Jason Lawrence
A20381993

Project

Section 0. How to Run

To run the code in the terminal or command prompt. Go to the bin directory for this project. Then type into the prompt ./main.py ..\Instances
Where main.py is the file that has the main function for the overall program, and ..\Instances is the directory with all of the instance files are located. The results of the program are stored into the Results directory.

Section 1. PseudoCode

The Greedy Algorithm was chosen for the project. The following is the pseudocode for the algorithm.

```
Def main():
    Set = []  ##O(1)
    Set = greedy(instanceObject) O( C^2P )
    Return Set ##O(1)

Def greedy(instanceObject):
    Set = []
    While true:   ## O(C)
        countPointers(instanceObject)     ## O(CP)
        Center = findMax(InstanceObject) ## O(C)
        Set.append(Center) ##O(1)
        coverPoints(Center, instance) ##O(P)
        If checkStatus(instanceObject) ##O(P)
            Break
        Elif set.length == Centers.length ## *
            Return []
    Return Set
```

* if all of the Centers are in the solution Set then there is no Solution.

```
Def checkStatus(instance):
    For Point in Pointers: ## list of points ##O(P)
        If Point is not covered:
            Return False
    Return True
```

```
Def coverPoints(Center, InstanceObject):
        For Point in Pointers: ## list of points  ##O(P)
                If Center covers Point and point is not covered:
                        point.Covered = true

Def findMax(InstanceObject):
        Max = -1
        Maxid = -1 ## ID of the Center with the most pointers
        For Center in Centers: ## list of  centers ## O(C)
                If Center.Count > max:  ## Center.Count = amount of pointers that center Covers
                        Maxid = Center.id
                        Max = Center.Count
        Return Maxid

Def countPointers(InstanceObject): ## O(C*P)
        For Center in Centers: ##O(C)
                Count = 0
                For Pointer in Pointers:    ## O(P)
                        Distance = sqrt((Pointer.x - Center.x)^2 + (Pointer.y - Center.y)^2)  ## *
                        If Distance is between 0 and 1 and Pointis not already covered:  ## **
                                Count += 1
                                Point.list.append(Center)    ## ***
                Center.Count = Count
```

* The distance Formula is used to tell if the point is in range of the Center.
** Checks the range and to see if the point is already Covered by a Center in the Solution Set.
*** Point.list attribute is used to see if the given Center Covers that point in the CoverPoints function

Section 2. Runtime Analysis

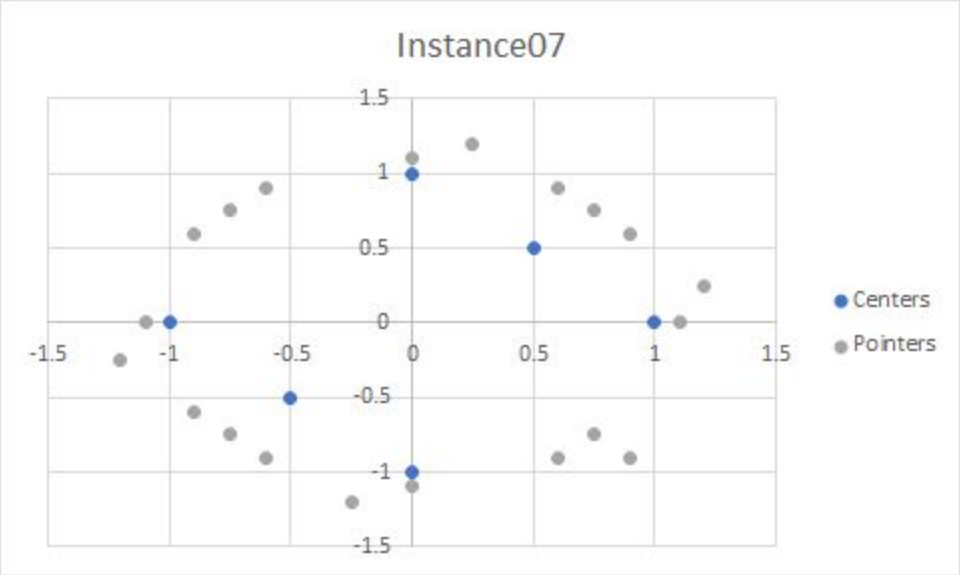
My implementation of the algorithm runs in O(C^2P). In the worst case scenario the while loop in the greedy() function will at most run C times which means all of the functions within that while loop will also run C times. Out of all of the sub functions in the while loop the countPointers() function has the greatest runtime of O(C * P) so multiplying out the C from the while loop you get O(C^2P). The following table contains all of the runtimes of the sub functions

Function	Runtime Analysis
main()	O(C^2P + C^2 + 2CP)
greedy()	O(C^2P + C^2 + 2CP)

countPointers()	O(C * P)
findMax()	O(C)
coverPoints()	O(P)
checkStatus()	O(P)

Section 3. Example of an Instance where the Greedy Algorithm Fails.

Looking at the provided instances, Instance07 does not provide the optimum solution when the greedy algorithm is ran on it.
The following graph depicts what instance07 looks like.



Here is the run of the algorithm.

Iteration	countPointers (Center.id = pointers)	findMax	coverPoints	checkStatus

1	1 = 7 2= 7 3 = 8 4 = 8 5 = 8 6 = 8	3	8	False
2	1 = 2 2= 7 3 = 0 4 = 8 5 = 5 6 = 5	4	16	False
3	1 = 2 2 = 2 3 = 0 4 = 0 5 = 2 6 = 2	1	18	False
4	1 = 0 2= 2 3 = 0 4 = 0 5 = 0 6 = 2	2	20	True

The solution that the greedy algorithm provides for this instance is S = [3, 4, 1, 2]
A better solution is S = [1, 4, 6]

Section 4. Project Report

Each instance was ran three times and the averages are given below in seconds.
Running times in seconds

Instance	Time in seconds	Size in KB
instance01	0.0043214956666666667	5
instance02	0.0728054824	79
instance03	0.1113684958666666667	8
instance04	0.0236034393266666667	4

instance05	0.033576647301	6
instance06	2.4541021185333333334	39
instance07	0.000997543332	1
instance08	0.0	1
instance09	0.000997304916381836	1

The data above was gathered using pythons time module. More specifically time.time() was used. For instances 7, 8 and 9 sometimes the algorithm would run so fast that it would return 0.0. In the case of instance08 this happened every single trial.

As far as the size of the instance files are concerned the smaller the file the faster the algorithm is. The only deviation from this is instance06. This is probably due to the fact that Instance06 has more centers that instance02. My Implementation depends more on how many centers there are more than the amount of points.