AN APPLICATION TO NETWORK DESIGN

ALGORITHMIC ASPECTS OF TELECOMMUNICATION NETWORKS

TE 6385.001

V. PURNA CHANDRA VINAY KUMAR

PXV171630

Contents

Introduction to Network Topology:

The arrangement of the various elements of the network such as nodes and links in topological structure is called network topology. The two different elements in the network are nodes which generate or absorbs the data or which transmits the data from source to destination and links which connects the nodes together. The network topology is constructed for the given number of nodes 'N' from the given values of unit link cost 'a_{ij}' of the links connecting the source and the destination nodes. The network can be constructed using shortest path algorithms like Dijkstra's algorithm, Bellman–Ford algorithm, Floyd–Warshall algorithm etc. each used for different purposes. These algorithms gives the least cost path between the source and the destination nodes. The total cost of the path between the two nodes is calculated by multiplying the traffic demand between the nodes and the total unit cost over the links connecting the two nodes which is obtained from shortest path algorithm. The network density is calculated by dividing the number of directed edges that are assigned non-zero capacity, with the total possible number of directed edges N(N-1).

Problem Statement:

The aim of the project is to design the network topology using fast solution method and to calculate the total cost of the network and the network density. The project aims at fulfilling the following tasks.

Tasks:

- The objective of the project is to create a program which constructs a network for the given number of nodes '*N*' from the unit cost values of the links.
- The program takes number of nodes '***N***', the unit cost values of the link '$\boldsymbol{a_{ij}}$' and the traffic demand values '$\boldsymbol{b_{ij}}$' between the nodes as input variables.
- It constructs the network topology for the given number of nodes '***N***' from the unit cost values '$\boldsymbol{a_{ij}}$' with network capacities assigned to the links '$\boldsymbol{b_{ij}}$', using the shortest path based fast solution approach.
- The program is divided into three modules.
    - *Module 1:* Generates the network parameters '$\boldsymbol{a_{ij}}$' and '$\boldsymbol{b_{ij}}$' for '***N***' number of nodes based on the given conditions and passes the values to the next main module.
    - *Module 2:* Carries main algorithm to construct the network graph. Dijkstra's shortest path algorithm is used to construct the network path between the nodes and the total cost of the network is calculated using fast solution method.

- *Module 3:* Module 3 is used to plot the required graphs and figures form the values generated from the module 2.

Approach:

The approach to solving the given problem is described in the following steps.

- Module 1 is used to generate the unit link cost values and the traffic demand values based on the condition given in the problem for the given number of nodes *N = 20*.
- The program i.e. module 1 generates a matrix of traffic demand values 'b_{ij}' between node '***i***' to node '***j***', based on the condition described in the problem statement.
- The program then initializes a matrix '***A***' of link cost values for the links between every node to all the other '***N-1***'. The link cost values are generated for different values of '*k*'.
- The link cost values 'a_{ij}' and the traffic demand values 'b_{ij}' are passed to the second module which generates the network topology using shortest path based fast solution approach.
- Dijkstra's algorithm is implemented on the matrix '**A'** to find the least cost network path from one node to another, which returns the total cost between the source node and the destination node and traces the path travelled to the destination. The network path and the total cost of the path is saved to an external excell file.
- The network density and the total link cost is calculated and the results are passed to the module 3.
- The module 3 then calculates and plots the required plots and graphs by reading the values from the module 2.

Algorithm:

The programming language Python v.3.6.4 is used to solve the given problem for generating network topology and to plot the required figures.

Module 1:

- Based on the rules described in the problem statement for generating the traffic demand values, module 1 takes 10 - digit student ID as input and calculated which is repeated

twice to make it 20 - digit number denoting the with $d_1, d_2, \ldots\ldots d_{20}$, then '$b_{ij}$' is calculated by the formula

$$b_{ij} = |d_i - d_j|.$$

The calculated 'b_{ij}' values are stored in a 20×20 matrix, where the values in the '*i th*' row and '*j th*' column denotes the traffic demand between the source node '*i*' to the destination node '*j*'.

- The link cost values are initialized depending on the values of *'k'*. A 20×20 matrix *'a'* is initialized. Uniform random number is used to generate *'k'* random numbers from '$1\ to\ N$' representing the *'j'* indices all different from each other and different from *'i'* that take the link cost value '1' and the values are replaced with the link cost values i.e.

$$a_{ij_1} = a_{ij_2} = a_{ij_3} = \cdots = a_{ij_k} = 1$$

- The remaining indeces $j \neq j_1, , , , , j_k$, are assigned the cost values of $a_{ij} = 100$.
- For $k = 3,4,5 \ldots \ldots 14$, random numbers are generated and the corresponding indices are given the link cost of '1'.

Module 2:

- Module 2 implements the Dijkstra's shortest path algorithm on the link cost values 'a_{ij}' and generates the least cost path and the total cost from node ***'i'*** to all the nodes $j \in N - 1$.
- The algorithm is implemented for all the nodes $i \in N$,and the generated path and values are stored in an external excell file for all the values of $k = 3,4,5 \ldots 14$.
- The network density is calculated by counting the number of links in a network certain value of '*k*' and divided by the total number of potential connection $N(N-1)$. The network density is calculated for all values of $k = 3,4,5 \ldots \ldots 14$.
- The total cost of the network is also calculated using the formula

$$Z_{opt} = \sum_{k,l} \left(b_{kl} \sum_{(i,j) \in E_{kl}} a_{ij} \right)$$

Module 3:

- The values calculated in the module 2 is used to plot the required graphs of total cost of the network Z_{opt} and the network density vs K.

Dijkstra's Algorithm:

Dijkstra's Algorithm is and algorithm used to find the shortest paths between nodes and graphs. It does use min-priority queue and runs over a time of $O(V^2)$ where V is the number of nodes.

The algorithm returns constructs the shortest path from source not all the other nodes. The algorithm is implemented on the graph which have link weights.

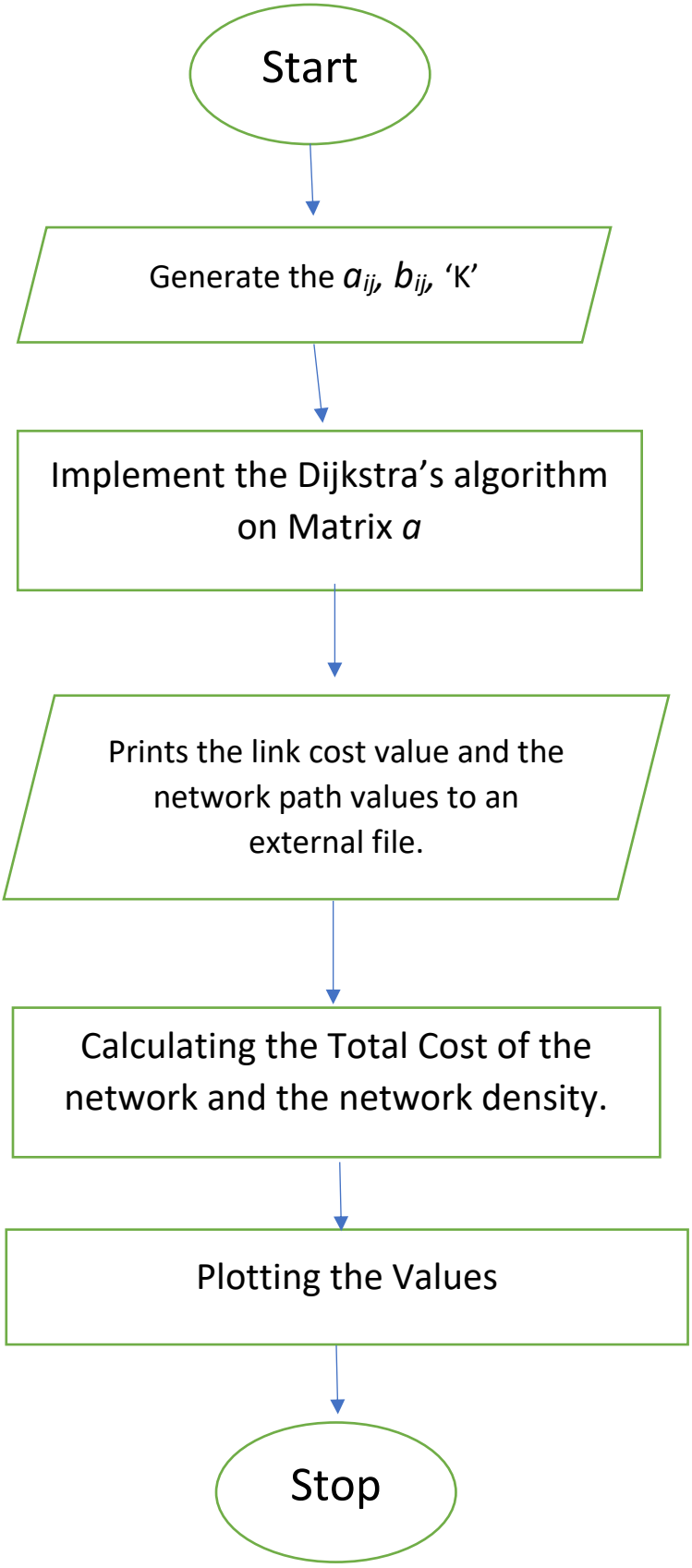
Algorithm:

1. Initially mark all the nodes as unvisited called the unvisited nodes. Create a table which stores the destination node, link cost and the previous node values.
2. Assign initial cost to all the nodes to infinity and zero to initial source node. Set initial node as current node and the current cost to zero.
3. Consider all the unvisited nodes, and calculate the link cost to all the adjacent unvisited nodes to the current node and update the table with the link cost to all the adjacent nodes and previous node value to current node value.
4. The least link cost node is selected from the table and the current node and current cost values are updated with the node value and the cost value. Then the step 3 repeated for the selected node.
5. If the current cost plus the link cost to the adjacent node is less the stored value of the table, then the table is updated with the new cost value and the previous node is updated with the current node value.
6. The process is repeated until all the nodes are visited.
7. This entire process is repeated for all the node values.

Pseudo Code[1]:

```
function Dijkstra(Graph, source):

    create vertex set Q

    for each vertex v in Graph:             // Initialization
        dist[v] ← INFINITY                  // Unknown distance from source to v
        prev[v] ← UNDEFINED                 // Previous node in optimal path from source
        add v to Q                          // All nodes initially in Q (unvisited nodes)

    dist[source] ← 0                        // Distance from source to source

    while Q is not empty:
        u ← vertex in Q with min dist[u]    // Node with the least distance
                                            // will be selected first
        remove u from Q

        for each neighbor v of u:           // where v is still in Q.
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:               // A shorter path to v has been found
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

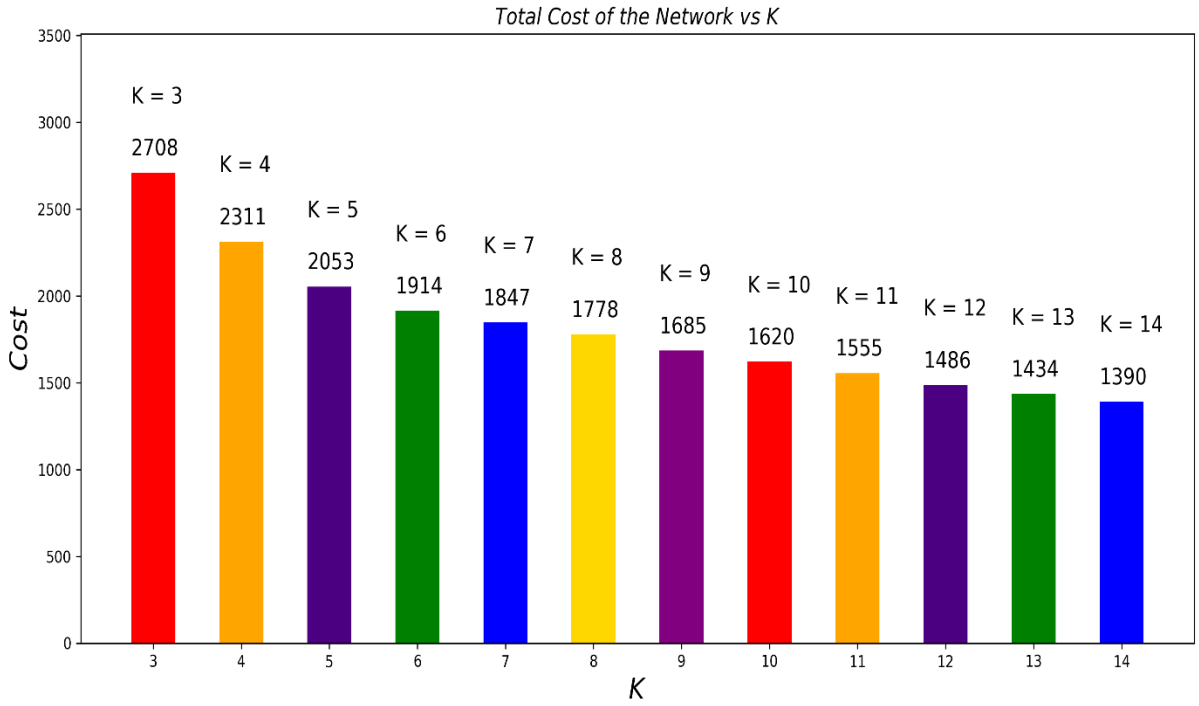
Flow Charts:


Start
Generate the a_{ij}, b_{ij}, 'K'
Implement the Dijkstra's algorithm on Matrix *a*
Prints the link cost value and the network path values to an external file.
Calculating the Total Cost of the network and the network density.
Plotting the Values
Stop

Bar Graph:

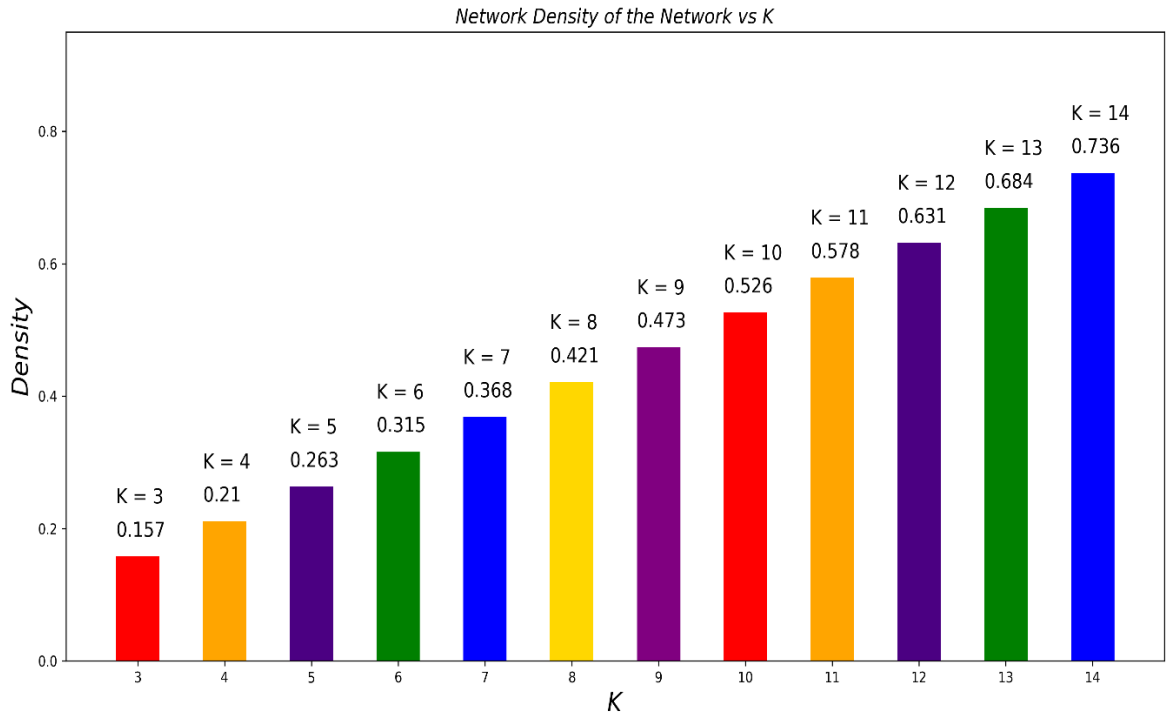
Cost vs K:

- The following graph indicates the total cost of the network versus ‘K’.
- As we can observe that the total cost of the network decreases with the increase in the value of ‘K’.
- The values of indicates the number of links, which have link cost of ‘1’
- Therefore, we can observe as the value of ‘K’ increases, the number of directed links increases which have cost of ‘1’ and therefore total cost of the network decreases for given set of data traffic values.



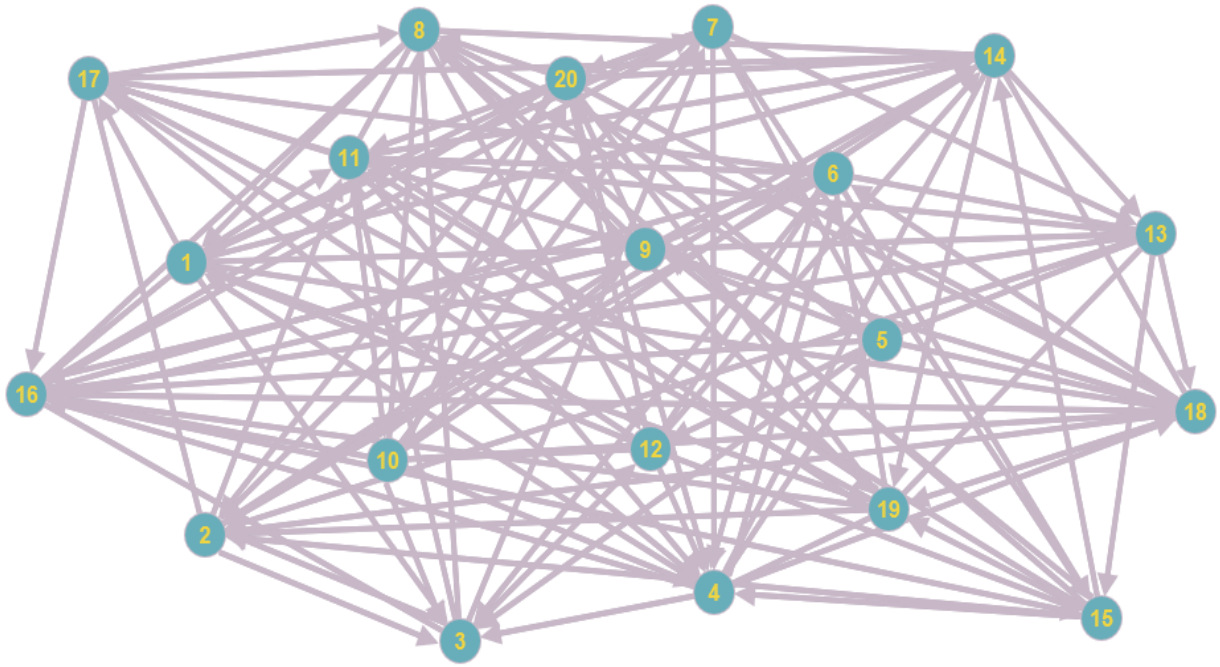
Network Demand vs K:

- The following graph indicates the network density vs K.
- The network density is defined as the number of directed edges that are assigned non-zero capacity, divided by the total possible number of directed edges $N(N-1)$.
- As the value of ‘K’ increases the network density increases since, the number of links which have link cost of ‘1’ increases and therefore, the number of connection between the nodes increases.

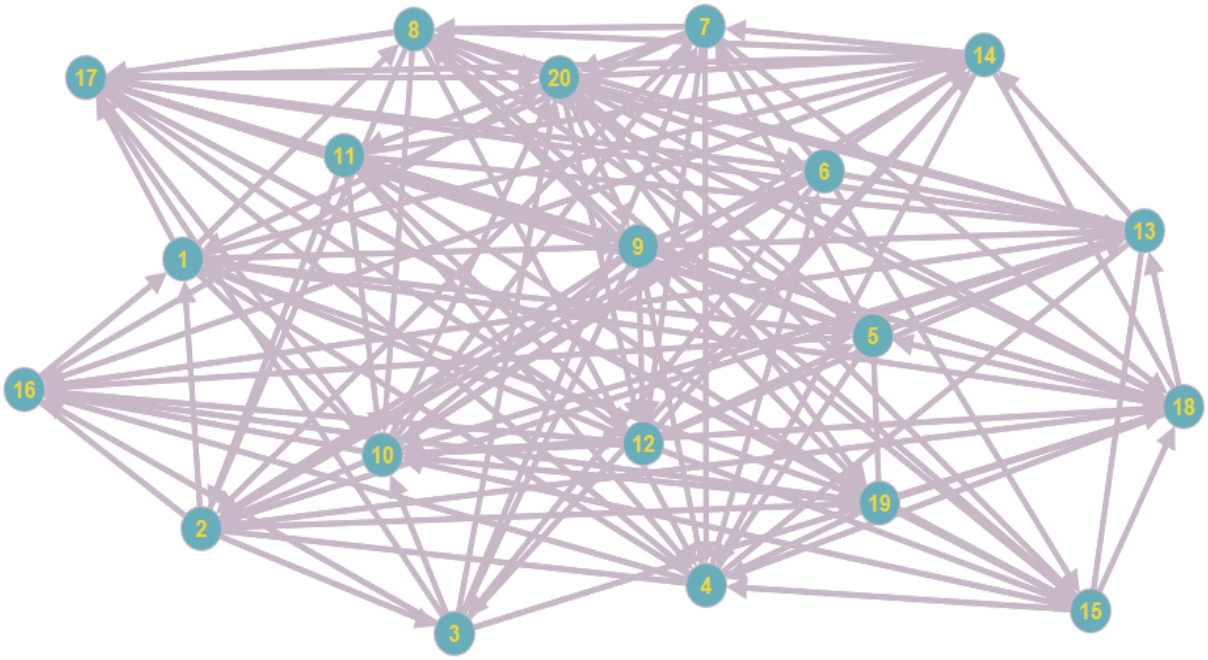


Network Topology:

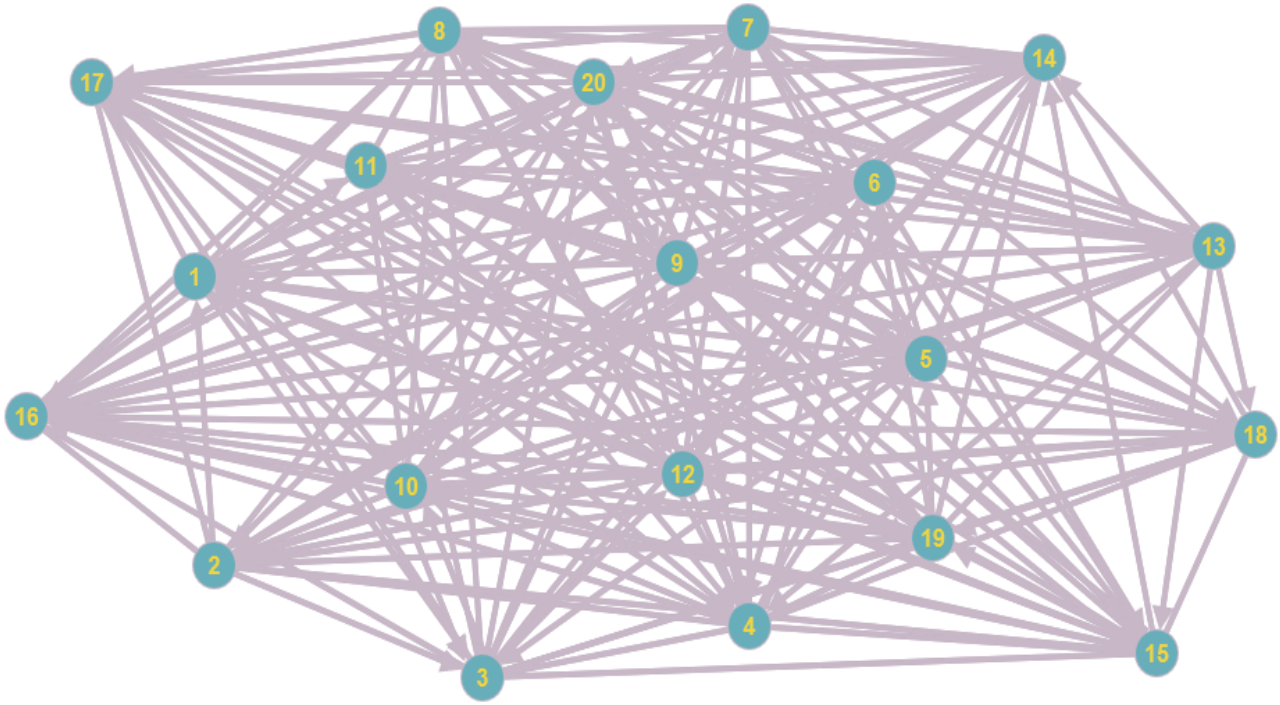
K = 3



K = 8


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

K = 14


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

References:

1. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
2. https://youtu.be/gdmfOwyQlcI
3. http://graphonline.ru/en/

Appendix:

Module 1 code:

```
from numpy import *

def generate_val():
    # The number of Nodes N = 20
    # The traffic demand between the pair of nodes (i,j) is b_{i,j}
    # The unit cost values for the link (i,j) is a_{i,j}
    # bij = |di - dj|

    # the flow through the link 'd'

    ID = '1210413658'
    ID = ID*2
    d = []

    for i in ID:
        d.append(int(i))


    # bij = |di - dj|
    b_ij = zeros([len(d),len(d)])

    for i in range(len(d)):
        for j in range(len(d)):
            if(i!=j):
                b_ij[i,j] = abs(d[i]-d[j])


    K = list(range(3,15))


    # A stores all the generated matrix of aij
    A = list()
    for k in K:
```

```
        a = zeros([len(d),len(d)])
        for i in range(len(d)):
            r = random.choice(len(d),k,replace=False)
            while(i in r):
                r = random.choice(len(d),k,replace=False)

            for j in range(len(d)):
                if(i!=j):
                    if(j in r):
                        a[i,j] = 1

                    else:
                        a[i,j] = 100

        A.append(a)

    return A,b_ij,K,d
```

Module 2 code:

```
import openpyxl as excell
import values as val

A,b_ij,K,d = val.generate_val()

file = excell.Workbook()
activeSheet = file.active

def Dijksters_Short_Cost(node_one,Row,visited,unvisited,p):
    # the statement first visites the current node i.e. Row node and
sets cost to '0'
    activeSheet.cell(row=Row,column=Row-p*21).value = str(Row-p*21-
1)+','+str(0)
    # initializing the current cost to '0'
    current_cost = 0
    node = Row-1-p*21
    # visiting the unvisited nodes till all the nodes are visited
    while(unvisited):
        # Now taking the node 'Row' i.e. visiting the 'Row'th node
        # and obtaining the minimum cost node

        for row in
activeSheet.iter_rows(min_row=Row,max_row=Row,min_col=1,max_col=len(d)
):
            #node = unvisited[0]
            cost =
float(activeSheet.cell(row=Row,column=node+1).value.split(',')[1])
            for RoW in row:
```

```
                if(float(RoW.value.split(',')[1])<cost and
(int(RoW.value.split(',')[0]) in unvisited)):
                    cost = float(RoW.value.split(',')[1])
                    node = int(RoW.value.split(',')[0])

        current_cost =cost
        current_node = node
        for j in unvisited:
            get_cost = node_one[current_node,j]
            if(get_cost==0):
                continue


if(float(activeSheet.cell(row=Row,column=j+1).value.split(',')[1])>=cu
rrent_cost+get_cost):
                activeSheet.cell(row=Row,column=j+1).value = \

str(int(activeSheet.cell(row=Row,column=j+1).value.split(',')[0]))\

+','+str(int(get_cost+current_cost))+','+str(int(current_node))


        visited.append(node)
        unvisited.remove(node)

        if(unvisited):
            node = unvisited[0]

    file.save('ATNexcell.xlsx')


def source_node(graph1,p):
    row = 1

activeSheet.merge_cells(start_row=row+p*21,start_column=23,end_row=row
+p*21+4,end_column = 27)
    activeSheet.cell(row=row+p*21,column=23).value = 'K = '+str(p+3)
    for i in range(len(d)):
        for j in range(len(d)):
            # stores (node,cost)
            activeSheet.cell(row=(i+1+p*21),column=j+1).value =
str(j)+',inf'

    for source in graph1:
        unvisited = list(range(len(d)))
        visited = []
        Dijksters_Short_Cost(graph1,row+p*21,visited,unvisited,p)
        row+=1


def network(A):
```

```
    for p in range(len(A)):
        source_node(A[p],p)

def total_cost(p):
    Z_opt =0
    for i in range(len(d)):
        for j in range(len(d)):
            # sum(d_kl*sum(a_ij))
            if(i==j):
                continue

            a_ij =
int(activeSheet.cell(row=i+1+p*21,column=j+1).value.split(',')[1])
            Z_opt += b_ij[i][j]*a_ij
    return Z_opt

def network_density(p):
    path = 0
    li = []
    for i in range(len(d)):
        for j in range(len(d)):
            if(i==j):
                continue

            c =
(int(activeSheet.cell(row=i+1+p*21,column=j+1).value.split(',')[0]),

int(activeSheet.cell(row=i+1+p*21,column=j+1).value.split(',')[2]))
            li.append(c)
    links = len(set(li))
    return links/(20*19)

def plot_values():
    T_cost = []
    N_dens = []
    network(A)
    for l in range(len(K)):
        T_cost.append(total_cost(l))
        N_dens.append(network_density(l))

    return T_cost,N_dens,K
```

Module 3 Code:

```
import Main_module as mn
import matplotlib.pyplot as plt
```

```
def plottings():
    Cost,Den,K = mn.plot_values()

    color =
mn.val.array(['red','orange','indigo','green','blue','gold','purple'])
    plt.subplots(figsize=(15,7))
    plt.bar(K,Cost,width=0.5,color=color)
    plt.xticks(list(range(3,15)))
    for i in K:
        plt.text(i-0.25,Cost[i-K[0]]+100,str(int(Cost[i-
K[0]])),fontsize=14)
        plt.text(i-0.25,Cost[i-K[0]]+400,'K = '+str(i),fontsize=14)

    plt.ylim((0,max(Cost)+800))
    plt.xlabel('$K$',fontsize=18)
    plt.ylabel('$Cost$',fontsize=18)
    plt.title('$Total \ Cost \ of \ the\ Network \ vs \
K$',fontsize=14)
    plt.savefig('Total Cost.png',dpi=300)

    color =
mn.val.array(['red','orange','indigo','green','blue','gold','purple'])
    plt.subplots(figsize=(16,8))
    plt.bar(K,Den,width=0.5,color=color)
    plt.xticks(list(range(3,15)))
    yval = []
    for i in Den:
        t = int(i*1000)
        yval.append(t/1000)


    for i in K:
        plt.text(i-0.25,Den[i-K[0]]+0.03,str(yval[i-
K[0]]),fontsize=14)
        plt.text(i-0.25,Den[i-K[0]]+0.08,'K = '+str(i),fontsize=14)

    plt.xlabel('$K$',fontsize=18)
    plt.ylim((0,0.95))
    plt.ylabel('$Density$',fontsize=18)
    plt.title('$Network \ Density \ of \ the\ Network \ vs \
K$',fontsize=14)
    plt.savefig('Density.png',dpi = 300)
    plt.show()
```

Read Me:

1. Python v.3.6.4 is used as a programming software. The three modules are saved in separate files.
2. To run the program, all the three files are saved in the same destination folder and is imported into the program for compilation.
3. Python Jupyter notebook environment is used to compile the program.
4. The third module is imported to the notebook, then the ‘plottings’ functions is run to generate the plots. It also creates excell file which stores the shortest path between the source and the destination.
5. Alternatively, the program can be compiled from the terminal window by importing the module 3 and then running the function ‘plottings’.