Linear-Time Computation of High-Coverage Backbones for Wireless Sensor Networks

Ian Johnson

Southern Methodist University
CSE 7350 - Algorithm Engineering
Professor Lee McFearin

October 22, 2016

Contents

1 Executive Summary **2**
- 1.1 Introduction . . . 2
- 1.2 Programming Environment . . . 3
- 1.3 References . . . 4

2 Wireless Sensor Network Backbone Report **5**
- 2.1 Reduction to Practice . . . 5
  - 2.1.1 Data Structure Design . . . 5
  - 2.1.2 Algorithm Description . . . 6
  - 2.1.3 Algorithm Engineering . . . 7
  - 2.1.4 Verification Walkthrough . . . 10
  - 2.1.5 Algorithm Efficacy . . . 13
- 2.2 Benchmark Results . . . 15
  - 2.2.1 Benchmark 1 . . . 15
  - 2.2.2 Benchmark 2 . . . 16
  - 2.2.3 Benchmark 3 . . . 17
  - 2.2.4 Benchmark 4 . . . 18
  - 2.2.5 Benchmark 5 . . . 19
  - 2.2.6 Benchmark 6 . . . 20
  - 2.2.7 Benchmark 7 . . . 21
  - 2.2.8 Benchmark 8 . . . 22
  - 2.2.9 Benchmark 9 . . . 23
  - 2.2.10 Benchmark 10 . . . 24
  - 2.2.11 Real-Time Graphics Interface . . . 24

Chapter 1

Executive Summary

1.1 Introduction

Widely distributed networks of sensors which relay information to one another, called sensor networks, are an area of interest for many natural and computer scientists. Because inexpensive sensors can easily be distributed around a physical area, sensor networks are a viable strategy for a wide variety of applications. For example, if one were interested in measuring rainfall over the surface of the earth, a globally-distributed network of rainfall-measuring sensors could be used. Connecting these sensors in an efficient and fault-tolerant way in a wireless environment is a difficult problem [1],[2]. In order to minimize energy consumption for the overall network of sensors and maximize node availability across the network, it is useful to define a set of nodes, called a backbone, which are responsible for transmitting data across the entire network. One practical strategy for computing backbones is using graph coloring algorithms [3],[4],[5]

In this report, we present an implementation of a graph-coloring algorithm to compute a high-coverage backbone for a wireless sensor network. The algorithm, given an randomized distribution of points (which represent sensors in a physical space) identifies a subset of points which provide maximal connectivity to the remaining points. This computation is performed in linear time with respect to the sum of the number of points and the number of connections between points. A smallest-last vertex ordering is used to compute a coloring of the graph, and high-frequency pairs of colors are evaluated as network backbones. The resulting networks, along with their backbones are visualized for a unit square distribution, a unit disk distribution, and a spherical distribution. Table 1.1 shows the performance of the algorithm for a set of benchmark cases. The algorithm we present performs well against a set of proposed benchmarks.

ID	Distribution	N_{total}	E	R	$N_{covered}$	Percent Coverage
1	Square	1000	32	0.101	992	0.992
2	Square	4000	64	0.071	3993	0.998
3	Square	16000	64	0.036	15976	0.998
4	Square	64000	64	0.018	63840	0.998
5	Square	64000	128	0.025	63960	0.999
6	Disk	4000	64	0.063	3992	0.998
7	Disk	4000	128	0.089	3995	0.999
8	Sphere	4000	64	0.253	3988	0.997
9	Sphere	16000	128	0.179	15950	0.996
10	Sphere	64000	128	0.089	61215	0.957

Table 1.1: A summary of the results of each of the 10 benchmark data sets

To acheive linear time complexity for backbone selection, a number of operations has to be performed in linear time. Most non-trivially, the smallest last first vertex ordering must be computed in linear time. The algorithm used to achieve this is described in Matula, D.W, Beck, L.L 1983 [6].

1.2 Programming Environment

The processes of random point generation, edge computation, and graph coloring are performed using Python 3, and use the Collections package from Python [7]. All computations were performed on one of two test machines. The first machine is a 2015 MacBook Pro running MacOS Sierra with a 2.5Ghz Intel I7 processor and 16GB of RAM. The second machine, which was used only for testing, and not in the generation of this final report, is a custom-built tower running Ubuntu 14.04 with an overclocked Intel I7 4770k processor and 32GB of RAM. However, a workhorse machine such as this one is not required to reproduce the output of our analyses.

Beyond the Python standard libary (Collections), no additional Python packages or 3rd party code were used. However, rendering was performed using the Processing framework [8]. This framework provides a simple interface for rendering 2D and 3D point distributions, as well as connections between those points.

When the largest benchmark test was performed, the process consumed 100% of available CPU clock cycles (it is single threaded, so this is the maximum possible usage), and 1.24GB of RAM. The benchmark in question is 64000 nodes with an average degree of 128 in a spherical distribution. Visualizing this benchmark would be computationally infeasible, as a significant amount of VRAM would be required. Considering the considerable size of the benchmark dataset, 1.24GB of RAM is a reasonable memory expenditure.

One final consideration with respect to the performance of the algorithm is that the entire process is run in Python inside the Processing environment, to make interfacing with the graphics library more simple. This adds considerable overhead to the total time it takes to perform computations on the datasets. However, because this is a constant overhead, this will not interfere with the measurements of computation time for the benchmark datasets relative to eachother.

Plots and figures for this report were generated using R with Sweave and LaTeX.

1.3 References

[1] Mahfoudh, Chalhoub, Minet, Misson, Amdouni, Node Coloring and Color Conflict Detection in Wireless Sensor Networks, *Future Internet*, 2010, 469-504

[2] Akyildiz, Ian F., et al. "Wireless sensor networks: a survey." Computer networks 38.4 (2002): 393-422.

[3] Clark, Brent N., Charles J. Colbourn, and David S. Johnson. "Unit disk graphs." Discrete mathematics 86.1-3 (1990): 165-177.

[4] Cardei, Mihaela, et al. "Wireless sensor networks with energy efficient organization." Journal of Interconnection Networks 3.03n04 (2002): 213-229.

[5] Gandham, Shashidhar, Milind Dawande, and Ravi Prakash. "Link scheduling in wireless sensor networks: distributed edge-coloring revisited." Journal of Parallel and Distributed Computing 68.8 (2008): 1122-1134.

[6] Matula, D.W, Beck, L.L, Smallest-Last Ordering and Clustering and Graph Coloring Algorithms, *Journal of the Association for Computing Machinery*, July 1983, 421-427

[7] Python Software Foundation. Python Language Reference, version 3.4. Available at http://www.python.org

[8] Reas, C. and Fry, B. Processing: programming for the media arts (2006). Journal AI and Society, volume 20(4), pages 526-538, Springer

Chapter 2

Wireless Sensor Network Backbone Report

2.1 Reduction to Practice

2.1.1 Data Structure Design

A number of data structures are utilized in the process of generating points, computing edges, coloring the graph, and identifying backbones. They are, in order of appearance in the algorithm:

- Points Container
- Adjacency List
- Colors
- Backbones

Points Container

The points container is a simple 2D python list, where each list at index i in the main list is a set of coordinates for a point. Therefore, if the points container is an MxN matrix, then there are M points in an N-dimensional space. The generalization of a 2D list allows for N-dimensional points to be used at any point in the algorithm, since the 2-or-3-dimensionality of the points is not ever explicitly required. This becomes very useful when switching from the Disk and Square distribution to the Spherical distribution.

Adjacency List

The adjacency list, like the points container, is a simple 2D python list. However, it is not a rectangular matrix, like the points container. The list at index i in the main list represents a list of the indeces of nodes in the points container which are connected to the node at index i in the points container. For example, if the 5th list in in the adjacency list contained a 1, a 2, and a 7, then the node whose location is encoded by the 5th list in the points container would be connected to the nodes at indeces 1, 2, and 7 of the points container. From this point forward, the set of coordinates at the ith index of the points container will be refered to as the ith point.

Colors

To store the colors of the points, a 1D python list is used which is parallel to the points container. At index i in the colors list sits a value representing the color of the ith point.

Backbones

The set of backbones is stored using a simple 2D python list. The ith element in the 2D list represents the ith backbone, and contains a list of the indeces of the nodes in the ith backbone. For example, if the 3rd list in the backbone set had a 1, a 3, and a 5, then the 3rd backbone would be the set of points 1,3,5.

2.1.2 Algorithm Description

The overall backbone selection algorithm can be split up into a number of parts, each of which will be explored in further sections:

- Random Point Selection
- Edge Discovery
- Graph Coloring
- Backbone Selection

Random Point Selection

The process of random point selection is slightly different for each of the distributions tested in this report:

- **Unit Square**: For the unit square distribution, random point selection is performed by computing two random values between 0 and 1 and using those values as the x and y coordinates of the point. This is iterated until the desired number of points is produced.
- **Unit Disk**: For the unit disk distribution, the same procedure from the unit square is used (where the random values can fall between -1 and 1, in this case); however, if points fall outside of a unit disk ($x^2 + y^2 > 1$), then the x and y coordinates are re-randomized until they do fall within the disk. The process still ensures that the desired number of points is produced.
- **Unit Sphere**: For the unit sphere distribution, three random values between -1 and 1 are chosen and assigned to the x, y, and z coordinates of the point. The point is then projected onto the surface of the sphere by dividing each of the 3 coordinates by $\sqrt{x^2 + y^2 + z^2}$.

Edge Discovery

Edge discovery is performed using a bucket strategy, so that the algorithm can run in linear time (this is discussed in more detail in the upcoming *Algorithm Engineering* section). In order to detect edges, the points are partitioned into a matrix of N^2 buckets, where $N = \frac{1}{R}$, such that the indeces of the bucket in which a point resides is an approximation of the actual location of the point in 2 dimensions. This is performed using a linear pass through the points in the points list. After that, every point in every bucket is compared to every other point in the bucket, as well as all points in adjacent buckets. This guarantees that all edges are found without directly comparing every node to every other node, which is an $O(N^2)$ algorithm.

Graph Coloring

The graph coloring algorithm utilizes a smallest-last vertex ordering to perform coloring in linear time. The smallest-last vertex ordering is performed in linear time per the algorithm described in Matula, D.W, Beck, L.L 1983 [6]. Once the smallest-last vertex ordering has been computed, the coloring is done by iterating through the smalles-last vertex ordering and, at each point, assigning the first available color to the point on the graph by looking at the colors of points adjacent to the point in question. This is a non-heuristic graph coloring algorithm and is guaranteed to find a minimal number of colors on the graph [6]. Verification of the linear-time claim for this algorithm can be found in the upcoming *Algorithm Engineering* section.

Backbone Selection

Once the graph coloring has been performed, the points in the graph are partitioned into k disjoint sets of point, where k is the number of colors, and the items in each set are all of the same color. The 4 sets with the highest cardinality are selected, and the 6 possible combinations of those 4 colors are identified as possible backbones. For each of those backbones, the coverage is calculated by identifying the number of vertices in the major component of the resulting graph when only the edges which include an item from the backbone as a vertex are included in the graph. Then, the top 3 backbones based on coverage are selected and used as the backbones.

2.1.3 Algorithm Engineering

Each part of the algorithm is optimized for performance. The various optimizations that were used are explained below.

Random Point Selection

The random point selection is optimized for the spherical distribution of points by projecting all points in a random cube of space onto the surface of the sphere. This guarantees that no points are wasted, or thrown away. For the square distribution of points, it is guaranteed that no points will be thrown away because the range of the possible results of the random location is equal to the range of possible locations on the unit square. The unit disk is slightly sub-optimal when compared to the other distributions, as it occasionally throws away points that are generated and fall outside of a unit disk. However, only approximately $\frac{1}{4}$ of points are thrown away, so the algorithm, on average still runs in linear time with respect to the number of vertices, just with a slighly higher constant.

Edge Discovery

Edge discovery, through the use of the the bucket algorithm, is performed in $\theta(|E| + |V|)$ time, where $|E|$ is the number of edges, and $|V|$ is the number of vertices. This is considered linear time with respect to the sum of the cardinalities of the sets of edges and vertices. Figure 2.1 shows the actual running time of the algorithm compared to the sum of the number of edges and vertices for each of the 10 benchmark sets.

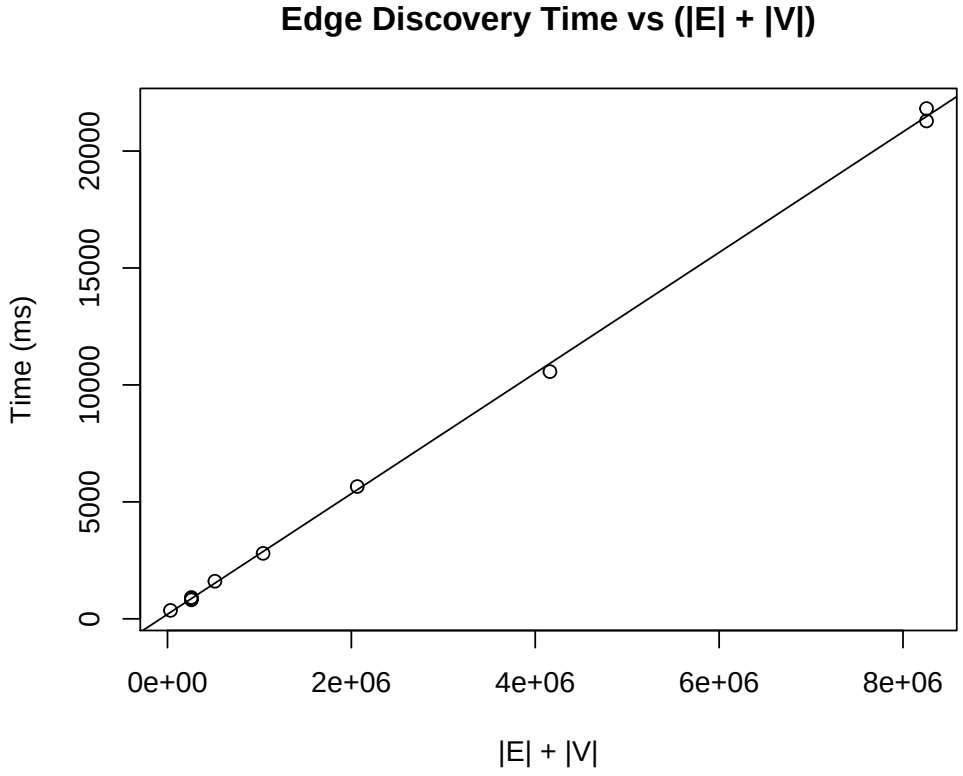


Figure 2.1 - Edge discovery time vs ($|E| + |V|$)

A cursory look at Figure 2.1 shows that there is, in fact, a linear relationship between running time and $(|E|+|V|)$. This linear relationship for the 10 benchmark sets can be treated as operational proof that the edge discovery algorithm is, in fact, $\theta(|E|+|V|)$.

Smallest-Last Vertex Ordering

Figure 2.2 shows a plot of the runtime for the smallest-last vertex ordering vs $(|E| + |V|)$. Much like figure 2.1, it shows a linear relationship between the two. Notice, however, that the slope of the regression line in this plot is much larger. This suggests that the constant in the $(|E|+|V|)$ function that represents the run-time of the ordering is much higher than that of the edge discovery process. The linear relationship between the two plotted factors provide proof that the smallest-last vertex ordering is, in fact $\theta(|E|+|V|)$. A mathematical proof of this can be found in Matula, D.W, Beck, L.L 1983 [6].

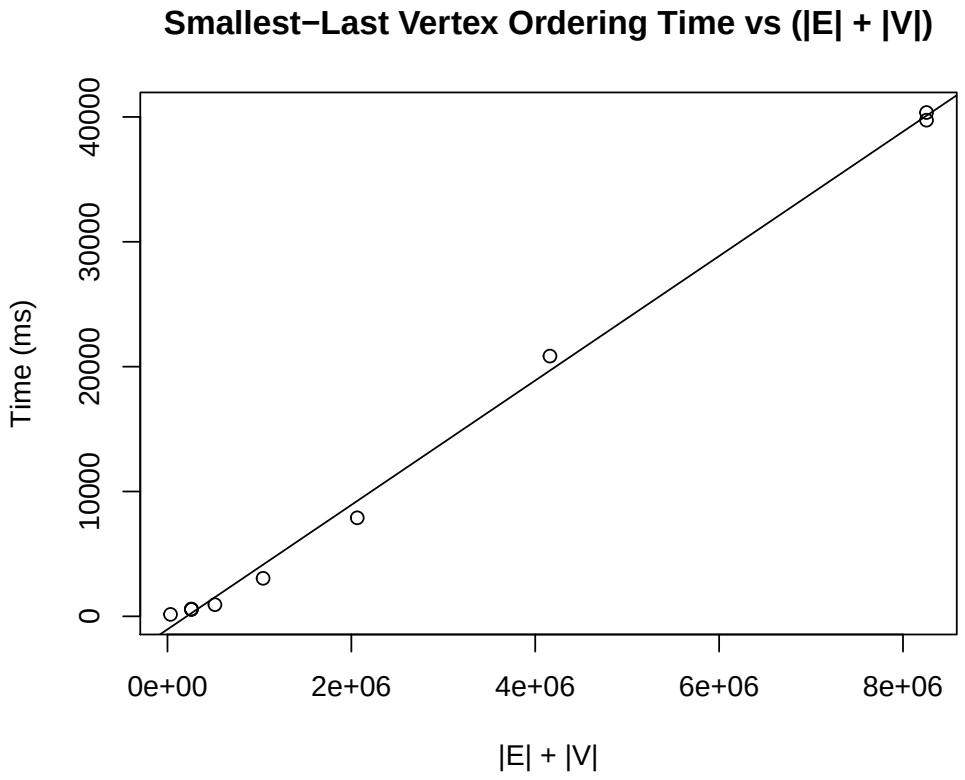


Figure 2.2 - Smallest-last vertex ordering time vs (|E| + |V|)

Graph Coloring

The graph coloring process, like the previously described processes, runs in linear time. Figure 2.3 shows the coloring time vs $(|E| + |V|)$, and, once again, identifies a linear relationship between the two. In this case, the constant on the function of $(|E| + |V|)$ is much smaller than it was for the previous two algorithms. In fact, a cursory look at Figures 2.3 and 2.2 shows that the graph coloring process, on average, takes about $\frac{1}{10}$ of the time of the smallest-last vertex ordering process. The strong linear correlation between graph coloring time and $(|E| + |V|)$ demonstrates that the graph coloring algorithm is $\theta(|E| + |V|)$.

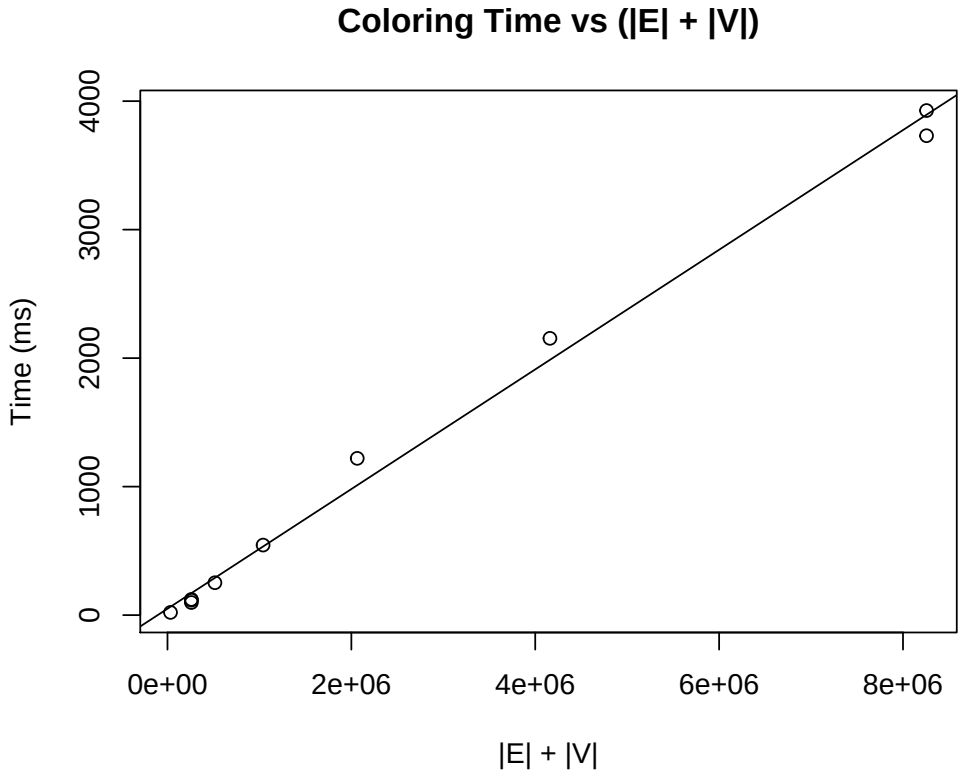


Figure 2.3 - Coloring time vs (|E| + |V|)

Backbone Selection

The backbone selection process is quite simple, and therefore has little algorithmic complexity. The top colors are selected (this is performed by performing a bucket sort on the color lists based on their cardinalities, a $\theta(N)$ operation), and then the 6 backbones are computed. The computation of the cardinality of the giant component of the backbones is performed using a depth-first search, which is also a $\theta(N)$ operation. The top backbones are then selected (based on size of giant component), once again, using a bucket sort and extracting the top 3 (this is also $\theta(N)$, where N is the number of backbones, which is actually constant in this problem (6)). Because the backbone selection process is comprised of 3 individual $\theta(N)$ operations, the entire process is $\theta(N)$. Since it is well-known that all 3 of the described processes do, in fact, run in linear time, no graph of the running time of this operation is provided.

2.1.4 Verification Walkthrough

For the graph shown in Figure 2.4, the smallest-last ordering, coloring, and backbone selection algorithms will be demonstrated. For this graph, there are 20 nodes ($N = 20$), the radius of each node is 0.4 ($R = 0.4$), and the average degree of each node is 6.

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

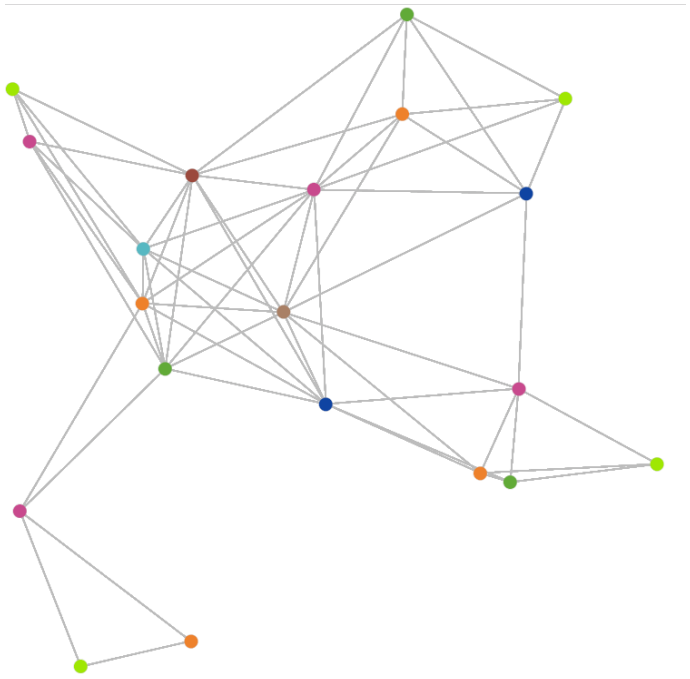


Figure 2.4 - Adjacency list and visualization of reference graph for verification

Smallest-Last Ordering

To perform the smallest-last ordering, a set of degree buckets will be used. These will be visualized using a table:

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

Current Ordering: {}

The first node to be added to the ordering is 12 (this is chosen arbitrarily between 12 and 17, as they're tied for the number of vertices adjacent to them). After removing the 12, the edge counts for all nodes adjacent to count decrease.

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

Current Ordering: {*12*}

The following node to be removed is 17. The remaining iterations are shown below.

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

Current Ordering: {*12, 17*}

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

Current Ordering: {*12, 17, 3*}

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

Current Ordering: {12, 17, 3, 18}

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

Current Ordering: {12, 17, 3, 18, 5}

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

Current Ordering: {12, 17, 3, 18, 5, 1}

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

Current Ordering: {12, 17, 3, 18, 5, 1, 9}

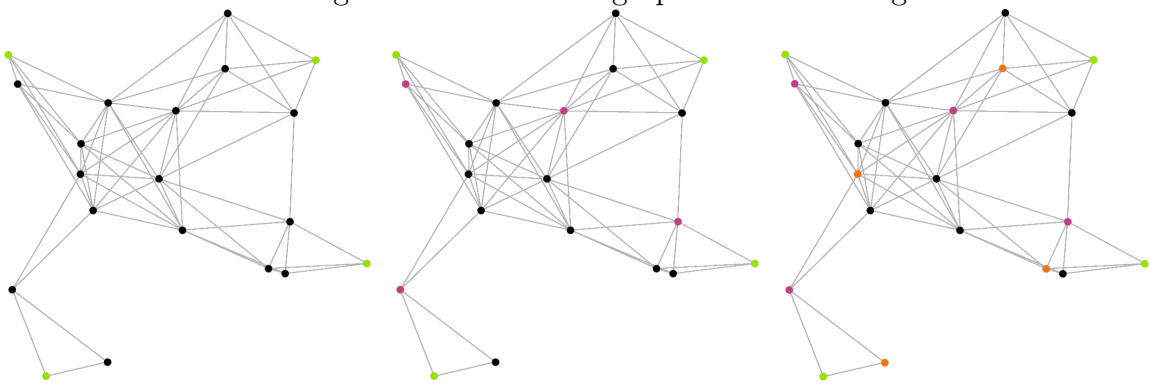
Degree	1	2	3	4	5	6	7	8	9
Vertices				10	4,8,16,19	2,13,14		7,15	0,11

Current Ordering: {12, 17, 3, 18, 5, 1, 9, 6}

This process continues until the final ordering is: {12, 17, 3, 18, 5, 1, 9, 6, 10, 4, 19, 16, 13, 15, 14, 11, 8, 7, 2, 0}. This list is then reversed for the coloring process.

Graph Coloring

The smallest-last ordering is traversed, and every node is assigned a color when it is reached in the list. The color is assigned by exploring the list of neighbors to a node and seeing which colors which already exist in the graph don't exist in the set of neighbors. If no such color exists, then a new color is added and assigned to the current node. Figure 2.4 shows the first few iterations of the coloring. The final colored graph is shown in Figure 2.3.



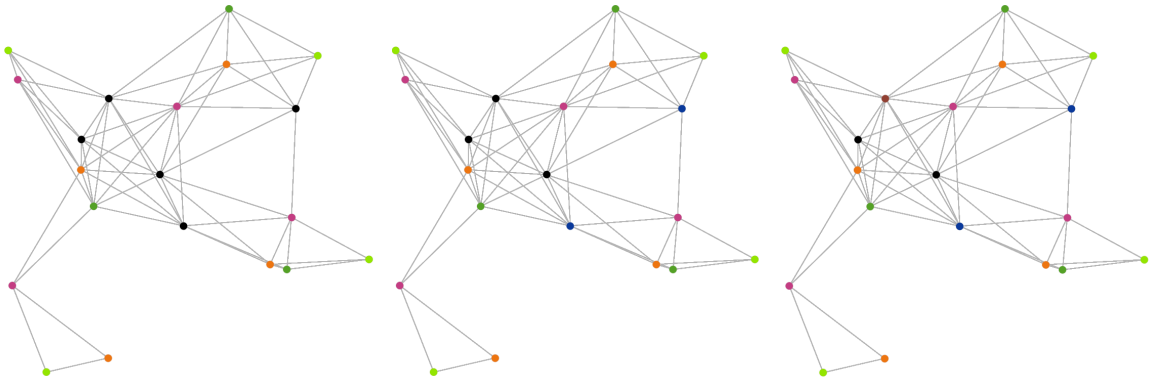


Figure 2.5 - First 6 iterations of coloring

Backbone Selection

After the colors have been identified, the top 4 colors are isolated and 6 possible backbones are created. The size of the giant component is then taken, and, in this case, the largest giant component is of cardinality 6. The backbone, therefore, is comprised of 6 nodes. Figure 2.6 shows the subgraph which contains the bes t backbone that was found. Note that the component of the subgraph on the right side of the graph is not part of the backbone, as it is not part of the giant component. The coverage of this backbone is 0.8, as 16 of the 20 nodes are reachable from the backbone.

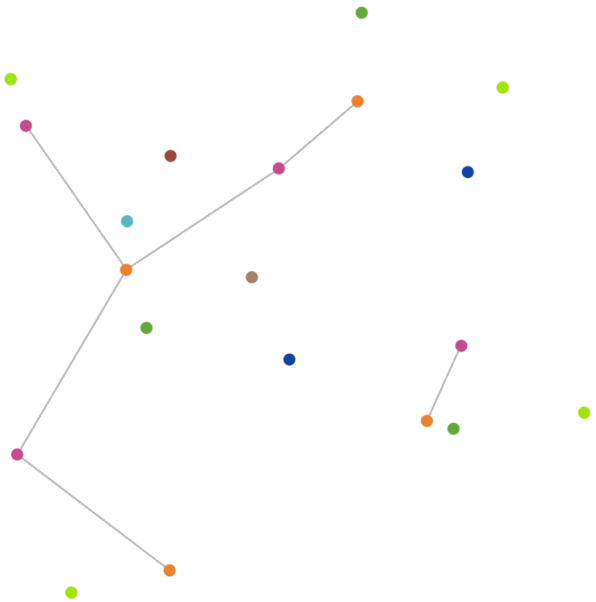


Figure 2.6 - A backbone for the verification network (note that only the left component of the graph is the actual backbone)

2.1.5 Algorithm Efficacy

Table 2.1 shows the coverages of the backbones computed for each of the 10 benchmarks (each of which will be explored more in the upcoming sections). Clearly, the algorithm performs quite well, as the coverages of the computed backbones are exceptionally high (the lowest is 0.996, or

99.6% coverage). Furthermore, all algorithms run in linear time, and the use of the Processing language allows for real-time visualization of the graphs built by the algorithm. This real-time rendering is interactive, and contains multiple view modes for looking at individual colors and backbones. The efficiency of the algorithm was evident in the run-time graphs shown in the *Algorithm Engineering* section, and the implementation in Python was even able to compute backbones for a graph of 2 million vertices with an average degree of 250 in a unit square distribution in only a few minutes.

ID	Distribution	N_{total}	E	R	$N_{covered}$	Percent Coverage
1	Square	1000	32	0.101	997	0.997
2	Square	4000	64	0.071	3993	0.998
3	Square	16000	64	0.036	15976	0.998
4	Square	64000	64	0.018	63840	0.998
5	Square	64000	128	0.025	63960	0.999
6	Disk	4000	64	0.063	3992	0.998
7	Disk	4000	128	0.089	3995	0.999
8	Sphere	4000	64	0.253	3988	0.997
9	Sphere	16000	128	0.179	15950	0.996
10	Sphere	64000	128	0.089	63915	0.999

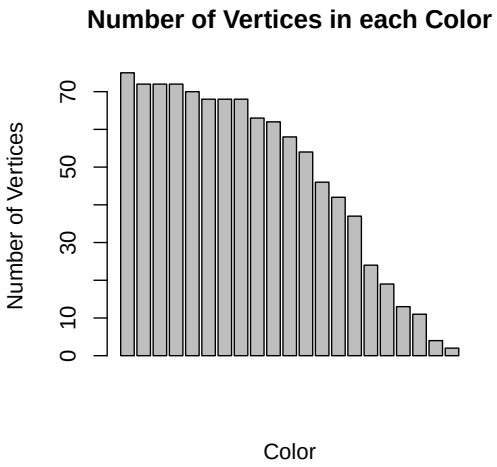
Table 2.1: A summary of the results of each of the 10 benchmark data sets

2.2 Benchmark Results

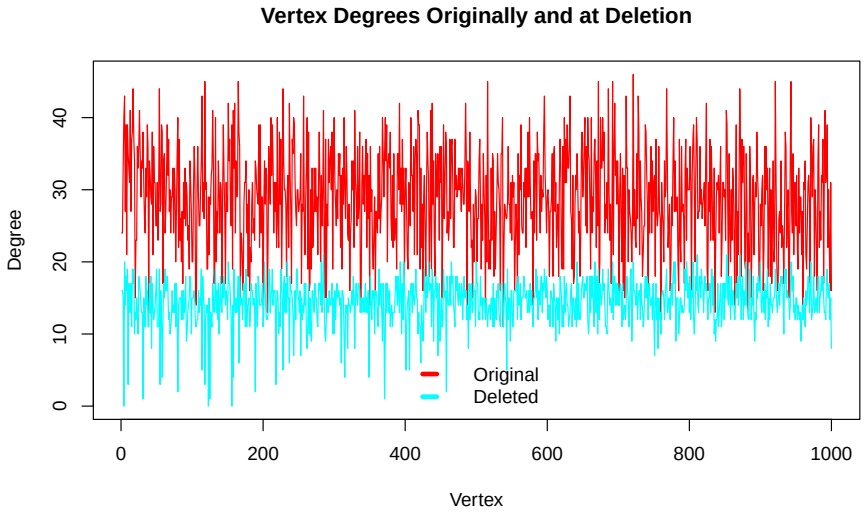
The remainder of the report will include visualizations of the 10 benchmark datasets.

2.2.1 Benchmark 1

ID	1
Distribution Type	Square
$\|V\|$ (Number of Vertices)	1000
R (Radius)	0.101
$\|E\|(Number of Edges)$	29352
$Degree_{min}$	8
$Degree_{max}$	49
$Degree_{del,max}$	21
Number of Colors	23
Max Color Class Size	78
Terminal Clique Size $\|V\|_{largestBackbone}$	144

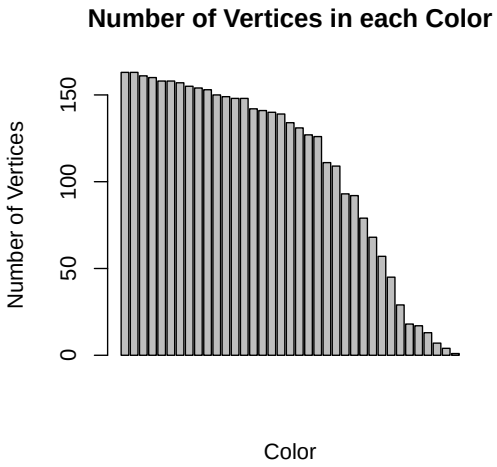


Backbone 1	
Vertices	148
Edges	173
Coverage	0.99
Backbone 2	
Vertices	148
Edges	180
Coverage	0.987

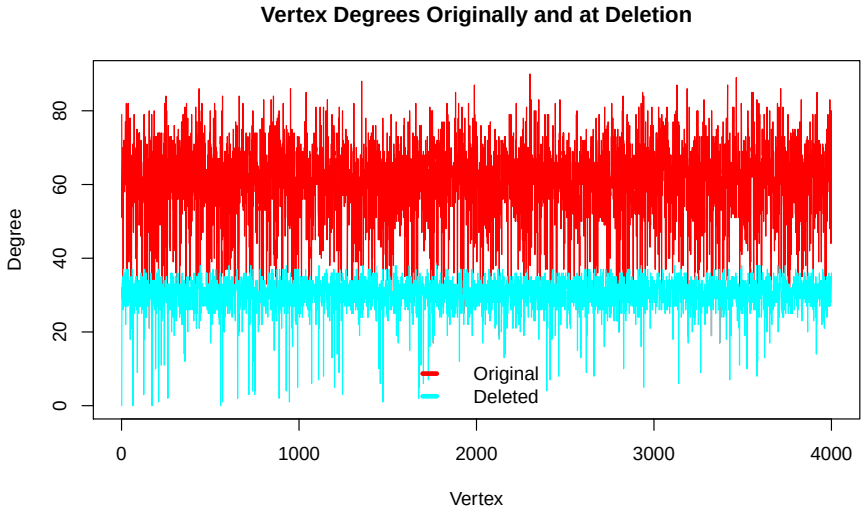


2.2.2 Benchmark 2

ID	2
Distribution Type	Square
$\|V\|$ (Number of Vertices)	4000
R (Radius)	0.0714
$\|E\|(Number of Edges)$	120609
$Degree_{min}$	17
$Degree_{max}$	90
$Degree_{del,max}$	38
Number of Colors	27
Max Color Class Size	163
Terminal Clique Size $\|V\|_{largestBackbone}$	321

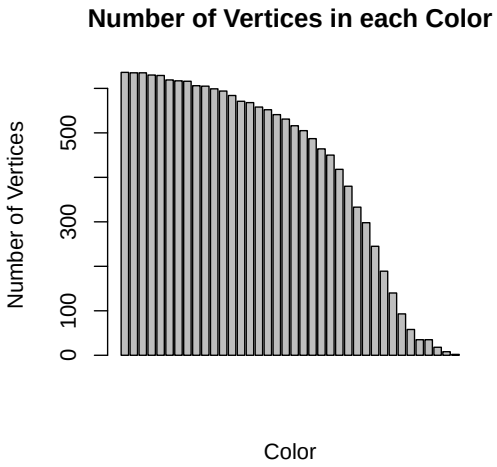


Backbone 1	
Vertices	321
Edges	389
Coverage	0.998
Backbone 2	
Vertices	326
Edges	411
Coverage	1.00

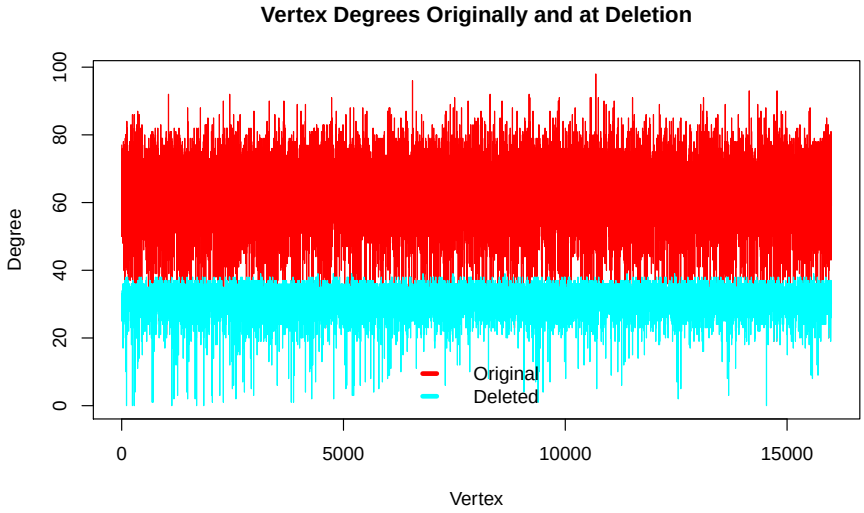


2.2.3 Benchmark 3

ID	3
Distribution Type	Square
$\|V\|$ (Number of Vertices)	16000
R (Radius)	0.0357
$\|E\|(Number of Edges)$	497435
$Degree_{min}$	19
$Degree_{max}$	98
$Degree_{del,max}$	39
Number of Colors	38
Max Color Class Size	636
Terminal Clique Size	
$\|V\|_{largestBackbone}$	1271

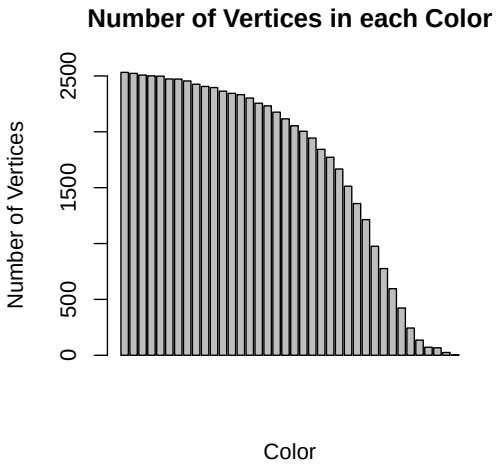


Backbone 1	
Vertices	1270
Edges	1634
Coverage	0.999
Backbone 2	
Vertices	1271
Edges	1610
Coverage	0.996

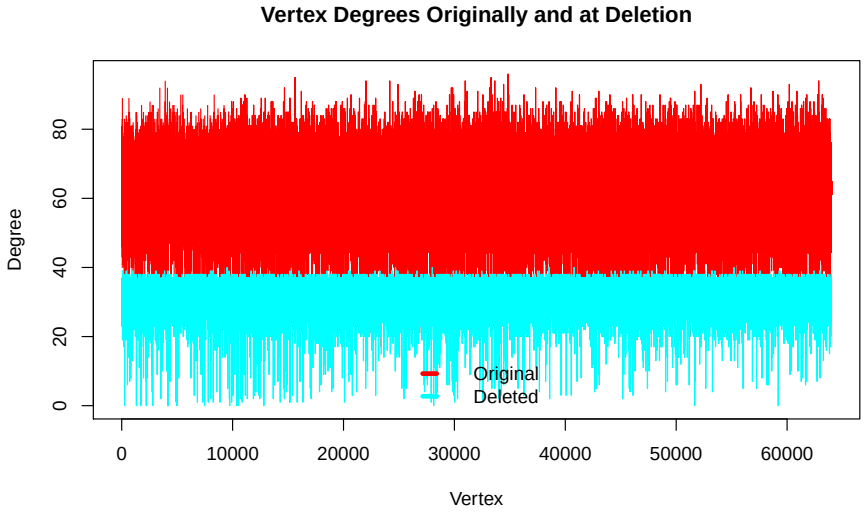


2.2.4 Benchmark 4

ID	4
Distribution Type	Square
$\|V\|$ (Number of Vertices)	64000
R (Radius)	0.0178
$\|E\|(Number of Edges)$	2018059
$Degree_{min}$	15
$Degree_{max}$	96
$Degree_{del,max}$	40
Number of Colors	38
Max Color Class Size	2532
Terminal Clique Size $\|V\|_{largestBackbone}$	5040

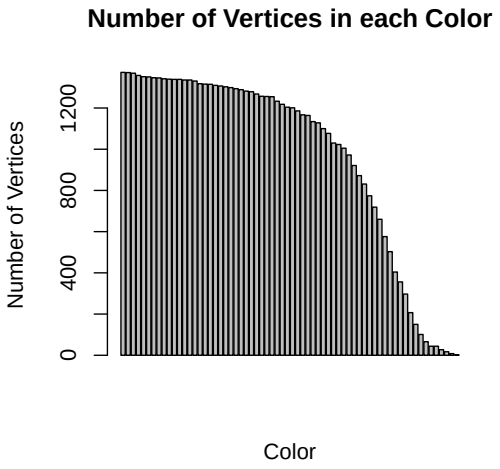


Backbone 1	
Vertices	5040
Edges	6510
Coverage	0.999
Backbone 2	
Vertices	5055
Edges	6562
Coverage	0.998

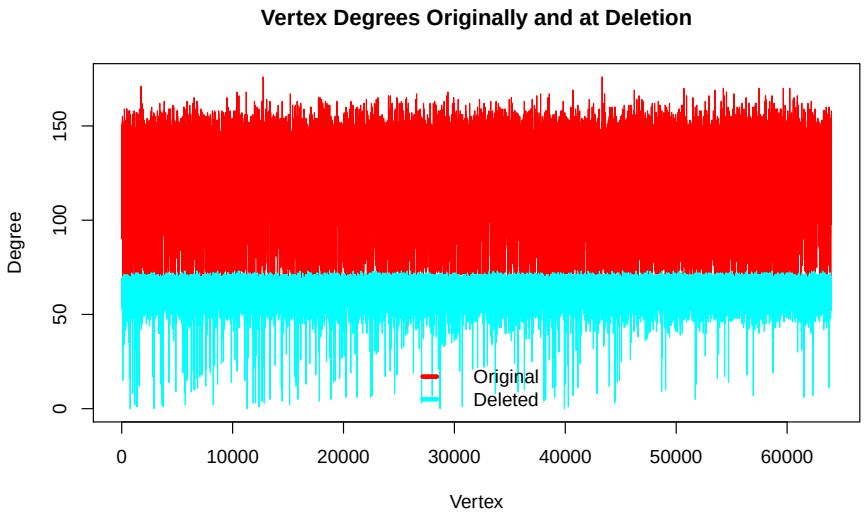


2.2.5 Benchmark 5

ID	5
Distribution Type	Square
$\|V\|$ (Number of Vertices)	64000
R (Radius)	.0252
$\|E\|(Number of Edges)$	4009517
$Degree_{min}$	36
$Degree_{max}$	179
$Degree_{del,max}$	72
Number of Colors	66
Max Color Class Size	1374
Terminal Clique Size $\|V\|_{largestBackbone}$	2734

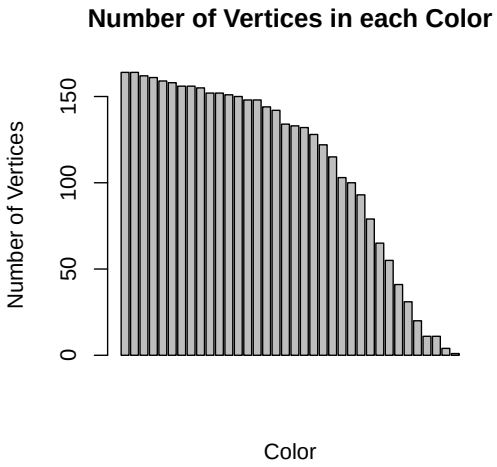


Backbone 1	
Vertices	2734
Edges	3714
Coverage	0.999
Backbone 2	
Vertices	2737
Edges	3755
Coverage	0.999

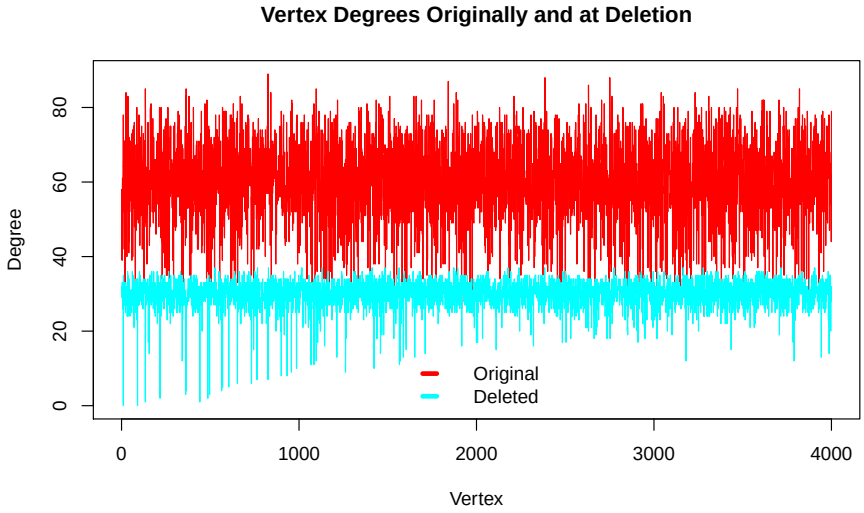


2.2.6 Benchmark 6

ID	6
Distribution Type	Disk
$\|V\|$ (Number of Vertices)	4000
R (Radius)	.0632
$\|E\|(Number of Edges)$	119871
$Degree_{min}$	23
$Degree_{max}$	89
$Degree_{del,max}$	37
Number of Colors	36
Max Color Class Size	164
Terminal Clique Size	
$\|V\|_{largestBackbone}$	328

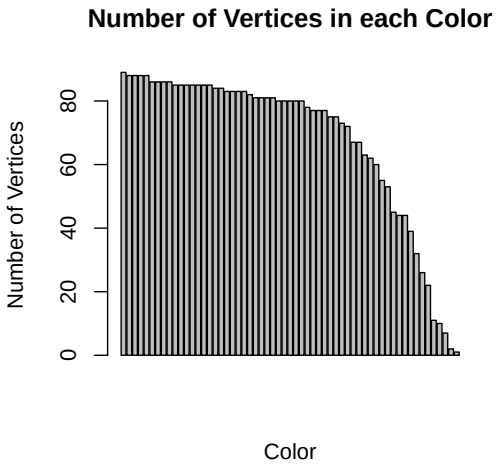


Backbone 1	
Vertices	328
Edges	415
Coverage	0.993
Backbone 2	
Vertices	326
Edges	420
Coverage	0.998

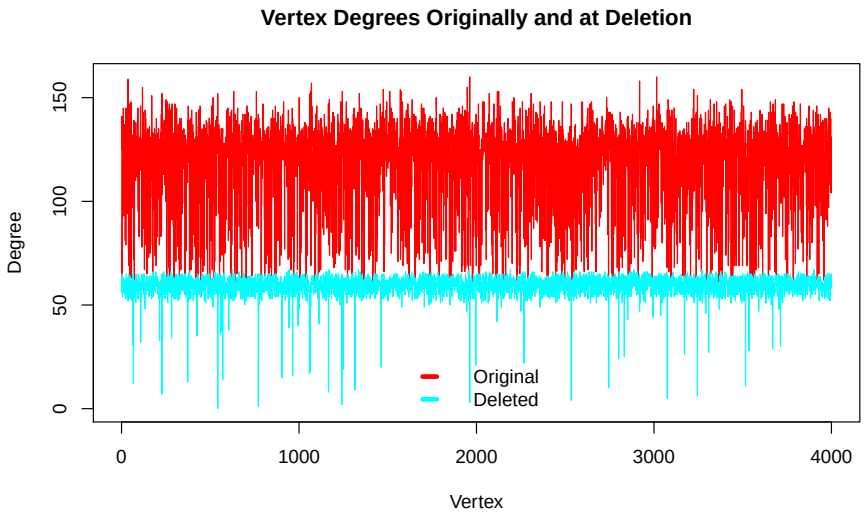


2.2.7 Benchmark 7

ID	7
Distribution Type	Disk
$\|V\|$ (Number of Vertices)	4000
R (Radius)	.0894
$\|E\|(Number of Edges)$	236763
$Degree_{min}$	51
$Degree_{max}$	160
$Degree_{del,max}$	67
Number of Colors	59
Max Color Class Size	89
Terminal Clique Size	
$\|V\|_{largestBackbone}$	176

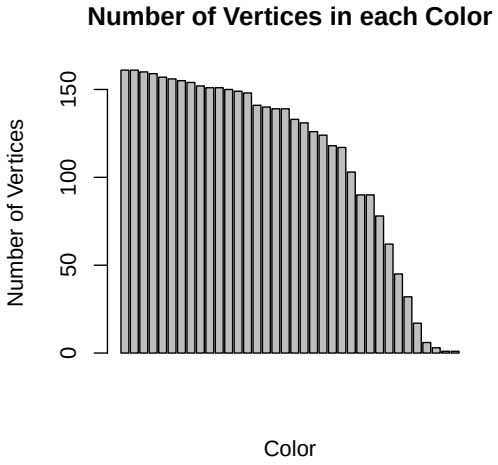


Backbone 1	
Vertices	176
Edges	238
Coverage	0.998
Backbone 2	
Vertices	177
Edges	230
Coverage	0.986

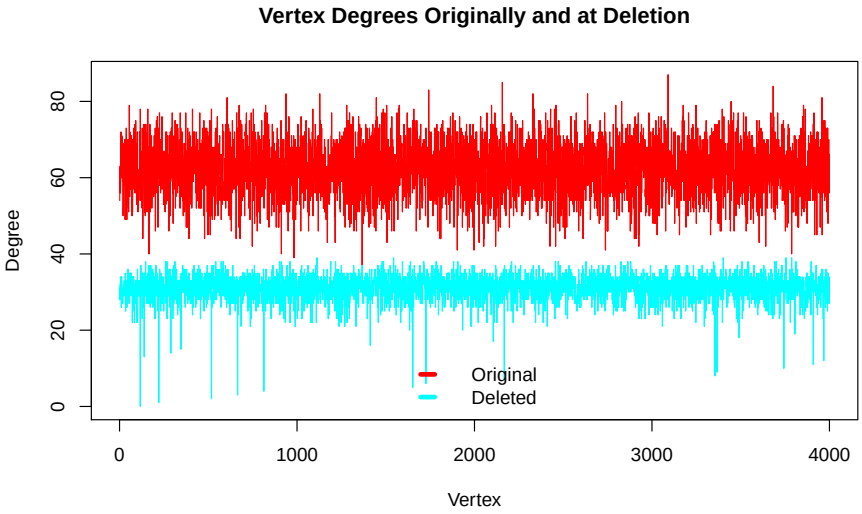


2.2.8 Benchmark 8

ID	8
Distribution Type	Sphere
$\|V\|$ (Number of Vertices)	4000
R (Radius)	0.252
$\|E\|(Number of Edges)$	124376
$Degree_{min}$	37
$Degree_{max}$	87
$Degree_{del,max}$	39
Number of Colors	36
Max Color Class Size	161
Terminal Clique Size	
$\|V\|_{largestBackbone}$	321
Faces on Largest Backbone	101

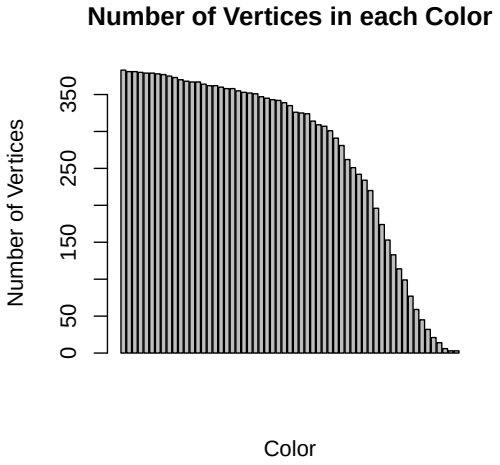


Backbone 1	
Vertices	321
Edges	420
Coverage	0.996
Faces	101
Backbone 2	
Vertices	319
Edges	429
Coverage	0.999
Faces	112

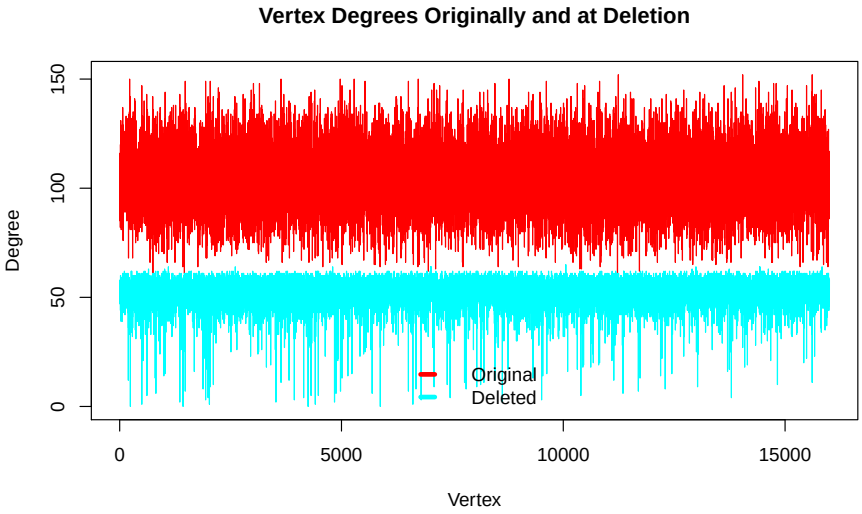


2.2.9 Benchmark 9

ID	9
Distribution Type	Sphere
$\|V\|$ (Number of Vertices)	16000
R (Radius)	0.179
$\|E\|(Number of Edges)$	836565
$Degree_{min}$	61
$Degree_{max}$	152
$Degree_{del,max}$	65
Number of Colors	59
Max Color Class Size	383
Terminal Clique Size	
$\|V\|_{largestBackbone}$	762
Faces on Largest Backbone	189

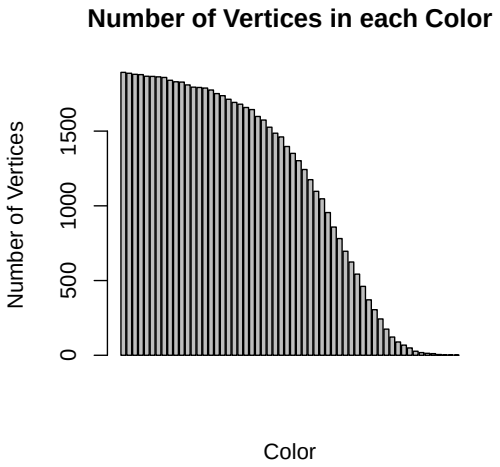


Backbone 1	
Vertices	762
Edges	949
Coverage	0.995
Faces	189
Backbone 2	
Vertices	763
Edges	938
Coverage	0.995
Faces	176

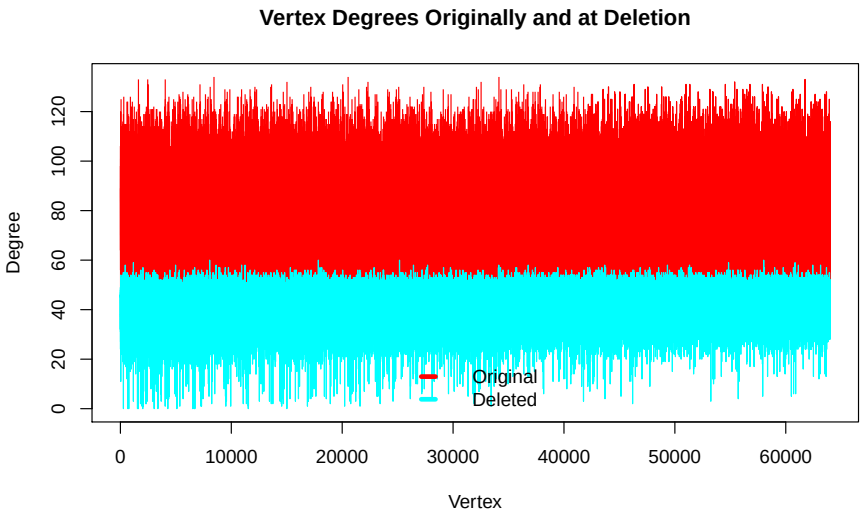


2.2.10 Benchmark 10

ID	10
Distribution Type	Sphere
$\|V\|$ (Number of Vertices)	64000
R (Radius)	.0894
$\|E\|(Number of Edges)$	2597009
$Degree_{min}$	35
$Degree_{max}$	134
$Degree_{del,max}$	60
Number of Colors	58
Max Color Class Size	1839
Terminal Clique Size $\|V\|_{largestBackbone}$	3758



Backbone 1	
Vertices	3758
Edges	4396
Coverage	.991
Faces	640
Backbone 2	
Vertices	3773
Edges	4438
Coverage	.994
Faces	667



2.2.11 Real-Time Graphics Interface

The graphs generated in this project can be visualized interactivelly using Processing. Included below are two screenshots of a sphere distribution. While rendering the sphere, one can drag the sphere around to see it from different perspectives, and press number keys to switch between view modes (available options include seeing the whole graph, seeing individual backbones, or seeing individual colors).

The code for the entire project, including the real-time visualization, is available at:

https://www.github.com/ianjjohnson/SensorNetwork