PRINCIPLES OF COMPUTER NETWORKS
COMP 3203

Evangelos Kranakis
Assignment 2: Project
Sensors and Domain Coverage
December 13th, 2016

Theodore Kachulis
100970278
Michael Stupich
100973305

Introduction

Group Information:

- Theodore Kachulis, 100970278
- Michael Stupich, 100973305

Need to Know:

- Log files are created when the program is run, and our log files are also available in the .zip associated with this document.

- In the .zip, there is an executable jar which, when extracted, will start our program upon being run. This can either be done through **cmd**, or by double clicking the **.jar** file.

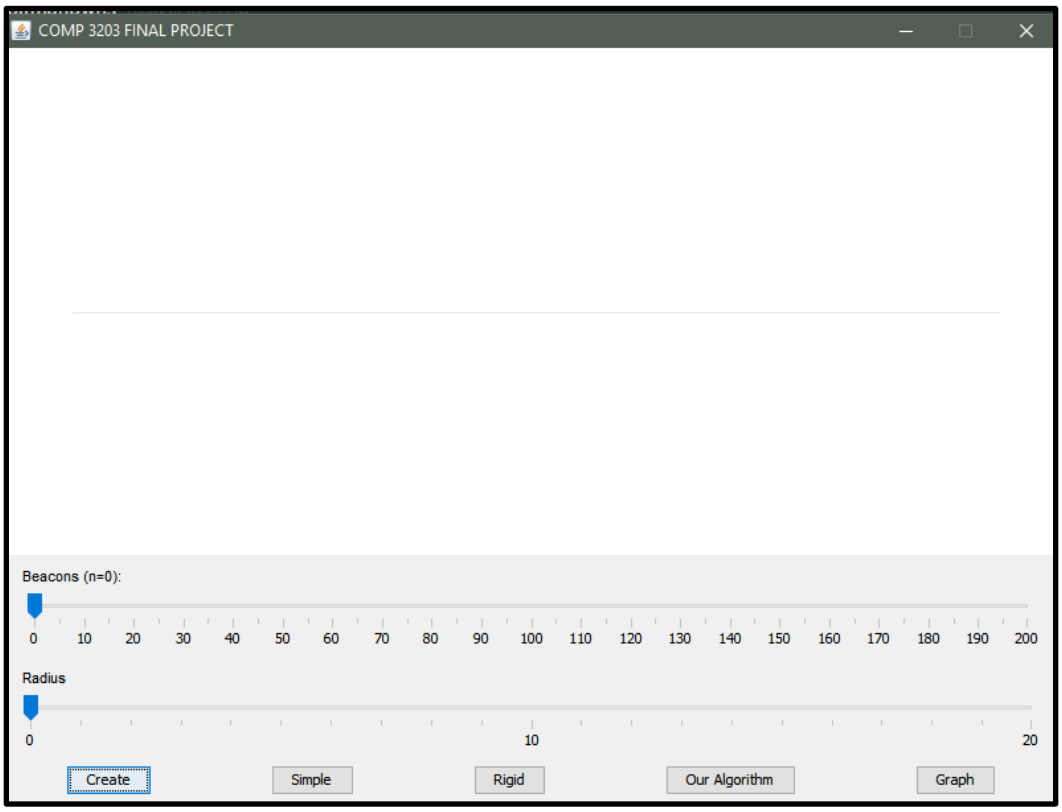
- Also in the .zip you can find our **source code**. Later in this document, an explanation of how to access this project's repository online will be available as well.

- The formatting of this **pdf** documentation is as follows - please refer to the below index section for quick reference to the information this document contains.

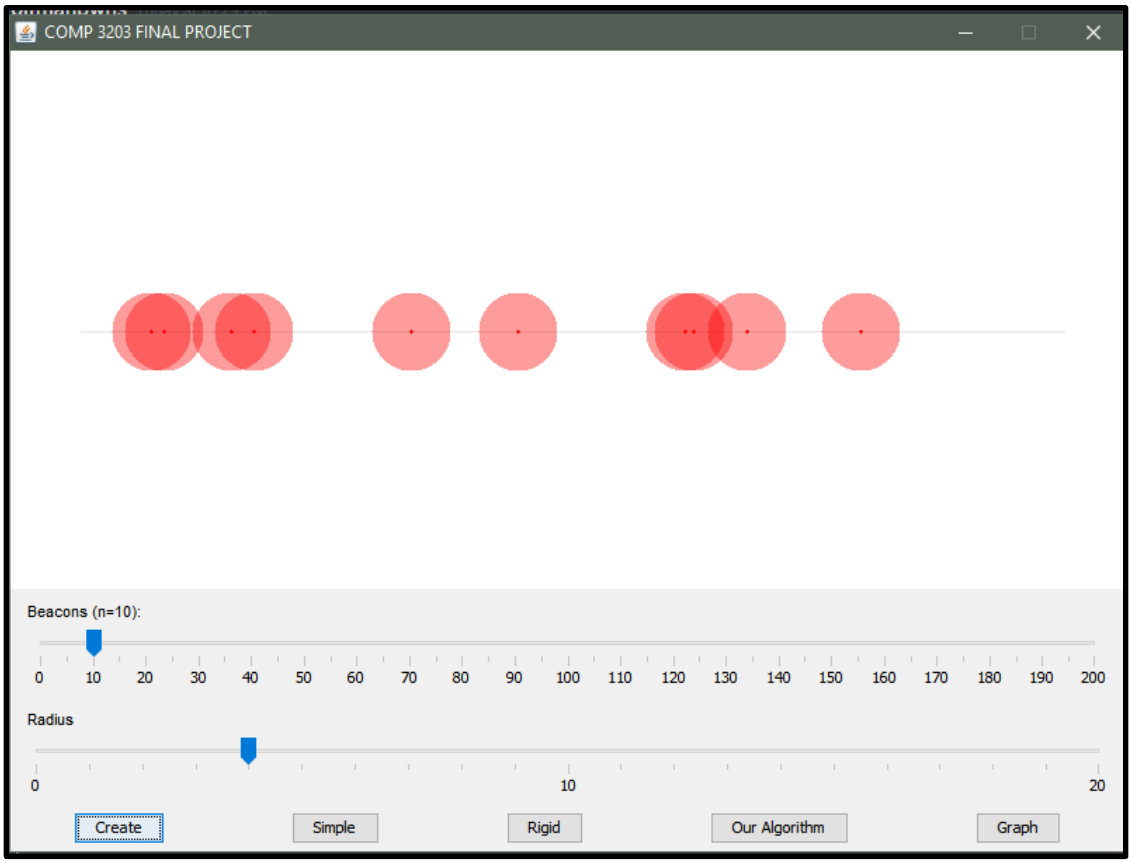
Index:

Part 1: Testing Environment

To test the functionality of our implementations, we needed a reliable display as well as a simplistic user interface. There are two sliders allowing the testing user to change the number of beacons (n) with a maximum of 200, as well as the radius that each beacon has up to a maximum of 20 units. This allows testing to be dynamic, and we are able to test a number of different situations.



Each button on the bottom of the window triggers a unique event. After the user has chosen slider values for the amount of beacons and their respective range, the **create** button will display the randomly placed beacons along the domain line. What follows is an example of a possible distribution of the domain when a user wishes to simulate 10 beacons with a radius of 5 being distributed randomly among the domain line of the network.



After the random network of beacons has been created, the remaining four buttons can trigger other actions which are taken on these beacons. When activated, each protocol shows the individual movements of each beacon before displaying the end result, to best show the user the process of the algorithm. Data regarding the movement of the beacons can be found inside the graph, which is viewed using the **graph** button.

The three remaining buttons are as follows; simple, rigid, and our algorithm. Each is designed to sort the beacons evenly among the length of the domain. Following this, you will find a brief description of each buttons use.

The three remaining buttons function as follows:

- **Simple**
  Triggers the simple protocol implementation algorithm which distributes the beacons evenly across the domain - leaves overlapping beacons in the position they are found.

- **Rigid**
  Triggers the rigid protocol implementation algorithm which distributes the beacons evenly across the domain. Unlike the simple protocol, the rigid implementation shifts all surplus beacons to the right of the domain.

- **Our Algorithm**
  Triggers our protocol implementation algorithm which distributes the beacons evenly across the domain, with overlap. Unlike the simple one however, the distribution of the beacons is uniform.

The following section of this report will include an in depth explanation and analysis of each of the three protocols listed above.

Part 2: Functionality of Protocol Implementations

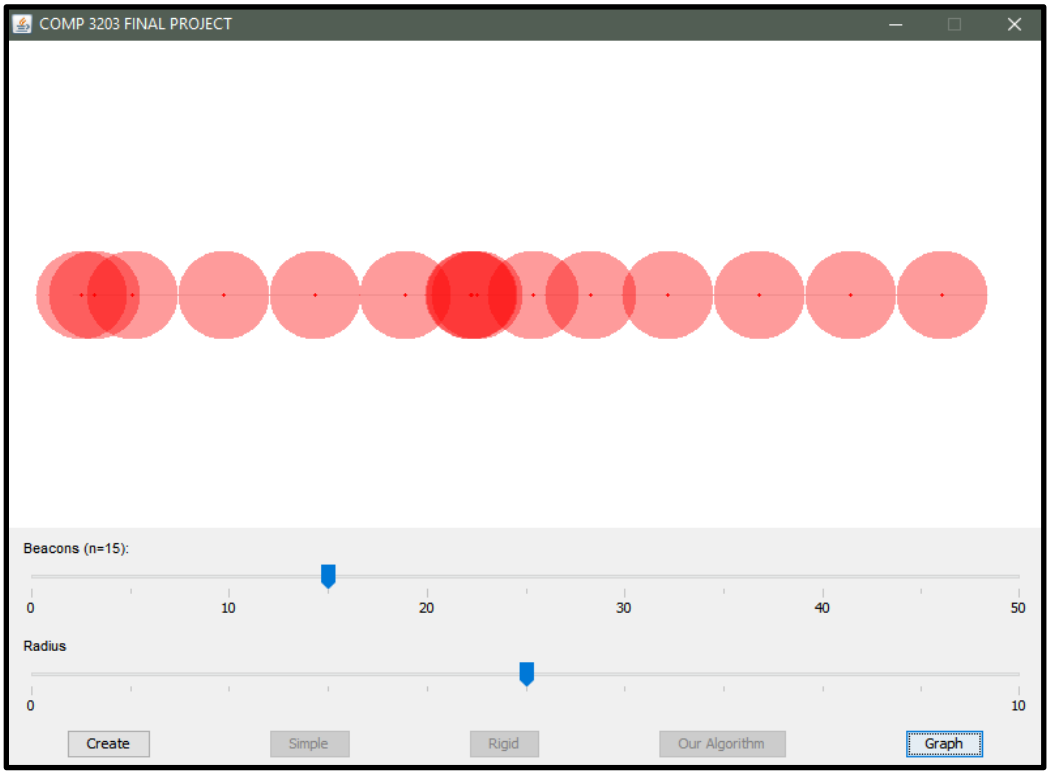
Below, each implementation method will have its own section. In other words, simple, rigid, as well as our own algorithm, will all have their own subsection including an example of functionality, a concise explanation of the implementation protocols, and a brief example of the algorithm in plain language for each.

1.0 - “Simple” Implementation

- **1.1 - Example of Functionality**

  Click **here** for an example of our simple protocol implementation *in motion* (gif).

  The following is the result one would receive when calling the simple protocol on a 15 beacon, 5 unit radius network:



  It is observed that unlike the rigid protocol, several beacons will remain unmoved, or left overlapping with other beacons. This is the main difference in end results between the

two protocols - to see the end result of a ***rigid*** protocol on a network with similar parameters, see **section 2.1**.

- **1.2 - Explanation of Protocols**

  The simple protocol aims to cover the entire area while leaving as much overlap as possible. This keeps as many redundancies as possible across the entire range to prevent errors in the case that a sensor fails. It r takes quite long when compared to the rigid sort as it has to iterate through the beacons both forwards and backwards to get rid of any gaps, however is much more robust in terms of dealing with beacon failure by allowing overlap/redundancies in the network.

- **1.3 - Algorithm in Plain Language (Pseudo Code)**

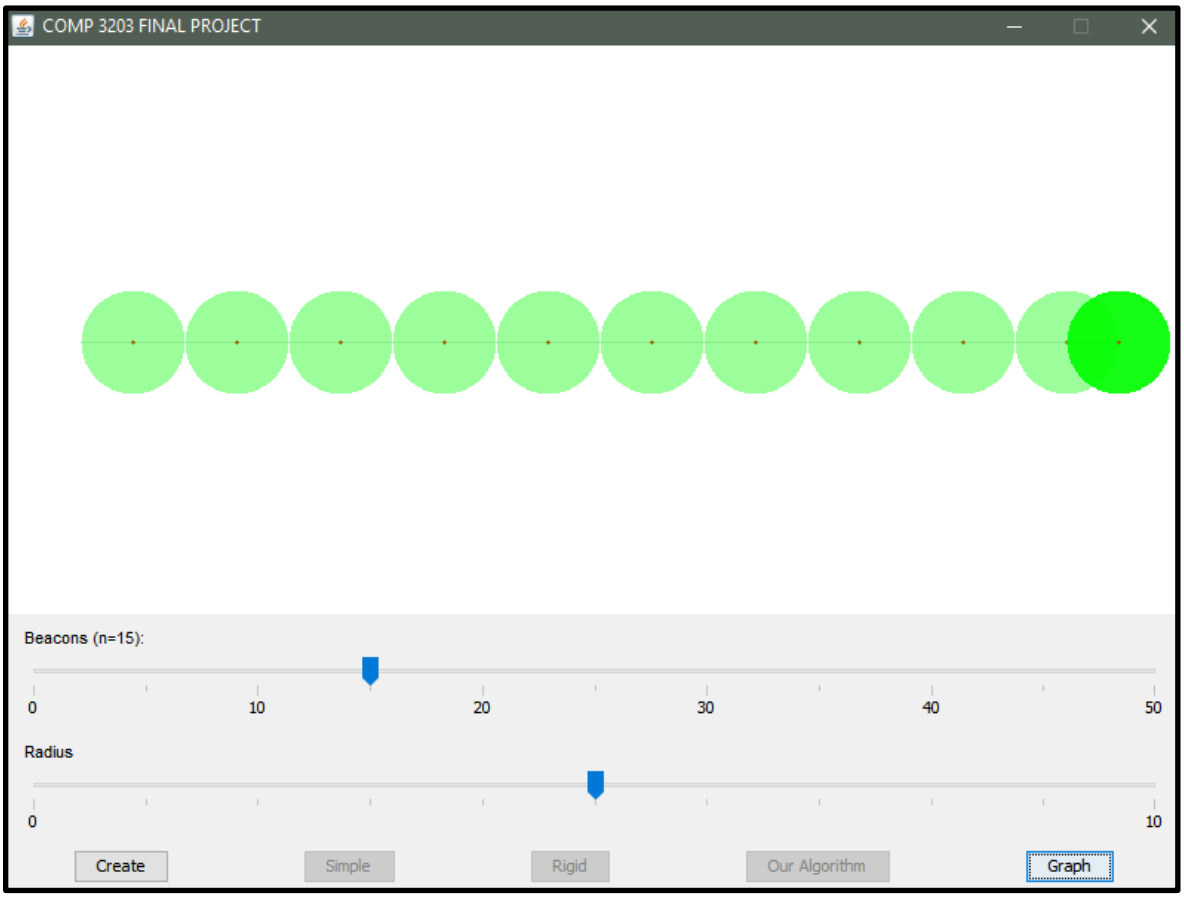
```
Sort beacons from left to right and iterate through them
      if(less than or = to number of beacons needed to fill perfectly
            Arrange: 2*radius away from each other starting at 1*radius
      else
            if(sensors overlapping)
                  leave them as is
            else if (gaps between)
                  Shift next beacon to 2*radius of last sensor
      Sort beacons from right to left and iterate through them
            if gaps of >2*radius
                  move next beacon to 2*radius away from current beacon
```

2.0 - “Rigid” Implementation

- **2.1 - Example of Functionality**

  Click **here** for an example of our simple protocol implementation *in motion* (gif).

  The following is the result one would receive when calling the simple protocol on a 15 beacon, 5 unit radius network:



As stated previously, it is observed that the rigid protocol ensures the displacement of surplus beacons to the right of the domain. Unlike the simple protocol, this rigid implementation does **not** allow for an overlap of beacon range. This is the main difference in end results between the two protocols - to see the end result of a ***simple*** protocol on a network with similar parameters, see **section 1.1**.

- **2.2 - Explanation of Protocols**

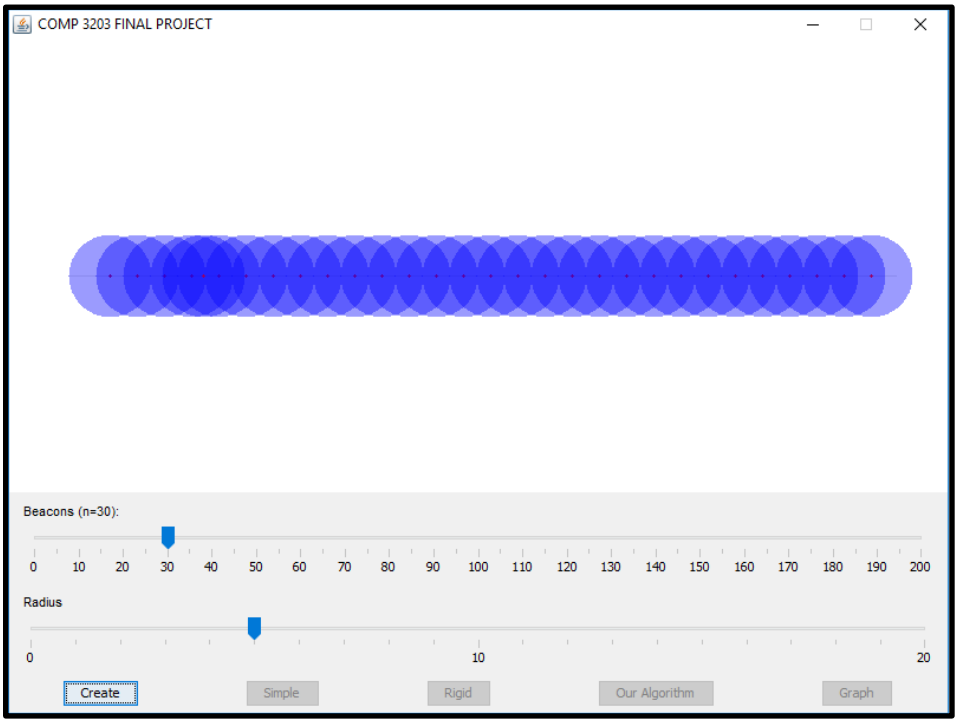
  The rigid protocol aims to cover the entire range by using **n** moves. This is accomplished by iterating through all the beacons, and moving them to cover exactly the range with no overlap except for at the far right. The case of the network having no overlap implies that if a beacon were to fail, there would be no redundancies to compensate. When compared to the simple protocol, this is much faster - however not as robust in a situation where beacons could fail, or transmission is unreliable.

- **2.3 - Algorithm in Plain Language (Pseudo Code)**

```
Sort beacons from left to right and iterate through them
      Arrange each 2*radius away from each other starting at 1*radius
            if (domain is filled)
                  if (excess beacon)
                        Shift excess beacon to the right end
                  else
                        done
            else
                  repeat
```

3.0 - Our Implementation

- **3.1 - Example of Functionality**
  Click **here** for an example of our simple protocol implementation *in motion* (gif).



- **3.2 - Explanation of Protocols**

  Our group thought of a third protocol which would cover the entire domain in a much more robust and efficient way than the Rigid protocol, and in a faster way than the Simple protocol. The protocol works in a way similar to the merge sort - by this, I mean that the protocol will split the beacons evenly across the domain by recursively calling a function on itself, similar to how merge sort works. This protocol will give the most redundancies to the system, while also being much faster than the Simple protocol.

- **3.3 - Algorithm in Plain Language (Pseudo Code)**

```
Call sort function with the beacon list, left and right as parameters
      Center = left+right /2
      Set x value of center beacon to center of the domain
      Call sort function with beacon list, left & center as parameters
      Call sort func. with beacon list, center +1 & right as parameters
```

Part 3: Graphing, Logs, and Results

Below, there are several examples of graphing results comparing the number of movements for each radius when a different amount of beacons is selected. Two graphs, (in separate windows) will populate when the **graph** button is clicked. In this section of the report, you will see first some information about our log files, as well as the graphical snapshots of the results after unique random simulations for protocols called on beacon amounts of n = 10, 25, 50, 100, and 200.

Log Files

Our log files can be found in the zip file with the name **LogFile.log**. Additionally, you can find our log files **here** on github. Note that the log file will also be created when a new program session is initiated, so you should be able to produce your own.

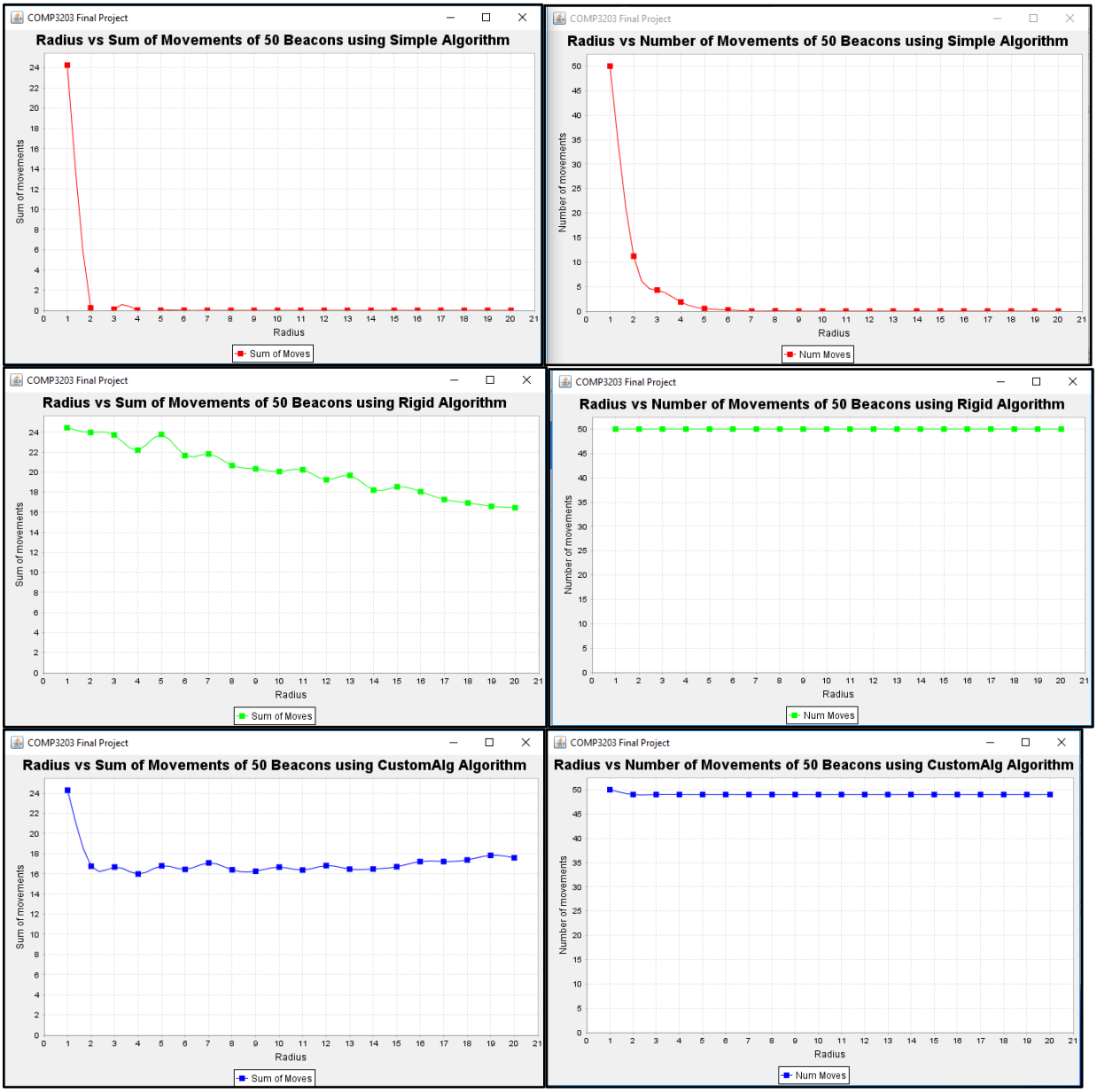
The log files contain an output of xml data, detailing the beacon number, and the action taken on it - i.e, “moved” or “unmoved”, along with other data.

Project Online

Our final project is also hosted on a github repository which we used for development of this applet as a team. The repository hosting our file network can be found at the following URL: https://github.com/Mike-Stupich/Computer_Networks_Final_Project

---

Comparing Graphs: 50 Beacons, Sample size of 20



From looking at these graphs, we can see that the simple protocol is the most effective, our protocol the second most, and the rigid algorithm the least. This is because the simple protocol doesn't move beacons unless it has to. Whereas both our algorithm, and the rigid protocol move all the beacons every time. However, a big difference not shown in the graph is the runtime of each protocol. The simple protocol takes a much longer time to go through the beacons than the other two protocols. This is because the simple protocol has to go both

forwards, and backwards through the list to verify that there are no holes, as well as perform two sorting operations on the list. The differences between our protocol and the rigid protocol are as follows:

- Our protocol has uniform overlap across the domain to provide redundancies in the case that a beacon would fail, whereas in the rigid protocol the excess beacons are all pushed to the right, which is much less beneficial.
- Our protocol performs less movement when 1 < radius < max_radius(20). This is due to the fact that rigid protocol will perform n movements n/2 distance on average, whereas ours will perform n movements of log(n) distance.

From these results we can conclude that the simple algorithm is the most efficient in number of moves and sum of movement. We can also conclude that our protocol is the most efficient in runtime and distribution redundant beacons. Another thing to note is that for the simple and rigid protocol, as the radius increases, the sum of the movements decreases. This is because the larger the radius, the less beacons it takes to cover the domain. In the case of the simple protocol, this means the chance of having to move a beacon as the radius increases, decreases exponentially. In the case of the rigid protocol, this affects the sum of the movements because the first beacon has to move less to get into its position 2*radius away from the previous beacon.

The most notable thing here is that our protocol is unaffected by the radius. This is because we set the position of the beacons to be evenly distributed across the domain by x value, not based on the radius.

A possible protocol to be tested in the future would be one that is based off of both our protocol and the simple protocol in such a way that you only move beacons when necessary, and if you move them, you do so in the style that our protocol does it. This would minimise the runtime of the protocol, and would keep the same (or very similar) sum of movement as the simple protocol.