9	→	[2, 3, 4, 5, 6, 8, 11, 14]
10	→	[7, 12, 13, 14]
11	→	[5, 6, 8, 9]
12	→	[10, 13, 14, 15]
13	→	[10, 12, 14, 15, 17]
14	→	[9, 10, 12, 13, 15]
15	→	[12, 13, 14, 17, 18,19]
16	→	[19]
17	→	[13, 15, 18]
18	→	[15, 17]
19	→	[15, 16]

Adjacency List

Now for vertex 9, adjacent vertices are 2, 3, 4, 5, 8, 9, 11 for none of them color is assigned from vertex color list so, vertex 9 gets the color 1 we update this vertexcolor list.

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

Updated vertexcolor list

Now we pick the next element from Smallestlastvertextorder list 8, and here in the adjacency list of 8 we have 9 which is already colored with color 1 so we color vertex 8 with color 2.

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

Updated vertexcolor list

We do the same process for the next elements 6, 3, 2,…., 16 from Smallestlastvertextorder list. Therefore the final vertexcolor list with the colors assigned to the vertices will look like

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

Below figure shows the colored vertices where no two adjacent vertices get same color.

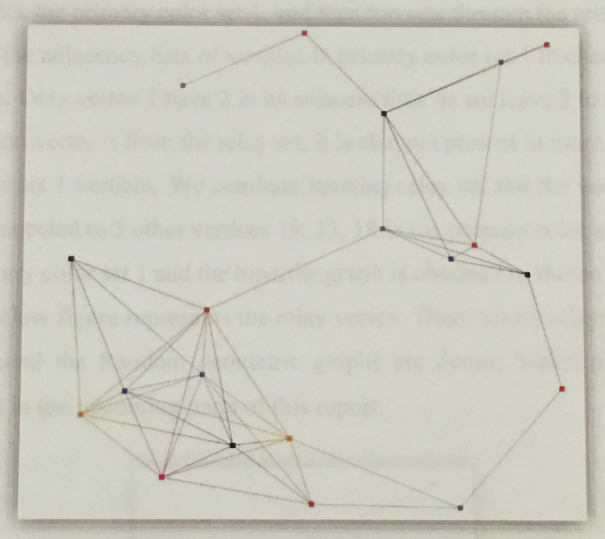

Fig 7. Colored Vertices

Now, we pick the 1/3rd color sets from the 6 color sets, that is, we pick first 2 color sets. And all other vertices in other color sets are considered as relay pools.

Primary color set 1	1,7,9,13,18,19
Primary color set 2	0,8,14,16,17
Relay set	2,3,4,5,6,10,11,12,15

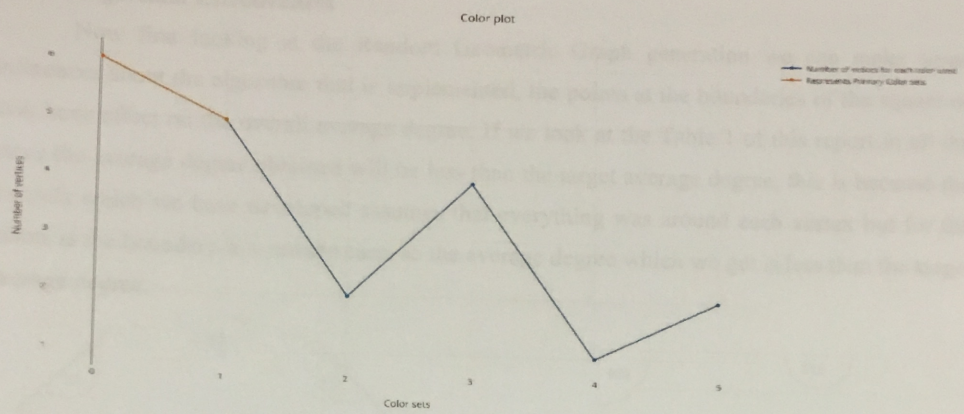

Fig 8. Color plot

Now first pick the primary color set 1, and then traverse through the relay set, first element is 2 now we check the adjacency lists of vertices in primary color set 1 to check if 2 is present in their adjacency lists. Only vertex 1 have 2 in its adjacent lists, so we leave 2 in the relay pool itself and now pick the next vertex 3 from the relay set, it is also not present in more than three adjacent list of primary color set 1 vertices. We continue iterating relay set and the vertex 15 satisfies the condition of it is connected to 3 other vertices 19, 13, 18 in the primary color set 1. Now this relay is added to the primary color set 1 and the bipartite graph is obtained as shown below. Vertex with black color in the below figure represents the relay vertex. There are no relays satisfying primary color set 2. In general the Random geometric graphs are denser which gives more bipartite subgraphs as shown in the upcoming parts of this report.

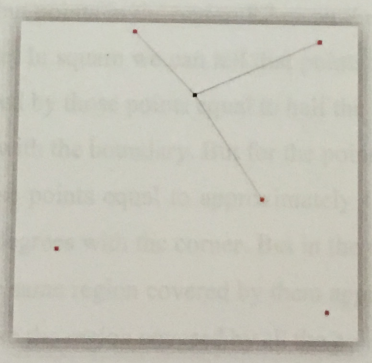

Fig 9. first bipartite subgraph.

2.1.5 Algorithm Effectiveness

Now first looking at the Random Geometric Graph generation we can make some inferences about the algorithm that is implemented, the points at the boundaries of the square or disk have effect on the overall average degree. If we look at the Table 1 of this report in all the cases the average degree obtained will be less than the target average degree, this is because the formula which we have developed assumes that everything was around each vertex but for the points at the boundary it is not the case, so the average degree which we get is less than the target average degree.

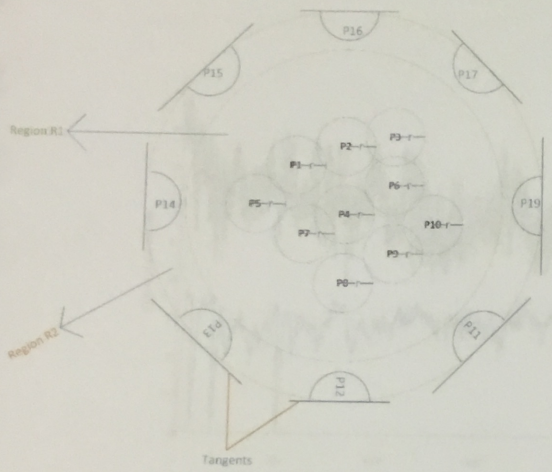

Fig 10. Boundary effect in Disk

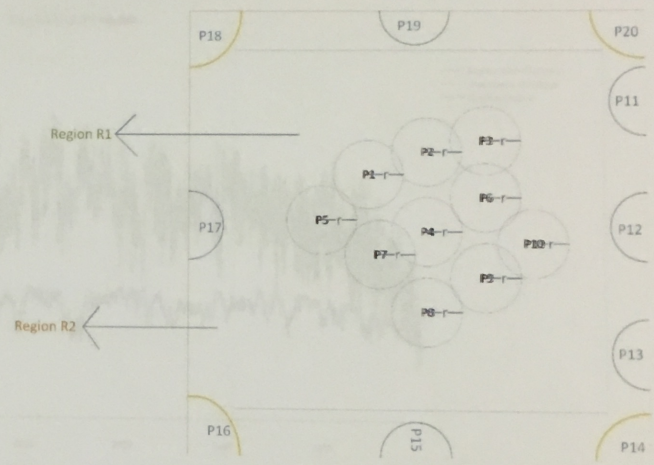

Fig 11. Boundary Effect in Square

From the above two figures we can tell that the points in the Region R1 have everything around each of them P1, P2, … , P10 but points in the region R2 or on the boundary they have boundary effects in both square and circle. In square we can tell that points in R2 like P11, P12, P13, P15, P17, P19 have the region covered by those points equal to half the region covered by the points in region R1, that is, 180 degrees with the boundary. But for the points in the corners P14, P16, P18, P20 the region covered by these points equal to approximately $1/4^{th}$ the region covered by the points in region R1, that is, 90 degrees with the corner. But in the disk almost all the points in the region R2 have the approximate same region covered by them approximately they make 180 with tangent to the disk at that point, so the region covered by all the points on the disk is half the region covered by each point in the region R1. So, the disk will be better applicable than square since there are no points like square where the coverage is $1/4^{th}$ which causes the artificial variation. So,

this makes the degree plot graph of the initial vertices that we pick from the circle more uniform since all those points have the degree of almost half the average degree but with slight variations since we pick the vertices with minimum degree. But the sphere is even better because every point anywhere in the sphere will have region around it, so almost the coverage of all the points in the sphere is almost the same. But the sphere is when we consider 3-Dimensions, if we want on a flat plane then the Disk will be the best one.

We can get many inferences from the degree plot, for example if we consider a degree graph plot for square of size N=4000 and d = 60 as shown in the below figure

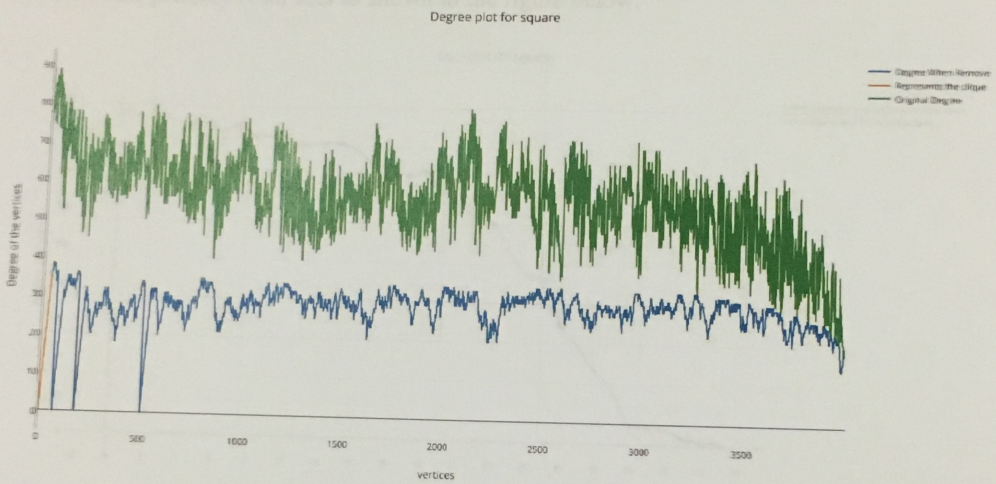

Fig 12. Example degree plot for square N= 4000 and d =60

Here initially the degree when remove curve and original degree curve starts from degree of approximately 14 (approximately 1\4th of the average degree 60) which is usually the degree of points at the corner and original degree most of the vertices have degree around 60 on the other side if we consider the degree when remove graph after removing vertices we get most of the vertices having degree around 30 because we remove vertices from the boundary towards the center of circle like a circular fashion which makes each vertex to have the region to pick from look like semi-circle which is equal half the average degree region to pick from. So, the average value of the degree when remove list curve is almost half of the average value of the original degree curve. If we look at the degree when remove curve and the original degree curves, original degree curve has a lot of variations when compared to that of the variations in the degree when remove curve because we cannot get the value greater than half of the average degree because we

always pick the minimum degree and the curve cannot get a very low because of the argument as follows, as the sum of all the values of degree when remove gives the number of edges and sum of all the values of the original degree gives twice the number of edges so the area under the degree when remove curve is half the area under the original degree curve. So, this makes degree when remove curve not to get low value since entire curve should average to get half the original degree curve. These two reasons make the degree when remove curve not to vary too much. This helps to understand how the color algorithm is working. The color plot tells us that the first 1/3rd sets have almost all the same number vertices than compared to other color sets with a total of 50% of vertices in the primary color sets as shown in the figure below.

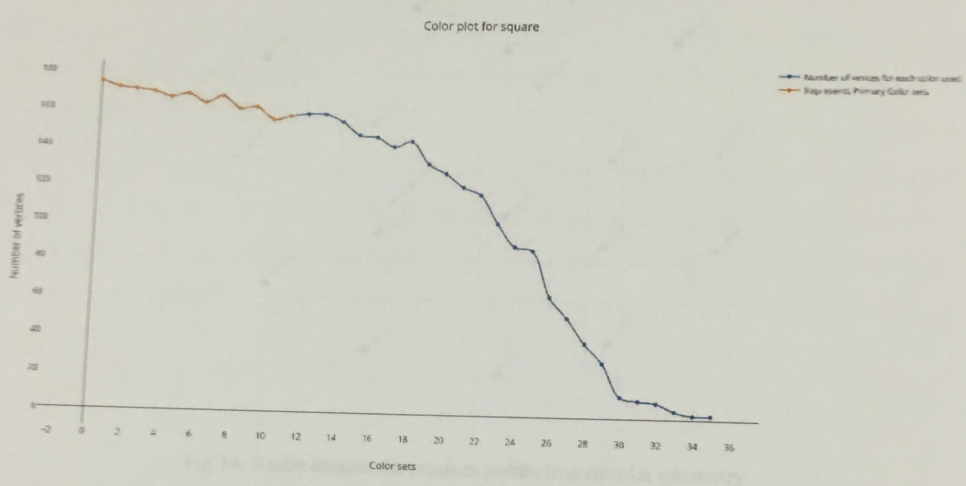

Fig 13. Example color plot for Square N = 4000 and d = 60

The coloring algorithm implemented here needs an extra overhead than compared to the greedy independent set selection algorithm to get the independent set, but the coloring algorithm gets many independent set at once instead of just one set, where the greedy independent set selection algorithm gets only one set and we need to rerun the whole algorithm to pick one more independent set. So, the coloring algorithm yields good efficiency which is one of the strengths of this algorithm.

2.1.6 Importance of these algorithms on the Wireless Sensor Networks

For this simulation each sensor is considered to be a point in one of the three geometries that are employed. The three geometries where the nodes can be deployed are the Square, Disc,