Algorithms Engineering Wireless Sensor Network

Sam Hunter

December 2016

Contents

1 Executive Summary

1.1 Programming Environment

For this project I developed the source code, generated the graph and created the graphics on an early 2014 macbook air. This particular macbook air has 8 gb of memory, a 1.7 GHz Intel Core i7 (four cores) and an integrated graphics card.

The source code for this project was written in python, throughout four separate modules. In addition to the language-included packages, I used numpy to generate the random numbers and plotly to create the graphical elements for this paper. Ultimately all of the report, figures and numbers were compiled in a latex document.

1.2 Summary Tables

#	Nodes	Average Degree	Topology	Radius	Edges	Min Degree	Average Degree	Max Degree
1	1000	32	Square	0.1	14798	9	29.596	49
2	4000	64	Square	0.071	120471	18	60.2355	86
3	16000	64	Square	0.035	496583	13	62.0728	92
4	64000	64	Square	0.0178	2015571	15	62.9865	105
5	64000	128	Square	0.0252	4011840	42	125.37	176
6	4000	64	Disk	0.126	99826	12	49.9132	85
7	4000	128	Disk	0.178	198704	28	99.3522	156

Table 1: Random Graph Summary Table

Number	Max Degree When Deleted	Colors	Max Color Set Size
1	22	32	84
2	37	55	172
3	39	56	677
4	43	58	2622
5	73	100	1411
6	61	48	147
7	107	89	76

Table 2: Coloring Summary Table

Number	Average Nodes	Average Edges	Average Degrees	Average Domination Ratio
1	128.16	147.66	2.3	0.128
2	369.5	442.5	2.395	0.092
3	1564.16	1979.66	2.531	0.097
4	6239	7960.416	2.55	0.0974
5	3631	4942	2.72	0.0567
6	87.5	100.416	2.295	0.0218
7	52.33	56.083	2.143	0.0131

Table 3: Backbone Summary Table

2 Reduction to Practice

2.1 Data Structure Design

In order to efficiently do each part of this project, data structures had to be effectively implemented. A large part of being able to process large volumes of data is storing them in a way that they are quickly accessible in the necessary format without adding too much bloat. The biggest challenge that come with designing effective data structures is creating structures that optimize for handling the needs of the solution and minimizing the size of the structures. In this project there are four primary parts, the construction of the random graph, finding the smallest last vertex ordering, coloring the graph based on this ordering and then creating the bipartite backbone from the sets of the colors.

Random Graph Construction Creating a randomly distributed uniform graph across the unit circle and unit square was the first task in this project. Initially the points are randomly generated and stored in a list. The list acts as a container before the points are separated into a two dimensional array of buckets, constructed as a list of lists. Once the buckets are filled with points in their respective spatial areas, they are ready to be connected. Using a radius determined as a function of the number of points and average degree points are connected to other points in surrounding buckets if they fall within the threshold of the radius. In order to store these points an adjacency list is created. The adjacency list is the primary data structure employed in the part of the project. The structure itself is a dictionary object that uses the point coordinates as a key and another dictionary structure as the value. This value contains data associated with the point (connected points, degree, etc.). This adjacency structure is then used for the next part, finding the smallest last vertex.

Smallest Last Vertex Ordering Using the adjacency list from the random graph construction as input, the smallest last vertex ordering section of the project harnesses an array of data structures to accomplish its task. To begin, all of the points are dumped into buckets according to their degree. Additionally, an auxiliary dictionary structure is populated with each point and its degree. The buckets are then iterated over to find the first non empty bucket from which to remove the smallest degree vertex from. Once removed from

the auxiliary dictionary, all of the neighbors of this node are shifted down in the buckets corresponding to their degree. This process is repeated as the smallest last vertex ordering appears.

Graph Coloring Once the smallest last vertex ordering is established, the ordering is used to assign colors such that no node of color x is connected to another node of the same color x. The structures needed to do this include an expansion of the adjacency list to include data about the color of the node and a list to keep track of colors. The list of colors is necessary to keep track of how many colors exist and so that another color can be added if necessary. By iterating over the vertices in order of the smallest last vertex ordering, a color can be assigned based on which colors are already assigned to its neighbors. Ultimately this part does not result in a new data structure, but rather a modification of the previous adjacency list structure.

Backbone Construction The last portion of the project involves creating a backbone from the combinations of a bipartite set of the largest four color classes. This part used a list to aggregate all points that are apart of a particular set of two color classes. Iterating over this list, a vertex is added to a stack structure temporarily. The neighbors in the bipartite set of the node on the stack are also added to the stack. In this way all neighbors are checked and each component is generated. These structures are the primary components of this section and allow for efficient running of the program.

2.2 Algorithm Analysis

While the data structures allow the algorithm to function, the actual operations and algorithm design enable the solution to this project. In order to assess the project as a whole, the algorithm used must be assessed critically to determine it's level of efficiency. Because this project consists of many parts, each section will be assessed individually.

Random Graph Construction The first step in random graph construction was generating the points for the graph itself. The next steps involve the actual algorithm for creating the graph from the nodes. To begin a matrix of buckets was created, representing a segment of the unit square or unit circle. In one scan over the adjacency structure, each point was placed in a bucket corresponding to its location. This operation took $\Theta(n)$ time. After this the adjacency list is iterated over again, this time with the goal of attaching adjacent points such that it will have the average degree inputted. To do this, each point in the surrounding buckets and same bucket are concatenated into a list. This list is then iterated over and compared to the desired radius to see if an edge should be created. This whole step requires one iteration over the list plus iteration over a small sub sample of the overall adjacency structure for each point. This can be simplified down to another $\Theta(n)$. This results in an overall running time of approximately $\Theta(2n)$ which is can be simplified further to $\Theta(n)$ where n is the number of vertices.

Smallest Last Vertex Ordering Using the adjacency structure from the previous part of the project this part aims to create an ordering of the nodes that will be used to color the graph later. The first step for this is iterating once over the adjacency list and placing each node into a bucket corresponding to its degree. This is a $\Theta(n)$ operation. The next step involves another scan over the adjacency structure in order to create a dictionary with each node and its corresponding degree. Again, this is a $\Theta(n)$ operation. The bulk of the work is involved in the next step. At this point a loop iterates n times (where n is the number of nodes). In each iteration the smallest degree node is found by getting a node from the smallest non empty bucket. This operation is $\Theta(b)$ where b is the length of the buckets (a negligible number relative to the number of nodes). After finding the smallest node x, x's neighbors need to have their degrees bumped down by one to reflect the removal of x from the graph. This requires iterating over the neighbors of x, finding the degree of the neighbor node using the auxiliary dictionary and then moving the point in the bucket structure down one bucket as well as decrementing its degree in the auxiliary dictionary. This process requires $\Theta(E)$ time, where E is the number of edges connected to the vertex. Therefore each iteration runs in $\Theta(E + b)$ time. Overall this makes the whole process of finding the smallest last vertex ordering $\Theta(V + 2E)$ because the algorithm ultimately iterates over each edge twice, once for each node its connected to. Because 2E is a constant times a variable the whole process' running time can be simplified to $\Theta(V + E)$.

Graph Coloring Using the smallest last vertex ordering, the graph coloring portion of the project assigns colors classes to each vertex such that no vertex of color a has a neighboring node of color a. To do this, the ordering is iterated over, requiring $\Theta(n)$ time. In each iteration node x's neighbors are iterated over to check their coloring. Vertex x is then assigned the first color class that isn't claimed by one of its neighbors. Should all the colors be used by its neighbors a new color is created and added to the array of existing colors. The operation inside each iteration takes $\Theta(e)$ where e is the number of neighbors of vertex x. Overall, this portion of the project takes $\Theta(V + 2E)$ because each edge is checked for each node either end. Once again this can be simplified to $\Theta(V + E)$.

Backbone Selection The last part of the project uses the six pair combinations of the four largest color classes and constructs bipartite backbones from them. For this section, the running time efficiency of one backbone construction will be analyzed. Given two color sets combined into one list, this algorithm is tasked with finding the largest component, the backbone. To accomplish this, the first element from the combined list of the two color classes was put on a stack. The first element would immediately be popped from the stack and any neighbors that also belonged to either of the color classes was put on the stack. A loop would iterate over each element of the stack popping it off first and then adding any neighbors in the bipartite set to the stack. When the stack finally was emptied the component would be completely searched and the next unvisited vertex in the bipartite set would be put on the bottom of the stack to find all of the vertices in its component. This process involved iterating over each of the vertices once and adding and removing them from the stack. For this reason the run time efficiency of the backbone selection is equivalent to $\Theta(V)$.

Overall Runtime Efficiency As a whole the run time efficiency of this project is simply a sum of its parts. If you sum all of them together you get an overall run time of $\Theta(4V + 2E)$. Similar to the steps taken above this can be simplified to the most dominant drivers of the run time, $\Theta(V + E)$, which ultimately represents a linear run time.

2.3 Verification Walkthrough

In order to effectively walk the reader through the process of this project each of the four parts are broken up into sections in which the process is described.

2.3.1 Random Graph Construction

The first part of the project entails creating a random graph with an arbitrary number of nodes uniformly distributed across the unit square or circle. This has bee done and is shown in figure 1. Because connecting all of the nodes would add too much noise to the figure, they have been omitted in the graph, however, in the structure they are connected.

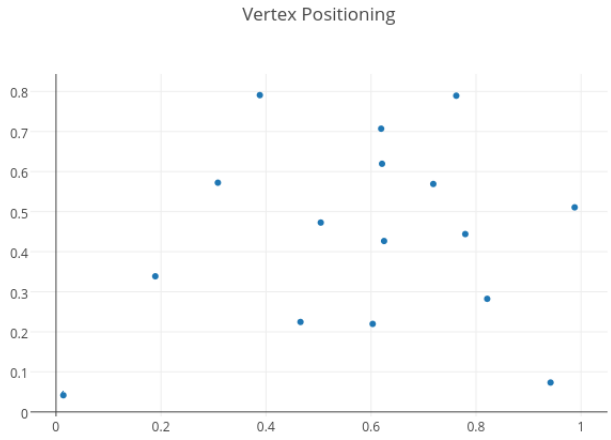


Figure 1: Vertex Scatter Plot

2.3.2 Smallest Last Vertex Ordering

The second part of the project involves finding a smallest last vertex ordering for the graph. In each iteration over all of the nodes, the vertex with the smallest degree is found and removed from the graph. The rest of the nodes' degrees are then updated before the process is repeated. This continues until there are no vertices left. In order to visualize how the degree of the nodes change during this step, figure 2 graphs both the original and deleted degrees as the smallest last vertex ordering happens.

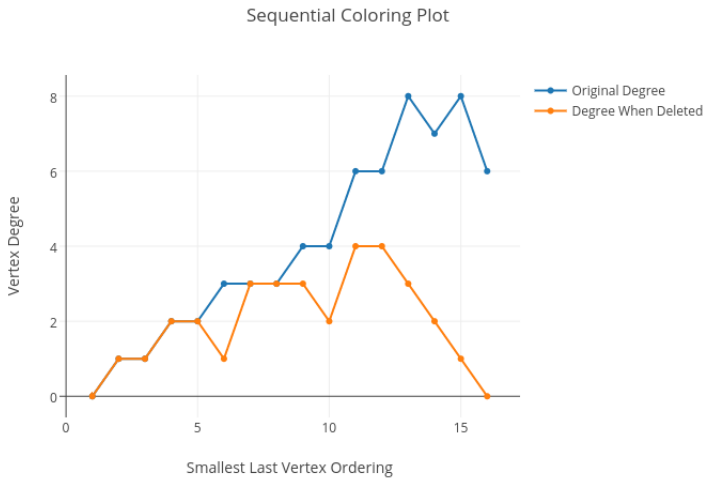


Figure 2: Sequential Coloring Plot

2.3.3 Graph Coloring

The third step in the project involves assigning a color to each node such that none of its neighbors are assigned the same color. This process results in an arbitrary number of color classes that can be combined to form a bipartite set. In this instance, 7 color classes were created. The distribution of these color classes is displayed in figure 3.

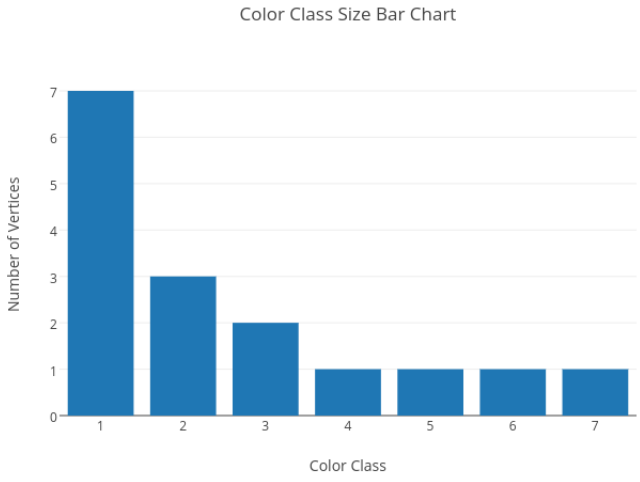


Figure 3: Color Class Size Plot

2.4 Backbone Creation

The last step allows a wireless sensor network to be created by building a backbone from the color classes. In particular, the four largest color classes are taken and each combination of the four is used to generate a backbone. A backbone is the largest component in the graph formed by the set of two colors in the combination. Figure 4 displays the most comprehensive backbone generated. This backbone happens to be very uninteresting, but the vertex coloring demonstrates that no edge connects two points of the same color.

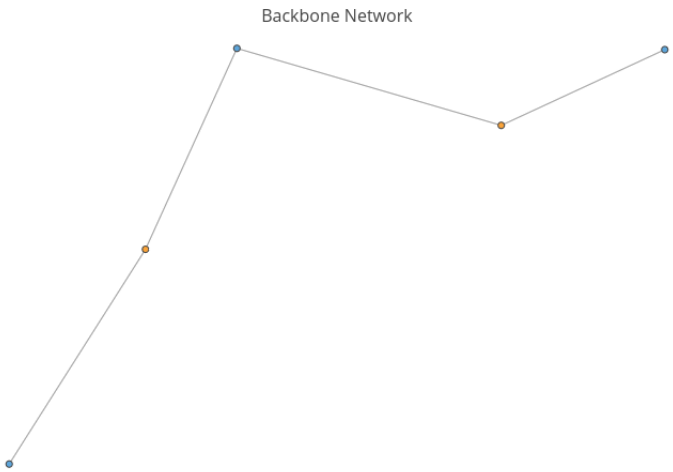


Figure 4: Wireless Sensor Network Plot

2.5 Algorithm Engineering

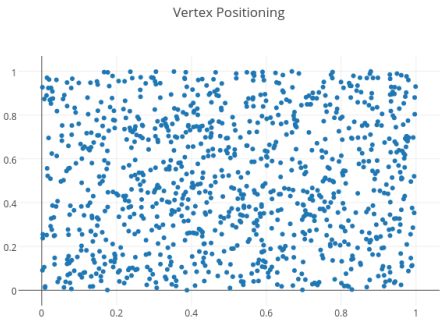
This project required a lot of outside-the-box thinking in order to bring the runtime down to linear. Straightforward thinking would have yielded a runtime that would not allow 64k nodes to be processed as an input. Minimizing the time the algorithm took to run involved effectively using data structures as needed, while optimizing so that not too many are used. Most data structures took the form of a dictionary or list, two built in python structures. Both provide advantages in terms of maintaining order and look up times. For items that needed to be found quickly, dictionaries provided constant time look up over linear time searching through unordered lists. On the other hand, lists maintained order that was necessary especially for smallest last vertex ordering.

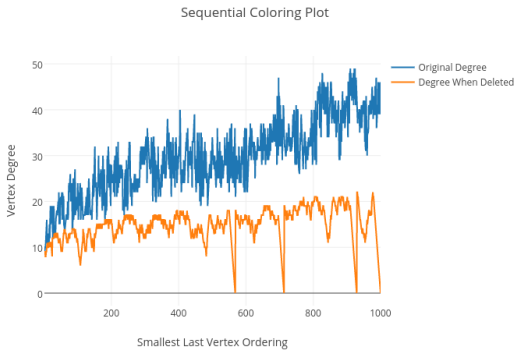
In combination with the right data structures, the next biggest challenge for this problem was setting up the data so that it could be effectively used in the algorithms. For instance, when creating the smallest last vertex ordering putting all of the vertices in buckets according to their degree only took one iteration over the list but saved time during the algorithm that would have come at the expense of making the algorithm quadratic. Furthermore, the dictionary containing the degrees that is also populated at the beginning of the algorithm saves a similar amount of time just by storing the node and its degree as a key value pair.

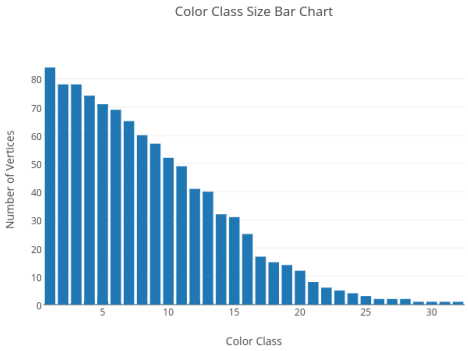
This project used a lot of unconventional thinking to effectively reduce the run time. By harnessing the right data structures and breaking the problem into many sub problems, this project emphasized critical thinking and outside-the-box thinking. Patterns can be transferred this project to other solutions to effectively reduce their run time as well. Ultimately the pursuit of a linear efficiency pushed the author to step back and reframe the conceptual problem many times and shed new perspective on using algorithms to effectively solve any problem.

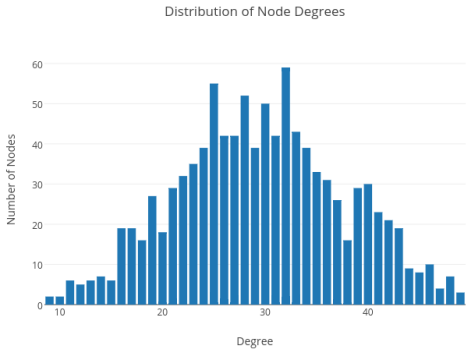
3 Result Summary

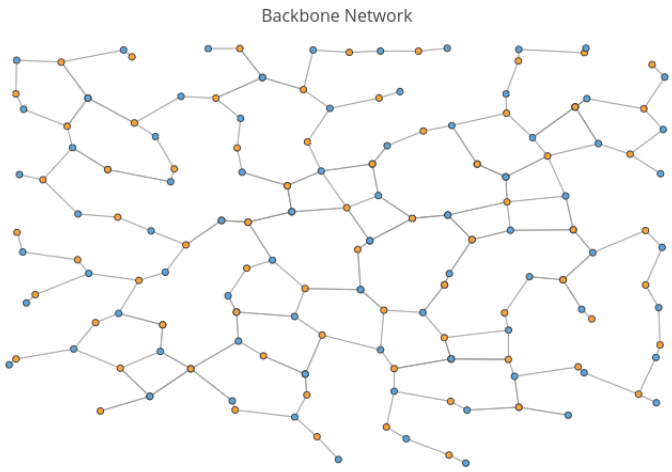
3.1 Number 1





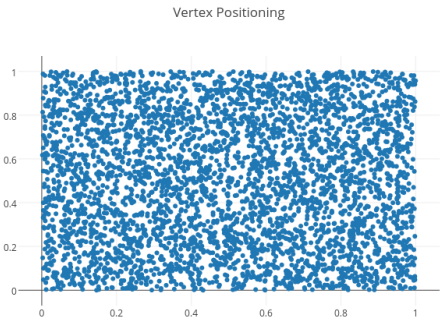


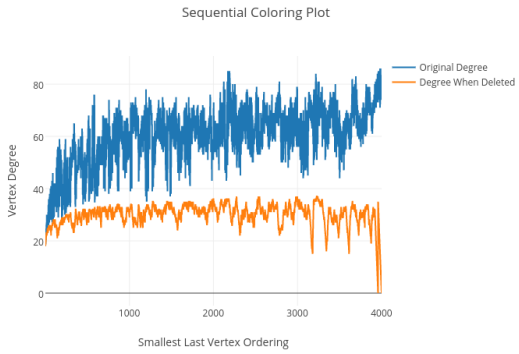


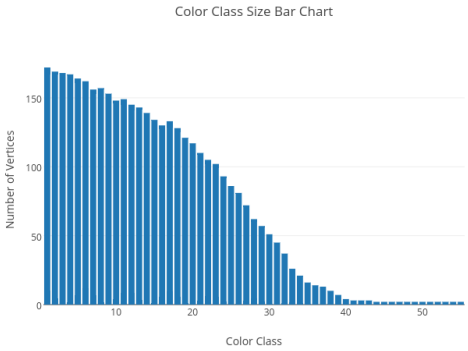


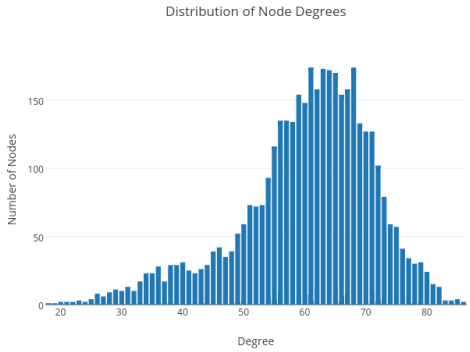
#	Nodes	Average Degree	Topology	Edges	Average Degree
7	4000	128	Disk	198704	99.3522

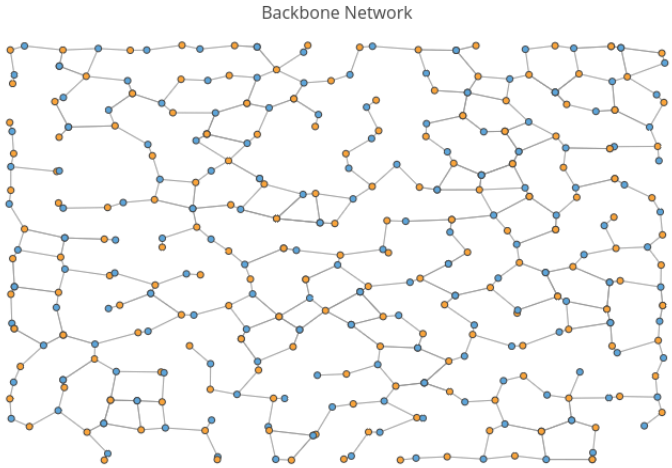
3.2 Number 2





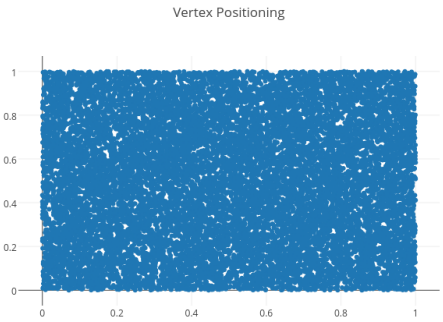


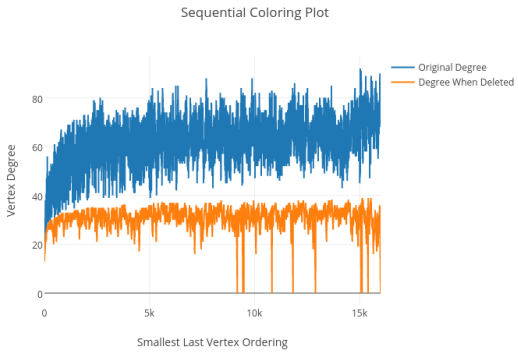


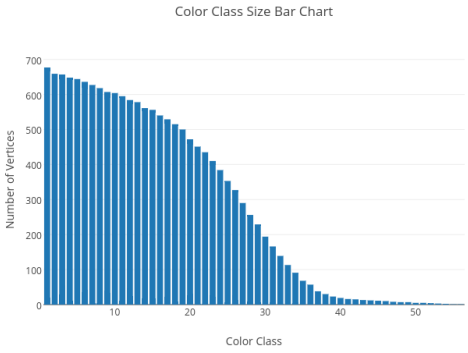


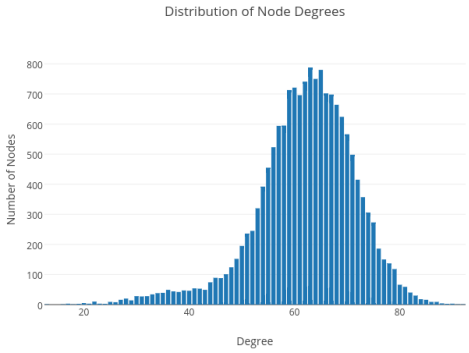
#	Nodes	Average Degree	Topology	Edges	Average Degree
2	4000	64	Square	120471	60.2355

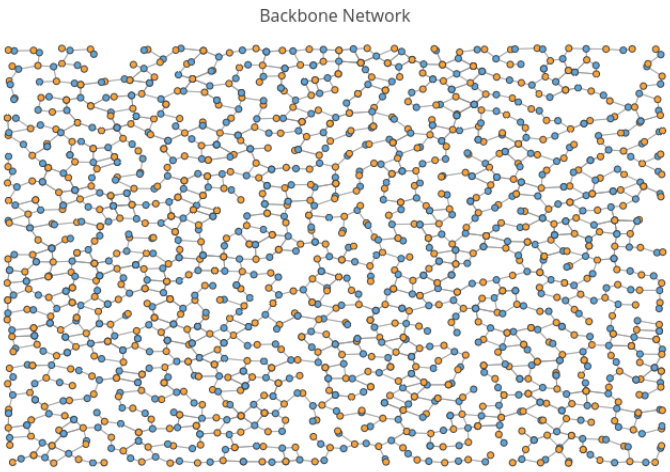
3.3 Number 3





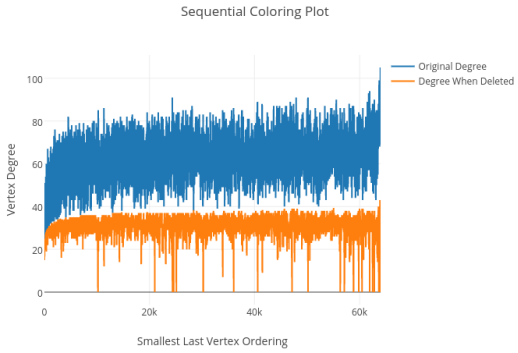


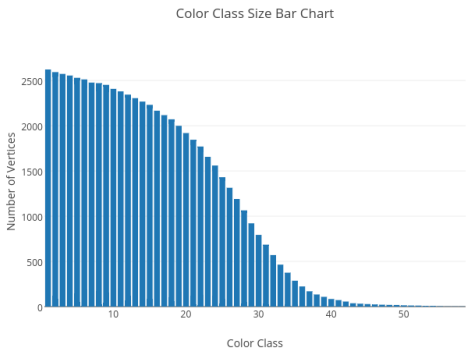


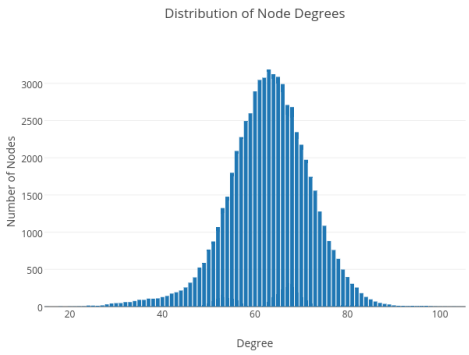


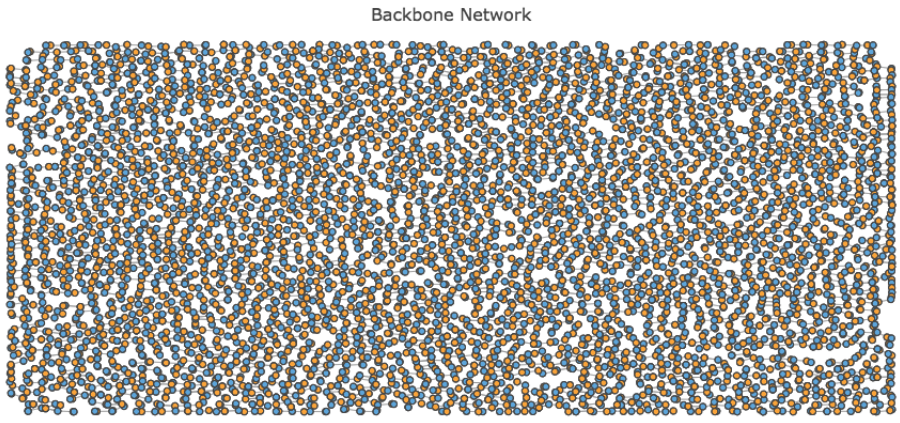
#	Nodes	Average Degree	Topology	Edges	Average Degree
3	16000	64	Square	496583	62.0728

3.4 Number 4



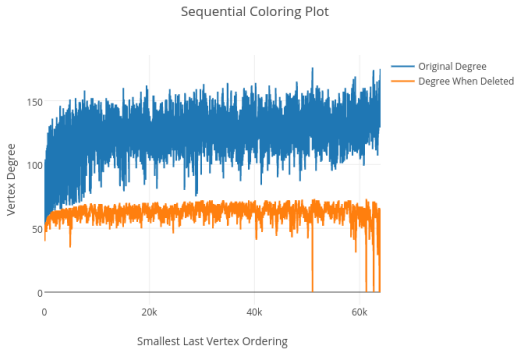


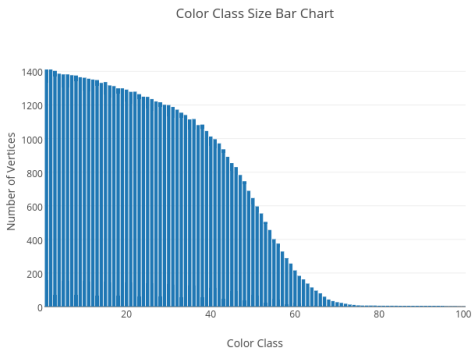


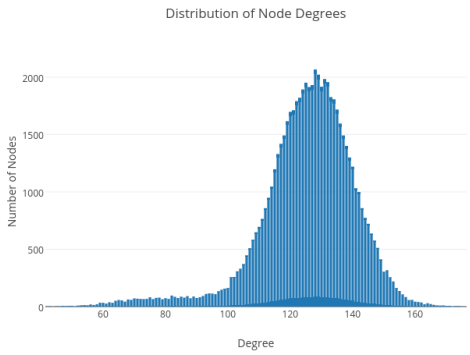


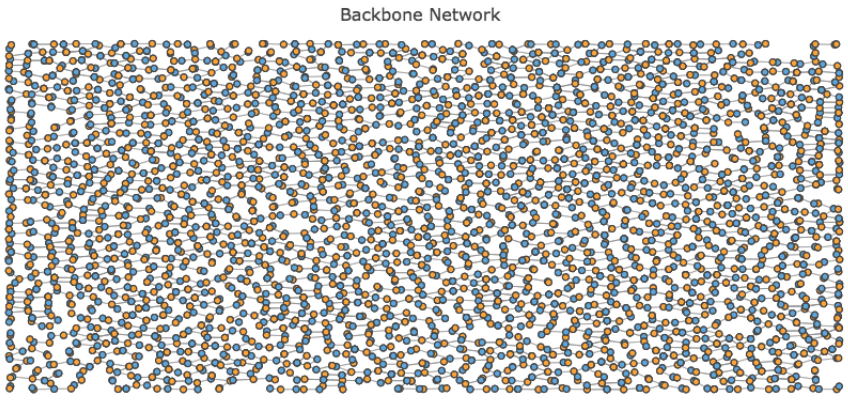
#	Nodes	Average Degree	Topology	Edges	Average Degree
4	64000	64	Square	2015571	62.9865

3.5 Number 5



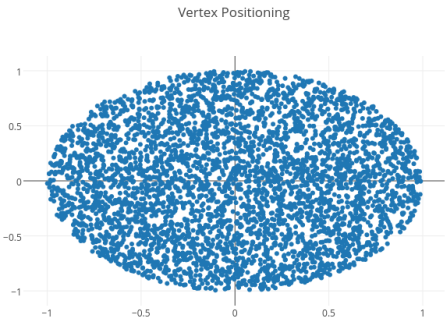


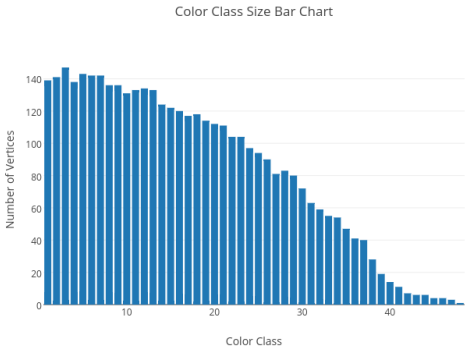


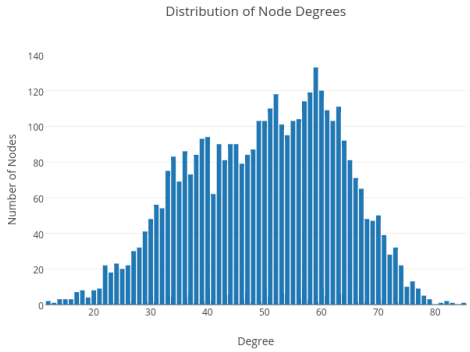


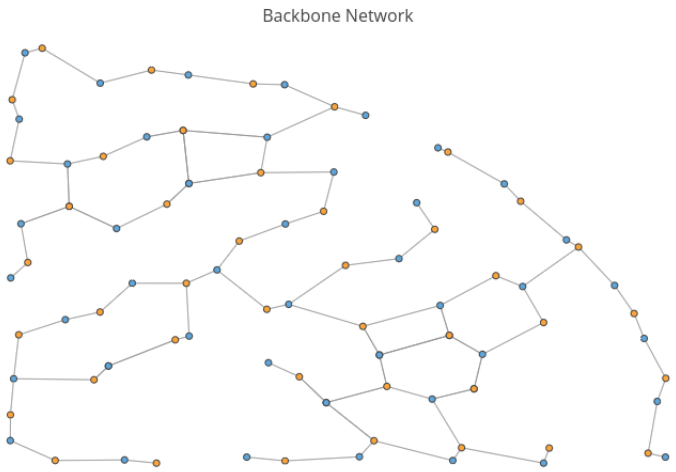
#	Nodes	Average Degree	Topology	Edges	Average Degree
5	64000	128	Square	4011840	125.37

3.6 Number 6



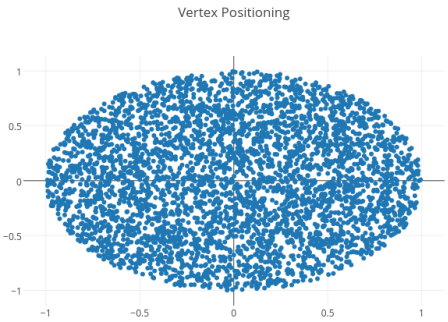


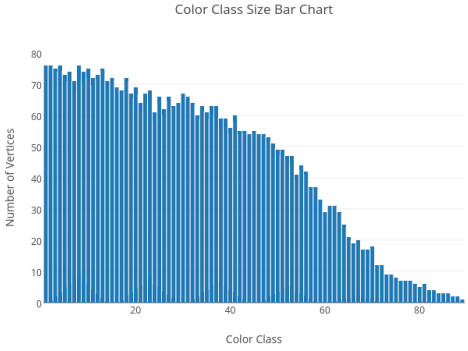


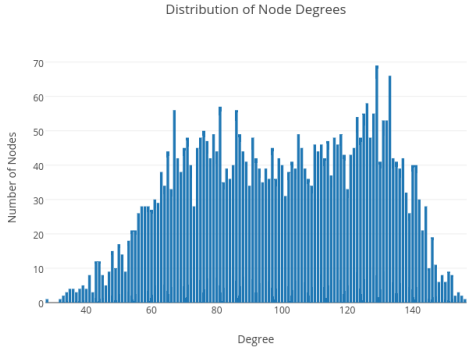


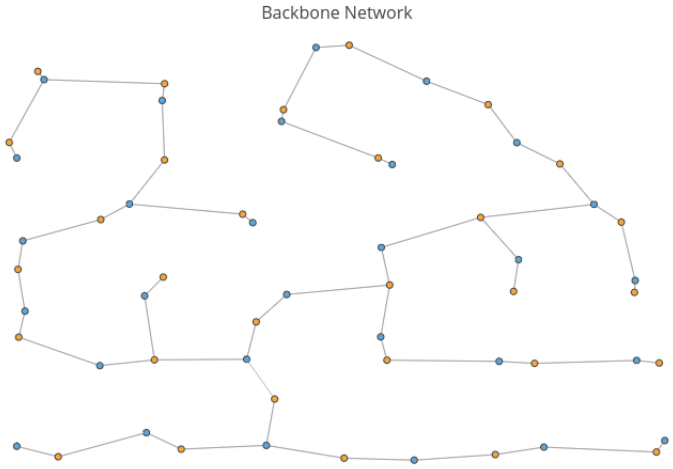
#	Nodes	Average Degree	Topology	Edges	Average Degree
6	4000	64	Disk	99826	49.9132

3.7 Number 7









#	Nodes	Average Degree	Topology	Edges	Average Degree
7	4000	128	Disk	198704	99.3522

References

[1] Matula, David W., and Leland L. Beck. "Smallest-Last Ordering and Clustering and Graph Coloring Algorithms." *Journal of the Association of Computing Machinery* 30.3 (1983): 417-27. Web. 9 Dec. 2016.