Jimi Ford
CSCI 351-01 Data Communications and Networks
Programming Project 3 Report

## 1. Program Inputs, Program Objective, Program Outputs

---

**SimulationStation.java**

This class is responsible for generating a network of randomly placed Space Stations in 3D space. It takes in a lower bound for the fewest amount of space stations to include, and an upper bound for the largest amount of space stations to contain in a network. Another parameter (or knob) that gets passed in to this program is the number of trials to run where a trial consists of randomly generating a network, and then extracting the relevant data from that network. Each trial runs Floyd-Warshall on the graph of space stations using the power required to send a signal from one station to another as the weight of an "edge" between two stations. The resulting 2D matrix contains the minimum total power needed for any station *A* to send a message to any other station *B* by asking stations along the shortest path to forward the message along until it reaches *B*. Every value above the reflexive 0 diagonal that is not infinite is averaged over all the trials for a given amount of space stations. This average appears on the plot for the average power needed for that single measurement. This process is repeated for every number of space stations the user requested at runtime. The program knows if a network is fully connected or not if it encounters an edge with infinite weight. A fully connected network has a way for every pair of stations to communicate and an infinitely weighted edge means there is no way for those two stations to communicate. Using this rule, the program keeps a count of how many connected networks it found and calculates the fraction of connected networks over the total number of trials for a given number of space stations. The objective of this program is to study random space station networks, how much power would be needed to communicate in them, and the probability of a space station network being connected given the number of space stations used within a finite 3D space.

**PlotHandler.java**

This program contains the necessary methods to write a plot file given a set of data and then display the results of that plot file. The input to this program is what plot(s) file to display and the output is a graphical representation of the given plot file(s). The objective of this program is to produce a visual or graphical representation of the results of running the above program.

## 2. Exact Command Line

---

**SimulationStation.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 SimulationStation <lower_bound_stations>
<upper_bound_stations> <increment_stations> <num_trials> <seed>
<file_prefix>
```

`<lower_bound_stations>` – Lower bound (inclusive) for number of vertices in space networks
`<upper_bound_stations>` – Upper bound (inclusive) for number of vertices in space networks
`<increment>` – amount to increment the number of space stations by for each simulation
`<num_trials>` – Number of space station networks to simulate
`<seed>` – the seed value for Prof. Alan Kaminsky's PRNG
`<file_prefix>` – Prefix for file output

**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 SimulationStation) to visualize in an X-Y plot

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

---

**4. Based on data generated by running your simulation program(s), discuss this question: What is the probability that the network is connected as a function of the number of stations, and why is the probability behaving this way?**

Looking at *Q5 Supporting Data*, we find an interesting trend in the probability of a network being fully connected depending on how many space stations are in it. The probability begins at about 16% and drastically decreases to 1%. This global minimum value of 1% occurs at 8 space stations. After 8 space stations, the probability of having a connected network shoots up quickly to 90% at about 60 space stations. From there we see the probability approaches 100% asymptotically and after about 175 stations, it is nearly guaranteed that we will have a

connected network. Observing Appendix A confirms that the frequency of 100% connectivity is especially prominent after 175 (or 174) space stations. Let us now look into why the probability of generating a connected graph is behaving this way.

To do this, consider the following oversimplification of our space station network model. This number line now represents the space we are confined to:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

We will represent randomly placing a space station by randomly picking 1 of these numbers.

| 1 | 2 | 3 | 4 | 5 | 6 | **7** | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Say that these numbers (or space stations) can only talk to other numbers that are either equal to themselves or directly adjacent to them. This, of course, models the constraint of the power limitations each space station has. The above green square represents that we picked 7 as our first number. This means that if we want the next number we pick to be able to talk to the first number, we have to pick either 6, 7, or 8 on our next try.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Assuming our method for picking numbers is uniformly random, the chances of us picking one of these numbers is $3/20$. This means with this model we have roughly 15% chance of ending up with a network of numbers of size 2 that is fully connected. The reason it isn't exactly 15% is because if we pick either 1 or 20 for our first number, in each case we only have $2/20$ chance of picking a second number that is either equal or directly adjacent to the first number. If we now decide to pick a third number in hopes of maintaining a connected network, here are the possible outcomes. Green denotes what we picked and yellow denotes the possible numbers for pick #3 that would maintain network connectivity.

**A)**

| 1 | 2 | 3 | 4 | 5 | 6 | **7** | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | **7** | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |

**B)**

| 1 | 2 | 3 | 4 | 5 | 6 | **7** | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | **6** | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**C)**

| 1 | 2 | 3 | 4 | 5 | 6 | **7** | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | **8** | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |

Each outcome, A, B, and C, has equal probability of happening. Recall that the second pick (represented by row 2) had roughly $3/20$ probability of being connected. To calculate the probability of row 3 happening (our third pick), we will look at each case individually.

Case A:

We picked 7 two times in a row. This is allowed since it is consistent with the original problem's constraint. Our original model allowed for space stations to occupy the same 3D space, so we will allow our space stations to occupy the same 1D space in this simplified model. We use $3/20$ for row 2's probability and multiply it by row 3's probability which is also $3/20$ to get $9/400.$ This means that row 3 has only a 2.25% chance of happening. (This is not exact because we are ignoring the special case of picking 1 or 20 first.) We went from roughly 15% chance of being connected with 2 numbers down to 2.25% chance of being connected with 3 numbers.

Case B:

We picked 6 after 7. This now means for our third pick, we have to pick either 5, 6, 7, or 8 in order to maintain connectivity. This has a $4/20$ chance of happening. To obtain the total probability of maintaining connectivity after this third pick we must multiply row 2's chance by row 3's to get $3/20 * 4/20 = 12/400$ which is equal to 3% probability of being connected with 3 random picks. Again this is a rough estimate since we are deliberately ignoring the special case to make this explanation simpler.

Case C:

We picked 8 after picking 7. The probabilistic consequences are identical to Case B.

Since row 2 had equal probability of happening, we can average row 3's probability across each case: $(12/400 + 12/400 + 9/400)/3 = 11/400$ which is 2.75%.

From this, we see that going from 2 random numbers which have roughly a 15% chance of being connected, to 3 random numbers results in a drastic decrease in probability of connectedness. In fact, picking 4 numbers will have an even lower probability of yielding a connected network. The probability will continue to decrease to a global minimum where it will then begin to increase again. This is because the more numbers we pick at random, the more we saturate the playing field with numbers and the more numbers we have to choose from that will maintain a connected network. We can show this visually like so:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

This is a highly probable outcome after picking anywhere from roughly 17~25 numbers. Note that the only 2 numbers left to pick that would yield a disconnected network are 19 and 20. From this, we can easily see that if we pick 1,000 numbers between 1 and 20 at random, we are almost guaranteed that the resulting network would be connected. The reason we are not guaranteed that the network would be connected is because since we are picking randomly, there is a ***very slim*** chance that we will never pick 19.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

In this case, since 19 wasn't picked, we would end up with the number 20 all alone (as long as it was picked at least once) and disconnected from the rest of the numbers.

We can extrapolate this 1D simplification out to 3D in order to explain why the probability of being connected dips down at first as the number of stations increases. It is rather intuitive why the chances of being connected increase after a certain global minimum. The cube of 3D-space that we are limited to begins to become saturated with stations limiting the possibility of placing a station in a position where it would be too far from everyone else to maintain connectivity. This explains why the probability of being connected is fairly high after a certain point in our original 3D space network model and still contains some rare cases where the network is not connected even with many stations. It is easier to spot these rarities in the table included in Appendix A. The plot in Q5 appears to have 100% success in connectedness after about 125 stations.

---

**5. Supporting Data**

---

*note: In order to save you time while grading, lower numbers of simulations are used. As a result, the plots produced are not extremely smooth but are smooth enough to convey the*

*general trend of network connectivity. The results in table form that we are concerned with are in Appendix A under the column labeled "percent_connected". 1 means 100% of the networks were connected, and 0 means none of the networks were connected.*

Commands used:

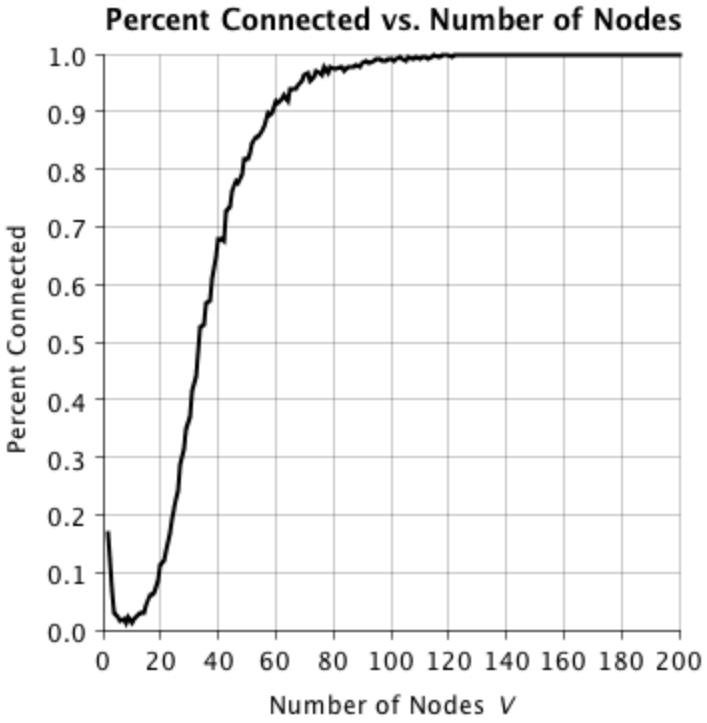
This first command will run the simulations and save *.dwg* files as well as a *.csv* file with the prefix *"space"*. (See Appendix A for this table data)

```
java pj2 SimulationStation 2 200 1 1000 12345019 space
```

```
(expected runtime 4 min 39 seconds with hyperthreading on 2 cores)
(expected runtime 2 min 19 seconds with hyperthreading on 4 cores)
```

After generating the *.dwg* files, you must run the following command to visualize the plots. The probability displayed goes from 0 to 1; 0 being 0 percent chance and 1 being 100% chance.

```
java PlotHandler space-probability-connected.dwg
```




**6. Based on data generated by running your simulation program(s), discuss this question: What is the average total power required to send a message from one station**

**to another as a function of the number of stations, and why is the average total power behaving this way?**

---

Observing *Q7. Supporting Data*, we see a strikingly similarly shaped graph compared to the graphs from project 1. Project 1 was all about the average distance between nodes in a random graph. The power needed to transmit from one station to another contains the distance to the other station in the equation $d^2$. In project 1, the distance was a measure of # of hops. We can equivalently say that the distance was not a measure of # of hops, but rather equally weighted edges where every edge's weight was equal to 1. Scaling this weight by a factor of 40 million and adding kilometers on as a unit gives us a model comparable to our space station model. In project 1, we took the average distance from every pair of vertices where each distance factored in was the minimum distance. In this project, we used Floyd-Warshall to also determine the weight of the shortest path between every pair of vertices. We see that the average power needed starts at a global minimum, increases to a global maximum, and decreases asymptotically. This is because the average distance between stations is doing the same thing. If the average distance for a network increases, since the power needed to transmit a message is dependent on the distance, the average power will increase with this average distance. If the average distance for a network decreases, so will the average power needed to transmit these messages.

## 7. Supporting Data

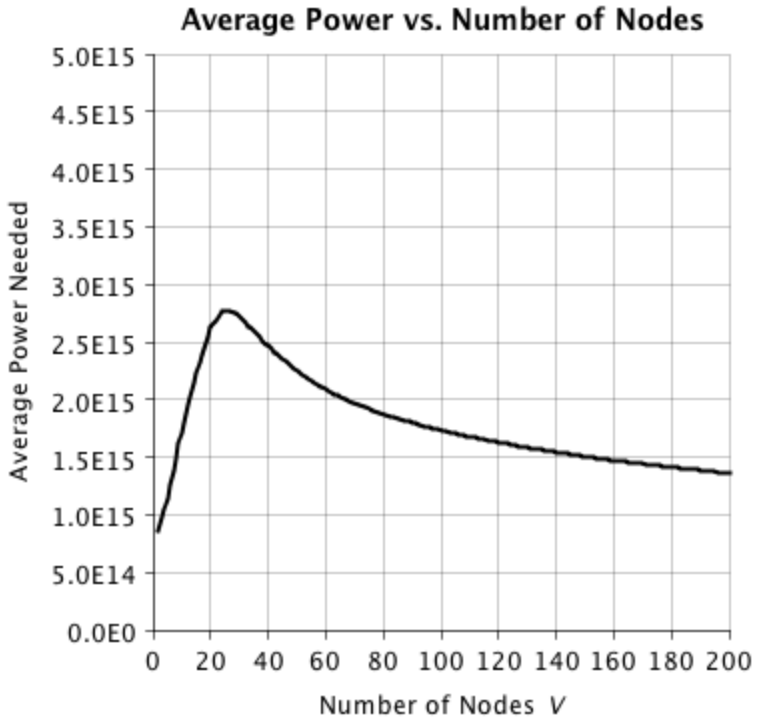
---

*note: In order to save you time while grading, lower numbers of simulations are used. As a result, the plots produced are not extremely smooth but are smooth enough to convey the general trend of the average power needed by each space station. The table results are found in Appendix A the "average_power" column.*

Commands used:

We will use the results generated from the previous run and just view the plot file generated by it:

```
java PlotHandler space-average-power.dwg
```




**8. What I learned**

---

This project was my favorite project we've done so far. This kind of problem is one I often fantasize about and it was fascinating to actually implement a program that modeled this problem and to observe the results for myself. I initially made predictions about what these results would look like: average power needed and probability of being connected. I was not too far off when I predicted that average power would decrease as more and more space stations were added, but I failed to correctly predict the global maximum that happens before the average power decreases. I correctly predicted that the probability of a network being connected would increase asymptotically to 1 (or 100%) but I failed to predict the initial dip in probability for very low numbers of space stations. I now have a heightened appreciation for the effects that the number of nodes have on networks and especially the effect average distance has on communication between nodes in a network. If I worked at NASA, I would now have a tool to accurately predict what would happen if we randomly placed stations out in space. As far as I understand, though, in a real network of space stations, they would need to be orbiting something. We would also not want to randomly place stations in orbit around celestial objects because as we saw, we have absolutely no guarantee that the network would be connected. It would cost us millions, if not billions of dollars to upgrade the capability of the space stations to reach further distances if we didn't plan accordingly and ended up with a disconnected network.

I would also be extremely mindful of calculating the average distance of the network because that is the determining factor of how much power it will take (on average) to communicate within the network.

**Appendix A)**

---

**Table generated**

| num_stations | average_power | percent_connected |
|---|---|---|
| 2 | 8.68E+14 | 0.162 |
| 3 | 9.46E+14 | 0.064 |
| 4 | 1.05E+15 | 0.024 |
| 5 | 1.13E+15 | 0.02 |
| 6 | 1.24E+15 | 0.011 |
| 7 | 1.32E+15 | 0.011 |
| 8 | 1.44E+15 | 0.01 |
| 9 | 1.55E+15 | 0.014 |
| 10 | 1.70E+15 | 0.013 |
| 11 | 1.80E+15 | 0.019 |
| 12 | 1.93E+15 | 0.025 |
| 13 | 2.02E+15 | 0.026 |
| 14 | 2.10E+15 | 0.03 |
| 15 | 2.24E+15 | 0.05 |
| 16 | 2.31E+15 | 0.054 |
| 17 | 2.45E+15 | 0.066 |
| 18 | 2.50E+15 | 0.078 |
| 19 | 2.55E+15 | 0.084 |
| 20 | 2.66E+15 | 0.11 |

| | | |
|---|---|---|
| **21** | **2.65E+15** | **0.122** |
| **22** | **2.68E+15** | **0.126** |
| **23** | **2.74E+15** | **0.17** |
| **24** | **2.77E+15** | **0.19** |
| **25** | **2.74E+15** | **0.222** |
| **26** | **2.75E+15** | **0.233** |
| **27** | **2.75E+15** | **0.279** |
| **28** | **2.73E+15** | **0.306** |
| **29** | **2.71E+15** | **0.334** |
| **30** | **2.69E+15** | **0.38** |
| **31** | **2.70E+15** | **0.403** |
| **32** | **2.66E+15** | **0.451** |
| **33** | **2.64E+15** | **0.483** |
| **34** | **2.62E+15** | **0.517** |
| **35** | **2.58E+15** | **0.541** |
| **36** | **2.55E+15** | **0.583** |
| **37** | **2.52E+15** | **0.577** |
| **38** | **2.50E+15** | **0.622** |
| **39** | **2.46E+15** | **0.648** |
| **40** | **2.46E+15** | **0.69** |
| **41** | **2.43E+15** | **0.691** |
| **42** | **2.40E+15** | **0.702** |
| **43** | **2.39E+15** | **0.74** |
| **44** | **2.36E+15** | **0.746** |
| **45** | **2.33E+15** | **0.753** |
| **46** | **2.31E+15** | **0.78** |

| | | |
|---|---|---|
| **47** | **2.30E+15** | **0.782** |
| **48** | **2.28E+15** | **0.795** |
| **49** | **2.26E+15** | **0.814** |
| **50** | **2.24E+15** | **0.822** |
| **51** | **2.22E+15** | **0.824** |
| **52** | **2.20E+15** | **0.861** |
| **53** | **2.19E+15** | **0.868** |
| **54** | **2.17E+15** | **0.861** |
| **55** | **2.16E+15** | **0.869** |
| **56** | **2.14E+15** | **0.871** |
| **57** | **2.13E+15** | **0.89** |
| **58** | **2.11E+15** | **0.891** |
| **59** | **2.10E+15** | **0.9** |
| **60** | **2.08E+15** | **0.915** |
| **61** | **2.07E+15** | **0.912** |
| **62** | **2.06E+15** | **0.923** |
| **63** | **2.04E+15** | **0.931** |
| **64** | **2.03E+15** | **0.922** |
| **65** | **2.02E+15** | **0.933** |
| **66** | **2.01E+15** | **0.945** |
| **67** | **2.00E+15** | **0.939** |
| **68** | **1.99E+15** | **0.943** |
| **69** | **1.98E+15** | **0.94** |
| **70** | **1.97E+15** | **0.962** |
| **71** | **1.96E+15** | **0.958** |
| **72** | **1.95E+15** | **0.954** |

| | | |
|---|---|---|
| 73 | 1.94E+15 | 0.959 |
| 74 | 1.93E+15 | 0.965 |
| 75 | 1.92E+15 | 0.964 |
| 76 | 1.91E+15 | 0.957 |
| 77 | 1.90E+15 | 0.967 |
| 78 | 1.89E+15 | 0.965 |
| 79 | 1.88E+15 | 0.975 |
| 80 | 1.87E+15 | 0.977 |
| 81 | 1.87E+15 | 0.967 |
| 82 | 1.86E+15 | 0.976 |
| 83 | 1.85E+15 | 0.975 |
| 84 | 1.84E+15 | 0.971 |
| 85 | 1.83E+15 | 0.98 |
| 86 | 1.83E+15 | 0.975 |
| 87 | 1.82E+15 | 0.972 |
| 88 | 1.81E+15 | 0.982 |
| 89 | 1.81E+15 | 0.977 |
| 90 | 1.80E+15 | 0.986 |
| 91 | 1.79E+15 | 0.984 |
| 92 | 1.79E+15 | 0.989 |
| 93 | 1.78E+15 | 0.984 |
| 94 | 1.77E+15 | 0.992 |
| 95 | 1.77E+15 | 0.993 |
| 96 | 1.76E+15 | 0.991 |
| 97 | 1.75E+15 | 0.989 |
| 98 | 1.75E+15 | 0.992 |

| | | |
|---|---|---|
| **99** | **1.74E+15** | **0.991** |
| **100** | **1.73E+15** | **0.988** |
| **101** | **1.73E+15** | **0.989** |
| **102** | **1.72E+15** | **0.994** |
| **103** | **1.72E+15** | **0.996** |
| **104** | **1.71E+15** | **0.994** |
| **105** | **1.70E+15** | **0.991** |
| **106** | **1.70E+15** | **0.995** |
| **107** | **1.69E+15** | **0.994** |
| **108** | **1.69E+15** | **0.992** |
| **109** | **1.68E+15** | **0.995** |
| **110** | **1.68E+15** | **0.994** |
| **111** | **1.67E+15** | **0.995** |
| **112** | **1.66E+15** | **0.997** |
| **113** | **1.66E+15** | **0.994** |
| **114** | **1.66E+15** | **0.998** |
| **115** | **1.65E+15** | **0.999** |
| **116** | **1.65E+15** | **0.997** |
| **117** | **1.64E+15** | **0.998** |
| **118** | **1.63E+15** | **0.999** |
| **119** | **1.63E+15** | **1** |
| **120** | **1.63E+15** | **0.998** |
| **121** | **1.62E+15** | **0.997** |
| **122** | **1.62E+15** | **0.998** |
| **123** | **1.61E+15** | **1** |
| **124** | **1.61E+15** | **0.999** |

| | | |
|---|---|---|
| 125 | 1.60E+15 | 1 |
| 126 | 1.60E+15 | 0.999 |
| 127 | 1.59E+15 | 0.998 |
| 128 | 1.59E+15 | 1 |
| 129 | 1.58E+15 | 0.999 |
| 130 | 1.58E+15 | 0.998 |
| 131 | 1.58E+15 | 0.998 |
| 132 | 1.57E+15 | 0.998 |
| 133 | 1.57E+15 | 0.999 |
| 134 | 1.57E+15 | 0.998 |
| 135 | 1.56E+15 | 0.997 |
| 136 | 1.56E+15 | 0.998 |
| 137 | 1.55E+15 | 0.999 |
| 138 | 1.55E+15 | 1 |
| 139 | 1.54E+15 | 0.999 |
| 140 | 1.54E+15 | 1 |
| 141 | 1.54E+15 | 0.998 |
| 142 | 1.53E+15 | 0.998 |
| 143 | 1.53E+15 | 0.999 |
| 144 | 1.53E+15 | 0.997 |
| 145 | 1.52E+15 | 0.999 |
| 146 | 1.52E+15 | 0.999 |
| 147 | 1.52E+15 | 0.998 |
| 148 | 1.51E+15 | 1 |
| 149 | 1.51E+15 | 1 |
| 150 | 1.50E+15 | 0.998 |

| | | |
|---|---|---|
| **151** | **1.50E+15** | **0.999** |
| **152** | **1.50E+15** | **1** |
| **153** | **1.49E+15** | **1** |
| **154** | **1.49E+15** | **0.999** |
| **155** | **1.49E+15** | **1** |
| **156** | **1.48E+15** | **1** |
| **157** | **1.48E+15** | **0.999** |
| **158** | **1.48E+15** | **1** |
| **159** | **1.47E+15** | **1** |
| **160** | **1.47E+15** | **0.999** |
| **161** | **1.47E+15** | **1** |
| **162** | **1.46E+15** | **0.999** |
| **163** | **1.46E+15** | **1** |
| **164** | **1.46E+15** | **1** |
| **165** | **1.46E+15** | **0.998** |
| **166** | **1.45E+15** | **1** |
| **167** | **1.45E+15** | **0.999** |
| **168** | **1.45E+15** | **1** |
| **169** | **1.44E+15** | **1** |
| **170** | **1.44E+15** | **0.999** |
| **171** | **1.44E+15** | **1** |
| **172** | **1.44E+15** | **0.999** |
| **173** | **1.43E+15** | **0.999** |
| **174** | **1.43E+15** | **1** |
| **175** | **1.43E+15** | **1** |
| **176** | **1.42E+15** | **1** |

| | | |
|---|---|---|
| 177 | 1.42E+15 | 1 |
| 178 | 1.42E+15 | 1 |
| 179 | 1.42E+15 | 1 |
| 180 | 1.41E+15 | 1 |
| 181 | 1.41E+15 | 1 |
| 182 | 1.41E+15 | 1 |
| 183 | 1.41E+15 | 1 |
| 184 | 1.40E+15 | 1 |
| 185 | 1.40E+15 | 1 |
| 186 | 1.40E+15 | 1 |
| 187 | 1.39E+15 | 1 |
| 188 | 1.39E+15 | 1 |
| 189 | 1.39E+15 | 1 |
| 190 | 1.39E+15 | 1 |
| 191 | 1.38E+15 | 1 |
| 192 | 1.38E+15 | 0.999 |
| 193 | 1.38E+15 | 1 |
| 194 | 1.38E+15 | 1 |
| 195 | 1.37E+15 | 1 |
| 196 | 1.37E+15 | 1 |
| 197 | 1.37E+15 | 1 |
| 198 | 1.37E+15 | 1 |
| 199 | 1.37E+15 | 0.999 |
| 200 | 1.36E+15 | 1 |

**Appendix B)**

## Source code

```java
//********************************************************************************
//
// File:    PlotHandler.java
// Package: ---
// Unit:    Class PlotHandler
//
//********************************************************************************

import java.io.File;
import java.io.IOException;
import java.text.DecimalFormat;
import edu.rit.numeric.ListXYSeries;
import edu.rit.numeric.plot.Plot;
import edu.rit.numeric.plot.Strokes;
import edu.rit.util.AList;

/**
 * Class PlotHandler is the delegate for dealing with visualizing the data
 * generated by the "number crunching" program, SimulationStation.
 * Its purpose is to be instantiated in SimulationStation with the data to plot,
 * where the write() method should then be called.
 *
 * Running this program and specifying in the command line arguments the plot
 * files previously generated will open a graphical representation of these
 * plots for each file.
 *
 * @author Jimi Ford
 * @version 4-4-2015
 *
 */
public class PlotHandler {

    // private data members
    private final String averagePowerFile;
    private final String probabilityFile;
    private final AList<SimulationResult> results;

    /**
     * Construct a new plot handler that plots average distances for a fixed
     * vertex count v, while varying the edge probability p
     *
     * @param   plotFilePrefix prefix to be used in the name of
     *          the plot file
     * @param   results collection of results of the finished set of
     *          simulations.
     */
    public PlotHandler(String plotFilePrefix,
            AList<SimulationResult> results) {
        averagePowerFile = plotFilePrefix + "-average-power.dwg";
        probabilityFile = plotFilePrefix + "-probability-connected.dwg";
        this.results = results;
    }

    /**
     * Save the plot information into a file to visualize by running
     * the main method of this class
     *
     * @throws IOException if it can't write to the file specified
```

```
59      */
60     public void write() throws IOException {
61         ListXYSeries averagePowerSeries = new ListXYSeries();
62         ListXYSeries probabilitySeries = new ListXYSeries();
63         SimulationResult result = null;
64         for(int i = 0; i < this.results.size(); i++) {
65             result = results.get(i);
66             if(!Double.isNaN(result.averagePower))
67                 averagePowerSeries.add(result.v, result.averagePower);
68             if(!Double.isNaN(result.percentConnected))
69                 probabilitySeries.add(result.v, result.percentConnected);
70         }
71 
72         Plot powerPlot = new Plot()
73             .plotTitle ("Average Power vs. Number of Nodes")
74             .xAxisTitle ("Number of Nodes <I>V</I>")
75             .xAxisTickFormat(new DecimalFormat("0"))
76             .yAxisTitle ("Average Power Needed")
77             .leftMargin(84)
78             .yAxisTitleOffset(60)
79             .yAxisTickFormat (new DecimalFormat ("0.0E0"))
80             .seriesDots(null)
81             .seriesStroke (Strokes.solid(2))
82             .xySeries (averagePowerSeries);
83         Plot.write(powerPlot, new File(averagePowerFile));
84         Plot probabilityPlot = new Plot()
85         .plotTitle ("Percent Connected vs. Number of Nodes")
86         .xAxisTitle ("Number of Nodes <I>V</I>")
87         .xAxisTickFormat(new DecimalFormat("0"))
88         .yAxisTitle ("Percent Connected")
89         .yAxisTickFormat (new DecimalFormat ("0.0"))
90         .seriesDots(null)
91         .seriesStroke (Strokes.solid(2))
92         .xySeries (probabilitySeries);
93         Plot.write(probabilityPlot, new File(probabilityFile));
94     }
95 
96     /**
97      * Open a GUI for each plot in order to visualize the results of a
98      * previously run set of simulations.
99      *
100     * @param args each plot file generated that you wish to visualize
101     */
102    public static void main(String args[]) {
103        if(args.length < 1) {
104            System.err.println("Must specify at least 1 plot file.");
105            usage();
106        }
107 
108        for(int i = 0; i < args.length; i++) {
109            try {
110                Plot plot = Plot.read(args[i]);
111                plot.getFrame().setVisible(true);
112            } catch (ClassNotFoundException e) {
113                System.err.println("Could not deserialize " + args[i]);
114            } catch (IOException e) {
115                System.err.println("Could not open " + args[i]);
116            } catch (IllegalArgumentException e) {
```

```
117                 System.err.println("Error in file " + args[i]);
118             }
119         }
120     }
121
122     /**
123      * Print the usage message for this program and gracefully exit.
124      */
125     private static void usage() {
126         System.err.println("usage: java PlotHandler <plot-file-1> "+
127                 "(<plot-file-2> <plot-file-3>... etc.)");
128         System.exit(1);
129     }
130 }
131
```

```java
//*********************************************************************
//
// File:    SimulationResult.java
// Package: ---
// Unit:    Class SimulationResult
//
//*********************************************************************

/**
 * Class SimulationResult is designed to be just a data container for recording
 * the results of running <I>n</I> simulations given a number of space stations
 *
 * @author Jimi Ford (jhf3617)
 * @version 4-4-2015
 */
public class SimulationResult {

    /**
     * the percentage of connected networks generated
     */
    public final double percentConnected;

    /**
     * the average power needed by the stations to transmit to any other station
     * in each network
     */
    public final double averagePower;

    /**
     * the number of vertices (i.e. nodes or space stations)
     */
    public final int v;

    /**
     * the number of trials that were run to generate this result
     */
    public final int trials;

    /**
     * Construct a SimulationResult
     * @param v number of space stations
     * @param trials number of trials that were run
     * @param connectedCount the number of connected networks produced
     * @param averagePower the average power needed by each space station to
     *      transmit messages to any other space station in the network
     */
    public SimulationResult(int v, int trials, int connectedCount,
            double averagePower) {
        this.v = v;
        this.trials = trials;
        this.percentConnected = connectedCount / (double) trials;
        this.averagePower = averagePower;
    }
}

```

```java
//********************************************************************************
//
// File:    SimulationStation.java
// Package: ---
// Unit:    Class SimulationStation
//
//********************************************************************************

import java.io.IOException;
import edu.rit.pj2.Task;
import edu.rit.util.AList;

/**
 * Class runs a number of trials simulating a network of space stations
 *
 * @author Jimi Ford (jhf3617)
 * @version 4-2-2015
 */
public class SimulationStation extends Task {

    // java pj2 SimulationStation
    // [0] = <lower_bound_stations>
    // [1] = <upper_bound_stations>
    // [2] = <increment_stations>
    // [3] = <seed>
    // [4] = <file_prefix>

    private static final int
        LOWER_INDEX = 0,
        UPPER_INDEX = 1,
        INCREMENT_INDEX = 2,
        TRIALS_INDEX = 3,
        SEED_INDEX = 4,
        FILE_INDEX = 5;

    /**
     * main method
     * @param args command line arguments
     */
    public void main(String[] args) {
        if(args.length != 6) {
            usage();
        }
        int lowerBound = 0, upperBound = 0, increment = 0, trials = 0;
        long seed = 0;
        String filePrefix = null;
        try {
            lowerBound = Integer.parseInt(args[LOWER_INDEX]);
            upperBound = Integer.parseInt(args[UPPER_INDEX]);
            increment = Integer.parseInt(args[INCREMENT_INDEX]);
            trials = Integer.parseInt(args[TRIALS_INDEX]);
            seed = Long.parseLong(args[SEED_INDEX]);
            filePrefix = args[FILE_INDEX];
        } catch (NumberFormatException e) {
            error("Detected non-numeric input where expected numeric value");
        }
        AList<SimulationResult> results = new AList<SimulationResult>();
        for(int vertices = lowerBound; vertices <= upperBound;
```

```
59                  vertices += increment) {
60              results.addLast(
61                      new Simulator(this, vertices, trials, seed).simulate());
62          }
63          try {
64              new PlotHandler(filePrefix, results).write();
65              new TableHandler(filePrefix, results).write();
66          } catch (IOException e) {
67              error("Error writing results file(s) using prefix: " + filePrefix);
68          }
69      }
70
71      /**
72       * print usage statement and gracefully exit
73       */
74      private static void usage() {
75          System.err.println("java pj2 SimulationStation "
76                  + "<lower_bound_stations> "
77                  + "<upper_bound_stations> "
78                  + "<increment_stations> "
79                  + "<num_trials> "
80                  + "<seed> "
81                  + "<file_prefix>");
82          System.exit(1);
83      }
84
85      /**
86       * print an error message and call the usage() method
87       * @param msg the error message to print
88       */
89      private static void error(String msg) {
90          System.err.println(msg);
91          usage();
92      }
93  }
94
```

```java
//******************************************************************************
//
// File:    Simulator.java
// Package: ---
// Unit:    Class Simulator
//
//******************************************************************************

import edu.rit.pj2.Loop;
import edu.rit.pj2.Task;
import edu.rit.pj2.vbl.DoubleVbl;
import edu.rit.pj2.vbl.IntVbl;
import edu.rit.util.Random;

/**
 * Class is responsible for the majority of the runtime of the program. It
 * generates the given number of networks in parallel by utilizing Prof. Alan
 * Kaminsky's PJ2 library.
 *
 * @author Jimi Ford (jhf3617)
 * @version 4-4-2015
 */
public class Simulator {

    private Task ref;
    private int v;
    private int trials;
    private long seed;
    private IntVbl.Sum countConnected;
    private DoubleVbl.Mean averagePower;

    /**
     * Construct a Simulator
     * @param ref the reference to the main task - necessary for utilizing the
     *      class's parallelFor method
     * @param v number of space stations (or nodes in the graph)
     * @param trials the number of random networks to generate
     * @param seed seed value for the PRNG used in instantiated classes
     */
    public Simulator(Task ref, int v, int trials, long seed) {
        this.ref = ref;
        this.v = v;
        this.trials = trials;
        this.seed = seed;
        countConnected = new IntVbl.Sum();
        averagePower = new DoubleVbl.Mean();
    }

    /**
     * Run all <TT>trials<TT>
     * @return a SimulationResult containing the findings of the given number of
     *          simulations
     */
    public SimulationResult simulate() {
        ref.parallelFor(0, trials - 1).exec(new Loop() {

            Random prng;
            DoubleVbl.Mean thrAverage;
```

```
59              IntVbl.Sum thrCount;
60
61              // (Non-javadoc)
62              public void start() {
63                  prng = new Random(seed + rank());
64                  thrAverage = threadLocal(averagePower);
65                  thrCount = threadLocal(countConnected);
66              }
67
68              // (Non-javadoc)
69              public void run(int i) throws Exception {
70                  SpaceNetwork sn = new SpaceNetwork(prng, v);
71                  if(sn.isConnected()) {
72                      thrCount.item++;
73                  }
74                  sn.accumulatePower(thrAverage);
75              }
76          });
77          return new SimulationResult(
78                  v,
79                  trials,
80                  countConnected.intValue(),
81                  averagePower.doubleValue());
82      }
83  }
84
```

```java
//********************************************************************************
//
// File:    SpaceNetwork.java
// Package: ---
// Unit:    Class SpaceNetwork
//
//********************************************************************************

import edu.rit.pj2.vbl.DoubleVbl;
import edu.rit.util.Random;

/**
 * Class models a network of space stations placed in random positions in 3D
 * space. The space stations' locations are limited to
 * 1E8 million kilometers X 1E8 million kilometers X 1E8 million kilometers.
 *
 * @author Jimi Ford (jhf3617)
 * @version 4-2-2015
 */
public class SpaceNetwork {

    /**
     * maximum dimension value allowed in 3D space
     */
    public static final double MAX_DIM = 1.0E8;

    /**
     * number of space stations
     */
    public final int n;

    // private data members
    private boolean connected;
    private double[][] adj;
    private double[][] shortest;
    private SpaceStation[] stations;

    /**
     * Construct a SpaceNetwork
     *
     * @param prng the pseudorandom number generator to use
     * @param n the number of space stations in this network
     */
    public SpaceNetwork(Random prng, final int n) {
        this.n = n;
        this.adj = new double[n][n];
        this.shortest = new double[n][n];
        this.stations = new SpaceStation[n];
        initStations(prng);
        initAdjacency();
        floydWarshall();
        checkConnectivity();
    }

    /**
     * initialize the coordinates of the <TT>n</TT> stations
     * @param prng the pseudorandom number generator to get random numbers from
     */
```

```
59      private void initStations(Random prng) {
60          double x, y, z;
61          for(int i = 0; i < n; i++) {
62              x = prng.nextDouble() * MAX_DIM;
63              y = prng.nextDouble() * MAX_DIM;
64              z = prng.nextDouble() * MAX_DIM;
65              stations[i] = new SpaceStation(i, x, y, z);
66          }
67      }
68
69      /**
70       * initialize the weights of the edges between nodes with the power needed
71       * to transmit from one station to another
72       */
73      private void initAdjacency() {
74          SpaceStation s1, s2;
75          double distance, power;
76          for(int i = 0; i < n; i++) {
77              adj[i][i] = 0; // not needed
78              s1 = get(i);
79              for(int j = i+1; j < n; j++) {
80                  s2 = get(j);
81                  distance = s1.distance(s2);
82                  if(distance > SpaceStation.MAX_DISTANCE) {
83                      power = Double.POSITIVE_INFINITY;
84                  } else {
85                      power = s1.powerNeeded(s2);
86                  }
87                  adj[i][j] = power;
88                  adj[j][i] = power;
89              }
90          }
91      }
92
93      /**
94       * Run Floyd-Warshall on the space network to determine all-pairs shortest
95       * paths. This will tell us the least amount of power a station needs to
96       * transmit to any other station in the network by forwarding the message
97       * along the shortest path to that station.
98       */
99      private void floydWarshall() {
100         System.arraycopy(adj, 0, shortest, 0, n);
101         double s_i_j, s_i_k, s_k_j;
102         for(int k = 0; k < n; k++) {
103             for(int i = 0; i < n; i++) {
104                 for(int j = 0; j < n; j++) {
105                     s_i_j = shortest[i][j];
106                     s_i_k = shortest[i][k];
107                     s_k_j = shortest[k][j];
108                     if(s_i_j > s_i_k + s_k_j) {
109                         shortest[i][j] = s_i_k + s_k_j;
110                     }
111                 }
112             }
113         }
114     }
115
116     /**
```

```
117      * Check if the network is connected
118      */
119     private void checkConnectivity() {
120         boolean connected = true;
121         double temp;
122         for(int i = 0; i < n && connected; i++) {
123             for(int j = 0; j < n && connected; j++) {
124                 temp = shortest[i][j];
125                 connected = !Double.isInfinite(temp);
126             }
127         }
128         this.connected = connected;
129     }
130 
131     /**
132      * get whether the network is connected or not
133      * @return true if the network is fully-connected
134      */
135     public boolean isConnected() {
136         return connected;
137     }
138 
139     /**
140      * get a space station
141      * @param n the unique identifier of the space station
142      * @return the space station with identifier = n
143      */
144     private SpaceStation get(int n) {
145         return stations[n];
146     }
147 
148     /**
149      * Accumulate the powers needed to transmit messages into a thread-local
150      * copy of a DoubleVbl.Mean. This is what averages the powers across
151      * multiple networks
152      * @param power
153      */
154     public void accumulatePower(DoubleVbl.Mean power) {
155         double temp;
156         for(int i = 0; i < n; i++) {
157             for(int j = i + 1; j < n; j++) {
158                 temp = shortest[i][j];
159                 if(!Double.isInfinite(temp) && temp != 0)
160                     power.accumulate(temp);
161             }
162         }
163     }
164 }
165 
```

```java
//*********************************************************************
//
// File:    SpaceStation.java
// Package: ---
// Unit:    Class SpaceStation
//
//*********************************************************************

/**
 * Class models a space station floating around in 3D space. This class contains
 * the math needed to calculate distances to other stations and the power needed
 * to transmit to them.
 *
 * @author Jimi Ford (jhf3617)
 * @version 4-2-2015
 *
 */
public class SpaceStation {

    /**
     * maximum distance a space station can transmit
     */
    public static final double MAX_DISTANCE = 40.0E6;

    /**
     * the station's x-coordinate
     */
    public final double x;

    /**
     * the station's y-coordinate
     */
    public final double y;

    /**
     * the station's z-coordinate
     */
    public final double z;

    /**
     * the station's unique identifier
     */
    public final int id;

    /**
     * Construct a new space station. It is assumed that all the parameters are
     * less than or equal to MAX_DIM.
     * @param x x-coordinate in 3D space
     * @param y y-coordinate in 3D space
     * @param z z-coordinate in 3D space
     */
    public SpaceStation(int id, double x, double y, double z) {
        this.id = id;
        this.x = x;
        this.y = y;
        this.z = z;
    }

```

```java
    /**
     * compute the straight line distance to another space station
     * @param other the other space station to compute the distance to
     * @return the Euclidean distance to this space station
     */
    public double distance(SpaceStation other) {
        return Math.sqrt(powerNeeded(other));
    }

    /**
     * compute the power needed to transmit to another space station
     * @param other the other space station to calculate the power needed
     * @return the power needed to transmit to the other space station
     */
    public double powerNeeded(SpaceStation other) {
        return  ((other.x - x)*(other.x - x)) +
                ((other.y - y)*(other.y - y)) +
                ((other.z - z)*(other.z - z));
    }
}

```

```java
//********************************************************************************
//
// File:    TableHandler.java
// Package: ---
// Unit:    Class TableHandler
//
//********************************************************************************

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import edu.rit.util.AList;

/**
 * Class handles writing the CSV file containing the results of the simulations.
 *
 * @author Jimi Ford (jhf3617)
 * @version 4-4-2015
 */
public class TableHandler {

    // private data members
    private final String file;
    private final AList<SimulationResult> results;

    /**
     * Construct a Tablehandler
     * @param prefix the prefix of the file name
     * @param results the collective results of the simulations
     */
    public TableHandler(String prefix, AList<SimulationResult> results) {
        this.file = prefix + "-table.csv";
        this.results = results;
    }

    /**
     * write a CSV file containing the results of the simulations
     */
    public void write() {
        SimulationResult temp;
        StringBuilder builder = new StringBuilder();
        builder.append("num_stations, average_power, percent_connected,"+'\n');
        for(int i = 0; i < results.size(); i++) {
            temp = results.get(i);
            builder.append(temp.v + ", " + temp.averagePower + ", " +
                    temp.percentConnected + ", " +'\n');
        }
        PrintWriter tableWriter = null;
        try {
            tableWriter = new PrintWriter(file);
            tableWriter.print(builder.toString());
        } catch (FileNotFoundException e) {
            System.err.println("Error writing table data to file \"" +
                    file+"\"");
        } finally {
            if(tableWriter != null) tableWriter.close();
        }
    }
}
```

59