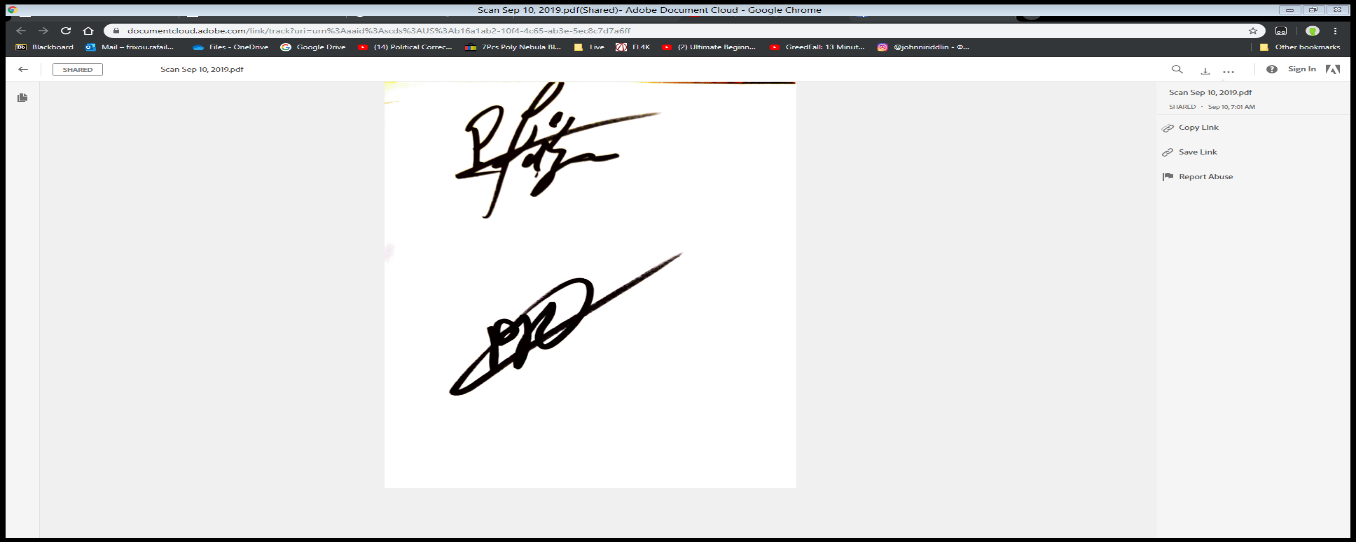
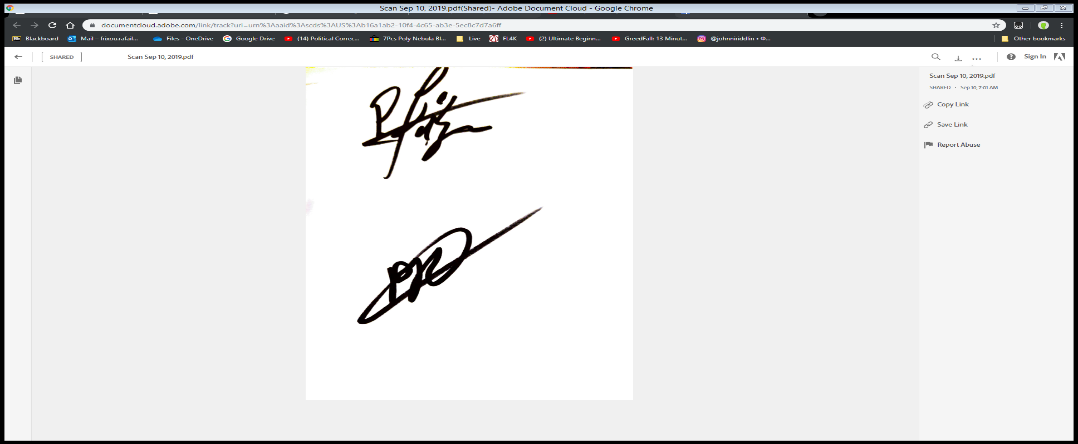
ECE 424 FINAL PROJECT

University of Cyprus [2020]



We hereby consent that this is our own work:

Rafail Frixou Panayiotis Procopi

# General Overview:

For the final project for this class we were asked to create a random component system simulation and calculate the system’s reliability by calculating the MTTF for the system. We were then asked to collect data for different ranges on input to the system and provide a comparative report on the data collected.

# Component System Generation/Modelling

For the purposes of this project we have chosen to model the component system as a directional graph. The first node of the graph represents the system input and the last node the system output. Each transition between the nodes of the graph generated is analogous to a wire extending from one corresponding component to the other. All graphs are generated based on two parameters. Density and Nodes. Density corresponds to the probability of an edge being generated between two nodes. Nodes corresponds to the number of components present in the simulated system (graph).

Our simulation run with the following parameter ranges:

* Density: 0.1 – 0.9 with a step-up of 0.1
* Nodes: 6 – 20 with a step-up of 2

We have modelled three different kinds of graphs in order to achieve a broader collection of data. The rules for generating the corresponding graphs are as follows:

1. The Graph is generated randomly and has no additional checks implemented.
2. The Graph is generated with the following rules in mind:
   1. The Edges generated can only be generated from a smaller indexed node to a larger one (Nn -> Nm where n > m). E.g. N3 -> N5.
3. The Graph is generated with the following rules in mind:
   1. The Edges generated can only be generated from a smaller indexed node to a larger one (Nn -> Nm where n > m). E.g. N3 -> N5.
   2. Each Node (excluding the Input and Output nodes, which respectively have only paths being led from and leading to them) must at least be led to and lead to another Node, which in conjunction with the first rule can guarantee that no dead ends exist in our Graph.

# Reliability Calculation:

In order to calculate the Reliability of our generated component system we employ exhaustive search. We calculate all combinations of components working/not working and create a polynomial reliability which we transform to produce the MTTF of our system. The data presented, at the end of this report, has been averaged over n iterations of the algorithm.

# Discussing the Code Implementation

Without delving into much detail we will present the general aspects of the code used to implement our Project. The code was fully written in C++ using the CLion IDE. While the project may have been aimed towards a more scientifically natured language, such as MATLAB, we chose to implement our project in C++ since we have a much more intuitive understanding of graph manipulation in it.

The Graph was represented in code by its adjacency matrix. The Adjacency matrix has a size of Nodes by Nodes. The Adjacency matrix is populated using the following code:

adj\_matrix.at(i).at(j) = rand() % 10;  
 if (adj\_matrix.at(i).at(j) >= dens-1) {  
 adj\_matrix.at(i).at(j) = 1;  
 }else{  
 adj\_matrix.at(i).at(j) = 0;

Iterating through the graph we assign a random value between 0 and 9 to each indexed position (each transition between components). Then we check if the random value assigned is greater than the value of our Density parameter. If it is then we assign a new value of 1. Else we assign a value of 0. This allows us to control the probability with which the value 1 appears. (E.g. if Density is equal to 6 then there is a 50% chance of the value of a certain position being 1). To calculate the probability corresponding to each integer Density we use the following formula: P = (11 – Density)/10.

In Graph Generations 2 and 3 we need to enforce the following rule:

* The Edges generated can only be generated from a smaller indexed node to a larger one (Nn -> Nm where n > m). E.g. N3 -> N5.

In code we do so using the following method:

for (int i = 0; i < nodes; i++) {  
 for (int j = i + 1; j < nodes; j++) { } }

By iterating through the Graph using the aforementioned method we make sure that the edge generated will lead from an n-Node to a greater one since the j index will always be greater than the i index.

The exhaustive search algorithm is achieved using the following steps.

1. We generate a bitset (a binary representation of an integer in C++) with an initial value of zero. Let’s call it bit.
2. For each individual bit belonging to the set we assign the following values:
   1. If bit == 1 we assign the value of R.
   2. If bit == 0 we assign the value of 1-R.
3. We multiply the polynomials assigned above together to produce the reliability of that specific combination.
4. We increase the integer value of bitset by one and iterate over all possible combinations.

Before we move on, we need to specify how we evaluate each polynomial. Not all polynomials produced above are useful to us since not all of them produce a path from the Input Node to the Output Node. We employ the use of Floyd-Warshall’s algorithm, which in turn is an algorithm for solving the issue of transitive closure. In broad strokes the algorithm checks if there is a path directly from all Nodes to all other nodes or if there is a path indirectly through other Nodes. If such a path exists, then the polynomial generated is of value to us and we add it to the Rall Polynomial.

When the full Rall polynomial is produced we use the following transformation to produce the system’s Mean Time to Failure:

                      mttf = mttf + (double) result.at(pol) \* ((double) 1 / pol);

                        avg\_mttf = (double)avg\_mttf + mttf \* ((double) 1 / lambda);

pol in the above functions stands for the result of the polynomial multiplication. All results are calculated with a lambda of 2.

# Discussing the Approximation Implementation

In order to approximate the Reliability of our component system we make the following assumptions. Our approximation considers only the components present in each path from the Input to the Output Node. By employing an algorithm based on BFS (Breadth First Search) of our Graph we calculate all paths from Input to Output. To calculate the Rall we employ the same process we used in the exhaustive search implementation and transform it to produce the system’s MTTF.

\* *In the charts below we show the results for each Graph Generation procedure over n = 100 iterations. The Third Generations was also generated for n=1000 iterations, the data for which can be observed on the first chart below.*

## Exhaustive Algorithm

What is noteworthy from the above data produced is that contrary to expectations the Highest Average\_MTTF is not produced from the Graph with the highest Density but between the ranges of 0,5 – 0,7. While someone would expect that more component connections would lead the system to failure slower that is not the case.

Furthermore, we observe that the Average\_MTTF is analogous to the number of Nodes in the Graph. The more Nodes the greater the value of Average\_MTTF. To make this observation we choose to exclude the results for Nodes:6,8 since their results seem unpredictable and counter intuitive. All methods of generating the Graph produce similar results to the ones described above, hence we will not delve into detail for each one.

## Approximation

The results for the approximation were not graphed since they were immensely irregular. We observed values for the Average\_MTTF greater than 60000 seconds which is not a very descriptive upper limit to the Exhaustive Algorithm. Our intentions when coding the Approximation were to produce a limit to describe the Exhaustive Algorithm’s results with much smaller complexity in the calculations. But we did not achieve a credible limit and we will not present any further details for the approximation.

The approximation under specific circumstances, we tend to believe that it would produce credible results, but we cannot discuss any further since, due to time restraints, were not able to produce enough simulations. The specific circumstances are as follows: If the Graph was produced with all components connected in series then the Approximation would produce credible results. Without absolute certainty we can say that the more components connected in series the more accuracy our Approximation would achieve in producing a limit for the Exhaustive Algorithm.