COMP 3203: Assignment 2

Ryan DiCintio - 100859074

Harth Majeed - 100896761

Abdulbadi Sabir - 100907779

Introduction

Mobile sensor deployment enables efficient access and information retrieval across communication environments. We can simulate random deployments of these sensors to prove reductions in manufacturing costs. The simulation involves deploying *n* sensors randomly over a domain interval [0,1], where every sensor has equivalent ranges, *r*, such that r > 0. The application we implemented displays an animation of sensors moving left or right to form barrier coverage of the domain. Observations from repeated simulations describe trends when we change the parameters n and r. We record the sum of distances all sensors were required to move to cover the entire interval. Pixels are the units we record for distance. Our results show that sensors with approximately 1/10th the interval as a radius require the least amount of movement.

Simulation Details

Each simulation considers two variables: the number of sensors and uniform ranges for the sensors. The sensors move their positions to meet the coverage requirements; sensor status is indicated by green or yellow colour, which represent in position and not in position respectively. After every simulation the domain interval is completely covered by the sensors. We measure the simulation results by calculating the sum of movement between sensors. Results are displayed in a graph by calculating the average values based on a number of simulations.

We consider two styles of coverage to solve our coverage problem: rigid coverage and simple coverage. Each case requires scanning sensors over the interval left-to-right; and, moving sensors from their initial random location if necessary.

Rigid Coverage:

Rigid coverage requires no overlapping between sensor range; therefore, a sensor cannot exist inside another sensor's radius. Sensors move positions right to close gaps and remove overlaps.

Simple Coverage:

Coverage is accomplished by removing gaps between sensor ranges; however, simple coverage ignores overlapping sensor ranges.

Application Implementation

The application was implemented using Java. Classes were written to define sensors, sensor animation, the interval interface which contains sensors, and the chart that displays results of a simulation.

The Sensor class declares the sensor object. A sensor contains a position, radius, and status (indicated by colour). When the system initializes a new sensor it requires an x and y position, along with its radius. Each sensor's coverage is coloured yellow or green to indicate their status as not in position or in position.

The SensorAnimation class creates sensors and a sensor bar. The sensor bar represents the interval which sensors must exist. A paintComponent function draws the sensor bar image and each sensor to the Graphics object. The constructor of this class creates arrays of sensors. When the class is initialized, each sensor is given a random position on the interval. The array of sensors orders sensors based on their random positions from left to right. Depending on the algorithm used, they are assigned a target position based on their index in the previous array.

The Presentation class builds the window that displays the components of application interface. This class contains a main function to launch the application. Additionally, instances used in the system are initialized here and added to the window such as sensor animation, the chart panel containing simulation results, and a status bar.

Rigid Coverage Algorithm

The algorithm works as follows:

```
Initialize rigid algorithm:
    temp j = radius
    Create sensorTarget array where size = number of sensors
    For every sensor
        Current sensorTarget  = j
        j gets the value: j + radius + 1
    For every sensor
        assign current target to corresponding sensorTarget

Start simulation for rigid algorithm:
    For every sensor
        sum the distance required to reach target position
    For every sensor
        If current x position is not the target position
            If current x position is greater than target position
                Move one position left
            else
                Move one position right

    Reset simulation for defined remaining simulations
    End algorithm
```

Simple Coverage Algorithm

This algorithm was not completed for the due date. Please see our intended implementation description in the “implementation difficulties” section.

Functionality

When the application is launched, users are presented with the main screen consisting of 3 components: animation panel, status bar, and chart panel.
Before starting a simulation, the user identifies the number of sensors and their radius for a scenario.

To begin a simulation, the user clicks the start button. Popup windows request values for number of sensors, radius of sensors, and number of times to run the coverage algorithm. The simulation will occur the desired amount of times, which is defined by the user.

The upper portion of the window illustrates the animation panel which contains the interval domain and the sensors that exist in a simulation. During an animation, sensors colours indicate whether they are in their target position; they are green on position. At the end of an animation, the sensors will be in their correct positions according to the type of coverage described in the simulation details.

The lower portion of the window displays a graph representing the results from simulations. Each time a simulation runs the graph updates to display the average number of movements.

Implementation Difficulties

Certain functionality and implementation of the project was left unresolved. We had difficulties implementing simulation of the simple coverage algorithm. If we had managed our time better, more time would have been given to this detail.

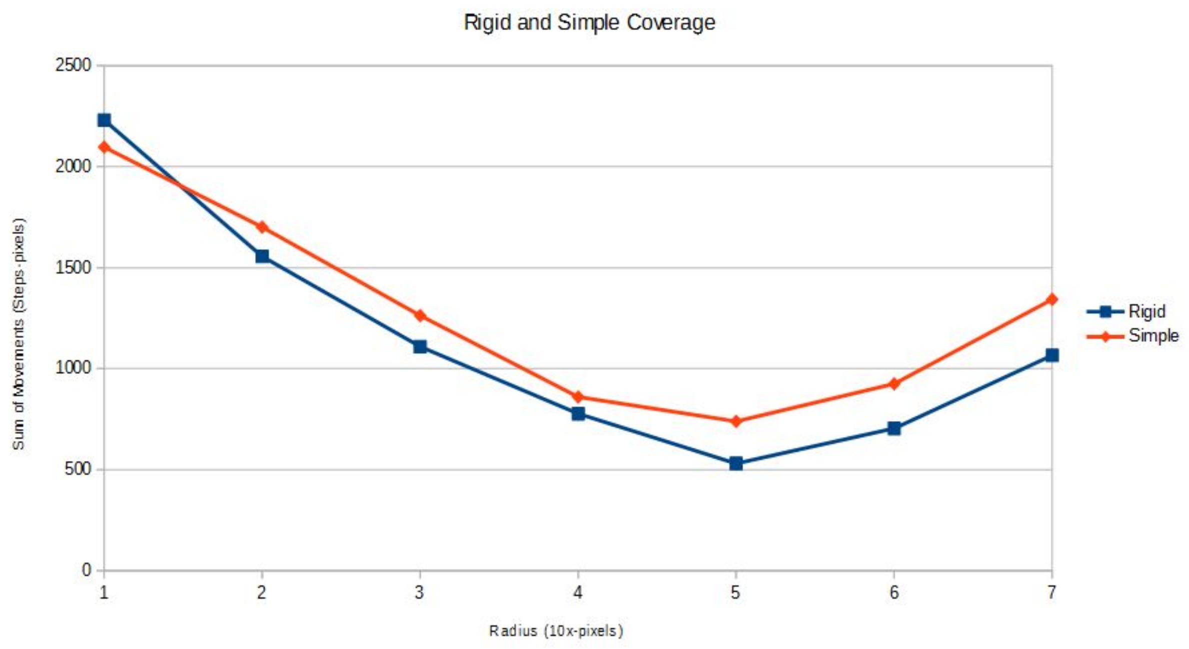
Still, we spent a considerable amount of time thinking about how we would implement the simple coverage algorithm. The first step was finding possible sensors overlapping such that sensor[i-1] and sensor[i+1] were overlapping; this would indicate that sensor [i] could be removed without creating a gap. Additionally, sensors removed in this way would be added to a new array for later use in the case of finding a new gap.

When we approach a gap as we scan the interval, we determine how many sensors are needed to fill a found gap. This could be determined by finding the difference of the gapping sensors and dividing the difference by the radius of the sensors. Our idea proceeding these steps would be to use the sensors in the array of removed sensors to fill these gaps; however, at this point we found ourselves stuck in implementation.

Other possible solutions to the simple coverage algorithm would be to use a threshold for ignoring overlapping. This would be a variable related to the number of sensors in the interval and their range; such that if two sensors overlapped by a large degree then one would instead move to a position as displayed by the rigid simulation. The threshold would ensure that the given amount of sensors would be able to overlap without the algorithm finishing without complete coverage.

Results

When our implementation of the rigid coverage algorithm was completed we ran multiple simulations with varied parameters. Most significant for our research was the relationship between the range of sensors and the movement required for complete coverage. We consistently ran 50 simulations for 10 sensors, each resulted with an average sum of movements for a given radius distance. The radius of a sensor was incremented by 10; and, the maximum sensor radius considered was 70. Although we did not complete the implementation of the simple coverage algorithm, tests were run on the attempted simulation. We used identical parameters for simple coverage as we did for the rigid coverage simulation. The results of our simulations can be found in the graph below.



Conclusion

The results from our simulations display the relationship between sensor range and the required movement between all sensors. The graph suggests the best results appear when the interval is divided between 10 sensors. Although we were not able to complete the simple coverage algorithm, it is likely that its performance may have proved more efficient. These algorithms prove useful for the deployment of sensors across environments where communication is necessary; the movement between them can be minimized using our algorithmic approaches.