Jimi Ford
CSCI 351-01 Data Communications and Networks
Programming Project 1 Report

1. Program Inputs, Program Objective, Program Outputs

MonteCarlo.java

This program takes in a seed value for a random number generator, the upper and lower boundaries for the number of vertices in each graph as well as a number to increment by, the upper and lower boundaries for the edge probability as well as a number to increment by, the number of random graphs to generate for each combination of *V* (vertices) and *p* (edge probability), and finally, a prefix for naming each plot generated by this program. After checking for valid input, this program loops through each combination of vertices and edge probabilities, running the specified number of simulations on each combination. Each random graph (or simulation) is generated by looking at every possible pair of vertices, generating a random floating point between 0 and 1, and marking these vertices with an edge connecting them if the random value is less than or equal to the specified edge probability (for that unique graph). In each simulation, the distance values of each graph are calculated with a breadth first search from vertex A to vertex B using the depth of the search as the distance from A to B.

PlotHandler.java

This program takes in a list of plot files generated by MonteCarlo.java. After the program ensures proper use of PlotHandler, it deserializes the files into objects that contain the plot data, and displays each plot in its own window. From there, each window opened has the capability of adjusting the plot size, formatting, textual attributes, and saving the plot to an image. Thanks to Prof. Alan Kaminsky's PJ2 library, PlotHandler is a very small program since all of the visual/GUI code is provided by PJ2.

2. Exact Command Line

MonteCarlo.java

note: This program must be invoked by Prof. Alan Kaminsky's Parallel Java 2 library with the CLASSPATH environment variable set as per Prof. Alan Kaminsky's instructions on this.

```
usage: java pj2 MonteCarlo <seed> <min_v> <max_v> <v_grain> <min_p>
<max_p> <p_grain> <num_simulations> <optional plotfile prefix>
```

`<seed>` – Seed value for Prof. Alan Kaminsky's PRNG
`<min_v>` – Lower bound (inclusive) for number of vertices in random graphs, *V*

`<max_v>` – Upper bound (inclusive) for number of vertices in random graphs, *V*
`<v_grain>` – Vertex granularity: amount to increment the number of vertices in the graph by for each round of *n* simulations
`<min_p>` – Lower bound (inclusive) for edge probability, *p*
`<max_p>` – Upper bound (inclusive) for edge probability, *p*
`<p_grain>` – Edge probability granularity: amount to increment the edge probability by for each round of *n* simulations
`<num_simulations>` – Number of random graphs to generate *per V,p* combination, *n*
`<optional plotfile prefix>` – (Optional) Prefix for file output (default = "plot")

PlotHandler.java

note: This program allows for any number (greater than 0) of plot files to be specified in the command line arguments.

```
usage:
java PlotHandler <plot-file-1> (<plot-file-2> <plot-file-3>... etc.)
```

`<plot-file-1>` – The plot file (generated by MonteCarlo) to visualize in an X-Y plot

3. Source Code *(See Appendix A for project's source code)*

4. For a given number of vertices *V*, what happens to the average distance as the edge probability increases, and why is this happening?

As evidenced by the data gathered in this report, *(Q5. Supporting Data)*, we see that the average distance between nodes in random graphs decreases and approaches 1 as the edge probability *p*, increases. It's important to note that the average distance doesn't continually decrease; it increases for a short period up to a single global maximum and then decreases and converges to 1. This is because as *p* increases, so do the number of edges in a graph. The more edges there are in a graph, the closer vertices become. For instance, let's say that there exists some path from vertex *A* to *B* to *C* but *A* and *C* are not immediately connected by a single edge. The more edges we add to this graph, the higher the chances are that we will end up with a path connecting vertex *A* and *C* directly, making the distance between the two vertices 1.

5. Supporting Data

Commands used:

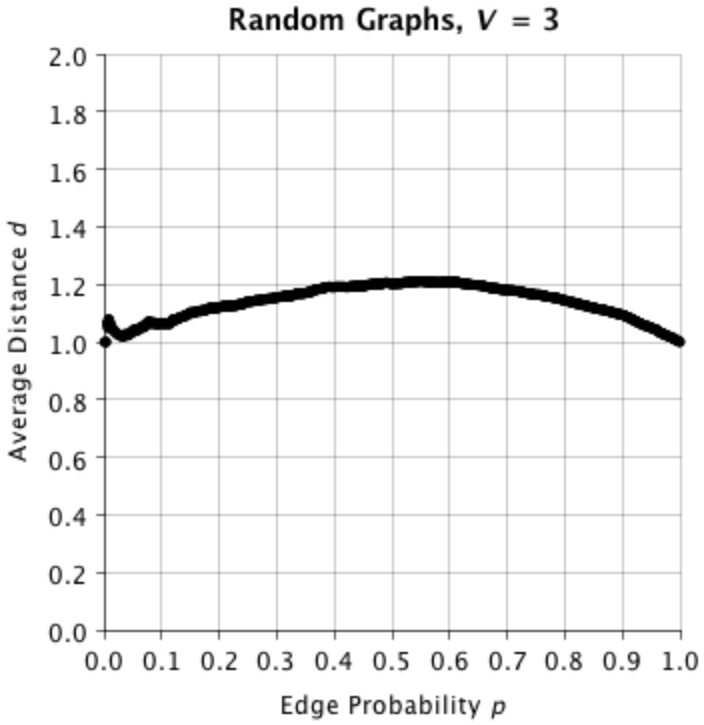
This first command will run the simulations and save *.dwg* files as well as a *.csv* file with the prefix *"plot-Q4"*.

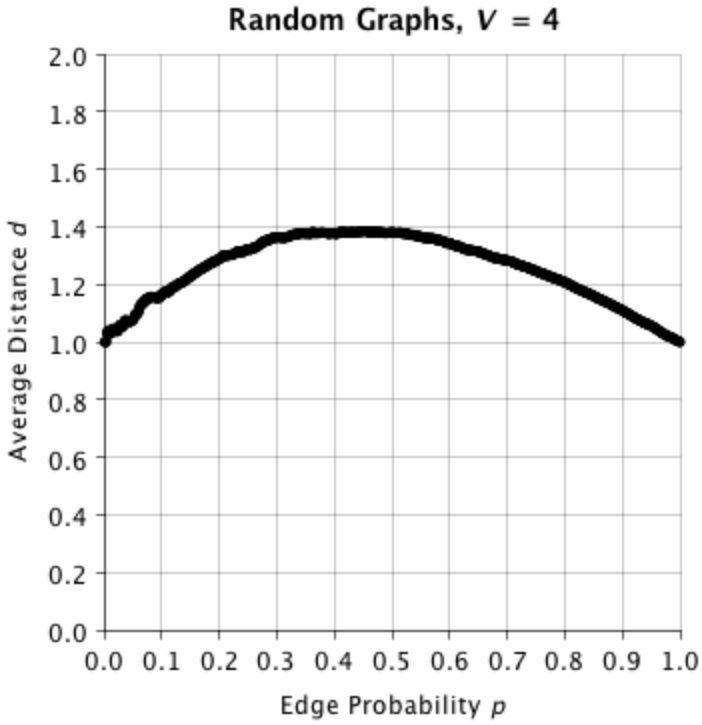
```
java pj2 MonteCarlo 100123456789 3 10 1 0 1 .001 1000 plot-Q4
```
(expected runtime 2-3 minutes on quad-core)

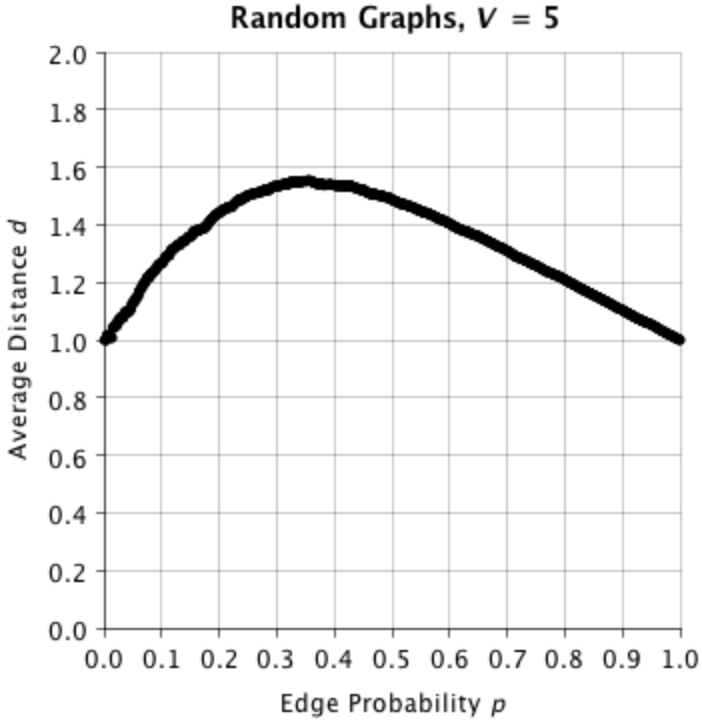
*note: In order to save you time while grading, 1000 simulations are used for each combination of **p** and **V**. As a result, the plots produced aren't extremely smooth but are smooth enough to convey the general trend of varying **p** and holding **V** constant.*

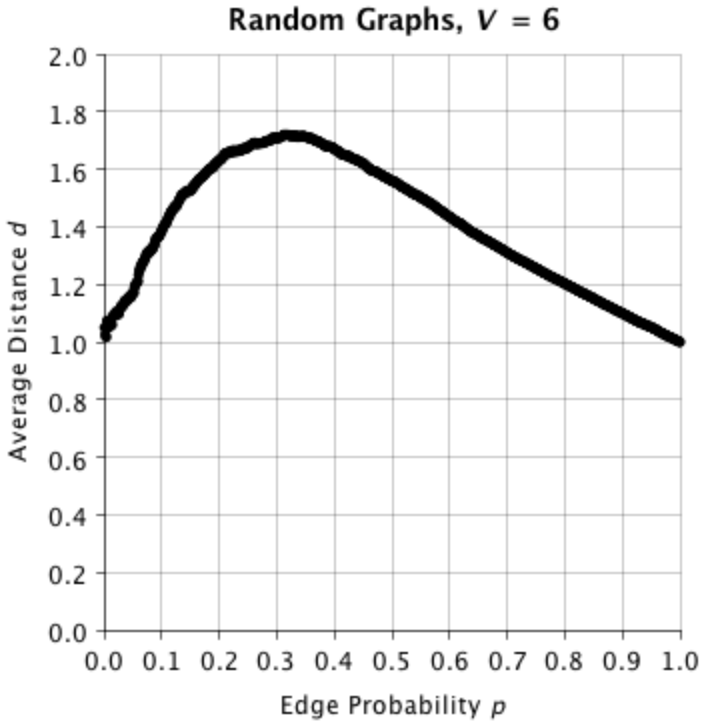
Then after generating the *.dwg* files, you must run the following command to visualize the plots.

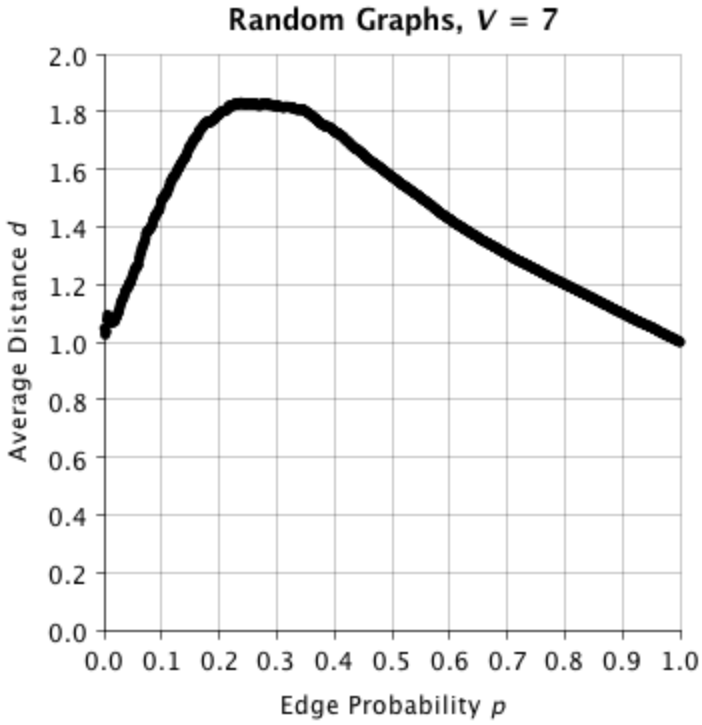
```
java PlotHandler plot-Q4-V-3.dwg plot-Q4-V-4.dwg plot-Q4-V-5.dwg
plot-Q4-V-6.dwg plot-Q4-V-7.dwg plot-Q4-V-8.dwg plot-Q4-V-9.dwg
plot-Q4-V-10.dwg
```

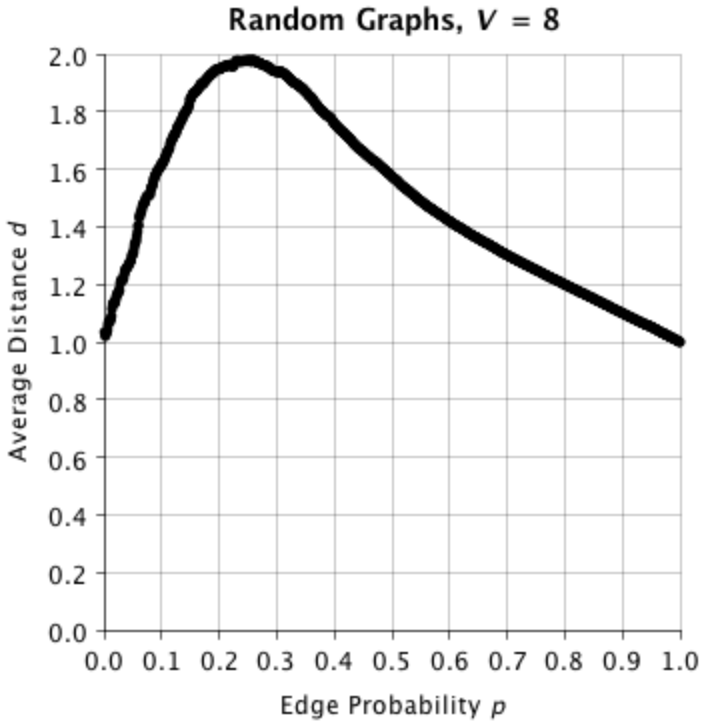


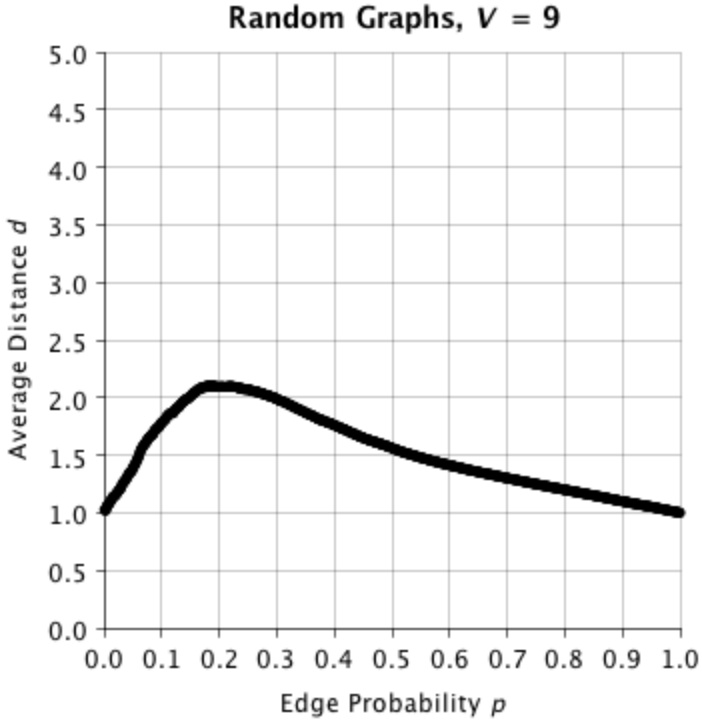

Random Graphs, $V = 4$
Average Distance d
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Edge Probability p

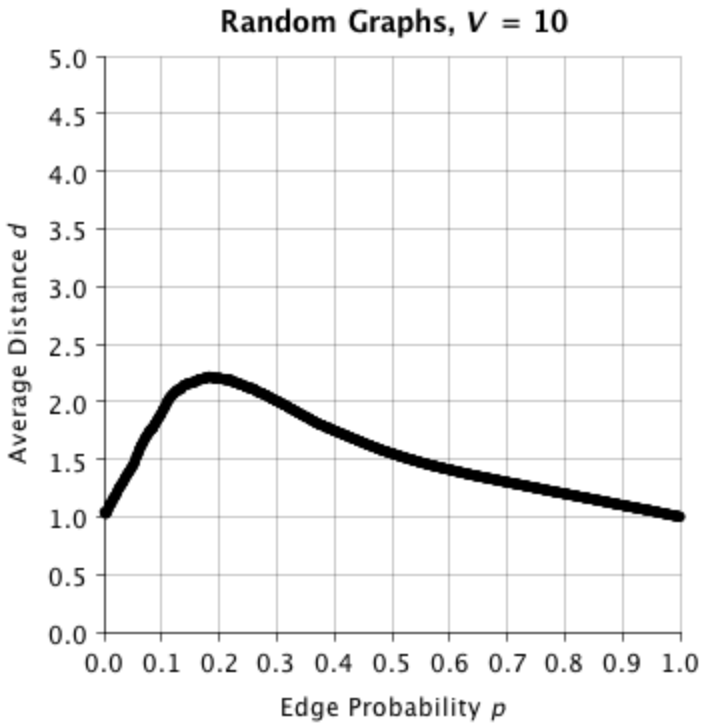

Random Graphs, $V = 5$
Average Distance d
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Edge Probability p


Random Graphs, V = 6
Average Distance d
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Edge Probability p


Random Graphs, V = 7
Average Distance d
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Edge Probability p


Random Graphs, V = 8
Average Distance d
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Edge Probability p


Random Graphs, V = 9
Average Distance d
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Edge Probability p



*note: The table included in this section is generated with a different command. This is because in order to produce dense plot data (above), a very low increment value for **p** must be used (.001) and by using this value, 1000 columns are generated in the table. To produce a table that will fit in this report, run the command listed below.*

```
java pj2 MonteCarlo 100123456789 3 10 1 0 1 .1 1000 plot-Q4a
```

This will generate a table (with fewer columns) in CSV format named “plot-Q4a-table.csv”.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
3	1.065217391	1.120171674	1.15450237	1.19047619	1.202253219	1.209094962	1.180533752	1.144140906	1.09494744	1
4	1.163027656	1.287384615	1.363950807	1.377513452	1.38004965	1.343414634	1.282342502	1.20648667	1.109998331	1
5	1.268370607	1.441808272	1.532845309	1.538985939	1.486505766	1.403758911	1.308493263	1.209061684	1.1042	1
6	1.385915493	1.632208158	1.708306504	1.669605339	1.558211983	1.435005467	1.309475942	1.202669336	1.100066667	1
7	1.493717438	1.789776499	1.822219195	1.730421605	1.573167097	1.42658118	1.304331234	1.200142857	1.099666667	1
8	1.620499182	1.947543113	1.93952269	1.758263198	1.574984556	1.419672013	1.301297402	1.199642857	1.100464286	1
9	1.780844835	2.095249625	1.988327227	1.760451228	1.56220631	1.412558424	1.300205601	1.199361111	1.099833333	1
10	1.899729811	2.203152181	2.003951685	1.747028088	1.549112822	1.408770643	1.300148919	1.199466667	1.100333333	1

6. For a given edge probability *p*, what happens to the average distance as the number of vertices *V* increases, and why is this happening?

For this answer, we will need to direct our attention to both *Q5. Supporting Data*, and *Q7. Supporting Data*. Observe the peak value in each plot and how it occurs at a smaller and smaller value for *p* as *V* increases. After this peak value, the average distance for every graph converges to 1. For given *p* values that are greater than the *p* value at which point the peak distance occurs, even as *V* increases, the average distance decreases. For given *p* values that are less than the *p* value at which point the peak distance occurs, as *V* increases, so does the average distance *(see Q5. Supporting Data)*. This is happening because for each combination of *p* and *V*, there is a point where there is optimal (maximal) distance. As more vertices are added to a graph, the upper bound on the maximum possible distance between two vertices is increased since the furthest two vertices can be from each other is *V* - 1. Once we reach that maximal distance for the graph, the distance must converge to 1 as discussed in the previous answer. Since the maximum possible distance increases with more vertices, the maximal (or peak) values for distance increase and have a steeper slope to climb to converge to 1. This explains why average distances at values for *p* to the right of this peak will decrease as *V* increases, because they approach the asymptote more rapidly.

7. Supporting Data

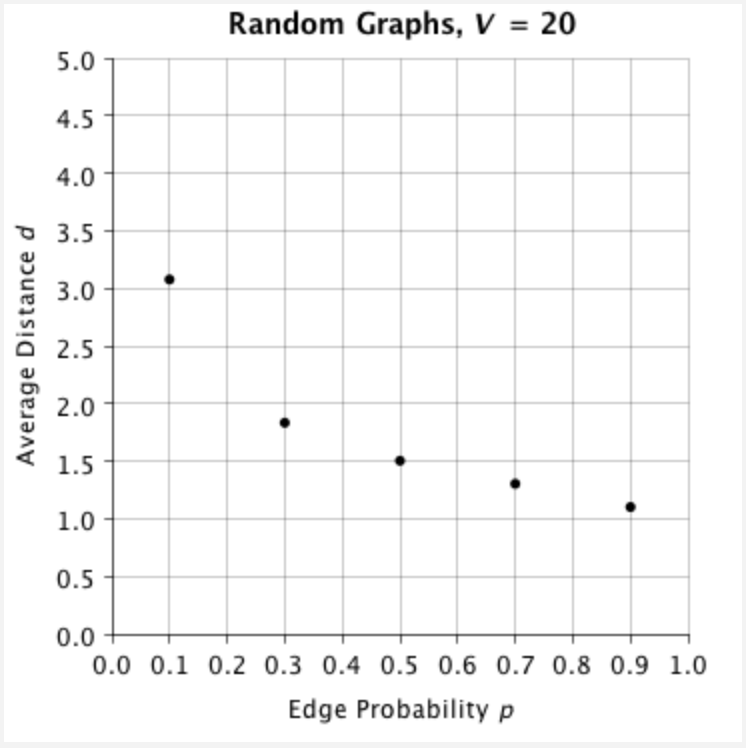
Commands used:

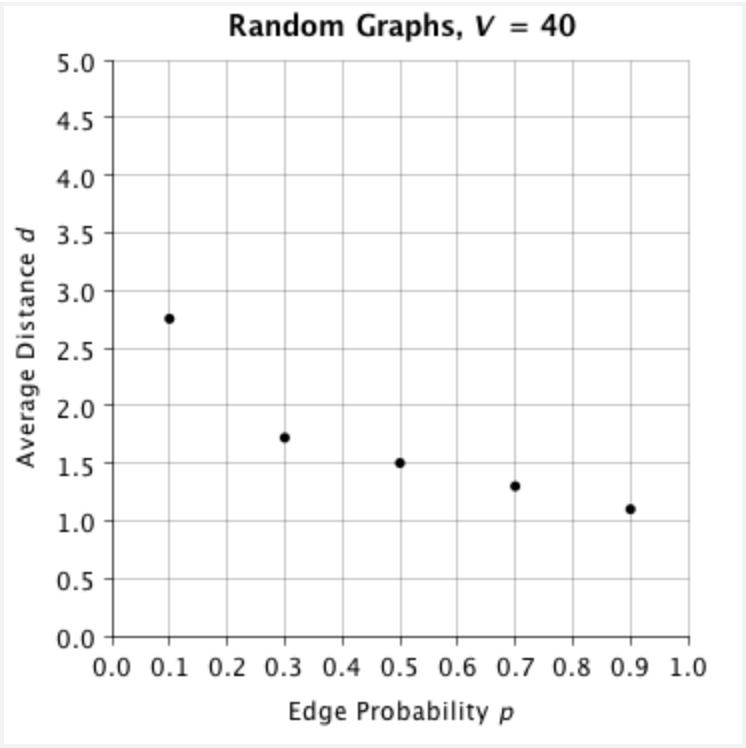
```
java pj2 MonteCarlo 100123456789 20 100 20 .1 .9 .2 100 plot-Q6
(expected runtime 2-3 minutes on quad-core)
```

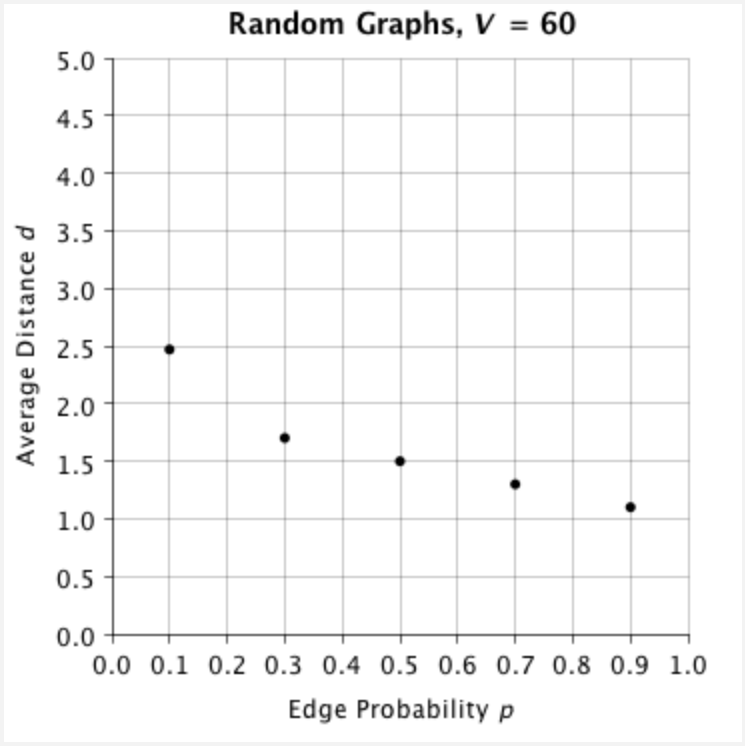
plot-Q6-table.csv:

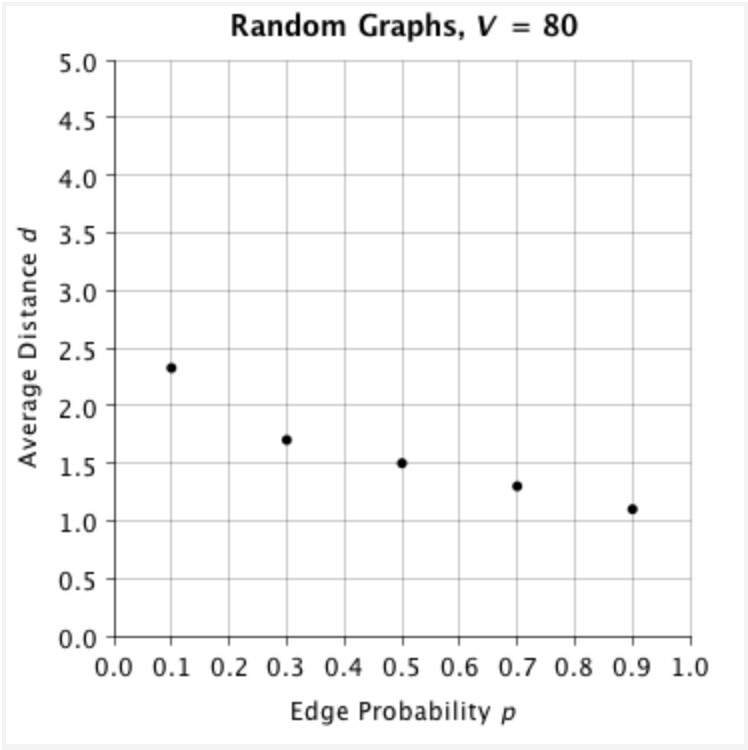
	0.1	0.3	0.5	0.7	0.9
20	3.077801449	1.834329542	1.504263158	1.303210526	1.101368421
40	2.754609476	1.720628205	1.501384615	1.300435897	1.100769231
60	2.469624655	1.702016949	1.499971751	1.299429379	1.100107345
80	2.326933494	1.700822785	1.500022152	1.299648734	1.100221519
100	2.242674747	1.700353535	1.500393939	1.300321212	1.100046465

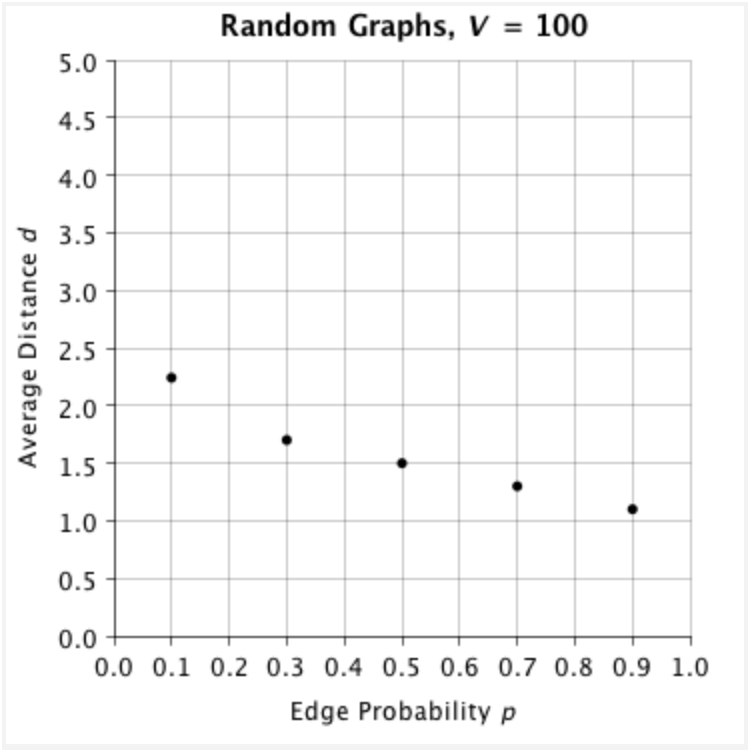
```
java PlotHandler plot-Q6-V-20.dwg plot-Q6-V-40.dwg plot-Q6-V-60.dwg
plot-Q6-V-80.dwg plot-Q6-V-100.dwg
```


Random Graphs, V = 20
Average Distance d
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Edge Probability p


Random Graphs, V = 40
Average Distance d
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Edge Probability p


Random Graphs, V = 60
Average Distance d
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Edge Probability p


Random Graphs, V = 80
Average Distance d
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Edge Probability p

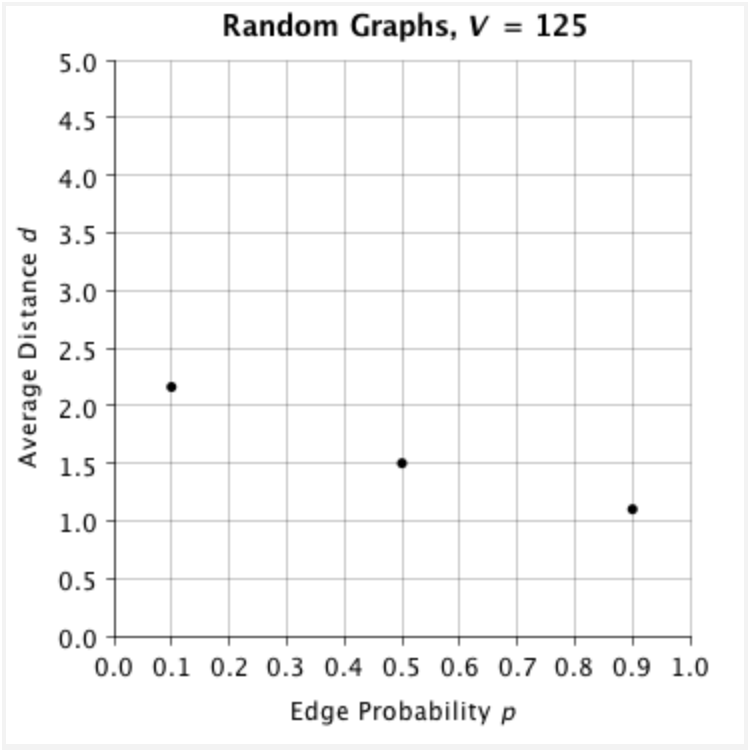

Random Graphs, V = 100
Average Distance d
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Edge Probability p

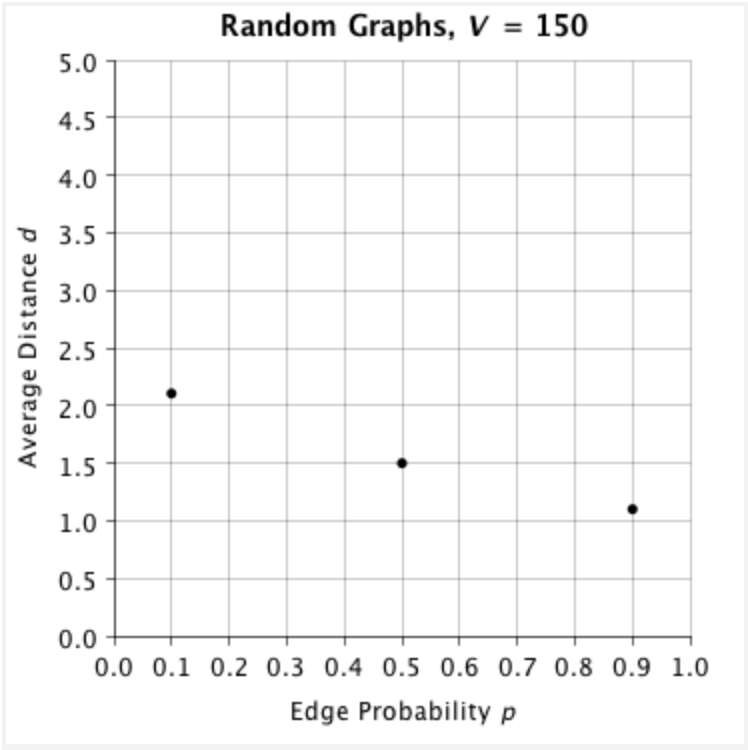
```
java pj2 MonteCarlo 100123456789 125 200 25 .1 .9 .4 10 plot-Q6a
(expected runtime: ~1 minute on quad-core)
```

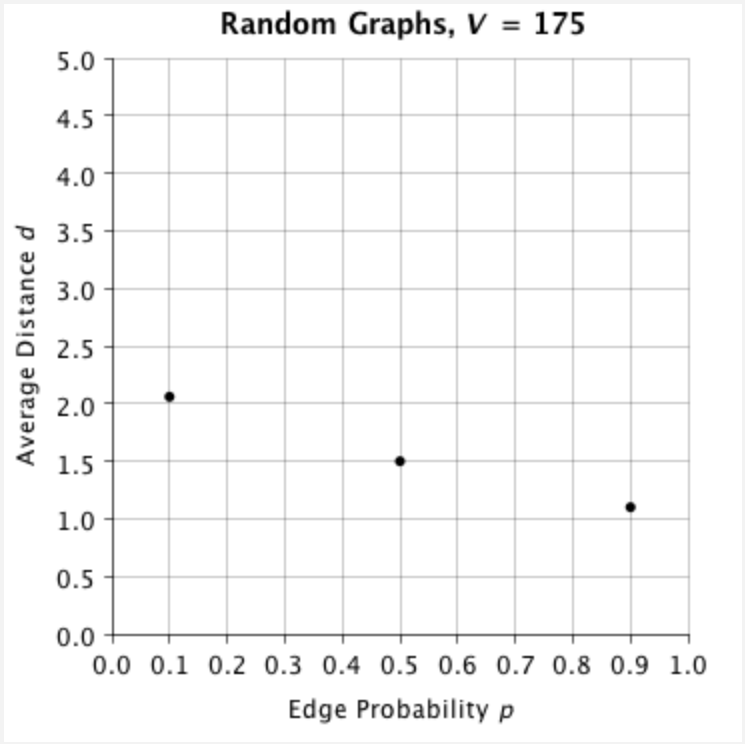
note: To save time, only 10 trials are used to obtain values for average distance with a large number of vertices like 200. Higher numbers used will yield more accurate results, but will take substantially longer to complete.

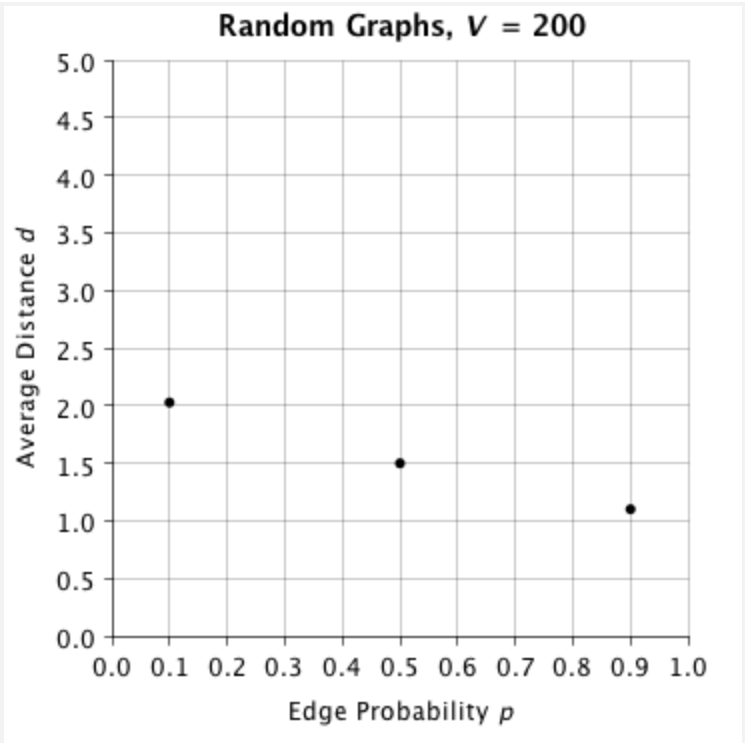
	0.1	0.5	0.9
125	2.165367742	1.500258065	1.101212903
150	2.103740492	1.498765101	1.100384787
175	2.056604269	1.499691297	1.100098522
200	2.022909548	1.500884422	1.100291457

```
java PlotHandler plot-Q6a-V-125.dwg plot-Q6a-V-150.dwg
plot-Q6a-V-175.dwg plot-Q6a-V-200.dwg
```


Random Graphs, V = 125
Average Distance d
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Edge Probability p


Random Graphs, V = 150
Average Distance d
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Edge Probability p





8. What I learned

This project taught me how to explore properties of random graphs that aren't necessarily obvious until thoroughly studied. The method of thorough study that I utilized is called the Monte Carlo method or a Monte Carlo simulation. This project helped me realize the value of using a Monte Carlo simulation for problems that are difficult to theorize or express mathematically. If I were a mathematician, I would have an extremely helpful tool to guide research towards finding a formula to represent average distance for a random graph as a function of *p* and *V*. I was surprised to learn that even in a graph with 100 vertices, the majority of edge probabilities produce distances that are less than 3 hops away (on average). For every number of vertices in a random graph, there is a value for *p* that produces a global maximum for the average distance in the graph. This global maximum increases in value as *V* increases and shifts to the left toward edge probability 0. In the first iteration of this project, I wrote the program single-threadedly. After I realized that it was going to take a long time to get accurate results with higher numbers for vertices, and calculating a greater number of edge probabilities, I decided to implement a parallel loop to harness the full power of what the computer was capable of. I learned that programming with PJ2 speeds up runtime substantially. On a quad core, the runtime was decreased by a factor of at least 3, and on an 8-core computer, the runtime was cut by 6 (at the very least). Also I finally learned how to incorporate PJ2 into eclipse! Thanks, Shane Hale.

Appendix A)

Source code