CS 6385 Project 1

bxl190001

October 2022

1 Program

1.1 Module 1

1.1.1 Initialization

```
    demands = np.empty((N, N), dtype=int)
    costs = np.full((N, N), 100)
    net = np.zeros((N, N, N), dtype=int)
    total_costs = np.zeros(N, dtype=int)
    densities = np.zeros(N)
```

Numpy arrays are used for efficiency. Densities is the only matrix with floats to store the results of floating point division. All of the weights of costs are initialized to 100 for simplicity. The net tensor represents multiple minimum cost networks, one for each value of k. The total_costs and densities array represent total costs and densities for each value of k. The size of net, total_costs, and densities is N for simplicity since the maximum value of k is N.

1.1.2 Demands

```
for i in range(len(id)):
    for i2 in range(len(id)):
        demands[i][i2] = abs(int(id[i]) - int(id[i2]))
```

Each demand is set to the absolute value of the difference so all demands are non-negative and demands for paths between the same nodes are 0.

1.1.3 Costs

```
    rng = np.random.default_rng()
    rand_set = tuple(range(N))

    for r in range(test_min_r, test_max_r):
        costs.fill(100)
        np.fill_diagonal(costs, 0)
        for i in range(N):
            indices = rng.choice(rand_set[:i] + rand_set[i + 1:], r, replace=False)
            costs[i][indices] = 1
```

Costs is reset every iteration of r because it is modified in the construction of the minimum cost network. The diagonals of the cost matrix is set to 0 because the cost from every node to itself is 0. The random number generator is seeded with a different value on each execution and each index generated is distinct (replace=False) and not equal to the row index (rand_set[:i] + rand_set[i + 1:]).

1.2 Module 2

1.2.1 Minimum Cost Network

See Algorithm.

1.2.2 Calculations

```
num_dir_edges = N * (N - 1)
total_costs[r] = np.sum(demands * costs)
densities[r] = float(np.count_nonzero(net[r])) / num_dir_edges
```

The demands and costs matrix are multiplied element wise and summed to get the total cost. The number of nonzero edges is casted to a float to ensure floating point division.

1.3 Module 3

1.3.1 Total Cost and Density Graph

```
def show(densities, net, total_costs):
    x = np.arange(test_min_r, test_max_r)

    plt.plot(x, total_costs[test_min_r:test_max_r])
    plt.xlabel('k')
    plt.ylabel('Total Cost')
    plt.savefig('costs.png')
    plt.clf()

    plt.plot(x, densities[test_min_r:test_max_r])
    plt.xlabel('k')
    plt.ylabel('Density')
    plt.savefig('densities.png')
    plt.clf()
```

Only values from 3 to 14 are plotted. The plot is cleared each time with plot.clf().

1.3.2 Network Graph

```
plt.figure(figsize=(20, 20))

for i in [3, 8, 14]:
    G = nx.from_numpy_matrix(net[i])
    layout = nx.shell_layout(G)
    plt.title(f'k = {i}')
    nx.draw(G, layout, with_labels=True)
    edge_labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos=layout, edge_labels=edge_labels)
    plt.savefig(f'graph{i}.png')
    plt.clf()
```

The figure size is increased and a shell layout is used to improve visibility.

2 Algorithm

2.1 Definition

```
def min_net(costs, demands, net):
```

Costs is a matrix with dimensions N x N that represents the unit cost for a given capacity. Demands is a matrix with dimensions N x N that represents a lower bound on the flow between two nodes. Net is a matrix with dimensions N x N which represents the resulting minimum cost network.

2.2 Initialization

```
min_paths = np.empty((N, N), dtype=int)

for i in range(N):
    min_paths[i] = np.arange(N)
```

A numpy array is used to store the minimum cost paths to use for assigning capacities to the minimum cost network. Each element is initialized to the corresponding column index which represents the destination of the edge because the initial shortest path known is directly from the source node to the destination node.

2.3 All Shortest Paths

```
for i in range(N):
    for i2 in range(N):
        for i3 in range(N):
            min_cost_i = costs[i2][i] + costs[i][i3]
            if costs[i2][i3] > min_cost_i:
                min_paths[i2][i3] = min_paths[i2][i]
                costs[i2][i3] = min_cost_i
```

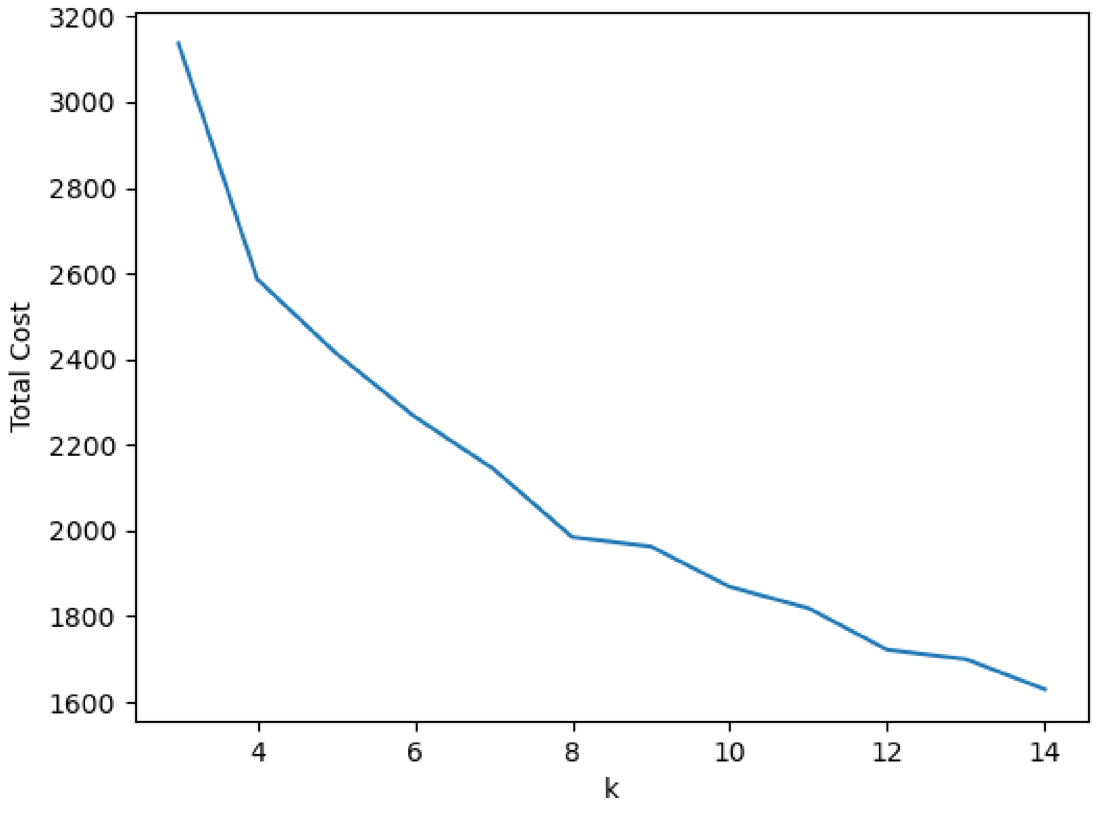
The all shortest paths algorithm is used to convert the costs into the minimum costs matrix for any path between two nodes and to construct the minimum paths matrix. The minimum paths matrix is a tree and only needs to store the next node because of the optimal substructure of a minimum path.

2.4 Minimum Cost Network

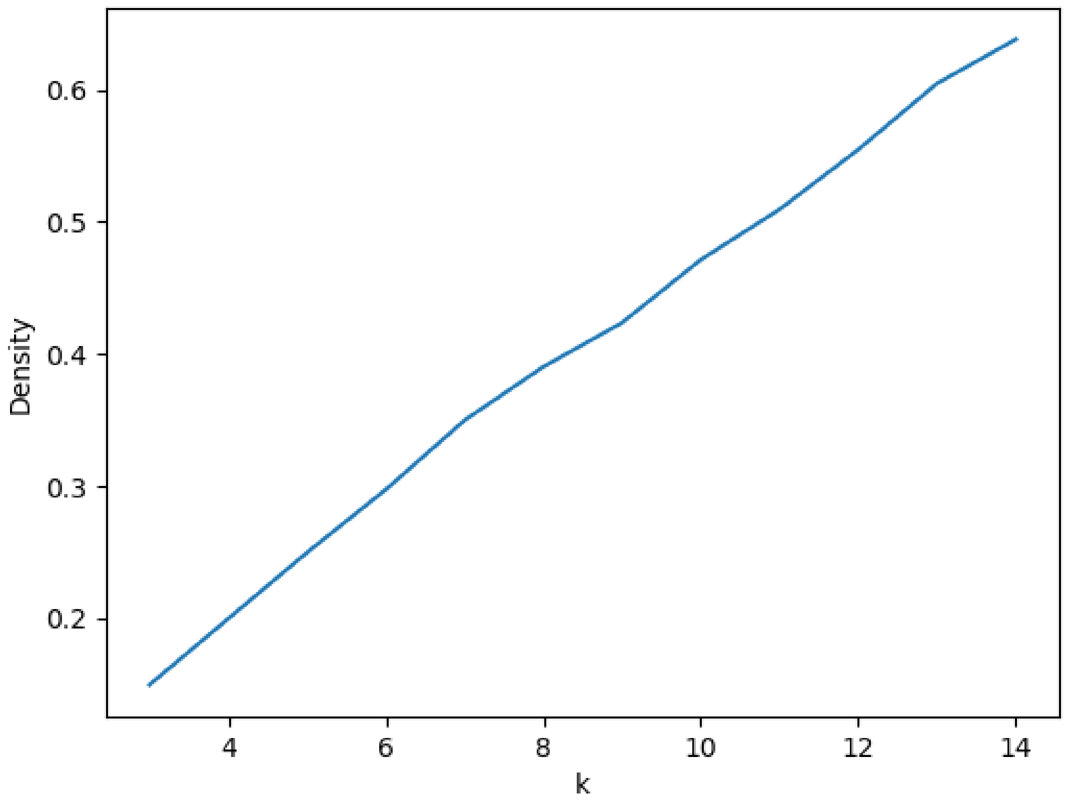
```
for i in range(N):
    for i2 in range(N):
        edge_src = i
        edge_dst = min_paths[i][i2]

        while edge_dst != i2:
            net[edge_src][edge_dst] += demands[i][i2]
            edge_src = edge_dst
            edge_dst = min_paths[edge_dst][i2]
        net[edge_src][edge_dst] += demands[i][i2]
```

The minimum cost network is constructed by using the minimum paths and demands matrix. The edges in all minimum paths between nodes are used to set the weights of all edges in the minimum cost network. The weight of an edge in the minimum cost network is the sum of the demands of all of the minimum cost paths that use the edge.

3 Results

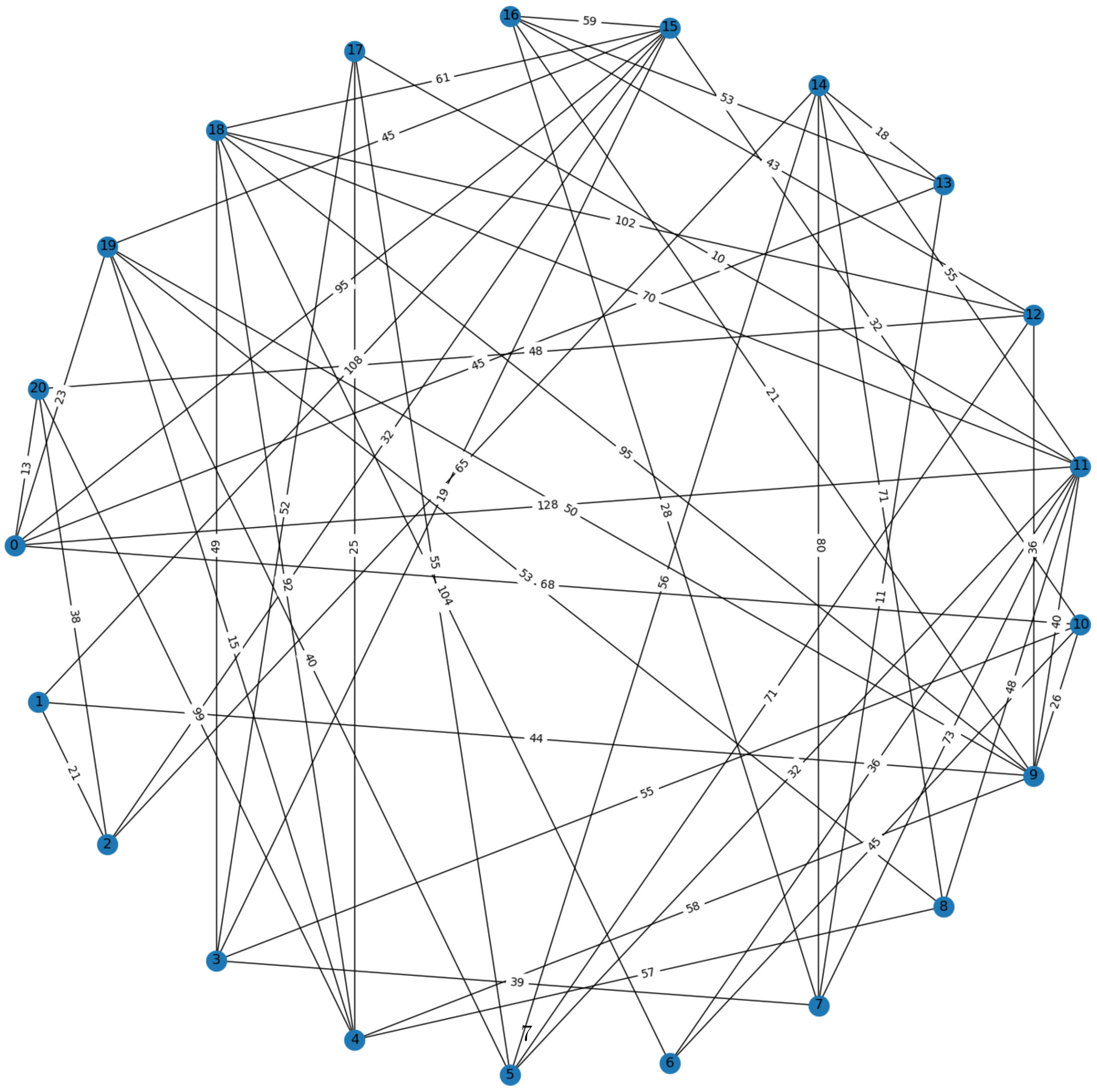


3200
3000
2800
2600
2400
2200
2000
1800
1600
Total Cost
4
6
8
10
12
14
k

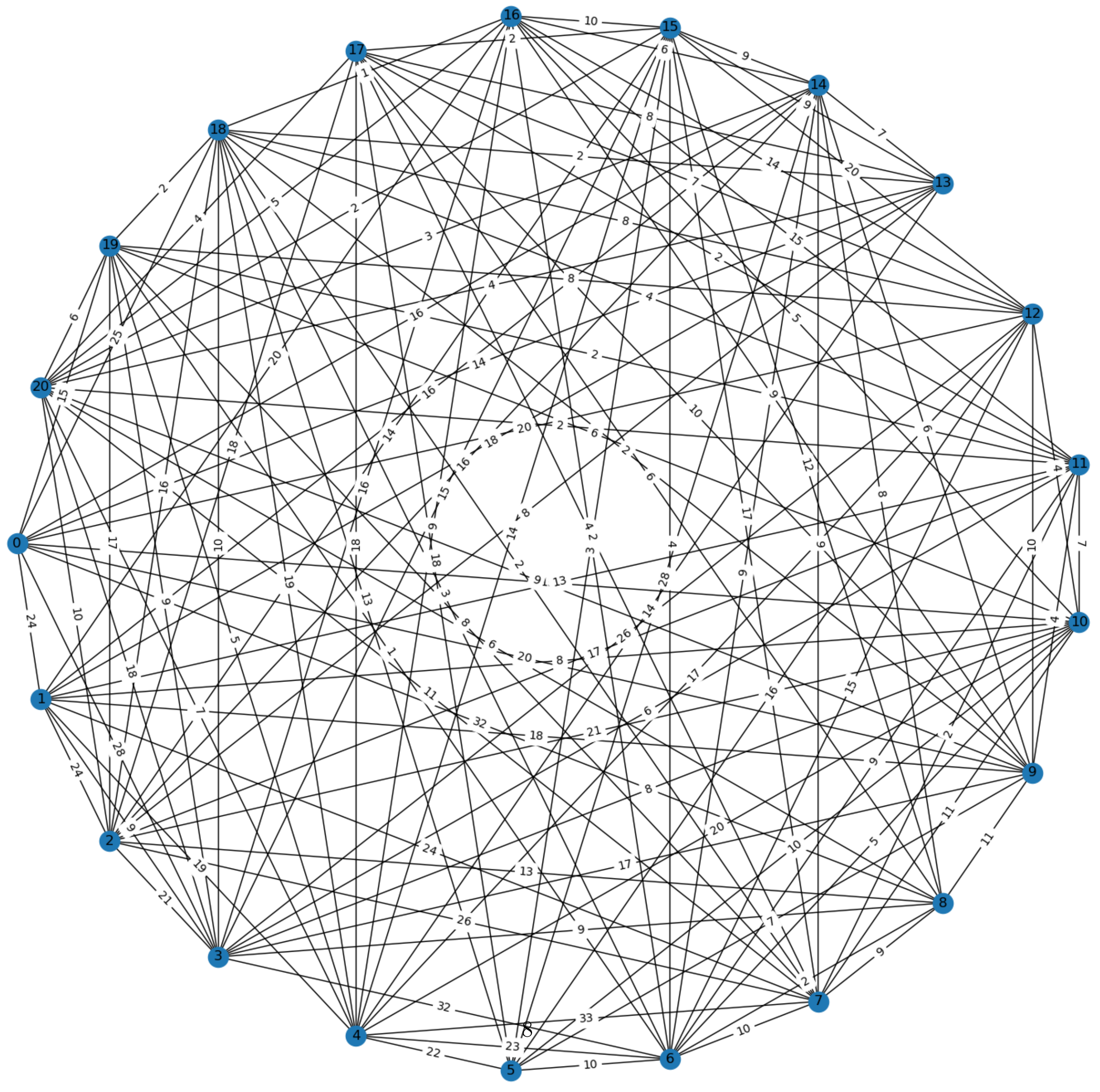


0.2
0.3
0.4
0.5
0.6
Density
4
6
8
10
12
14
k

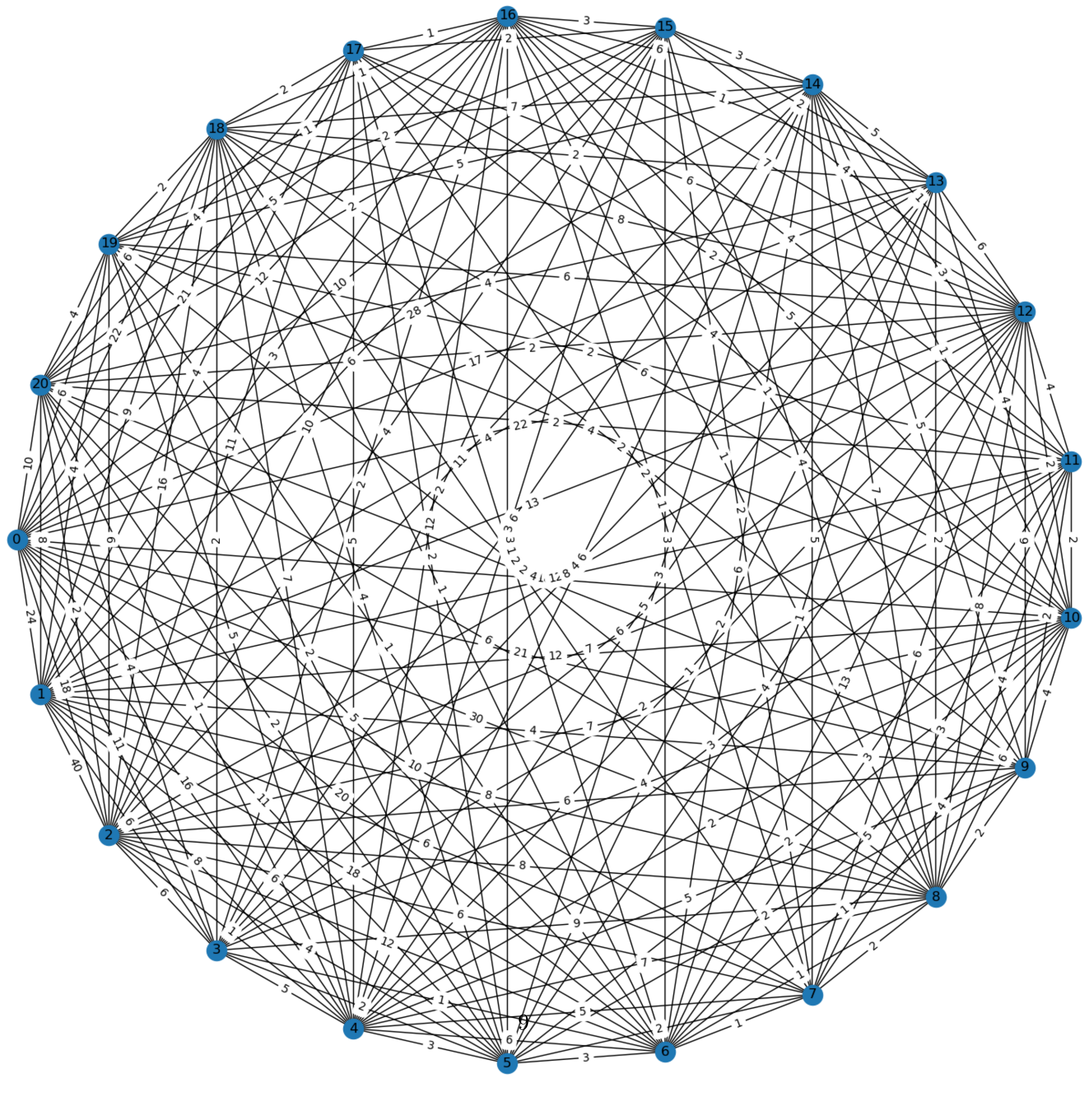
k = 3



k = 8



k = 14



4 Explanation

The total cost generally decreases with k because k determines the number of edges out of a node with a weight of 1 as opposed to 100. More edges with smaller weights will generally decrease the cost.

The density generally increases with k because k determines the number of edges out of a node with a weight of 1 as opposed to 100. More edges with smaller weights increases the number of edges that will be used in the minimum cost network.

The visual representation of the graphs confirms the increase in density with respect to k and also the decrease in total cost with respect to k. More edges can be seen as k increases and the weights of each edge are on average lower as k increases.

5 Instructions

Uses python 3.10.8, numpy 1.23.4, matplotlib 3.5.2, and networkx 2.8.7. Tested on Asahi Linux aarch64.

To run the program, write the source code to a file and execute

$ python3 <filename>

where filename is the name of the file.

6 Appendix

6.1 Notes

The UTD ID was permuted for privacy reasons.

6.2 Source Code

```
#!/usr/bin/env python3
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np

N = 21 # number of nodes
id = '648274120264827412026'
test_min_r = 3
test_max_r = 14 + 1
```

```
# module 2
def min_net(costs, demands, net):
    min_paths = np.empty((N, N), dtype=int)

    for i in range(N):
        min_paths[i] = np.arange(N)

    for i in range(N):
        for i2 in range(N):
            for i3 in range(N):
                min_cost_i = costs[i2][i] + costs[i][i3]
                if costs[i2][i3] > min_cost_i:
                    min_paths[i2][i3] = min_paths[i2][i]
                    costs[i2][i3] = min_cost_i

    #print(min_paths)
    #print(costs)

    for i in range(N):
        for i2 in range(N):
            edge_src = i
            edge_dst = min_paths[i][i2]

            while edge_dst != i2:
                net[edge_src][edge_dst] += demands[i][i2]
                edge_src = edge_dst
                edge_dst = min_paths[edge_dst][i2]
            net[edge_src][edge_dst] += demands[i][i2]

# module 3
def show(densities, net, total_costs):
    x = np.arange(test_min_r, test_max_r)

    plt.plot(x, total_costs[test_min_r:test_max_r])
    plt.xlabel('k')
    plt.ylabel('Total Cost')
    plt.savefig('costs.png')
    plt.clf()

    plt.plot(x, densities[test_min_r:test_max_r])
    plt.xlabel('k')
    plt.ylabel('Density')
    plt.savefig('densities.png')
    plt.clf()
    plt.figure(figsize=(20, 20))

    for i in [3, 8, 14]:
        G = nx.from_numpy_matrix(net[i])
        layout = nx.shell_layout(G)
        plt.title(f'k = {i}')
        nx.draw(G, layout, with_labels=True)
        edge_labels = nx.get_edge_attributes(G, 'weight')
        nx.draw_networkx_edge_labels(G, pos=layout, edge_labels=edge_labels)
        plt.savefig(f'graph{i}.png')
        plt.clf()

def main():
    demands = np.empty((N, N), dtype=int)
    costs = np.full((N, N), 100)
    net = np.zeros((N, N, N), dtype=int)
    total_costs = np.zeros(N, dtype=int)
    densities = np.zeros(N)

    # module 1
    for i in range(len(id)):
        for i2 in range(len(id)):
            demands[i][i2] = abs(int(id[i]) - int(id[i2]))

    rng = np.random.default_rng()
    rand_set = tuple(range(N))
    num_dir_edges = N * (N - 1)

    for r in range(test_min_r, test_max_r):
        costs.fill(100)
        np.fill_diagonal(costs, 0)

        for i in range(N):
            indices = rng.choice(rand_set[:i] + rand_set[i + 1:], r, replace=False)
            costs[i][indices] = 1

        min_net(costs, demands, net[r])
        total_costs[r] = np.sum(demands * costs)
        densities[r] =  float(np.count_nonzero(net[r])) / num_dir_edges

    show(densities, net, total_costs)

if __name__ == '__main__':
    main()
```