CSE 551 – Foundations of Algorithms

Spring 2019

PROJECT FINAL REPORT

TEAM MEMBERS

VINISHA SUKAMETI (1216125638)
YASH JAIN (1215175494)
SUMIT RAWAT (1216225348)
HARSH LALWANI (1215204614)

INTRODUCTION

The goal of relay node placement problem is to maximize "connectedness" in a wireless sensor network by deploying limited number of relay nodes subject to a fixed budget constraint. Here, we define "connectedness" by two metrics:

- First metric [1] is to measure number of connected components in the network. Fewer number of connected components accounts to higher degree of connectedness.
- Second metric [2] is to measure the size of largest connected component in the network. Larger size of largest connected component accounts to higher degree of connectedness.

PROBLEM FORMULATION

Budget Constrained Relay Node Placement with Minimum Number of Connected Components (BCRP-MNCC)

Given the locations of n sensor nodes in the Euclidean plane,
P = {p1, p2, . . . , pn}, positive integers C, R as communication range of sensor nodes and relay nodes, and a budget B1 on the number of available relay nodes. Objective is to find a subset of Q = {q1, q2, . . . , q|B1|} points, in the same plane where relay nodes can be deployed, so that the number of connected components in the graph G0 = (V 0 , E0) corresponding to the point set P and Q is at most C.

Budget Constrained Relay Node Placement with Maximum size of Largest Connected Component (BCRP-MLCC)

Given the locations of n sensor nodes in the Euclidean plane P = {p1, p2, . . . , pn}, positive integer C, R as communication range of sensor nodes and relay nodes and a budget B2 on the number of available relay nodes. Objective is to find a subset of Q = {q1, q2, . . . , q|B2|} points in the same plane where relay nodes can be deployed, so that the size of the largest connected component in the graph G0 = (V 0 , E0) corresponding to the point set P and Q is at least C.

PROBLEM SOLUTION

(1) Solution for BCRP-MNCC

One way to find a graph with high level of connectedness is to create a complete graph for given terminal nodes. Using this graph, we construct a MST T' and assign a weight to every edge based on the Euclidean distance between the two edges of the node. The weight of an edge represents the number of relay nodes required to connect two nodes of the edge. Using this information, we evaluate every edge and compare it with the range R of the nodes. If the length of the nodes is greater than R, we will have to assign a relay node for successful connection else we know that the nodes are connected. Very likely, the number of edges that require relay nodes may exceed the number of relay nodes available 'B'. To deal with this shortage, we remove edges from the tree with maximum weights until the demand - supply is met. The resultant tree is the highly connected graph as per definition **[1]**

Prim's algorithm for minimum spanning tree

Prim's algorithm is a Greedy algorithm that starts with an empty tree and vertices are added one by one on finding the closest vertex. To start with, we select a vertex randomly and add it to MST. Then, at every step we create a candidate set of vertices that are connected to nodes in MST and select the one with minimum weight. The selected node is added to MST. We repeat this until all nodes of the graph are added to the tree.
The idea behind Minimum spanning tree is to connect two disjoint subsets using a parameter (weighted edge in our case).

Algorithm:

Step 1: For given terminal points, construct a Minimum Spanning Tree T'.
Step 2: For each edge E of T' assign weight using:
Weight = ceil (length(E)/R) - 1
Step 3: If sum of weight of all edges > Available relay nodes, (ie. budget is lesser than required nodes) goto step 4. Else goto step 5.
Step 4: Using greedy approach find edge with maximum weight in T' and remove it. Goto Step 3.
Step 5: Return resulting T'

(2) Solution for BCRP-MLCC:

As the name suggests, BCRP-MLCC problem tries to maximise the largest connected component in the graph. To achieve this, we need to compute a minimum spanning tree for a subgraph that connects k vertices at the least (initially k= the number of terminal points). This is equivalent to the k-Minimum Spanning Tree problem. The weights of the edges in the resulting k-MST are calculated as: *ceil(length(E)/R) - 1*. The length of the edge between two terminal points (length(E)) is the measure of the number of relay nodes that are required to connect the two sensors at the end of this edge. If the number of relay nodes required for this k-MST comes out to be less than the budget, then we have found our solution for the problem. Otherwise, we decrement the value of k and compute another k-MST until we find our solution.

To compute k-MST:

It is known that finding the optimal solution for k-MST is NP-hard as its computational complexity is very high. As a workaround, we use an approximation algorithm to solve this problem. An approximation algorithm will provide a solution to the k-MST problem in polynomial time, but this solution will not necessarily be the optimal solution.

k-MST using Branch and Bound: Branch and bound algorithm considers all possible set of candidate solutions and discards them based on the lower and upper bound of the evaluation parameter. In our case, this parameter is evaluated using *ceil(length(E)/R) – 1*, and all candidates whose sum exceed the budget of relay nodes are discarded.

Algorithm:

Step 1: Set the value for k equal to n (the number of terminal points).
Step 2: For a subgraph G' with k terminal points, construct an approximate k-Minimum Spanning Tree
Step 3: For each edge E of G' assign weights to the edges using:
Weight = ceil(length(E)/R) - 1

Step 4: If the sum of weight of all edges <= Available relay nodes, (ie. the required number of relay nodes are less than the budget) go to step 6. Else go to step 5.
Step 5: If k=2, go to step 7. Else, decrement k by 1 and go to step 2.

Step 6: Return this k-MST as the solution for the BCRP-MLCC problem and terminate.
Step 7: Return any random sensor point as solution.

INPUT FORMAT

Code expects file path as input.
Details on file content and input pattern is as below:

For BCRP-MNCC:

```
11                  //Number of nodes in graph
300                 //Value of R
4                   //Value of B
0 1 1000            //node1 node2 length
0 2 1000            //node2 node5 length
0 3 1000            . .
. .                 . .
. .                 //nodei nodej length
```

For BCRP-MLCC

```
11                  //Number of nodes in graph
300                 //Value of R
4                   //Value of B
0 1 1000            //node1 node2 length
0 2 1000            //node2 node5 length
0 3 1000            . .
. .                 . .
. .                 //nodei nodej length
```

CODE EXECUTION STEPS

Programming Language – Python

For BCRP-MNCC:
execute command:
python Algorithm4.py

For BCRP-MLCC:

execute command:

python Algorithm5.py

RESULTS

For BCRP-MNCC:

Input File:

11
300
4
0 1 1000
0 2 1000
0 3 1000
0 4 1000
0 5 1000
0 6 1000
0 7 1000
0 8 1000
0 9 1000
0 10 1000
1 2 778
1 3 94
1 4 288
1 8 948
2 4 740
2 9 902
3 6 270
3 7 578
4 5 772
4 7 751
4 9 395
5 9 893
5 10 640
6 7 46
6 8 152
6 9 714
6 10 847
7 8 765
7 9 408

7 10 67
8 9 931
9 10 716

Output:

```
(venv) C:\Users\vinis\PycharmProjects\FoaAlgorithml>python Algorithm4.py
Enter path of input file
C:\Users\vinis\PycharmProjects\FoaAlgorithml\sampleinput.txt
Resulting Forest for BCRP-MNCC is [[1, 3, 0], [3, 6, 0], [6, 7, 0], [7, 10, 0], [6, 8, 0], [1, 4, 0], [4, 9, 1], [1, 2, 2]]
```

For BCRP-MLCC:

Input:
11
300
4
0 1 1000
0 2 1000
0 3 1000
0 4 1000
0 5 1000
0 6 1000
0 7 1000
0 8 1000
0 9 1000
0 10 1000
1 2 778
1 3 94
1 4 288
1 8 948
2 4 740
2 9 902
3 6 270
3 7 578
4 5 772
4 7 751
4 9 395
5 9 893
5 10 640
6 7 46
6 8 152

6 9 714
6 10 847
7 8 765
7 9 408
7 10 67
8 9 931
9 10 716

Output:

```
(venv) C:\Users\vinis\PycharmProjects\FoaAlgorithml>python Algorithm5.py
Enter Path of input file
C:\Users\vinis\PycharmProjects\FoaAlgorithml\sampleinput2.txt
Resulting Forest for BCRP-MLCC problem is  [[1, 3, 0], [3, 6, 0], [6, 7, 0], [7, 10, 0], [6, 8, 0], [1, 4, 0], [4, 9, 1], [4, 5, 2]]
Size of Largest connected component is  9
```

CONCLUSION

1. As per metric [1] in Introduction, we maximized the connectedness of graph by dividing the graph in minimum number of connected components using Prims Algorithm and Algorithm 4 on BCRP-MNCC.

2. As per metric [2] in Introduction, we maximized the connectedness of graph by finding connected component with largest size using Branch and Bound Algorithm.

PEER EVALUATION

Member Name	Contribution	Marks
VINISHA SUKAMETI		
SUMIT RAWAT		
HARSH LALWANI		
YASH JAIN		

REFERENCES

[1] A. Mazumder, C. Zhou, A. Das and A. Sen, "*Budget Constrained Relay Node Placement Problem for Maximal "Connectedness"*", MILCOM 2016.

[2] https://coderbyte.com/algorithm/find-minimum-spanning-tree-using-prims-algorithm

[3] F. A. Chudak, T. Roughgarden, and D. P. Williamson, "*Approximate k-msts and k-steiner trees via the primal-dual method and lagrangean relaxation*," Mathematical Programming, vol. 100, no. 2, pp. 411–421, 2004.

[4] https://en.wikipedia.org/wiki/K-minimum_spanning_tree

[5] https://networkx.github.io/documentation/stable/reference/generators.html

[6] https://blog.rieder.io/solving-kmst-using-branch-and-bound/