Algorithmic Aspects of Telecommunication Networks

Project 2

Kevin Chen

E-mail: nkc160130@utdallas.edu

Table of Contents

Overview

The purpose of this project is to study how the total network reliability depends on individual link reliabilities. It also studies how random errors in the dataset will affect the total network reliability. The software will perform the following functions:

1. *Input:* The software receives the parameter p (or k, depending on what experiment is run) from the user. If this parameter is p, the reliability of link i ($i = 1,2,...,10$), is calculated with the following formula:

$$p_i = p^{\left\lceil \frac{d_i}{3} \right\rceil}$$

where d_i is the i^{th} digit in the 10-digit student ID. There is also a fixed network topology, which is a complete undirected graph with $n = 5$ nodes and $m = 10$ edges. Every node is connected with an undirected edge to every other node but itself.

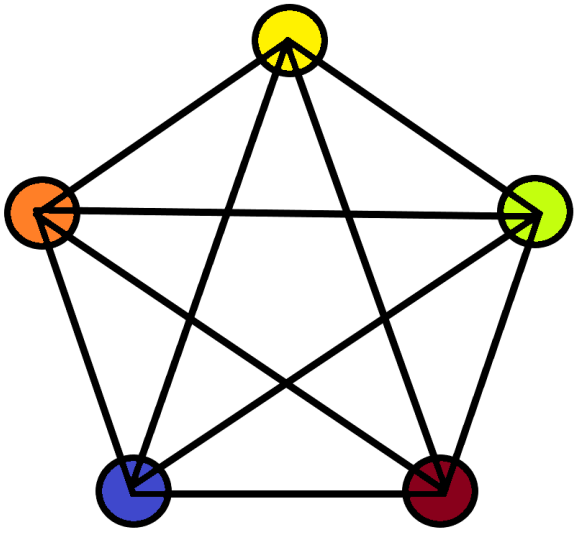

Figure 1. A visual representation of the undirected graph with $n = 5$ nodes and $m = 10$ edges, used to simulate a network topology

2. *Output:* The software calculates and prints the total network reliability, using the method of exhaustive enumeration.

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. JGraphT [1] is used to render the network topology.

To calculate the network reliability of the network, the software first generates all possible states that the network topology can be in; that is, all the possible combinations of up and down links in the network. This can be generated using a very simple algorithm.

```
graphStates = Array<String>

for i = 0, i++, while i < 2^(numEdges):
    binaryRep = i.toBinaryString()
    while binaryRep.size < numEdges:
        binaryRep = "0" + binaryRep
    graphStates.add(binaryRep)
```

Figure 2. Pseudocode for generating graph states

The graph states are generated by simply generating binary strings of length m of all integers from 0 to 2^m-1. This will generate all possible states of the network, from completely disconnected to completely connected

(0000000000, 0000000001, 0000000010, ..., 1111111111).

Then, going through every possible state of the network, the network topology of that state is generated, and from that, an algorithm is run to determine whether the network is connected or not. If the generated network is connected, the network is up; if the generated network is not connected, the network is down. This is how the up/down value for each state is generated.

```
visitedNodes = Set<Int>
graphEdges = Array<Int>
connected = Boolean
numNodes = Int

for edge in graphEdges:
    visitedNodes.add(edge.source)
    visitedNodes.add(edge.destination)

if visitedNodes.size == numNodes:
    connected = True
else if visitedNodes.size < numNodes:
    connected = False
```

Figure 3. Pseudocode for algorithm for determining graph connectivity

Since all of the edges of the graph are known to the software, determining whether the graph is connected is trivial. The algorithm goes through all the edges of the graph, adding the source and destination nodes to a set, which will only take unique nodes. If the size of the set is equal to the total number of nodes in the graph, then the graph is connected. If the size of the set is smaller than the total number of nodes in the graph, then the graph is not connected.

After the network up/down states are generated, the total network reliability is calculated by taking the up states only (as down states' reliabilities will automatically be 0), and calculating the total network reliability for each state using the following formula:

$$R_{network} = \sum_{i=1}^{2^m - d} \prod_{j=1}^{m} \begin{cases} R_j, & s_j = 1 \\ 1 - R_j, & s_j = 0 \end{cases}$$

where s_j is the state of the link (i.e. $s_j = 1$ when the link is up, $s_j = 0$ if it is down), and d is the number of down states. This is implemented by the following pseudocode:

```
totalNetworkReliability = Double 0.0
upDownStates = Array<Boolean>
pValue = Array<Double>

for state in upDownStates:
    if state == true:
        thisReliability = Double 1.0

        i = Int 0
        for edge in state:
            if edge == '1'
                thisReliability *= pValue[i]
            else if edge == '0'
                thisReliability *= (1 - pValue[i])

            i += 1

        totalNetworkReliability += thisReliability
```

Figure 4. Pseudocode implementing the above equation for $R_{network}$

Justification for Structuring of Program

The code implementation is also structured with debugging, checking correctness, and building algorithm changes in mind. A design decision that I made to support all three of these is using a separate class for the above algorithms, in the NetworkReliability class, and separating them from a Runner class. Arrays such as p values, up/down states, and possible states are also stored in this class. Also, these algorithms are implemented in different methods within the NetworkReliability class. This makes it easy to debug, because if one algorithm has incorrect code, it is much easier to isolate this problem to the problematic code. Also, at each step in the process of finding the total reliability, I can check the output of every algorithm by printing the output of the method and determining if it is correct. In terms of building algorithm changes, especially with regards to k, I just need to accept a new parameter, add a new variable to the NetworkReliability class, create a new random index generator method in the class, and change the up/down state algorithm to flip the system condition for the aforementioned random indices if k is set.

Observations

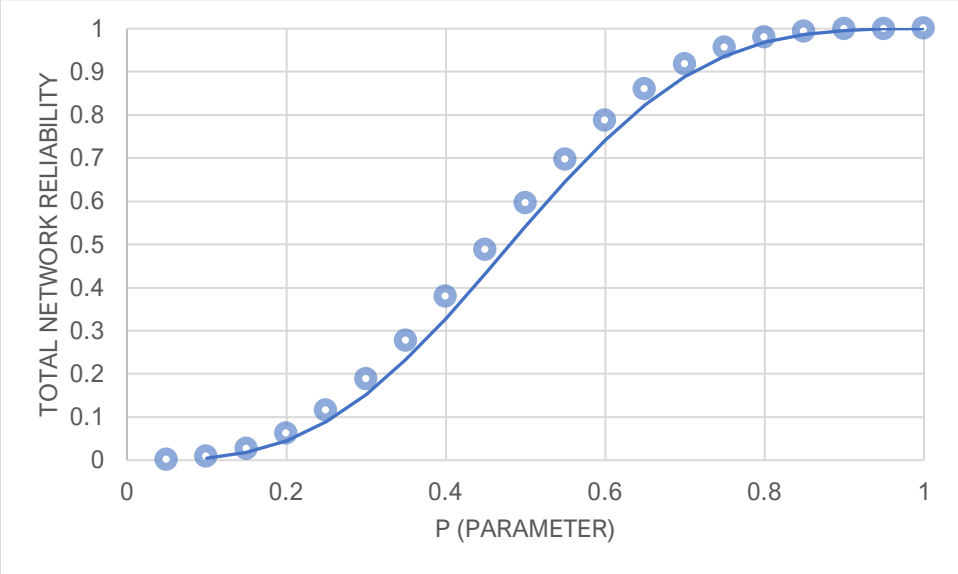

Figure 5. Total Network Reliability vs. p

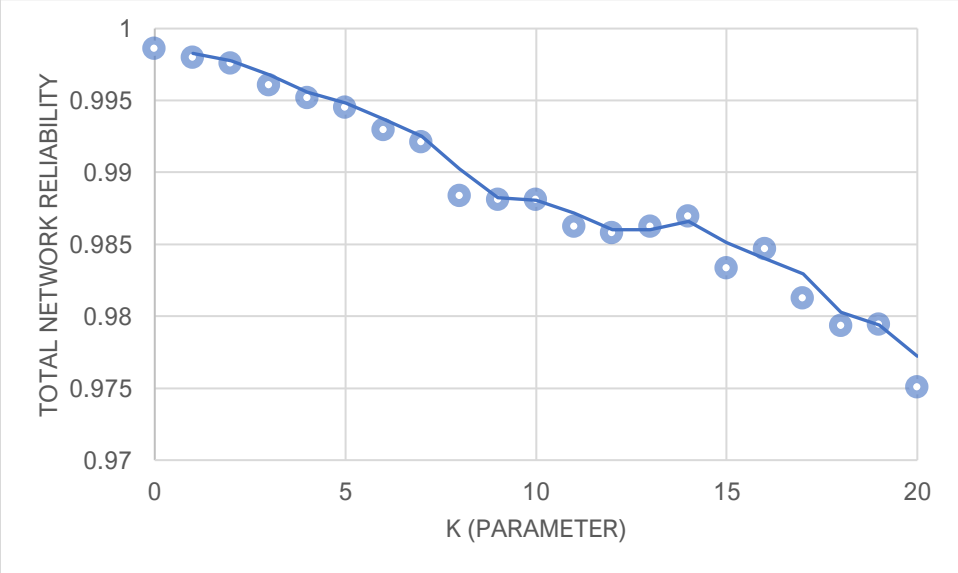

Figure 6. Total Network Reliability vs. k

Justification for Obtained Diagrams

Figure 5: Total network reliability increases when p increases. This is simply because, when p is larger, the value of each individual p_i increases. (Consider $p = 0.05$ vs. $p = 0.5$: if $p_1 = 0.05^2$ vs. $p_1 = 0.5^2$, then $p_1 = 0.0025$ vs. $p_1 = 0.25$). This diagram looks like the sigmoid function, which is a function commonly used to convert values into probabilities. This resemblance to the sigmoid function makes sense; consider that the total network reliability is a probability, so it is between 0 and 1, and each link reliability is also a probability, and hence must also be between 0 and 1.

Figure 6: Total network reliability generally decreases when k increases, for the same value of p $= 0.9$. There is a less concrete relationship between total network reliability and k, simply because the actual k indices which are flipped are randomly chosen. It seems that, generally, as the number of indices that are flipped (k) increases, it is possible that more system conditions that were up are now marked as down, decreasing the total network reliability. Conversely, it is also possible that more system conditions that were down are now marked as up, possibly increasing the network reliability. Even if all the states that were flipped were down, however, it would not affect total network reliability as much as if all the states that were flipped were up, as evidenced by the decreasing slope to the right for $p = 0.9$ in Figure 5 vs. the increasing slope to the left. This can also be seen by looking at a distribution plot of the different random values of total network reliabilities in Figure 7, for $k = 20$. To minimize the effect of randomization on the variance of the total network reliabilities, each k value was run 500 times on the program.

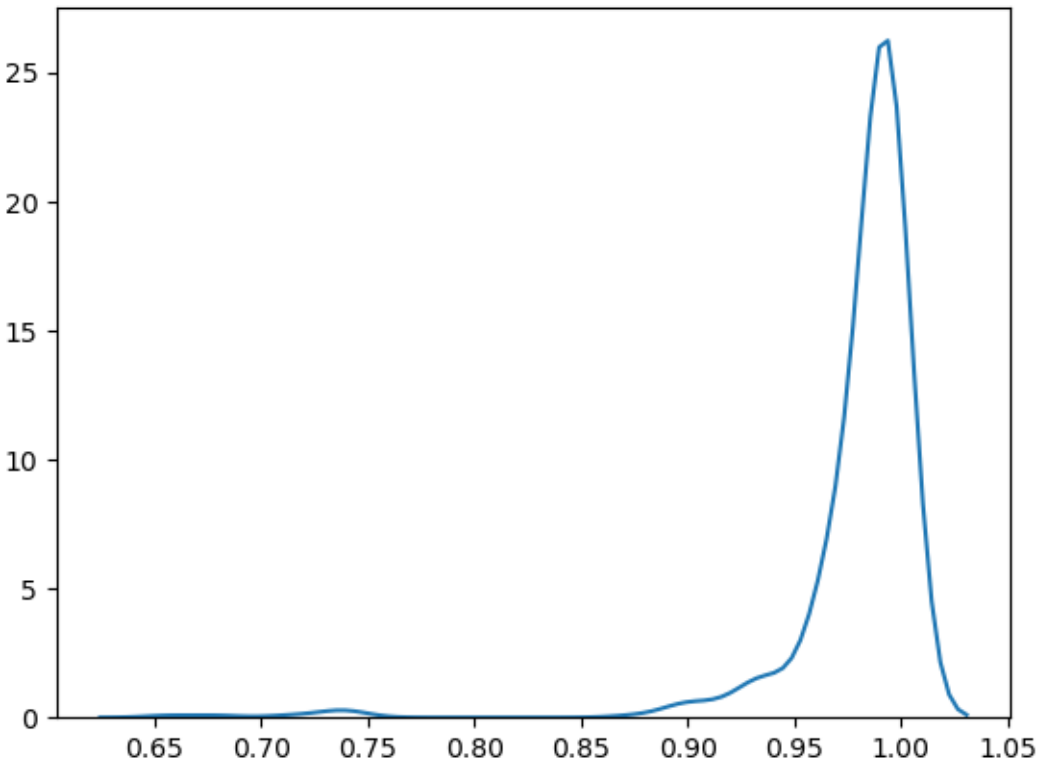

Figure 7. Total network reliability distribution when $k = 20$

The median is close to the no-error value for $p = 0.9$ (when $k = 0$), but the average is dragged down by lower values; the distribution is skewed to the left.

Readme

Instructions on how to run Project 2:

Note 1: These instructions assume that you already have Maven and Java 11 installed. If not, please install these first before following these steps.

1. Create a new folder, with a descriptive name (e.g. `project-2`) and go to this directory.
2. Copy the `pom.xml` to the root of this folder.
3. Create new folders within this folder, with this structure:
   `src/main/kotlin/com/cs6385`
4. Copy the `NetworkReliability.kt` and `Runner.kt` files to this folder.
5. Go back to the root `project-2` 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 project2-1.0-SNAPSHOT.jar <isKValue> <parameter>`
   in the terminal.

Note 2: `isKValue` is a Boolean value. If `isKValue` is `true`, the parameter will be k. If `isKValue` is `false`, the parameter will be p.

Project Files

Runner.kt

```
/*
    Runner.kt
    @author Kevin Chen
    Class: CS 6385
    Assignment: Project 2
*/
package com.cs6385

import org.jgrapht.graph.DefaultUndirectedGraph
import org.jgrapht.graph.DefaultWeightedEdge
import java.lang.IndexOutOfBoundsException
import java.lang.RuntimeException
import java.text.DecimalFormat
import kotlin.math.ceil
import kotlin.math.pow

// Constants
const val numNodes = 5
const val stuID = "1234567890"

fun main(args: Array<String>) {
    // This program requires 2 parameters
    if (args.size != 2)
        throw RuntimeException("ERROR: Invalid number of
arguments!\nArguments: <isKValue> <value>")

    // Inputs for algorithm
    val networkTop = DefaultUndirectedGraph<Int,
DefaultWeightedEdge>(DefaultWeightedEdge::class.java)
    val pVals = mutableListOf<Double>()
    val edgeIndices = mutableMapOf<Int, Pair<Int, Int>>()

    // Generate all inputs

    // Generate the graph
    // Generate graph vertices
    for (i in 0 until numNodes)
        networkTop.addVertex(i)

    // Generate graph edges
    var k = 0
    for (i in 0 until numNodes) {
        for (j in 0 until numNodes) {
            if (i != j && !networkTop.containsEdge(i, j)) {
                networkTop.addEdge(i, j)
                edgeIndices[k] = Pair(i, j)
                k++
```

```
            }
        }
    }

    val isKValue = args[0].toBoolean()
    var pParam = if (isKValue) 0.9 else args[1].toDouble()

    // p parameter not between 0 and 1
    if (pParam < 0 || pParam > 1)
        throw IndexOutOfBoundsException("ERROR: p value needs to be between 0
and 1!")

    // Generate pi (reliability) values
    for (i in 0 until networkTop.edgeSet().size)
        pVals.add(pParam.pow(ceil(Integer.parseInt("" + stuID[i]) / 3.0)))

    val netReliability =
        if (isKValue)
            NetworkReliability(pVals, networkTop, edgeIndices,
args[1]).getNetworkReliability()
        else
            NetworkReliability(pVals, networkTop,
edgeIndices).getNetworkReliability()

    if(isKValue)
        println("Total Network Reliability with p = 0.9 and k = " + args[1] +
": $netReliability")
    else
        println("Total Network Reliability with p = $pParam: $netReliability")
}
```

NetworkReliability.kt

```kotlin
package com.cs6385

/*
    NetworkReliability.kt
    @author Kevin Chen
    Class: CS 6385
    Assignment: Project 2
*/

import org.jgrapht.GraphTests.isConnected
import org.jgrapht.graph.DefaultUndirectedGraph
import org.jgrapht.graph.DefaultWeightedEdge
import java.lang.IllegalArgumentException
import java.security.SecureRandom
import kotlin.math.pow

class NetworkReliability {
    private val pVal: MutableList<Double>
    private val networkTop: DefaultUndirectedGraph<Int, DefaultWeightedEdge>
    private val edgeIndices: MutableMap<Int, Pair<Int, Int>>
    private val kVal: String

    private val randErrorIndices: MutableList<Int>
    private val networkStates: MutableList<String>
    private val upDownStates: MutableList<Boolean>

    constructor(
        pVal: MutableList<Double>,
        networkTop: DefaultUndirectedGraph<Int, DefaultWeightedEdge>,
        edgeIndices: MutableMap<Int, Pair<Int, Int>>
    ) {
        this.pVal = pVal
        this.networkTop = networkTop
        this.edgeIndices = edgeIndices
        this.kVal = ""
        this.randErrorIndices = mutableListOf()

        this.networkStates = genNetworkStates()
        this.upDownStates = genUpDownStates()
    }

    constructor(
        pVal: MutableList<Double>,
        networkTop: DefaultUndirectedGraph<Int, DefaultWeightedEdge>,
        edgeIndices: MutableMap<Int, Pair<Int, Int>>, kVal: String
    ) {
        this.pVal = pVal
        this.networkTop = networkTop
        this.edgeIndices = edgeIndices
```

```
        this.kVal = kVal
        if (kVal.toIntOrNull() == null)
            throw IllegalArgumentException("ERROR: k value not an integer!")
        else if (kVal.toInt() < 0 || kVal.toInt() > 20)
            throw IndexOutOfBoundsException("ERROR: k value needs to be
between 0 and 20!")

        this.networkStates = genNetworkStates()
        this.randErrorIndices = genRandErrorIndices()
        this.upDownStates = genUpDownStates()
    }

    private fun genRandErrorIndices(): MutableList<Int> {
        val randIndices = mutableListOf<Int>()

        while (randIndices.size < kVal.toInt()) {
            var randIndex = SecureRandom().nextInt(networkStates.size)
            while (randIndex in randIndices)
                randIndex = SecureRandom().nextInt(networkStates.size)
            randIndices.add(randIndex)
        }

        return randIndices
    }

    private fun genNetworkStates(): MutableList<String> {

        val nStates = mutableListOf<String>()

        // Generating states by getting the binary representation of base 10
numbers from 0 to (2^number of edges)-1
        for (i in 0 until
2.0.pow(networkTop.edgeSet().size.toDouble()).toInt()) {
            var binI = Integer.toBinaryString(i)

            // Make each binary representation size equal to the number of
edges
            while (binI.length < networkTop.edgeSet().size)
                binI = "0$binI"

            nStates.add(binI)
        }

        return nStates
    }

    private fun genUpDownStates(): MutableList<Boolean> {
        val udStates = mutableListOf<Boolean>()

        // Go through all possible states
        for ((i, state) in networkStates.withIndex()) {
            val networkStateTop = networkTop.clone() as
```

```
DefaultUndirectedGraph<Int, Int>

            // Remove the edges as indicated by the state
            for ((j, edge) in state.withIndex()) {
                if (edge == '0')
                    networkStateTop.removeEdge(edgeIndices[j]?.first,
edgeIndices[j]?.second)
            }

            // Determine whether this state's topology is connected or not
            // Also, if k is set, flip the truth value of connection of the
graph randomly
            if (i in randErrorIndices)
                udStates.add(!isConnected(networkStateTop))
            else
                udStates.add(isConnected(networkStateTop))
        }

        return udStates
    }

    fun getNetworkReliability(): Double {
        var totalNetworkReliability = 0.0

        for ((i, udState) in upDownStates.withIndex()) {
            // State is up (true)
            if (udState) {
                // Iterate through all edges, find reliability for the edge
and
                // compute for total reliability
                var thisReliability = 1.0
                for ((j, edge) in networkStates[i].withIndex())
                    thisReliability *= if (edge == '1') pVal[j] else 1 -
pVal[j]

                totalNetworkReliability += thisReliability
            }
        }

        return totalNetworkReliability
    }
}
```

pom.xml

```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>project2</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>com.cs6385 Project 2</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>
        <java.version>11</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.jgrapht</groupId>
            <artifactId>jgrapht-core</artifactId>
            <version>1.4.0</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>
    </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://jgrapht.org/