PROJECT – 1

APPLICATION OF NETWORK DESIGN

CS-6385

ALGORITHMIC ASPECTS OF TELECOMMUNICATION NETWORKS

Submitted by:

PALLAVI DASARAPU

PXD210008

Contents:

OBJECTIVE:

The main objective of this project is to implement a basic network design module in order to create different network topologies with capacities assigned to links in accordance with the model, “An application for network design” that employs a fast solution based on the shortest path.

- It takes as inputs - # of nodes (n), traffic demand values between node pairs (b_{ij}), and unit cost values for any prospective links (a_{ij}).
- The program then uses the shortest path based rapid solution method to generate a directed graph (network topology) with capacity given to the directed edges (links) in line with the model under study. In addition, it calculates the network's overall cost.

The shortest path between every pair of nodes is determined by the Dijkstra algorithm for every value of K, is used by the program. [1]

Total cost of all edges that form the shortest paths in the topology, is calculated together with the density for each value of k. For various values of k, total cost and density are examined and plotted.

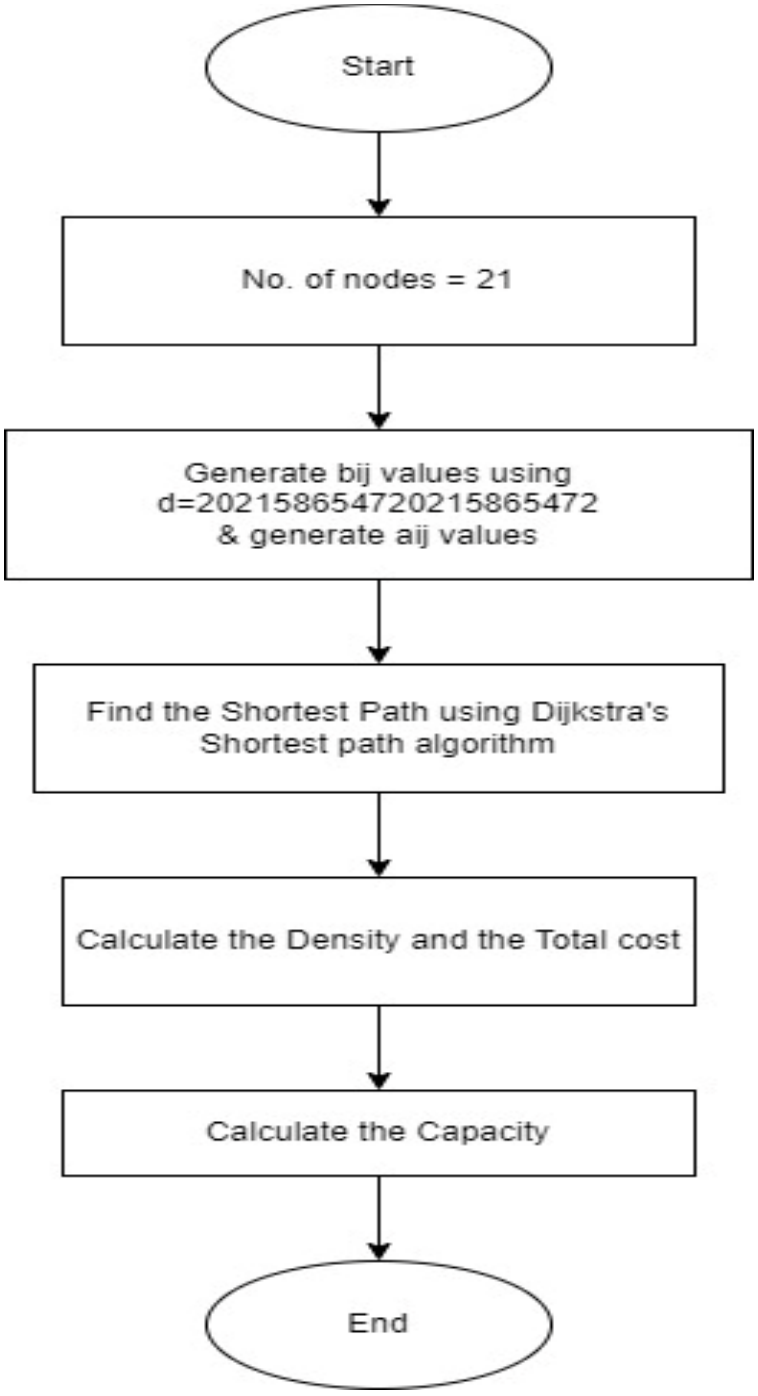
DIJKSTRA’s ALGORITHM:

The Dijkstra algorithm helps in determining the shortest paths from a source vertex in graph G to each of its vertices.

Given a source node as the root node, the algorithm creates a shortest path tree like Prim's Minimum Spanning Tree. [1]

- Make two sets of nodes (one set to keep track of visited nodes and the other to track unvisited) and initially set all nodes as unvisited.
- Set the starting node to 0 and assign some dummy distance values to all the nodes as infinity.
- Calculate approximate distance from current node to every next node. With the smaller value, compare the currently calculated value to the previously allocated value and assign.
- Add this current node to the visited set and remove from unvisited.
- The algorithm is complete once the destination node is visited or if the estimated distance in unvisited set is infinity.
- Otherwise, loop to step 3 while using the node with the shortest distance (approx.) as current node. [2]

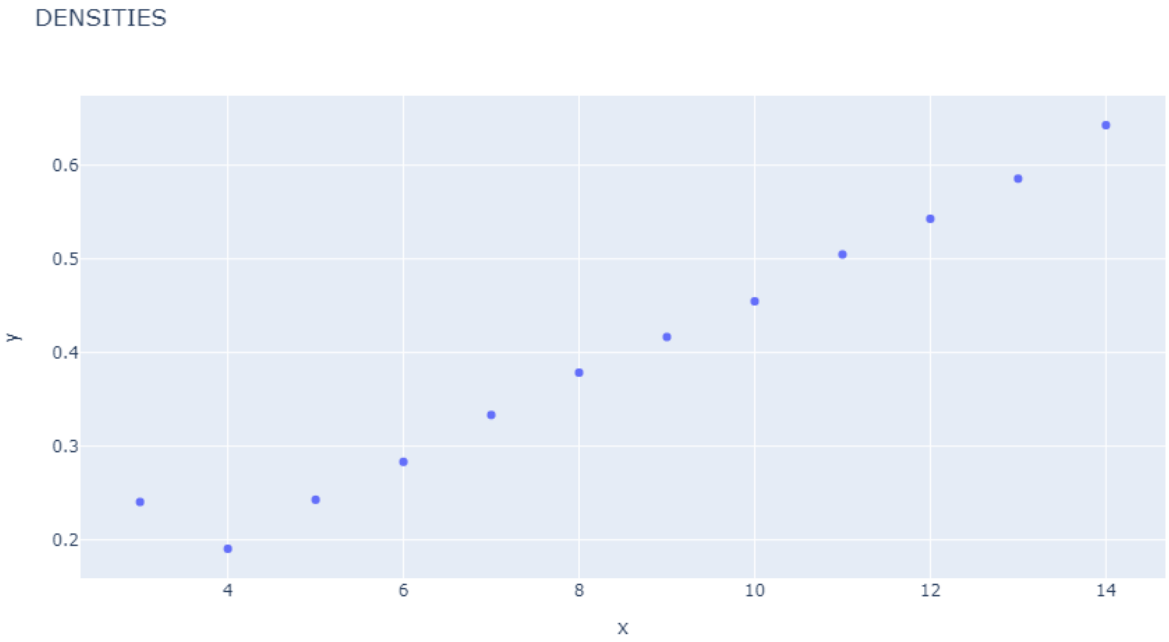
DESIGN:


Start
No. of nodes = 21
Generate bij values using
d=202158654720215865472
& generate aij values
Find the Shortest Path using Dijkstra's
Shortest path algorithm
Calculate the Density and the Total cost
Calculate the Capacity
End

ANALYSIS:

Density versus k-values:

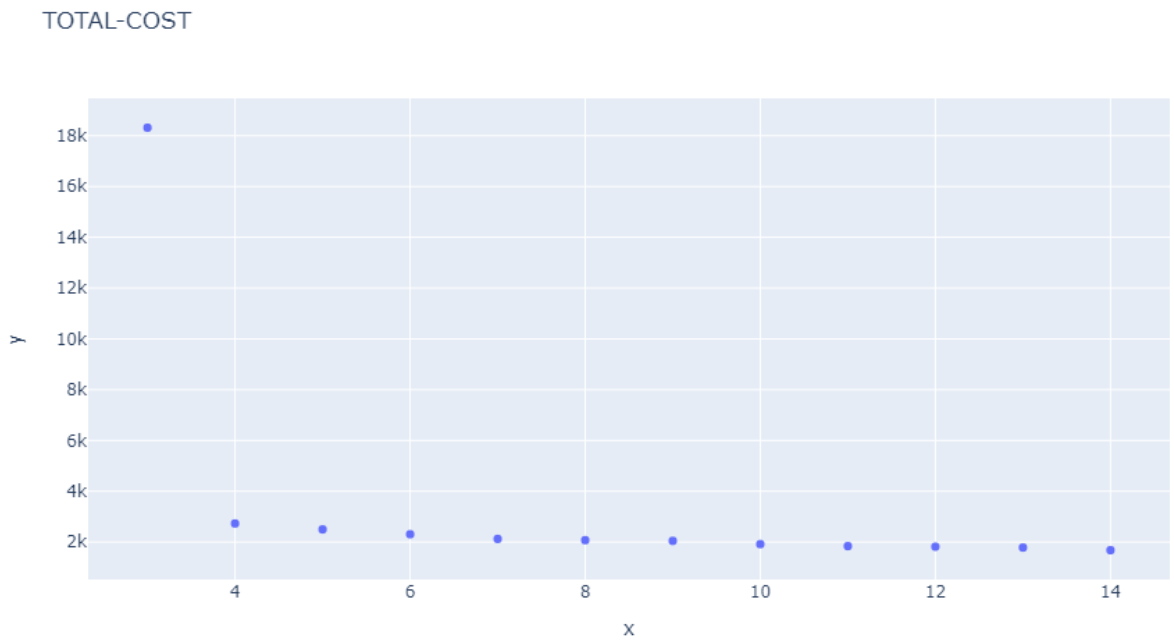
Here, Y-axis holds the Density values of the network topologies against the X-axis that holds k-values.



The number of links forming the shortest paths to the total number of links feasible should increase (as seen by the graph) when the maximum outgoing degree for each node (k) increases because, as k increases, more routes to other nodes are discovered, further clogging the network. It follows that the density should rise.

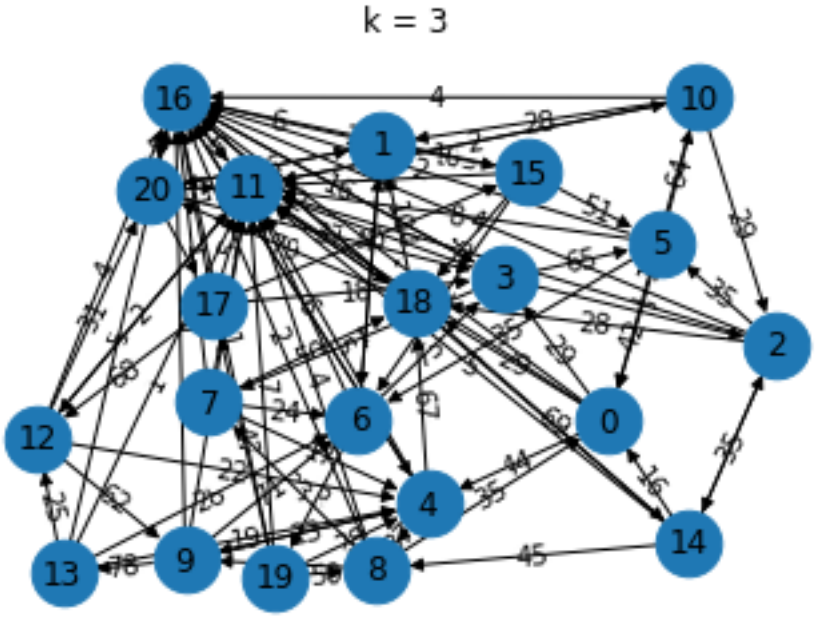
Total final cost versus K-values:

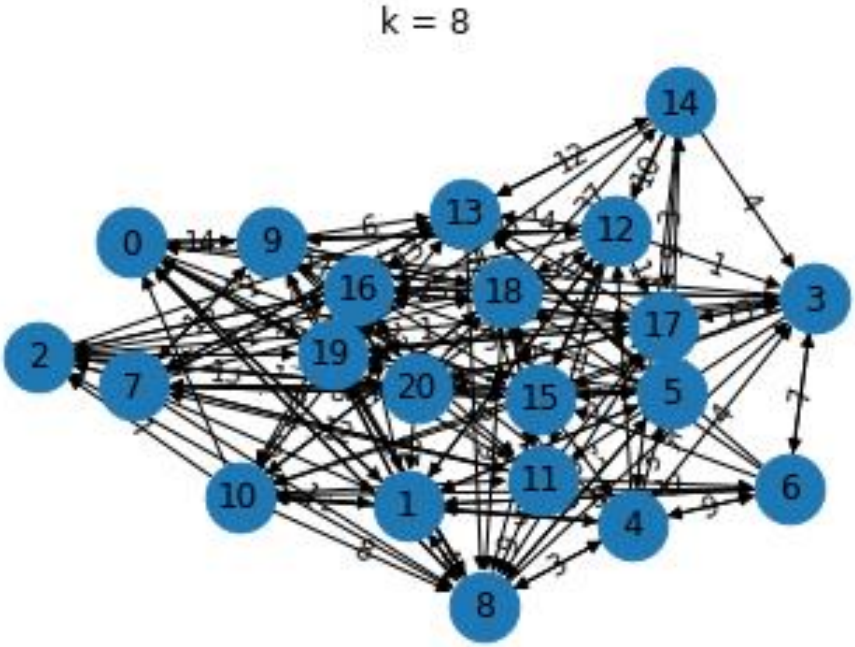
Here, Y-axis holds the Total final cost of the potential links of the network topologies against X-axis that holds k-values.

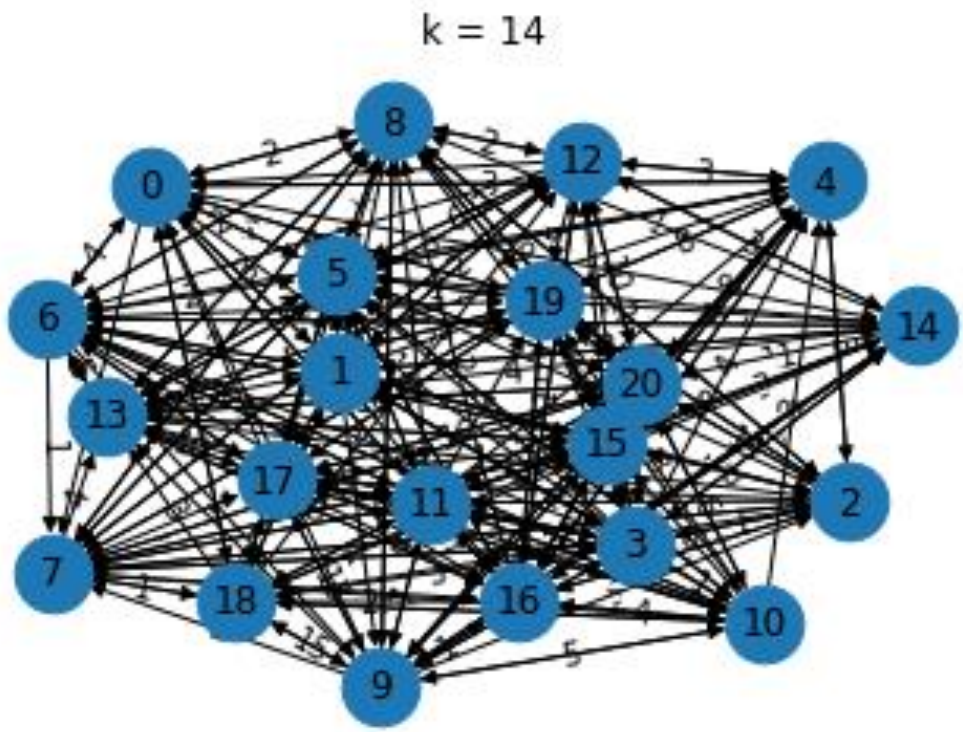


The total cost of the network traceable to all the links comprising the shortest paths should drop as the maximum outgoing degree for each node (k) grows because, as k increases, more routes to other nodes are found, leading to the discovery of additional shorter paths. Thus, it follows that their total cost sum should decrease.

TOPOLOGIES:


k = 3
16
10
1
15
20
11
5
3
17
18
2
7
6
0
12
4
14
13
9
19
8
45


k = 8
14
13
12
0
9
16
18
3
17
19
2
7
20
15
5
11
6
10
1
4
8



These are the graphs for the K values (3, 8, 14) that were asked for.

It is clear that the density and K values are inversely correlated. The graph becomes much thicker because of the rise in the number of edges to $1(a_{ij})$.

REFERENCES:

1. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
2. https://stackoverflow.com/questions/66840275/dijkstra-algorithm-for-the-shortest-path-using-adjacency-matrix
3. https://www.programiz.com/dsa/dijkstra-algorithm
4. https://plotly.com/python/network-graphs/#create-network-graph
5. https://stackoverflow.com/questions/67463450/printing-paths-in-dijkstras-shortest-path-python3
6. https://www.geeksforgeeks.org/optimal-binary-search-tree-dp-24/
7. https://python-course.eu/machine-learning/data-representation-and-visualization-data.php

APPENDIX:

main.py

```
import random as rdm
from algos import ShortestPath
from gen_graphs import *

# demand values
def bijgraphs(dvals):
    bijgraph = {}
    d_len = list(range(len(dvals)))
    for ival in d_len:
        for jval in d_len:
            if dvals[jval] >= dvals[ival]:
                bijgraph[ival,jval] = (dvals[jval] - dvals[ival])
            else:
                bijgraph[ival,jval] = (dvals[ival] - dvals[jval])
    return bijgraph

#aij graphs
def aijgraphs(ns, ks):
    vals = list(range(ns))
    Dict = {}
    for i in vals:
        temp_lst = []
        ks_lst = rdm.sample(vals, ks)
        for j in ks_lst:
            if j not in temp_lst and j != i:
                temp_lst.append(j)
        aij_lst = temp_lst
        for x in vals:
            if y in aij_lst:
                Dict[x, y] = 1
            else:
                Dict[x, y] = 100
        Dict[x, x] = 0
    aijcost = Dict
    aij_graph = [[None] * ns for _ in vals]

    for i in vals:
        for j in vals:
            aij_graph[i][j] = aijcost[i, j]
    return aij_graph

def gen_cap_graphs(ns, grphaij, grphbij):
    cap_grphs = [[0] * ns for _ in range(ns)]

    shrtst = ShortestPath().shrtstpaths_forall(grphaij)
    lst = list(range(ns))
    for i in lst:
        for j in lst:
            ijshort = shrtst[i][j]
            ijdem = grphbij[i, j]
```

```
            ilen = list(range(1,len(ijshort)))
            for spind in ilen:
                src_ind = ijshort[spind - 1]
                dst_ind = ijshort[spind]
                cap_grphs[src_ind][dst_ind] += ijdem
    return cap_grphs

# total final cost of the designed network topology
def calc_final_cost(ns, grphaij, cap_grphs):
    final_cost = 0
    nlen = list(range(ns))
    for i in nlen:
        for j in nlen:
            final_cost += (cap_grphs[i][j] * grphaij[i][j])
    return final_cost

# to calculate the final densities
def calc_final_densities(ns, cap_grphs):
        des_not0 = 0
        nlen = list(range(ns))
        for x in nlen:
            for y in nlen:
                if cap_grphs[x][y] != 0:
                    des_not0 += 1
        val = ns*(ns-1)
        final_density = des_not0 / val
        return final_density

if __name__ == "__main__":
    dval = [2,0,2,1,5,8,6,5,4,7,2,0,2,1,5,8,6,5,4,7,2]
    no_nodes = 21
    cap_grphs = []
    final_costs = []
    dens_vals = []
    k_max =14
    klist = list(range(3, k_max+1))

    for ks in klist:
        # aij graphs & demand values
        aijgrph = aijgraphs(no_nodes, ks)
        bijgrph = bijgraphs(dval)

        cap_grph = gen_cap_graphs(no_nodes, aijgrph, bijgrph)
        fin_cost = calc_final_cost(no_nodes, aijgrph, cap_grph)
        dens_val = calc_final_densities(no_nodes, cap_grph)

        # capacities, total costs & densities
        cap_grphs.append(cap_grph)
        final_costs.append(fin_cost)
        dens_vals.append(dens_val)

    GenGraphs.scatter_plot(klist, final_costs,
                           "TOTAL-COST", "k-values", "total-cost")
    GenGraphs.scatter_plot(klist, dens_vals,
                           "DENSITIES", "k-values", "density-values")

    #graphs for k=3, k=8, k=14
```

```
    GenGraphs.topology_grphs(cap_grphs[0], no_nodes, "k = 3")
    GenGraphs.topology_grphs(cap_grphs[5], no_nodes, "k = 8")
    GenGraphs.topology_grphs(cap_grphs[11], no_nodes, "k = 14")
```

algos.py

```
class ShortestPath:
    shrtst_pathlist = []

    def __init__(self):
        pass

    # shortest paths from all nodes to all nodes
    def shrtstpaths_forall(self, grph):
        shrtst_paths = []
        for src in range(len(grph[0])):
            gr_len = len(grph)
            c_lst = range(len(grph[0]))

            dst = [float("Inf")] * gr_len
            prnt = [-1] * gr_len
            dst[src] = 0

            dlist = range(len(dst))
            q_lst = []

            for i in range(gr_len):
                q_lst.append(i)

            while q_lst:
                minval = float("Inf")
                minindx = -1
                for i in dlist:
                    if minval >= dst[i]:
                        if i in q_lst:
                            minval = dst[i]
                            minindx = i
                x = minindx
                q_lst.remove(x)
                for i in c_lst:
                    if grph[x][i] and i in q_lst and \
                            (dst[x] + grph[x][i] < dst[i]):
                        dst[i] = dst[x] + grph[x][i]
                        prnt[i] = x

            all_shrtst_paths = [[] * len(dst) for _ in dlist]
            for i in dlist:
                self.shrtst_pathlist = []
                self.src_to_dstn(prnt, i)
                all_shrtst_paths[i] = self.shrtst_pathlist

            shrtst_paths.append(all_shrtst_paths)
        return shrtst_paths
```

```
    # shortest path from source to destination
    def src_to_dstn(self, prnt, i):
        if prnt[i] == -1:
            self.shrtst_pathlist.append(i)
            return
        self.src_to_dstn(prnt, prnt[i])
        self.shrtst_pathlist.append(i)
```

gen_graphs.py

```
import networkx as ntx
import pandas as pds
import pylab as pl
import plotly.express as pe


class GenGraphs:

    # to display the scatter plot
    def scatter_plot(x_val, y_val, ttl, xlbl, ylbl):
        vals = dict(x=x_val, y=y_val)
        lbls = {xlbl: ylbl}
        df_vals = pds.DataFrame(vals)
        fgr = pe.scatter(df_vals, "x", "y", title=ttl, labels=lbls)
        fgr.show()

    # to display network topology graphs
    def topology_grphs(cap_grph, n, ttl):
        gr = ntx.DiGraph()

        nlen = list(range(n))
        for i in nlen:
            for j in nlen:
                cap = cap_grph[i][j]
                if cap != 0:
                    x = str(i)
                    y = str(j)
                    gr.add_edges_from([(x, y)], weight=cap)

        edgs = dict([((m, n,), dval['weight'])
                     for m, n, dval in gr.edges(data=True)])
        node_labels = {}

        for i in nlen:
            x = str(i)
            node_labels[x] = x

        layout = ntx.spring_layout(gr)
        ntx.draw_networkx_edge_labels(gr, layout, edgs)
        ntx.draw_networkx_labels(gr, layout, node_labels)
        ntx.draw(gr, layout, node_size=600)

        pl.title(ttl)
        pl.show()
```

READ ME:

Packages:

install python 3.9.12

- pip install plotly = = 5.6.0
- pip install pandas = =1.4.2
- pip install networkx = = 2.7.1

How to execute:

- Copy and paste the code from the snippets on separate files in IDE. Maintain the same mentioned names for the files and keep all three files in same folder.
- Run main.py file and the results will be displayed.