Algorithmic Aspects of Telecommunication Networks

Project 1

Kevin Chen

E-mail: nkc160130@utdallas.edu

Table of Contents

Overview

The goal of this project is to create software that performs the following two functions:

- *Input*: The software receives the number of nodes (N), traffic demand values (b_{ij}, in Mbit/s), a constant k and the unit cost values for potential links (a_{ij}).
- *Output*: The software generates a network topology with capacities assigned to links according to the shortest path based fast solution method, and calculates the total cost of the network.

The total optimal cost of the network can be represented by the following equation:

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

where k, l are the starting vertex and ending vertex in which the shortest path and cost is calculated between, respectively, E_{kl} is the set of edges that are on the minimum cost path $k \to l$, as determined by the algorithm, and (i, j) is an edge that is a member of the set of minimum edges E_{kl}.

Implementation

This software was coded with Kotlin, which is a general-purpose programming language designed to maintain full compatibility and interoperability with Java code, while simplifying Java syntax, especially with regards to typing. A tool used in this project is Maven, for build automation in the project and integrating external libraries.

The input variables a and b were generated as follows:

- Each a_{ij} value is generated by picking k number of indices $j_1,...,j_k$, which are unique and does not equal to i. Then, the values $a_{ij_1},..., a_{ij_k} = 1$ and, for all other values of $a_{ij} = 250$.
- Each b_{ij} value is generated with a 25-digit number. All of the b_{ij} values will be calculated using this number, with the formula $b_{ij} = |d_i - d_j|$, where d_i and d_j are two digits in the 25-digit number.

The shortest path algorithm, in this software, is implemented using Dijkstra's Shortest Path First Algorithm. The algorithm works as follows [1]:

- Initialize all nodes as unvisited and have a set of visited and unvisited nodes.
- Assign to every node a tentative distance value, with the starting node at 0 and all other nodes at infinity.
- For the current node, calculate the tentative distances from the current node to all its neighbors. Compare the newly calculated value to the current assigned value, with the smaller value assigned.
- Move the current node from the unvisited set to the visited set.
- If the destination node is visited, or the tentative distance between the nodes in the unvisited set is infinity, the algorithm is finished.
- Else, loop back to step 3, with the smallest tentative distance node as the current node.

For implementing this algorithm into the software, I utilized a library called JGraphT [2], which contains a Java class called `DijkstraShortestPath`. This class implements Dijkstra's algorithm using a graph class `SimpleDirectedWeightedGraph` from the same library. GraphViz [3] was used to generate a graphical representation of the network topology.

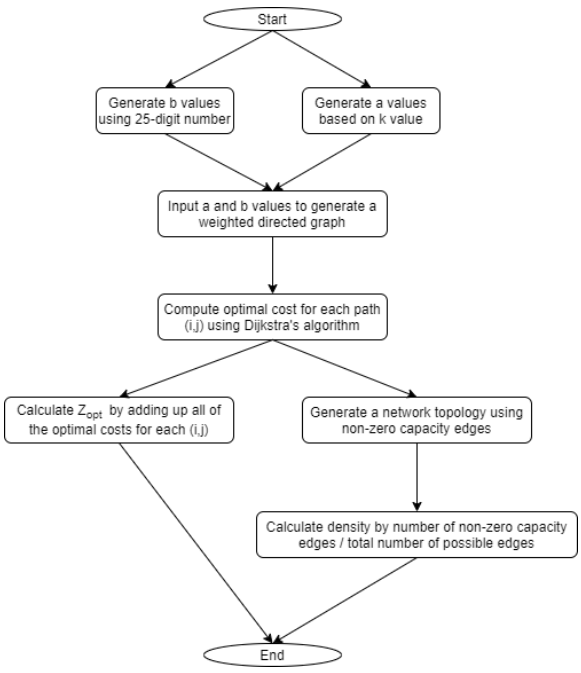

Figure 1. Flowchart detailing how the software generates the network graph and calculates total cost and density

Observations

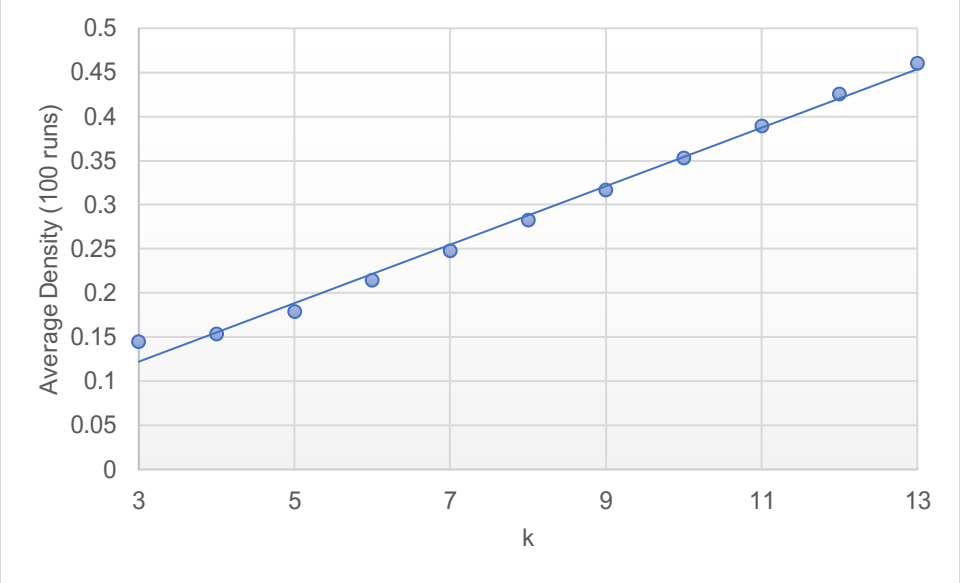

Figure 2. Average Density (100 runs) vs. k

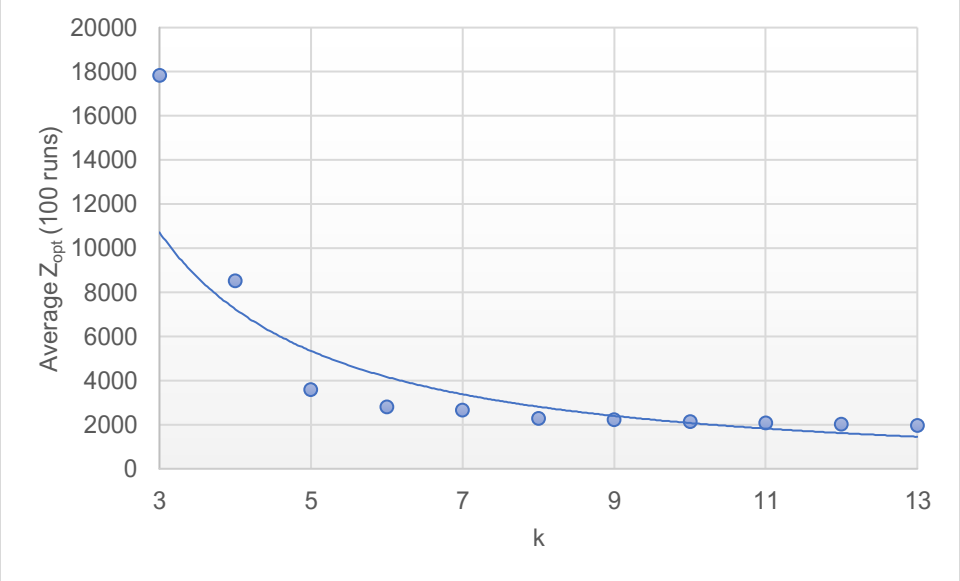

Figure 3. Average Z_{opt} (100 runs) vs. k

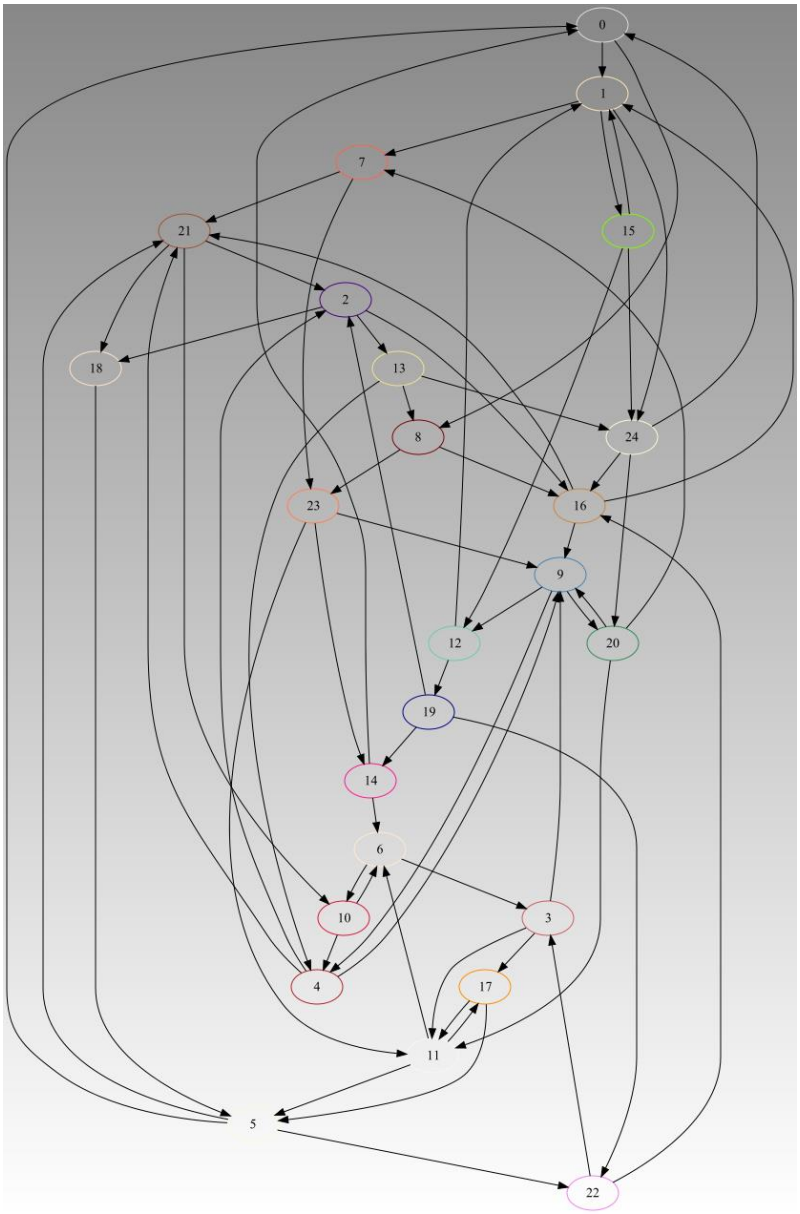

Figure 4. Network topology when k = 3

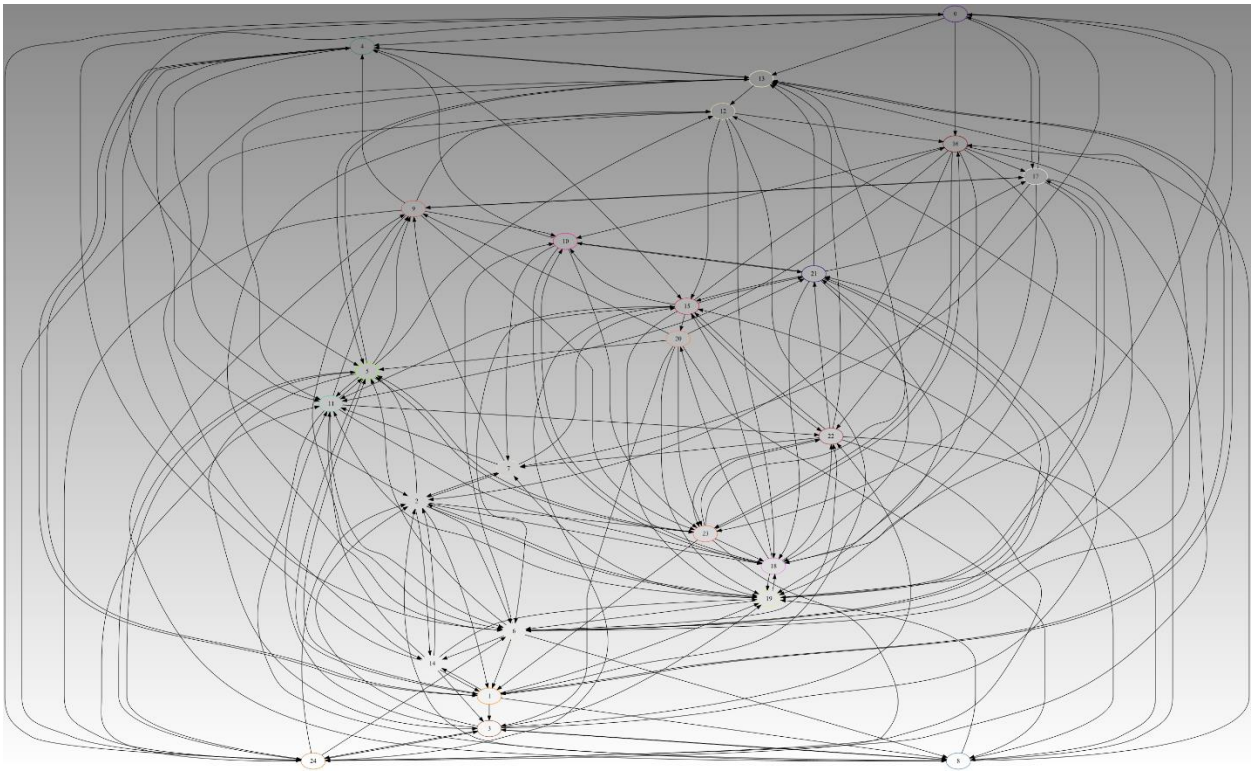

Figure 5. Network topology when k = 8

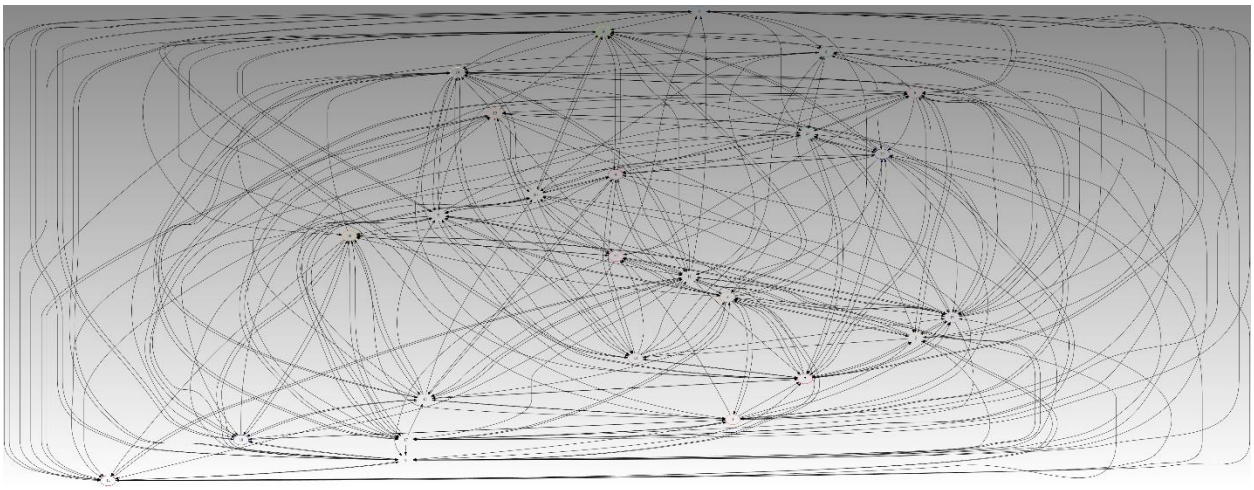

Figure 6. Network topology when k = 13

Justifications for Obtained Diagrams

Density: As seen in Figure 2, density increases linearly with an increase in k. The reason for this is that, as k increases, the number of edges that are assigned a values of 1 instead of 250 increase. Because of this, more outgoing edges from a node will be included in shortest path algorithms, and thus there are less edges that are assigned a capacity of 0.

Z_{opt}*:* As seen in Figure 3, the total cost of the network decreases approximately logarithmically with an increase in k. This is because, as k increases, there are more possible shorter paths out of a node because there are more edges with a cost of 1 instead of 250. It is also more probable, then, that either the shortest path from a vertex i to vertex j will be the edge directly from i to j, or a shorter path that passes from i through other vertices before reaching j, which reduces the number of edges the shortest path algorithm needs to traverse, reducing the total cost.

Network Topologies: As seen in Figures 4, 5 and 6, as k increases, there are more total number of edges in the graph. The reason for this is the same reason as to why the network density increases when k increases; the more outgoing edges that have an a value of 1, the more edges that are included in the shortest path algorithms from one vertex to another.

Readme

Instructions on how to run Project 1:

NOTE: This assumes you already have Maven and Java 11 installed. If not, follow instructions on how to install the software.

1. Create a new folder, with a descriptive name (e.g. `project-1`) and go into this folder.
2. Copy the `pom.xml` file to the root of this folder.
3. Create new folders within this folder, with this structure:
   `src/main/kotlin/com/cs6385`
4. Copy the `NetworkTopGen.kt` and `Runner.kt` files to this folder.
5. Go back to the root `project-1` folder and run
   `mvn clean package`
   in the terminal.
6. A new `target` folder should have been created after the build is successful. Go into this target folder and run
   `java -jar project1-1.0-SNAPSHOT.jar <k-value>`
   in the terminal.

Project Files

Runner.kt

```kotlin
/*
    Runner.kt
    @author Kevin Chen
    Class: CS 6385
    Assignment: Project 1
*/
package com.cs6385

import java.security.SecureRandom
import kotlin.math.abs

const val stuId = "12345678901234567890123456"


fun main(args: Array<String>) {

    // Get k value as command line argument
    if (args.size != 1) {
        throw RuntimeException("Invalid number of arguments, expected 1 (for k
value)")
    } else if (Integer.parseInt(args[0]) <= 2 || Integer.parseInt(args[0]) >=
14) {
        throw RuntimeException("Invalid k value (must be between 3 and 13)")
    }
    val kVal = Integer.parseInt(args[0])

    // Generating a (cost for unit capacity) values for all connections
    val aList = mutableListOf<MutableList<Int>>()
    for (i in stuId.indices) {
        // Generating random indices to have cost of 1
        val randomIndices = mutableListOf<Int>()
        val rand = SecureRandom()

        for (j in 1..kVal) {
            var randValue = rand.nextInt(25)
            while (randValue in randomIndices || randValue == i)
                randValue = rand.nextInt(25)
            randomIndices.add(randValue)
        }

        // Compute a values for a connection
        val nodeConnect = mutableListOf<Int>()

        for (j in stuId.indices) {
            if (j in randomIndices) nodeConnect.add(1)
            else nodeConnect.add(250)
        }
```

```
        aList.add(nodeConnect)

    }

    // Generating b (traffic demand) values for all connections
    val bList = mutableListOf<MutableList<Int>>()
    for (i in stuId.indices) {
        val nodeConnect = mutableListOf<Int>()

        // Compute b values for a connection
        for (j in stuId.indices)
            nodeConnect.add(abs(stuId[i].toInt() - stuId[j].toInt()))

        bList.add(nodeConnect)
    }

    // Running optimization for the network through NetworkTopGen
    var netOptimizer = NetworkTopGen(25, aList, bList)
    println("Total cost of network: " + netOptimizer.genOptimalTopology())

}
```

NetworkTopGen.kt

```
/*
    NetworkTopGen.kt
    @author Kevin Chen
    Class: CS 6385
    Assignment: Project 1
*/
package com.cs6385

import guru.nidi.graphviz.attribute.Color
import guru.nidi.graphviz.attribute.Font
import guru.nidi.graphviz.attribute.GraphAttr
import guru.nidi.graphviz.engine.Format
import guru.nidi.graphviz.engine.Graphviz
import org.jgrapht.alg.shortestpath.DijkstraShortestPath
import org.jgrapht.graph.SimpleDirectedWeightedGraph
import org.jgrapht.graph.DefaultWeightedEdge
import java.lang.IndexOutOfBoundsException
import guru.nidi.graphviz.model.Factory.*
import java.io.File
import java.security.SecureRandom

class NetworkTopGen {
    private var numNodes: Int

    private var aList: MutableList<MutableList<Int>>

    private var bList: MutableList<MutableList<Int>>

    private var netGraph: SimpleDirectedWeightedGraph<Int,
DefaultWeightedEdge>

    constructor(num_nodes: Int, a_list: MutableList<MutableList<Int>>, b_list:
MutableList<MutableList<Int>>) {
        this.numNodes = num_nodes
        this.aList = a_list
        this.bList = b_list

        this.netGraph = SimpleDirectedWeightedGraph<Int,
DefaultWeightedEdge>(DefaultWeightedEdge::class.java)

        // Generate graph vertices
        for (i in 0 until num_nodes) {
            netGraph.addVertex(i)
        }
        // Generate graph edges
        for (i in 0 until num_nodes) {
            for (j in 0 until num_nodes) {
                if (i != j)
                    netGraph.setEdgeWeight(netGraph.addEdge(i, j),
a_list[i][j] / 1.0)
```

```
            }
        }
    }

    private fun findShortestPathCost(startV: Int, endV: Int): Int {
        // Vertex or vertices are out of range
        if (startV < 0 || startV >= numNodes || endV < 0 || endV >= numNodes)
            throw IndexOutOfBoundsException("Start and/or end vertex is out of
range\nStart vertex: $startV \nEnd vertex: $endV")

        val shortestPath = DijkstraShortestPath.findPathBetween(netGraph,
startV, endV)

        // Get weights of edges on the path
        var shortestPathWeightSum = 0
        for (edge in shortestPath.edgeList)
            shortestPathWeightSum += netGraph.getEdgeWeight(edge).toInt()

        return shortestPathWeightSum
    }

    private fun findShortestPathEdges(startV: Int, endV: Int) :
MutableList<Pair<Int, Int>> {
        // Vertex or vertices are out of range
        if (startV < 0 || startV >= numNodes || endV < 0 || endV >= numNodes)
            throw IndexOutOfBoundsException("Start and/or end vertex is out of
range\nStart vertex: $startV \nEnd vertex: $endV")

        val shortestPath = DijkstraShortestPath.findPathBetween(netGraph,
startV, endV)

        // No edges in shortest path
        if(shortestPath.vertexList.size < 2)
            return mutableListOf<Pair<Int, Int>>()

        val edgeList = mutableListOf<Pair<Int, Int>>()

        var i = 0
        while(i+1 < shortestPath.vertexList.size) {
            edgeList.add(Pair(shortestPath.vertexList[i],
shortestPath.vertexList[i+1]))
            i++
        }

        return edgeList
    }

    fun genOptimalTopology(): Int {
        val optimalCostMap = HashMap<Pair<Int, Int>, Int>()
        val pathVisited = HashMap<Pair<Int, Int>, Boolean>()
        for (i in 0 until numNodes) {
            for (j in 0 until numNodes) {
```

```
            optimalCostMap[Pair(i, j)] = 0
            pathVisited[Pair(i, j)] = false
        }
    }

    var totalOptimalCost = 0

    // Get optimal cost for all edges (links)
    for (i in 0 until numNodes) {
        for (j in 0 until numNodes) {
            var thisCost = bList[i][j] * findShortestPathCost(i, j)

            for(edge in findShortestPathEdges(i, j))
                pathVisited[edge] = true

            optimalCostMap[Pair(i, j)] = thisCost
            totalOptimalCost += thisCost
        }
    }

    // Generate graph colors
    var rand = SecureRandom()
    var colorList = mutableListOf<Color>()
    for (i in 0 until numNodes) {
        var randValue: Int
        var thisColor: Color
        do {
            randValue = rand.nextInt(25)
            when (randValue) {
                0 -> thisColor = Color.ALICEBLUE
                1 -> thisColor = Color.CORAL
                2 -> thisColor = Color.VIOLET
                3 -> thisColor = Color.MEDIUMAQUAMARINE
                4 -> thisColor = Color.STEELBLUE
                5 -> thisColor = Color.MAROON
                6 -> thisColor = Color.DEEPPINK
                7 -> thisColor = Color.SEAGREEN
                8 -> thisColor = Color.SIENNA
                9 -> thisColor = Color.NAVYBLUE
                10 -> thisColor = Color.PERU
                11 -> thisColor = Color.BLANCHEDALMOND
                12 -> thisColor = Color.LEMONCHIFFON
                13 -> thisColor = Color.MOCCASIN
                14 -> thisColor = Color.INDIGO
                15 -> thisColor = Color.CORNSILK
                16 -> thisColor = Color.CRIMSON
                17 -> thisColor = Color.GAINSBORO
                18 -> thisColor = Color.BISQUE
                19 -> thisColor = Color.INDIANRED
                20 -> thisColor = Color.TOMATO
                21 -> thisColor = Color.FIREBRICK
                22 -> thisColor = Color.DARKORANGE
```

```
                    23 -> thisColor = Color.KHAKI
                    24 -> thisColor = Color.LAWNGREEN
                    else -> thisColor = Color.BLACK
                }

            } while (thisColor in colorList)
            colorList.add(thisColor)
        }

        // Generate graph image
        var optGraph = mutGraph("Optimal Network
Topology").setDirected(true).graphAttrs()
            .add(Color.WHITE.gradient(Color.rgb("888888")).background().angle(
90))
        for (i in 0 until numNodes) {
            for (j in 0 until numNodes) {
                if (pathVisited[Pair(i, j)] == true && optimalCostMap[Pair(i,
j)] != 0)
                    optGraph.add(mutNode("" +
i).add(colorList[i]).addLink(mutNode("" + j)))
            }
        }

Graphviz.fromGraph(optGraph).render(Format.PNG).toFile(File("optGraph.png"))

        println("Density of network: " + String.format("%.5f",
optGraph.edges().size / ((numNodes * numNodes).toDouble() - numNodes)))
        return totalOptimalCost
    }


}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.cs6385</groupId>
    <artifactId>project1</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>com.cs6385 Project 1</name>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <kotlin.version>1.3.72</kotlin.version>
        <kotlin.code.style>official</kotlin.code.style>
        <junit.version>4.12</junit.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.13.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-slf4j-impl</artifactId>
            <version>2.13.0</version>
        </dependency>
        <dependency>
            <groupId>org.jetbrains.kotlin</groupId>
            <artifactId>kotlin-stdlib</artifactId>
            <version>${kotlin.version}</version>
        </dependency>
        <dependency>
            <groupId>org.jetbrains.kotlin</groupId>
            <artifactId>kotlin-test-junit</artifactId>
            <version>${kotlin.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
```

```
        <dependency>
            <groupId>org.jgrapht</groupId>
            <artifactId>jgrapht-core</artifactId>
            <version>1.4.0</version>
        </dependency>
        <dependency>
            <groupId>guru.nidi</groupId>
            <artifactId>graphviz-java</artifactId>
            <version>0.16.2</version>
        </dependency>
    </dependencies>

    <build>
        <sourceDirectory>src/main/kotlin</sourceDirectory>
        <testSourceDirectory>src/test/kotlin</testSourceDirectory>

        <plugins>
            <plugin>
                <groupId>org.jetbrains.kotlin</groupId>
                <artifactId>kotlin-maven-plugin</artifactId>
                <version>${kotlin.version}</version>
                <executions>
                    <execution>
                        <id>compile</id>
                        <phase>compile</phase>
                        <goals>
                            <goal>compile</goal>
                        </goals>
                    </execution>
                    <execution>
                        <id>test-compile</id>
                        <phase>test-compile</phase>
                        <goals>
                            <goal>test-compile</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <version>3.2.0</version>
                <configuration>
                    <archive>
                        <manifest>
                            <addClasspath>true</addClasspath>
                            <classpathPrefix>libs/</classpathPrefix>
                            <mainClass>
                                com.cs6385.RunnerKt
                            </mainClass>
                        </manifest>
                    </archive>
```

```
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.3</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

References

[1] https://en.wikipedia.org/wiki/Dijkstra's_algorithm

[2] https://jgrapht.org/

[3] https://graphviz.org/