An Application to Network Design

Saideep Sambaraju, MS in Computer Science, The University of Texas at Dallas,
email:sxs125932@utdallas.edu

***Abstract*—This project tackles the problem of network design using a basic algorithm. In a network made up of V nodes, there is a cost associated with connecting any two nodes as well as the demand of traffic flow between any two nodes in the network. The purpose of this project is to design a network of these V nodes in such a way that the overall cost of connecting these nodes is kept at a minimum while making sure that the traffic flow demands between any two nodes are met by this network.**

I. INTRODUCTION

THIS project attempts to solve the problem of cost effective network design. Consider a network of V nodes. Any pair of nodes (i, j) amongst these V nodes would have a unit cost a_{ij} associated with connecting these two nodes and a demand value b_{ij} associated with the demand for the traffic flow from i to j.

Let's say the capacity we implement on a link (i,j) is z_{ij}. This capacity is calculated by summing all the demand values b_{kl} which use the link (i,j) along the path from k to l.
Then the overall cost of the network Z can be given as

$$Z = \sum a_{ij} z_{ij}$$

The task of this program is to construct these links such that the overall cost of constructing the links is minimized while the traffic demand value b_{ij} between any two nodes i and j are met.

II. APPROACH

We solve this problem by a simplified approach. Given two nodes i and j the traffic demand is given by b_{ij}. The cheapest way to transmit this traffic from i to j along the path in such a way that the sum of unit costs of edges along this path $\sum a_{kl}$ is minimum.

This is achieved by first finding the shortest path from i to j taking a_{kl} values as edge weights and then computing the overall cost to transmit data from i to j which will be

$$\text{Overall cost to transmit data from i to j} = b_{ij} \sum a_{kl}$$

This approach tries to minimize the overall cost of the network by finding all pairs shortest paths with a_{ij} values for edge weights and then computing the overall cost with by multiplying these edges with b_{kl} values for all {k, l} for which (i, j) lies along the shortest path from k to l.

III. DATA STRUCTURES

The solution can better be described by understanding the data structures involved.

There are primarily three data structures involved in the solution

a. Graph : Used to store network graph in the form of an adjacency matrix, and methods for initializing the graph as well as shortest path computation method. The graph data structure also contains a nextNode matrix which stores the information about the next node to be visited along the shortest path. E.g $nextNode_{ij}$ points to the next node to be visited along the shortest path that starts at i and ends at j.

   Graph
   adjMatrix : adjacency matrix of edge costs

   nextNode : nextNode matrix for the shortest path

   floydWarshall(): computes all pairs shortest path over the adjacency matrix and stores the next hop information in nextNode[2].

b. Cost Matrix,a: Cost matrix a contains all a_{ij} values of unit cost for links between nodes i and j.
c. Demand Matrix,b : The Demand Matrix d contains all d_{ij} values for traffic demand from node i to j.

IV. ALGORITHM

There are two primary steps involved in the algorithm
a. Test Case Generation
b. Cost Computation

Test Case Generation:
Test case generation follows the specific set of requirements mentioned in the problem statement.

V : initialize to 40
a_{ij}: initialize to 30 if i≠j; 0 if i=j
b_{ij}: initialize to random integers from [0,1,2,3] if i≠j; 0 if i=j
k: this values takes from [3,4..,24,25] is fixed for each test case.
for each i:=1..V
  for each j: pick k random integers from{1,2…V} – {i}
    a_{ij}:=1
Graph G(V, a_{ij}) : Initializes the graph G with V vertices and edge weights given by a_{ij}

Cost Computation:
Now that we have the Graph G initialized with edge weights a_{ij} values, we would perform the all pairs shortest path algorithm and then do cost computation

```
e : initialize a VxV matrix with 0s. This would record the
edge capacities of the final graph

G.floydWarshall() : initializes G.allPairs Matrix of
dimensions VxV and G.nextNode Matrix of dimensions VxV
with the shortest distance as well as the next node along the
shortest part route values respectively.

totalCost:=costComputation(G, a, b, V, e,  k): this function
call will fill out the e matrix while simultaneously
computing the overall cost of the the network.
```

```
costComputation(G, a, b, V, e, k)
c : c is a overall cost computation matrix which stores
the cost  of constructing an edge from i to j with
sufficient capacity to transfer all the data that passes
through the edge (i ,j)
totalCost : initialize to 0
for i := 1 .. V
   for j := 1 .. V
      if i≠j
totalCost : initialize to 0
for i := 1 .. V
   for j := 1 .. V
      if i≠j
         c[i,j]:=pathCost(i, j, G, a, b, e)
         totalCost:= totalCost + c[i, j]
```

```
pathCost(i, j, G, a, b, e)
   start:=i
   dest:=j
   pathCost:=0
   while(start≠dest)
     nextNode:=G.nextNode[start, dest]
     pathCost:=a[start,nextNode] + pathCost
     e[start,nextNode]:=e[start,nextNode]+ b[i,j]
    start:=G.nextNode[start, dest]

   pathCost:=pathCost X b[i, j]
   return pathCost
```

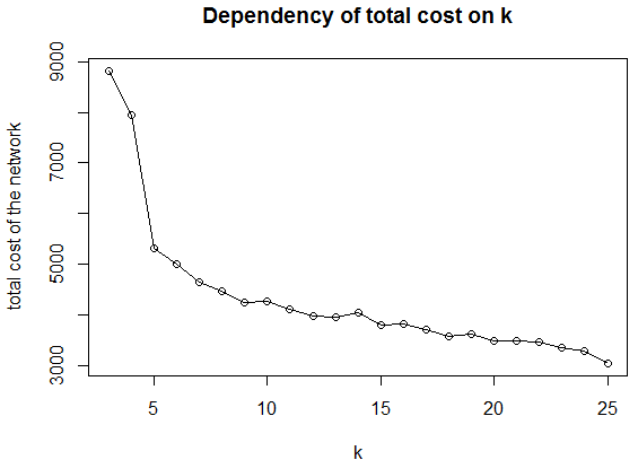
At the end of the execution of the algorithm the edge capacities matrix is filled with appropriate values as well as the total cost is computed. What we have towards the end is an edge capacity matrix and the overall cost matrix as well as the total cost value.

We use the edge capacity matrix, e, to plot the topological graphs as well as to compute the density of the graph by computing the proportion of e's where e[i, j]≠ 0. We plot graphs of k vs total cost and k vs density to study the impact of k on the network cost and density.

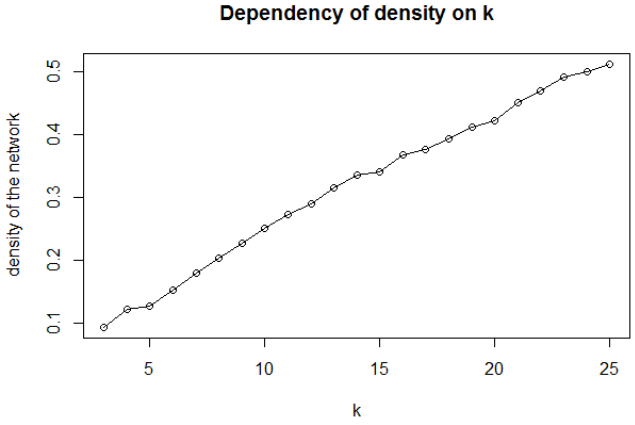
V. Results

This program has been tested on k values [3,4,..,24,25] and with V=40. The results are computed and stored in files. The following diagrams and graphs were obtained by running R language scripts over these files. All files are available through github repository made for the project[1].

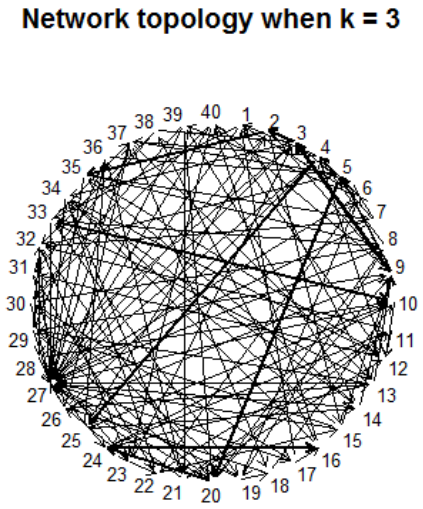
i. Total cost dependence on k



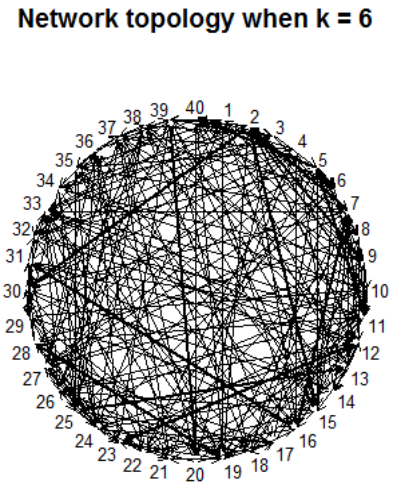
ii. Density dependence on k



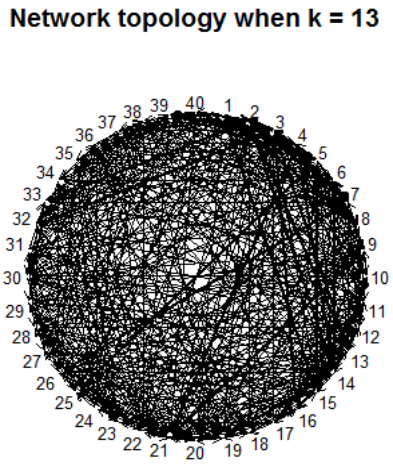
iii. Network topology for k=3



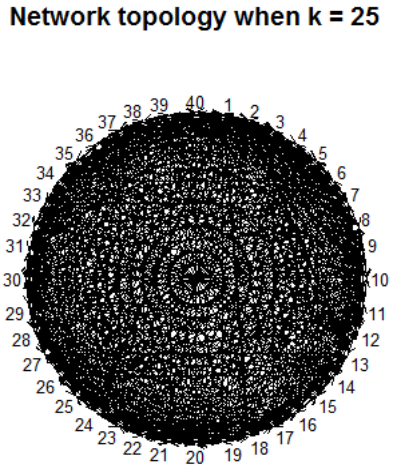
iv. Network topology for k=6



v. Network topology for k=13



vi. Network topology for k=25



More network topology plots can be found in the github site for the project [1].

VI. Observations from the results

i. Total Cost Dependence on K

When the K value is high, there are more number of "cheaper" edges in the graph. This means that more number of shortest paths within the graph will take more of these cheaper nodes thereby bringing the overall cost of the network down. Hence the graph displays a decreasing trend.

ii. Density Dependence on K

Here we describe the density as the proportion of non-zero edges from the overall possible edges in the graph. When k value is low, the few "lighter" edges will be popular and will be repeatedly used by different paths. The more expensive edges are avoided altogether. Hence, the density of the graph is low when k is small.

When K is large there are a lot of "lighter" edges forming paths between different nodes, this means a lot more of the edges become part of the network than when there were only a few of the popular "lighter" edges. Hence, the density increases as K increases.

iii. Network Topology Dependence on K

The network topology diagrams simply reflect the trends presented by the Cost and Density trends.

In the given network topology diagrams, the nodes are presented across the circumference of a circle. These nodes are numbered 1 through 40. All the nodes with non zero capacities are linked together with a black line. The thickness of the line is determined by the edge capacity values i.e. the thicker the line the higher the capacity.

We can see from the diagrams that when K=3, the network is very sparse but the connecting lines are thicker and more bold than when K=25. For higher K values the network topology is much denser but the actual thickness of the connecting edges is low.

VII. Conclusion

From the results observed we can conclude that in order keep the overall cost of network construction low, we should connect a lot of nodes with low capacity links than connecting a few with high capacity links. By having more low capacity interconnections we can ensure that the overall traffic demands for all the nodes are met while the overall cost of the network is kept to a minimum.

References

Basic format for books:

[1] GitHub Link for project source code,data and plots https://github.com/ssdeep/aatn-project1

[2] Floyd-Warshall All Pairs Shortest Path algorithm explanation http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

[3] R Diagram package for visualizing simple graphs http://cran.r-project.org/web/packages/diagram/vignettes/diagram.pdf