**WIRELESS SENSOR NETWORKS**

SWATI SHIRISH SAVAJI

47220684

FALL 2016

**Introduction and summary:**

This project is based on the implementation of real life Wireless sensor networks. Wireless sensor networks monitor conditions and pass their data co-operatively to a main location through the network. For passing this data the sensors communicate with each other with the help of set of frequencies. For using the same set of frequencies again and again, the limitation for wireless sensors is that they can communicate only within a certain distance. Because of this the sensors are spatially distributed in such a way that maximum area is covered with minimum number of sensors. One of the strongest feature of this project is that it has a linear time implementation. This project is divided in three parts.

Part 1: Distribution of sensors(N)

Part 2: Smallest last vertex ordering

Part 3: Maximal subgraph detection.

Part 1:

In this project a distribution of sensors over a unit square, a unit disk and a unit sphere is done. A square represents a town block, a disk represents a city and a sphere represents the entire globe. Input to the project consists of the number of sensors distributed over a certain region. We use pseudo random number generator for distribution of points. Random co-ordinates are generated using mathematical formulae. This is useful as we want them to be distributed over a unit square or disk or sphere only (as required); which means geometrically the co-ordinates should be inside these figures.

Part 2:

The number of sensors a sensor can communicate with are its neighbours and this is known as the average degree of a sensor. An adjacency list is created to maintain the list of neighbours. We use an array of hashmap to create the adjacency list. To identify the neighbours (points within a specified distance) we have a function neighbourhood distance (R) of average degree and all points.

The entire area of (square/sphere/disc) is divided into cells to avoid unnecessary comparisons to determine neighbours. In this way we do not have to compare the points which are far away from each other. The area of cell is defined as a function of neighborhood distance. If the Euclidian distance between points is less than or equal to the radius of neighbourhood distance they are neighbors. We put them in adjacency list. Also every sensor outside the minimum distance will have a different set of frequencies. This is illustrated here by graph coloring using smallest last vertex ordering. We use smallest last vertex ordering algorithm on the graph obtained from the adjacency list. A doubly linked list is used for implementation. We store the references to every element of linked list in an array for faster deletion and improving the runtime. We obtain a vertex order from this algorithm which is useful in assigning colors.

Part 3:

Now we have to find the color distribution and the bipartite subgraph. Bipartite subgraph is the one with the largest backbone considering with the two largest colors (colors spanning maximum vertices). Adjacency list and colors assigned from previous parts are used for this.

**Summary Table:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | R | N | Min degree | Average Degree | Maximum Degree | Distribution | Realized average deg | Edges in largest subg |
| 1 | 0.107331263 | 1000 | 8 | 32 | 51 | SQUARE | 32 | 137 |
| 2 | 0.075894664 | 4000 | 16 | 64 | 100 | SQUARE | 67 | 297 |
| 3 | 0.037947332 | 16000 | 16 | 64 | 107 | SQUARE | 69 | 1130 |
| 4 | 0.018973666 | 64000 | 16 | 64 | 107 | SQUARE | 71 | 4501 |
| 5 | 0.026832816 | 64000 | 38 | 128 | 190 | SQUARE | 141 | 2436 |
| 6 | 0.137875306 | 4000 | 14 | 64 | 101 | DISK | 65 | 304 |
| 7 | 0.194985128 | 4000 | 22 | 128 | 207 | DISK | 127 | 170 |

**Programming environment description:**

A windows 8.1 64 bit laptop with Intel(R) Core i5 processor @ 2.20GHz and 8GB RAM (DDR3) was used to develop this project. Programming language used is Java SE. Processing is used for visualizations.

**References:**

[1] D.W. Matula, Wireless Sensor Network Project, www.lyle.smu.edu/cse/7350/, 2014

[2] <http://lyle.smu.edu/~zizhenc/file/Wireless%20Sensor%20Network.pdf>

[3] <https://www.youtube.com/watch?v=cMh-VNRoLWE>

**Reduction to Practice:**

1. **Data Structure Design:**

Part 1:

To compute the x, y co-ordinates of each point we use an RPGenerator class; this takes as input the number of nodes and generates random co-ordinates. Every node is known as a point has following features: An integer id for uniqueness, 2 double precision co-ordinates (x and y) and floating point circular co-ordinates (r and theta) used to derive their x, y co-ordinates.

The sine and cosine values for angles 0 to 360 are used to calculate the Cartesian co-ordinates. But if they are to be computed as and when required then the load increases.

So they are computed and stored in a .csv file and can be extracted when needed.

Part 2:

An array of hashmaps is used to store the adjacency list. Each vertex stores the vertex id of its neighbour as a key in the hashmap for easy retrieval. We have to store vertices to compare them. An array of ArrayLists is used for this. The adjacency list generated is given as input to the smallest last vertex ordering algorithm. Node data structure is used to indicate vertices which improves the time complexity. Our Doubly Linked List uses references which are in Node. Node also keeps track of degree by using a variable integer degree. It also has an integer id for vertex id.

Part 3:

In this part we find four largest colors. We have 4C2 combinations available to select bipartite subgraph. We select the ones with maximum edges.

1. **Algorithm Description:**

Co-ordinates for disk and square are generated and stored in an array of vertices N. N is the total number of vertices. For generating the co-ordinates of a square we use a pseudo-random number generator.

Formula for generating co-ordinates for disk:

x = sqrt(R)\* cos(theta)

y = sqrt (R) \* sine(theta)

(r and theta are randomly generated.)

To calculate the neighbourhood radius following formula is used:   
sqrt(avg/N) \* correction, where avg is the average degree. Correction is used for accuracy. This is used to obtain the number of cells. For square x and y co-ordinates of each vertex are assigned cells. For disk, R and theta of each vertex are assigned cells. To check if the vertices are adjacent or not, Euclidean distance of every vertex from one cell to every vertex in another is computed. After this step an adjacency list is created as input for graph coloring part. For graph coloring we use smallest last vertex ordering algorithm. Our customized Linked list used for this implementation has an array for keeping track of references (nodes). Next we list the vertices according to their degree. We now use a structure which has bins to store the vertices with the same degree. Traversing this structure can get us the non-empty list of smallest degree. The first node is now assigned to vertex order. Now for graph coloring we have to iterate through vertex order and assign color to each vertex. Since the color of the vertex and its neighbour should not match we should check the colors of the neighbours first. For this we maintain an array to store the colors of the neigbours. We can iterate through this array to find an unused color.

1. **Algorithm Engineering:**

Part 1: This part has two main components: First is generating pseudo random numbers and then calculating co-ordinates from them mathematically. Time required for both of them is asymptotically constant. So we have linear time complexity O(N) for all N vertices.

Part 2: In this part vertices are assigned proper cells. Now there are two cases for calculating complexity: one is unit square and other is unit disk. For a square there are 1/R^2 cells. Here R is sqrt(avg/N). A cell undergoes 4 comparisons at the most. Assuming that each cell has average number of points each vertex can get compared with other 4\*average vertices. Average is calculated as (total vertices)\* R^2. We will have a complexity of O(N^2R). For disk, number of cells become 8/R. So the number of cells each cell is compared with reduces to utmost 3. So each vertex is compared with 3\*average no of vertices. Hence time complexity for disk also remains same as that of square.

But, Total degree = N \* average which is 2 \* |E|

Thus N \* N \* R^2 = 2|E| (since average = N \* R^2)

Thus the algorithm is linear time algorithm.

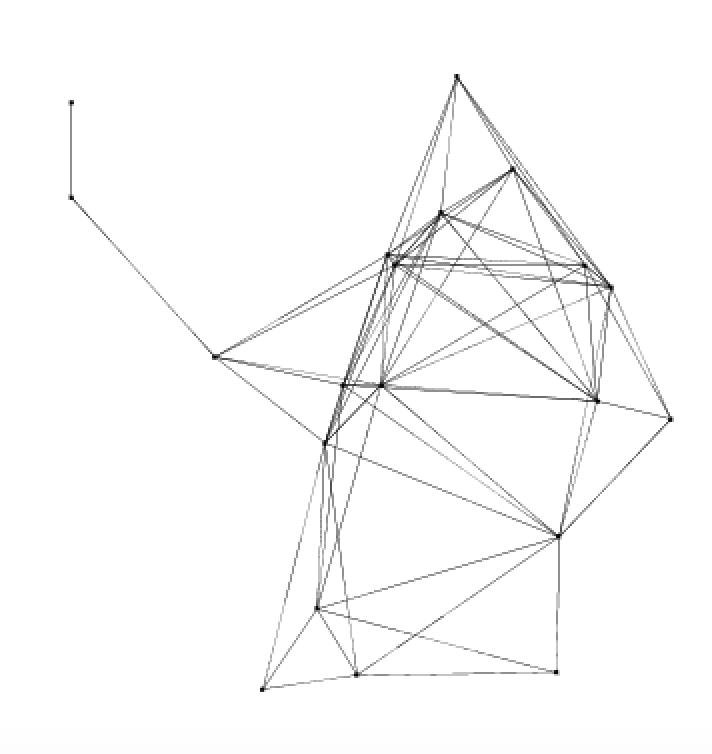
From [1] we know that the smallest last vertex ordering has a time complexity of O(|V| + 2|E|). In our implementation, vertex coloring is done in two iterations hence complexity becomes: O(4|E|).

Part 3: To generate backbone bipartite subgraph in part three we need to obtain colors of all vertices. As described earlier each vertex with same degree goes in same bin. This requires O(N) time. After this, we can extract colors with their corresponding vertices. In O(C\_max) we copy above data structure to obtain the four largest colors and their vertices. We can now create a adjacency list for bipartite subgraphs. To obtain the backbone a DFS traversal of that list is done. Now, Time complexity for Traversal of graph is: O(|V| + |E|) and of DFS is: O(V|s+ |E|s). (bipartite graph:s)

Thus linear time complexity is achieved for this part also.

|  |
| --- |
|  |

1. **Verification Walkthrough:**



We divide the region in minimum 4 to maximum 8 cells for this. Now the radius R is used to create the adjacency list below.

 Adjacency List indexing:

Column 1 : vertex id

Remaining: Neighbor ids

**Adjacency List for the above graph:**

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

This List goes as input to smallest last vertex ordering algorithm. From this algorithm we get the minimum colors required to color the entire graph (all vertices)

Backbone bipartite subgraph:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 16 | 19 | 10 | 5 | 8 | 14 | 9 | 13 |

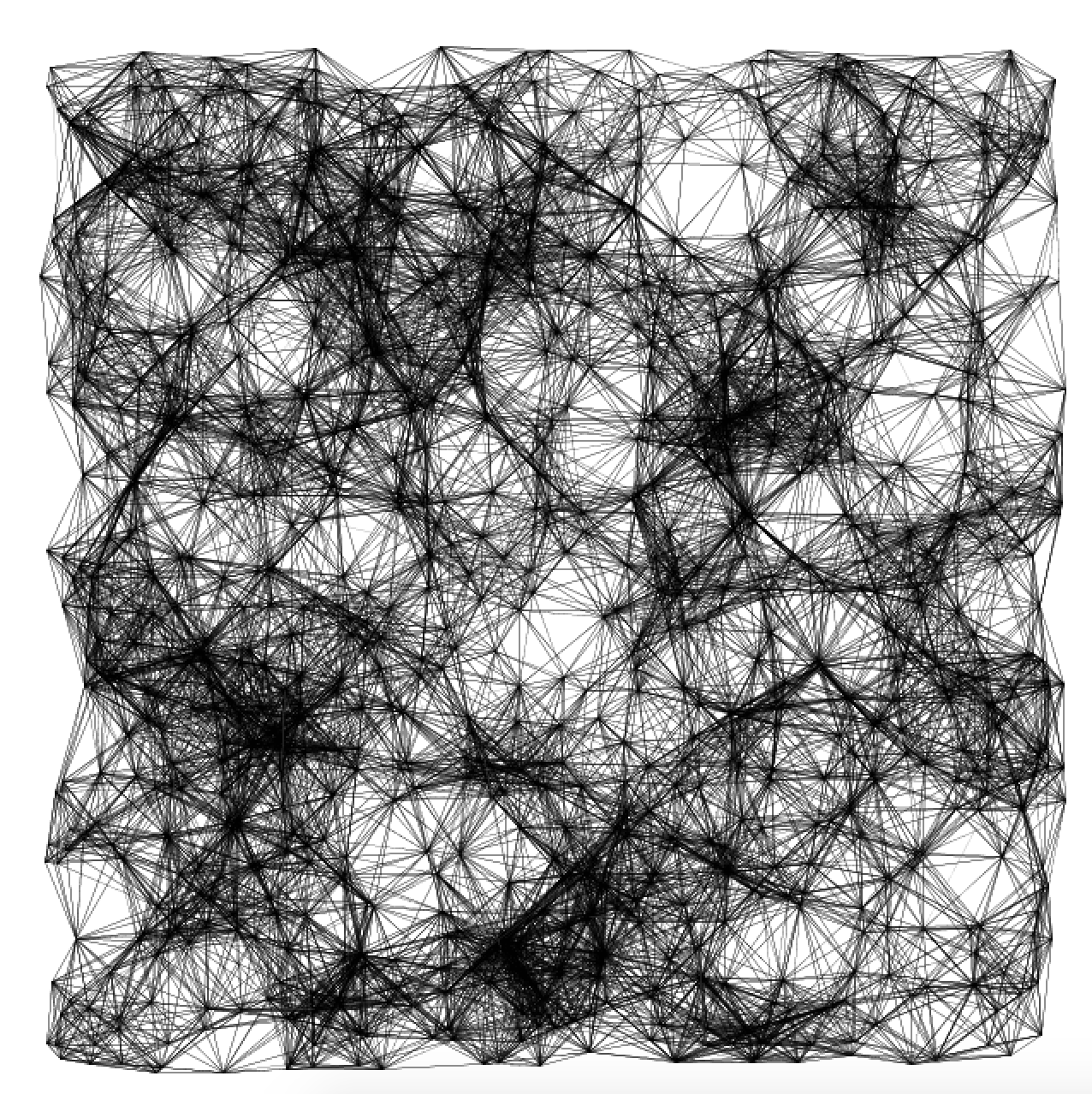
Colors used:

|  |  |
| --- | --- |
| Color | Number of vertices |
| 1 | 5 |
| 2 | 4 |
| 3 | 4 |
| 4 | 2 |
| 5 | 2 |
| 6 | 1 |
| 7 | 1 |
| 8 | 1 |

Orignal Degrees and Degrees Deleted.

|  |  |
| --- | --- |
| Original Degree | Degree when deleted |
| 9 | 5 |
| 11 | 6 |
| 6 | 6 |
| 7 | 4 |
| 10 | 7 |
| 4 | 4 |
| 12 | 3 |
| 1 | 1 |
| 6 | 5 |
| 10 | 2 |
| 3 | 3 |
| 10 | 5 |
| 10 | 1 |
| 8 | 0 |
| 2 | 1 |
| 3 | 3 |
| 9 | 5 |
| 11 | 5 |
| 9 | 4 |
| 5 | 3 |

Uniform square:



Uniform Unit Disk:

