**Network Planning: Constrained Access Points Placement (30 points)**

The goal of this problem is to find locations where to place access points (AP) in order to cover an area A. The constraint is that the location of access points is limited to a set of positions L={l1,l2,…,ln}. The goal is to minimize the deployment cost of the wireless network and therefore the number of access points. The reason behind the location constraint is that wireless network operators have to negotiate with buildings to be able to place an access points (or base station). In a practical setting each location has a cost and allows for some coverage (radius R). In this problem we assume that all locations have the same cost and the same coverage.

You will have to find some heuristics/algorithms to select a minimum number of locations from the set L such that any point in the area A is within range from at least one access point. Assume that A is 2000x2000 m2, and the access points range is 200m.

**[10 points] First Step: Randomly generate**L**.**

In practice the set L is given. But to be able to evaluate your algorithms you will need some input data. This part is about randomly generating the input data. There are many ways to generate the input data. Here is one that you can use:

1. Start with L=. While the whole area A is still not covered randomly pick a point that is not covered by any AP and add it the set L.
2. At this point all A is covered. Let m be the size of L. Randomly pick m other locations and add them to L. This second step is to provide some redundancy in coverage.

**[10 points] Second Step: Heuristics.**

1. Heuristic H0. One heuristic to find select locations is as follows: start with L′={}. As long as there are areas within A that are not covered by an AP select an AP outside the covered area and add it to L′. Implement H0.
2. Propose two heuristics (H1, and H2) for selecting L′ a subset of L such that every point in the area is covered. The goal of the heuristics is to minimize L′ (cost of network). Implement the heuristics using Python/Java.

**[10 points] Third Step: Comparison.** Through simulation of steps 1 and 2, compare your two heuristics to H0, to each other and to best(H0,H1,H2). Plot the performance of each heuristic and best(H0,H1,H2) averaged 100 simulation runs. Discuss.

**Heuristics:**

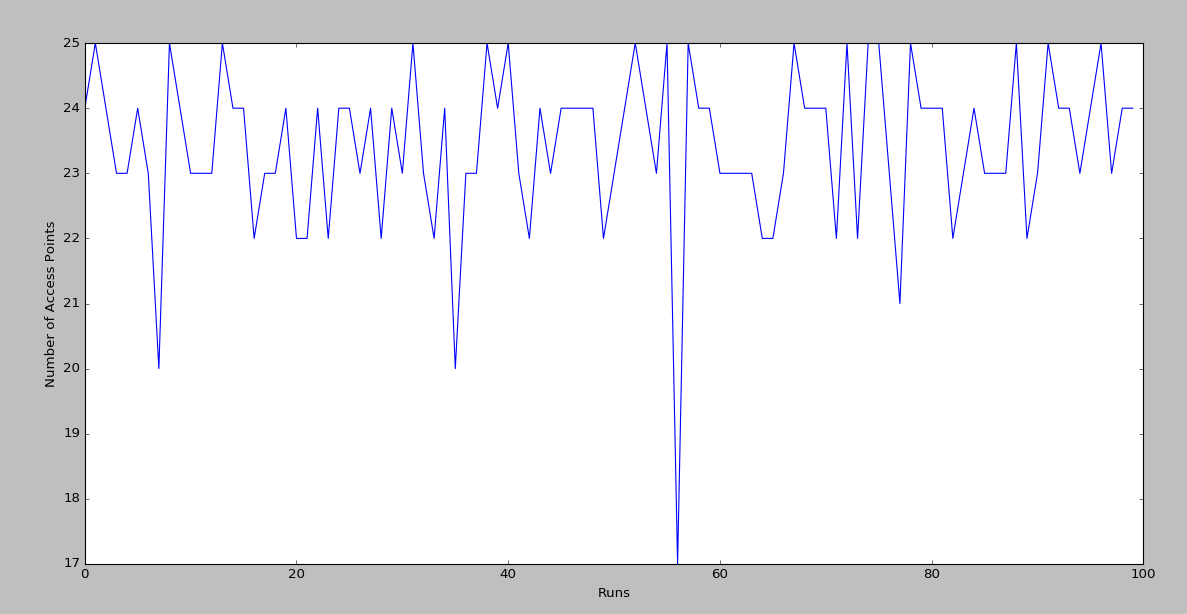
**Heuristic 1:**

Pseudo Code

// Initialize the list  
APList = list()  
// Initialize the XCoverage and YCoverage variables that keep a track of the coverage  
while(currentCoverage != totalCoverage){  
//increment Current XCoverage by taking up the next point  
   
XCoverage+=200  
// Traverse through the list to increment current Coverage in Y-Direction

}

Time complexity of this method is O(n2)



**Heuristic 2:**

Pseudo Code

//Sort the list

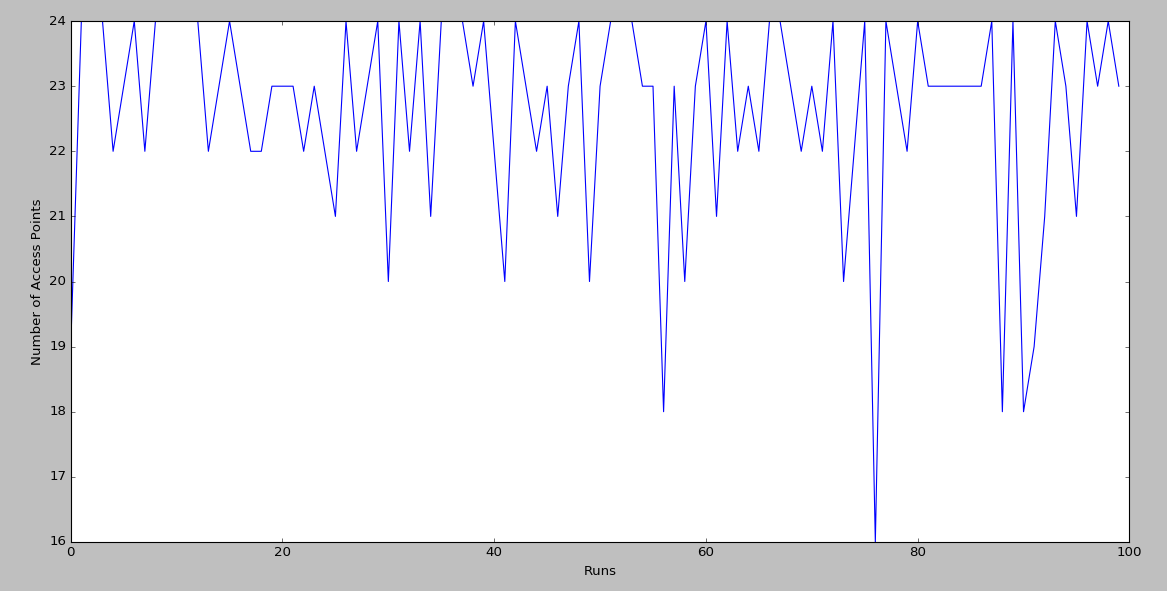
// Sorted list will decrease the number of times the inner loop runs

// Choose list[0] as the starting point  
// Initialize the list  
APList = list()  
// Initialize the XCoverage and YCoverage variables that keep a track of the coverage

//  
while(currentCoverage != totalCoverage){  
//increment Current XCoverage by taking up the next point  
   
XCoverage+=200  
// Traverse through the list to increment current Coverage in Y-Direction

}

Time Complexity of the code is O(nlogn + n2)



**Heuristic 3:**

**Heuristic 3 follows the greedy approach by picking up the first element in the sorted list and finding the farthest element in distance of 400m from it since, that will be least likely to cause significant overlap in the coverage area but still cover the entire area as the coverage radius of AP is 200m**

Pseudo Code

//Sort the list

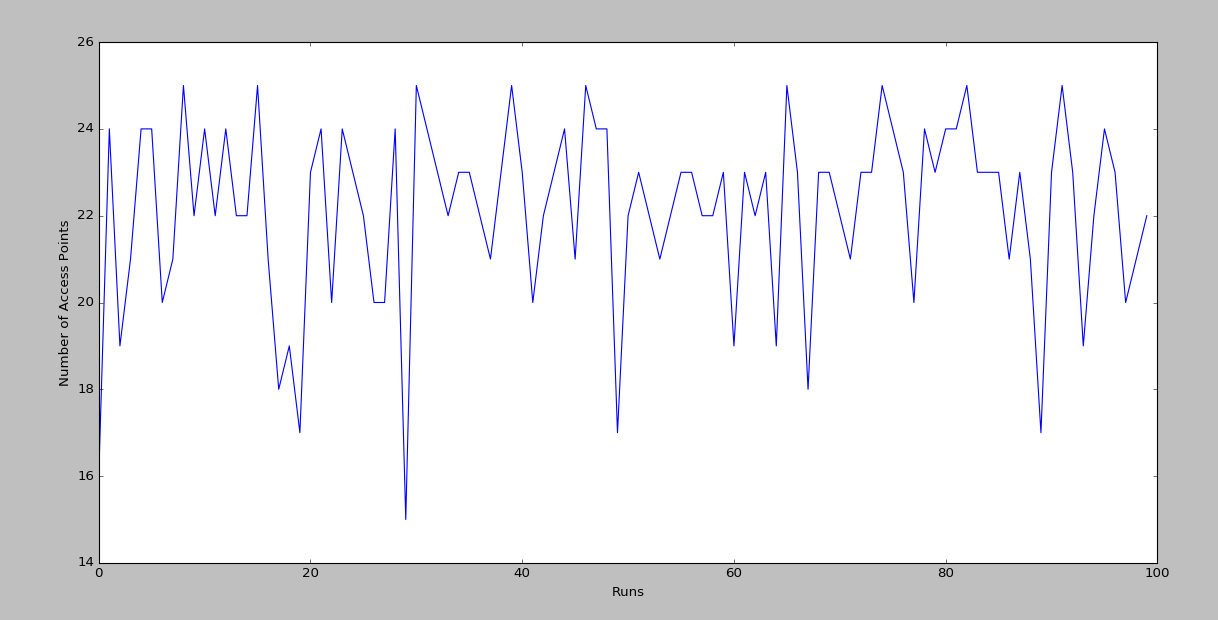
// Sorted list will decrease the number of times the inner loop runs

// Choose list[0] as the starting point  
// Initialize the list  
APList = list()  
// Initialize the XCoverage and YCoverage variables that keep a track of the coverage

//  
while(currentCoverage != totalCoverage){  
//increment Current XCoverage by taking up the next point  
   
XCoverage+=400  
// Traverse through the list to increment current Coverage in Y-Direction

}

Time Complexity of the code is O(nlogn + n2)



**Comparison**:

As it can be seen in the graph, Greedy approach in Heuristic 3 results in best results by decreasing the amount of access points required from Heuristic 2 and Heuristic 1

