IDENTIFICATION OF OPTIMUM DRUG COMBINATION FOR CANCER USING BOOLEAN NETWORKS

Submitted by

ABHISEK KARMAKAR (13000114006)

ARGHA NANDAN (13000114015)

BISHAL SAHA (13000114023)

Submitted for the partial fulfillment for the degree of Bachelor of Technology in Computer Science and Engineering

Techno India
EM 4/1, Salt Lake, Sector – V, Kolkata – 700 091.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Prof. Tapan Chowdhury of the department of Computer Science and Engineering, whose role as project guide was invaluable for the project. We are extremely thankful for the keen interest he took in advising us, for the books and reference materials provided for the moral support extended to us.

Last but not the least we convey our gratitude to all the teachers for providing us the technical skill that will always remain as our asset and to all non-teaching staffs for the gracious hospitality they offered us.

Place: Techno India, Salt Lake

Date:

……………………………….

……………………………….

……………………………….

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Briefing

Cancer results in loss of thousands of lives, but the underlying cause for it is very simple and basic. Cancers are caused due to uncontrolled growth of cells in our body [1]. This growth happens due to the presence of faults and malfunctions in the genetic and signaling networks. This uncontrolled cell proliferation is usually accompanied by gene mutations and alterations in the cell.

Given a signaling pathway, a Boolean Network is modeled and the faulty nodes are identified. The input vector, which uniquely generates an output vector, is obtained. Lastly, the optimum combination of drugs which can nullify most of the fault in the network is identified. The result would be the simulation of the network traversal and the combination of the drugs [1].

This will help us understand the workings of proteins in a simple manner. Fault location will convey that every gate in the Boolean Network is a potential fault in the system. Assumption of more than one fault at a given time will produce a more realistic solution to cancerous conditions, while figuring out the drug combination for the faulty network.

1.2. Problem Domain

Data Mining from Boolean Networks

1.3. Related Studies

1.3.1. Signaling pathways

A signaling pathway consists of a group of molecules, usually proteins, that work together to accomplish a single biological function, such as cell division and death. Signals are transmitted in the form of biochemical reactions, and the final result at the end of the pathway decides the function to be carried out.

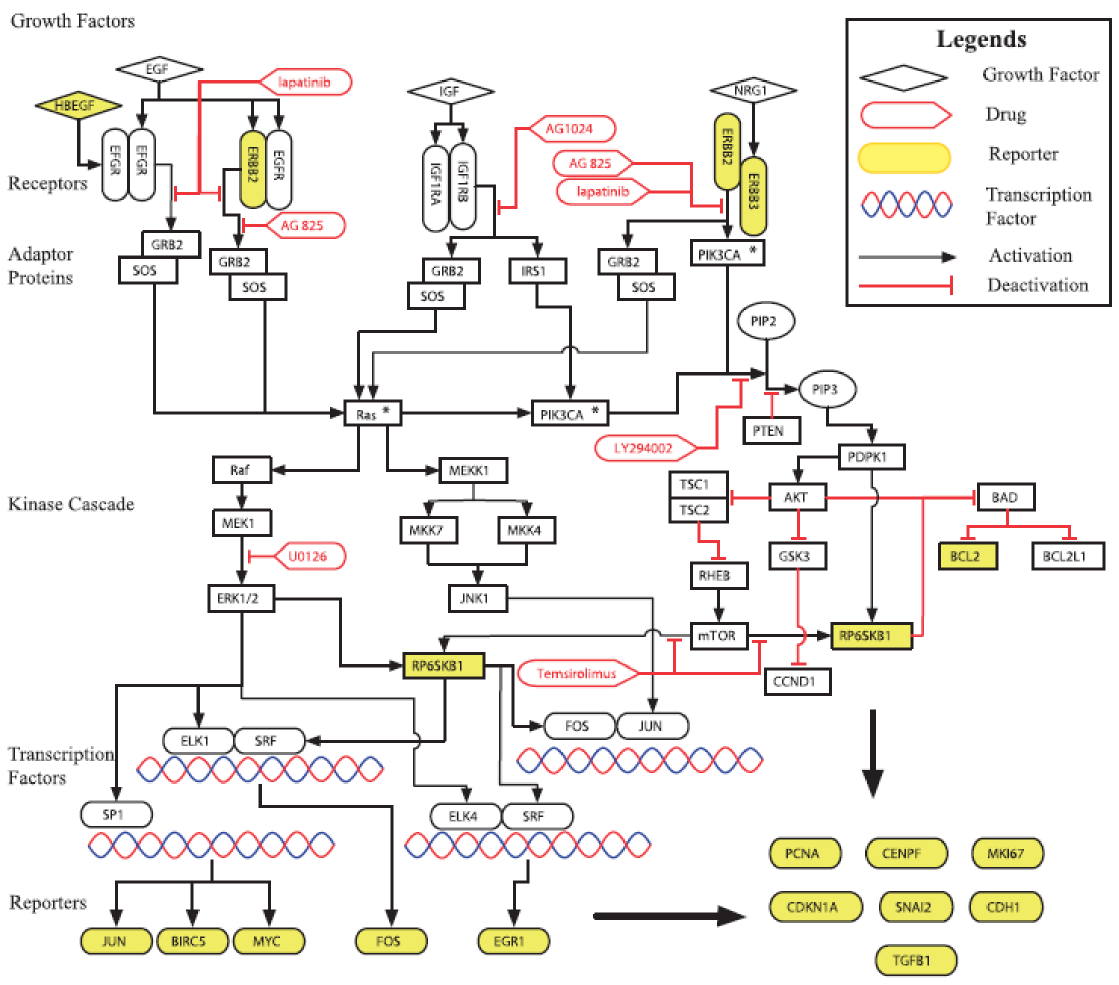


Figure 1.1: Growth factor signaling pathway

Figure 1.1 shows a growth factor "GF" signaling pathway, which governs cellular division in eukaryotic cells, i.e. cells having nucleus. Presence of malfunctions in the GF pathway results in outputs which trigger cell division or mitosis or proliferation without needing the necessary inputs, which are generally growth factors and metabolites.

1.3.2. Boolean Networks

A BN [1], $B=(V,F)$, on n genes/proteins is defined by a set of nodes (genes/proteins) $V=\{x_1,\ldots,x_n\}$, $xi\in\{0,1\}$, $i=1,\ldots,n$, and a list $F=(f_1,\ldots,f_n)$, of Boolean functions, $f_i:\{0,1\}n+m\rightarrow\{0,1\}$, $i=1,\ldots,n,m\geq 0$ is the number of external inputs e.g. GFs, stresses, metabolites, etc. Each node x_i represents the state/expression of the gene i, where $x_i=0$ means that gene/protein i is OFF (unexpressed or in active according to their biological significance) and $x_i=1$ means that gene/protein i is ON (expressed or active). The function f_i is called the *predictor function* for gene/protein i. Updating the states of all genes/proteins in B is done synchronously at every time step according to their predictor functions. If the predictor functions are known, the dynamics of the BN will solely depend on the set of input variables. The dynamic behavior of the BN is quite different in the presence of different external inputs. For *m* different external inputs, we can model the dynamical system as 2^m different closed BNs, each one of which we may define as a ‘context’. The switching between contexts here occurs in response to changes in the activity status of external input variables and is, therefore, deterministic.

1.3.3. Faults in Boolean Networks

A ‘fault’ [1] is defined by any structural error of the physical system, such that the dynamics become aberrant. For example, the accumulation of mutations in the genomic DNA may cause the signaling pathways to behave erratically leading to proliferation. On the other hand, sometimes the fault may not be in the genetic code of a particular protein, but rather it is in the protein synthesis factory ribosome, or in some control mechanism of alternative splicing.

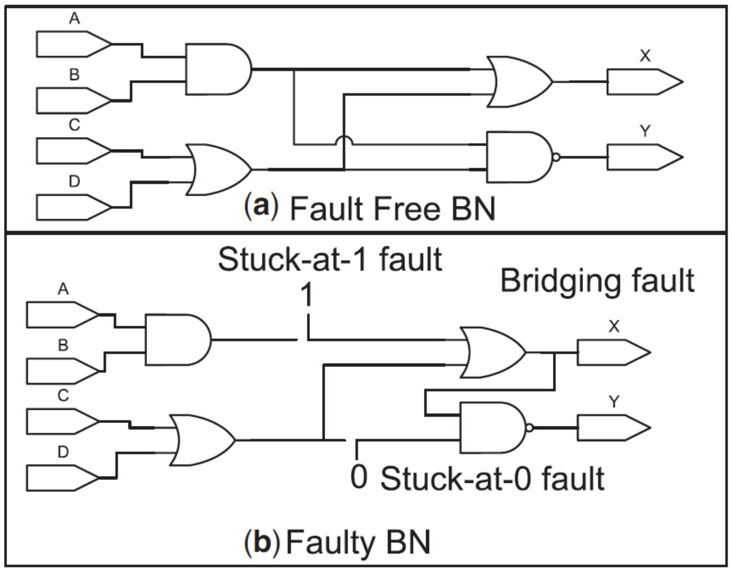


Figure 1.2: Faults in Boolean network

In a BN, faults are broadly classified into two types:

- Stuck-At Fault: A stuck-at fault means that a point in the network circuitry is stuck to a particular value. As a result, the incoming information is no longer communicated beyond the faulty point; instead, only the stuck-at value is passed on to the outgoing port.

- Bridging Fault: A bridging fault refers to the disruption of old interconnections and incorporation of new interconnections in the network. Bridging faults also make biological sense. The molecular signal transduction relies on the sequences and 3D conformations of the molecules involved. So, any variation in the sequence and 3D conformation of a molecule (mainly protein) will alter its functionality. As a result, many pathways involving that molecule will become inactive while the altered molecule may open up new ones. Without any loss of generality, this kind of aberrant behavior could be modeled as a bridging fault in the BN.

1.3.4. Drug therapy

Drugs are chemical substances which interfere with the proteins which they are designed to attack. If a drug is applied to any given protein P, P will be inhibited, irrespective of the Boolean function corresponding to that protein in the Boolean network.

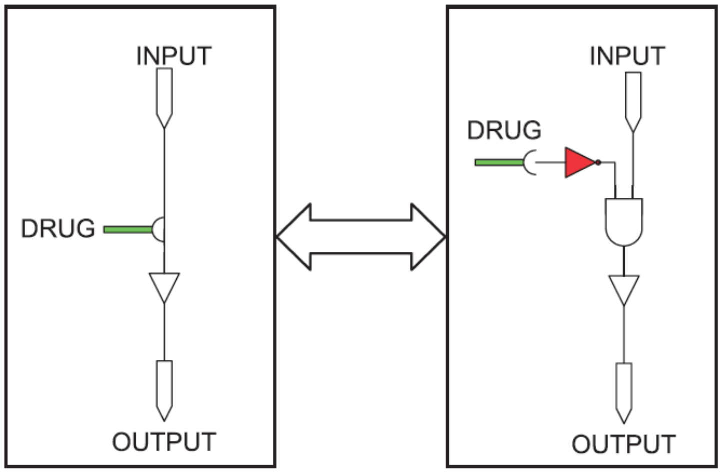


Figure 1.3: Drug interference in signaling pathway and Boolean network

The drugs in use in this project are as follows:

TABLE 1
DRUGS AND THE CORRESPONDING PROTEINS INHIBITED

Drug	Proteins inhibited in GF signaling pathway
LAPATINIB	EGFR/ERBB2, EFGR, ERBB2/3
AG825	EGFR/ERBB2, ERBB2/3
AG1024	IGFR1A/B
U0126	MEK1
LY294002	PIK3CA
TEMSIROLIMUS	mTOR

1.3.5. CUDA

CUDA[8] (Compute Unified Device Architecture) is a parallel computing platform developed by NVIDIA and programming model that makes using GPU (Graphics Processing Unit) for general purpose computing simple and elegant.

NVIDIA provides a system software that allows our programs to communicate with the CUDA-enabled hardware.
Using this driver we transfer our data to be processed to the GPU and process data in parallel and transfer the output back to the CPU for further usage.

Using this architecture we can efficiently process all the parallel jobs in very less time instead of depending upon CPU for batch processing of similar data.

The kernel code (Executable in GPU) is written using C Language.

1.3.6. PyCUDA

PyCUDA [9]is a Python Programming environment for CUDA. It lets us access Nvidia's CUDA Parallel computation API from Python.
It maps all CUDA into Python and enables run-time code generation for flexible, fast, automatically tuned codes. It is robust and convenient since there is a minimum need to handle all the hardware resources.

1.4.Glossary

<u>GRN</u> [3]: A network that has been inferred fromgene expression data a "gene regulatory network," is briefly denotedas GRN.

<u>Boolean Network</u>(BN) [1]: A Boolean Network B (v, f), on n genes / proteins is defined by a set of nodes $V=(x_{1......}x_n)$, $x_i \in \{0, 1\}$ where I = 1… n and a list $F = (f_1.....f_n)$ of boolean functions, f_i: $\{0, 1\}^{n+m} \rightarrow \{0, 1\}$, i= 1….n, m>=0 is the number of external inputs e.g. GFs, stresses, metabolic, etc. Each node x_i represents the state / expression of the gene I, where $x_i = 0$ means the gene / protein i is OFF (inactive) and if $x_i = 1$, that means gene / protein is ON (active).

<u>Faults</u>: A 'fault' is defined by any structural error of the physical system, such that the dynamics become aberrant. For example, the accumulation of point mutations in the genomic DNA may cause the signaling pathways to behave erratically leading to proliferation. These faults could cause structural changes in the regulatory network, thereby changes its dynamic and steady-state behavior.

<u>Data Frame</u>: Data Frame is a 2-D data structure which has columns of different types. It is the most used structure in the pandas package.

2. PROBLEM DEFINITION

2.1. Scope

The Boolean Network derived from the GRN would not only enable us to simplify the network to understand the logic but also to introduce faults and analyze how the drug will act at particular points of the BN. This will help us understand the behavior of the network when a drug will be introduced to it. The mutations caused by cancer are irreversible but a perfect drug combination can help us nullify the faults in the network which will contribute to save lives in the near future.

2.2. Exclusions

The part for analyzing which particular genes are affecting the growth factors to send the messages to the cells to proliferate is excluded.

2.3. Assumptions

The quantized or Boolean nature of proteins is assumed throughout, although their actual levels are not quantized, but continuous.

Only Stuck-At faults [1] are considered assuming there are no bridging faults.

The drug which produces the output vector closest to the fault-less network, is optimum for the therapy.

The binary numbers used in this document are written as $b_1b_2b_3b_4$ instead of [b1, b2, b3, b4] for ease of understanding. They are lists, not numbers.

3. PROJECT PLANNING

3.1.Software Life Cycle Model

In our Project we have followed the Iterative Waterfall Model [7]:

The problems with the waterfall model created a demand for a new method of developing systems which could provide faster results, require less up-front information and offer greater flexibility. Iterative model not only divides the project into small parts but also allows the development team to demonstrate results earlier on the process and obtain valuable feedback from system users. Each iteration is actually a mini-waterfall process with the feedback from one phase providing vital information for the design of the next phase.

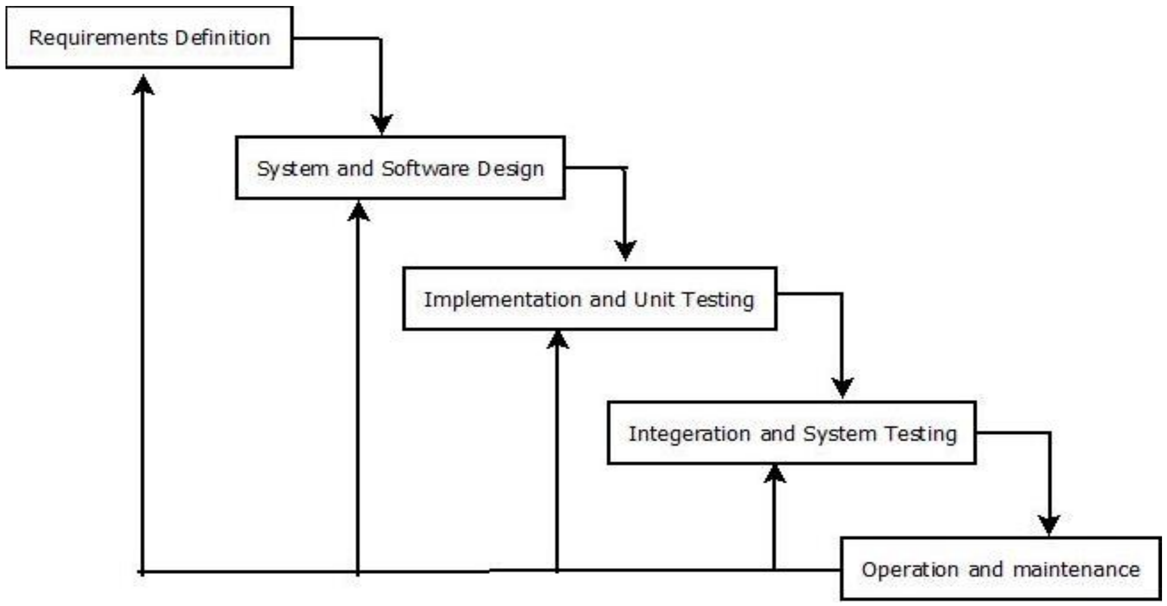
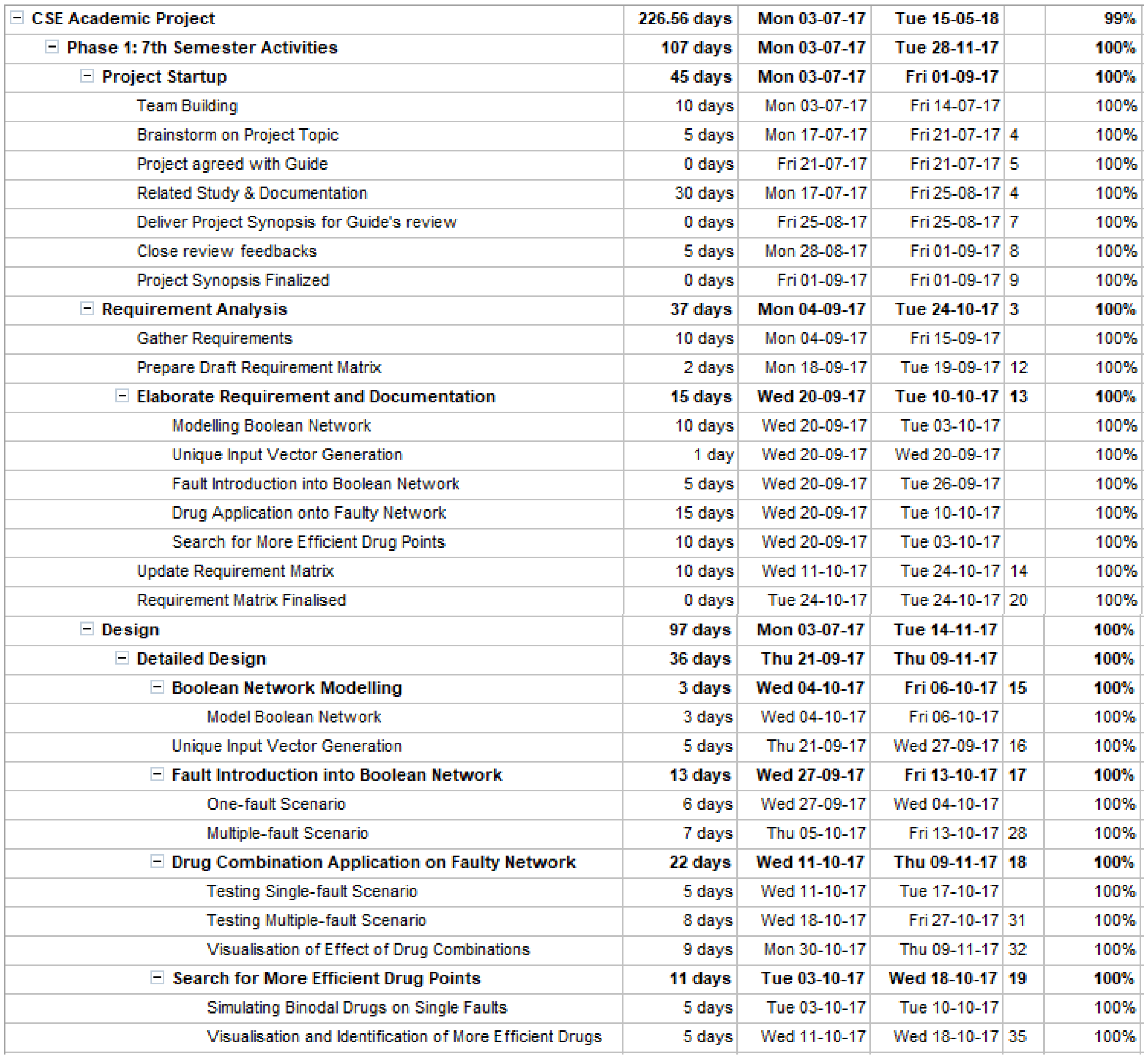

Requirements Definition
System and Software Design
Implementation and Unit Testing
Integeration and System Testing
Operation and maintenance

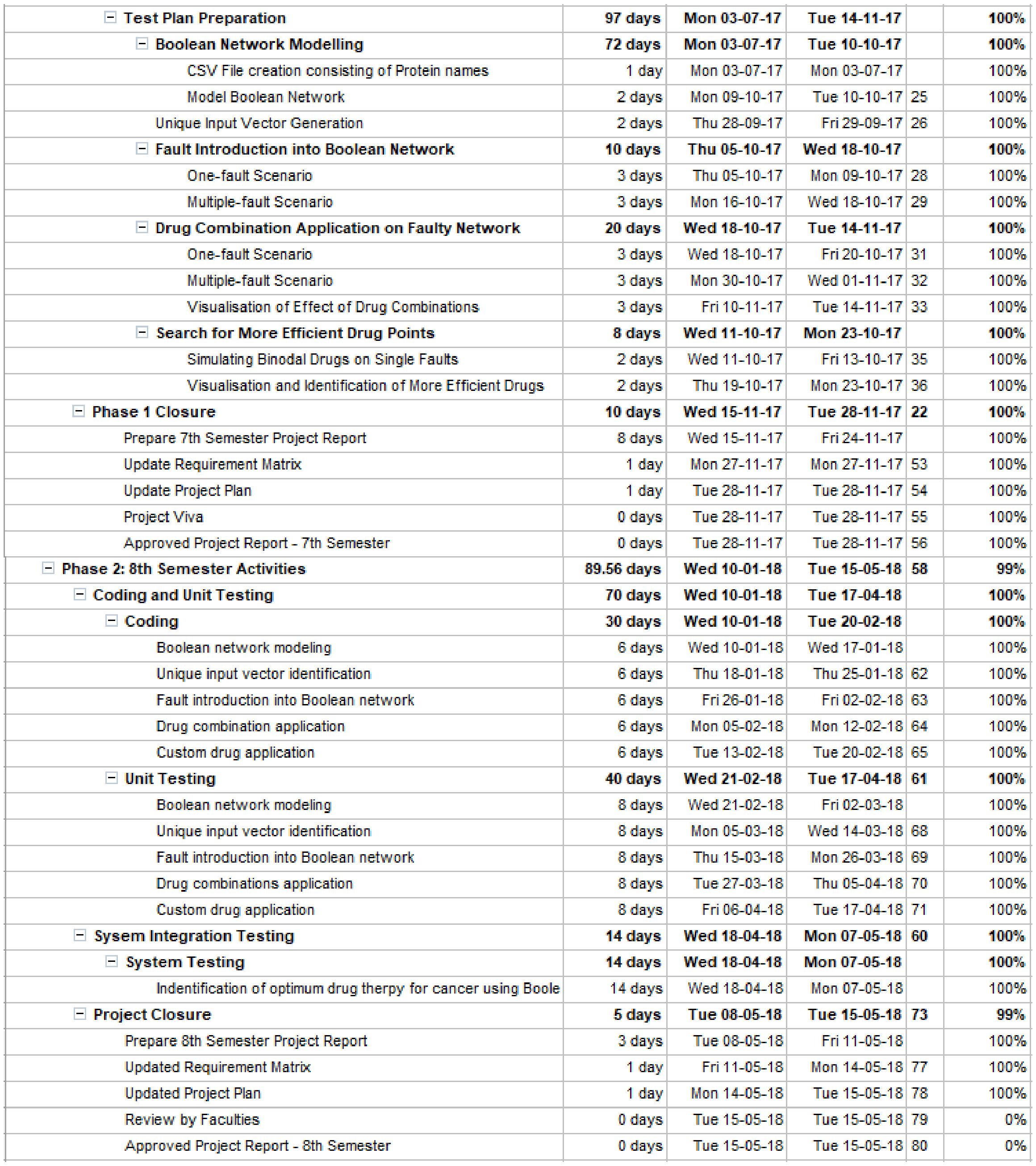
Figure 3.1: Iterative Waterfall Model

3.2.Scheduling

3.2.1. Project Schedule



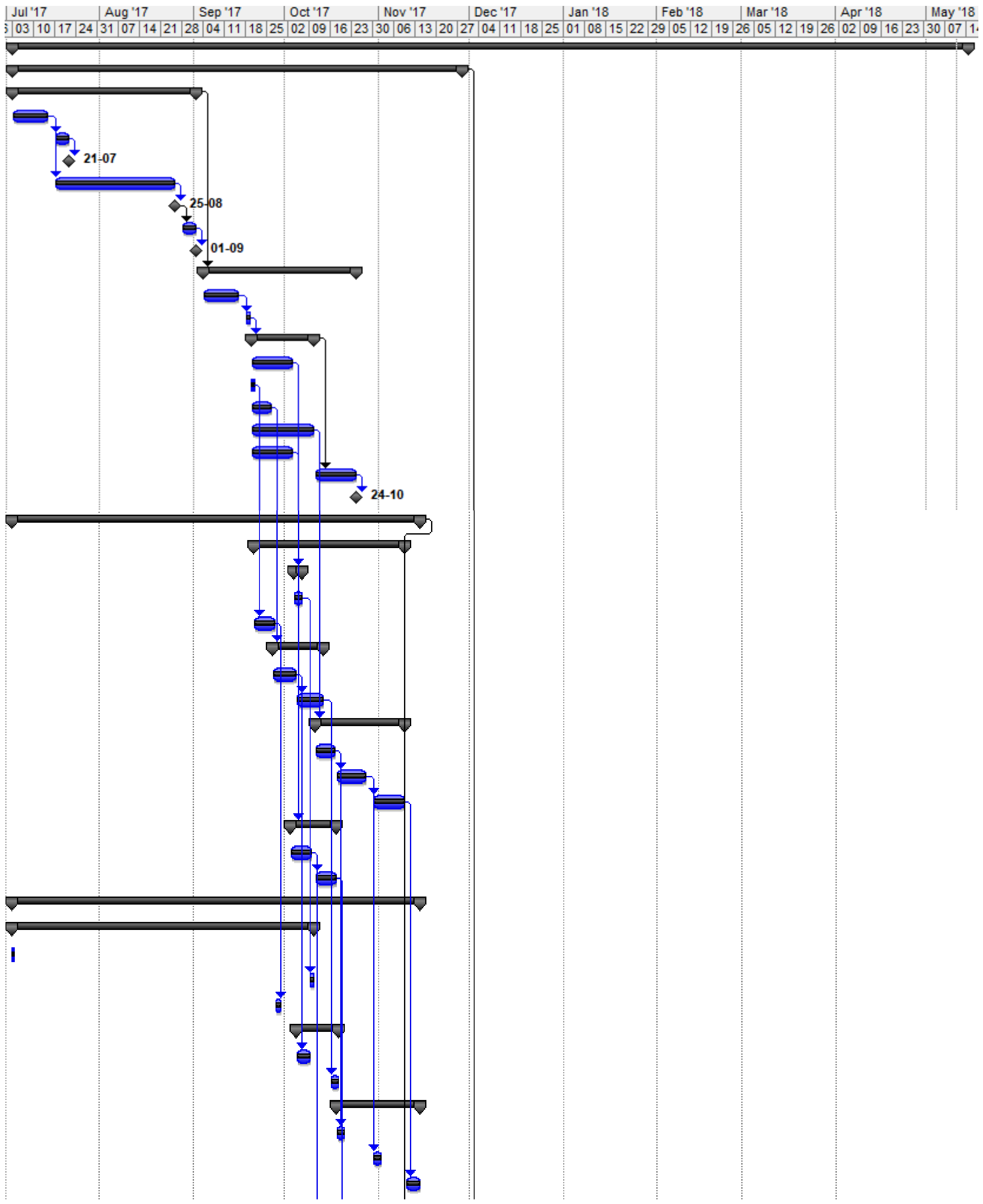
CSE Academic Project	**226.56 days**	**Mon 03-07-17**	**Tue 15-05-18**		**99%**
Phase 1: 7th Semester Activities	**107 days**	**Mon 03-07-17**	**Tue 28-11-17**		**100%**
Project Startup	**45 days**	**Mon 03-07-17**	**Fri 01-09-17**		**100%**
Team Building	10 days	Mon 03-07-17	Fri 14-07-17		100%
Brainstorm on Project Topic	5 days	Mon 17-07-17	Fri 21-07-17	4	100%
Project agreed with Guide	0 days	Fri 21-07-17	Fri 21-07-17	5	100%
Related Study & Documentation	30 days	Mon 17-07-17	Fri 25-08-17	4	100%
Deliver Project Synopsis for Guide's review	0 days	Fri 25-08-17	Fri 25-08-17	7	100%
Close review feedbacks	5 days	Mon 28-08-17	Fri 01-09-17	8	100%
Project Synopsis Finalized	0 days	Fri 01-09-17	Fri 01-09-17	9	100%
Requirement Analysis	**37 days**	**Mon 04-09-17**	**Tue 24-10-17**	**3**	**100%**
Gather Requirements	10 days	Mon 04-09-17	Fri 15-09-17		100%
Prepare Draft Requirement Matrix	2 days	Mon 18-09-17	Tue 19-09-17	12	100%
Elaborate Requirement and Documentation	**15 days**	**Wed 20-09-17**	**Tue 10-10-17**	**13**	**100%**
Modelling Boolean Network	10 days	Wed 20-09-17	Tue 03-10-17		100%
Unique Input Vector Generation	1 day	Wed 20-09-17	Wed 20-09-17		100%
Fault Introduction into Boolean Network	5 days	Wed 20-09-17	Tue 26-09-17		100%
Drug Application onto Faulty Network	15 days	Wed 20-09-17	Tue 10-10-17		100%
Search for More Efficient Drug Points	10 days	Wed 20-09-17	Tue 03-10-17		100%
Update Requirement Matrix	10 days	Wed 11-10-17	Tue 24-10-17	14	100%
Requirement Matrix Finalised	0 days	Tue 24-10-17	Tue 24-10-17	20	100%
Design	**97 days**	**Mon 03-07-17**	**Tue 14-11-17**		**100%**
Detailed Design	**36 days**	**Thu 21-09-17**	**Thu 09-11-17**		**100%**
Boolean Network Modelling	**3 days**	**Wed 04-10-17**	**Fri 06-10-17**	**15**	**100%**
Model Boolean Network	3 days	Wed 04-10-17	Fri 06-10-17		100%
Unique Input Vector Generation	5 days	Thu 21-09-17	Wed 27-09-17	16	100%
Fault Introduction into Boolean Network	**13 days**	**Wed 27-09-17**	**Fri 13-10-17**	**17**	**100%**
One-fault Scenario	6 days	Wed 27-09-17	Wed 04-10-17		100%
Multiple-fault Scenario	7 days	Thu 05-10-17	Fri 13-10-17	28	100%
Drug Combination Application on Faulty Network	**22 days**	**Wed 11-10-17**	**Thu 09-11-17**	**18**	**100%**
Testing Single-fault Scenario	5 days	Wed 11-10-17	Tue 17-10-17		100%
Testing Multiple-fault Scenario	8 days	Wed 18-10-17	Fri 27-10-17	31	100%
Visualisation of Effect of Drug Combinations	9 days	Mon 30-10-17	Thu 09-11-17	32	100%
Search for More Efficient Drug Points	**11 days**	**Tue 03-10-17**	**Wed 18-10-17**	**19**	**100%**
Simulating Binodal Drugs on Single Faults	5 days	Tue 03-10-17	Tue 10-10-17		100%
Visualisation and Identification of More Efficient Drugs	5 days	Wed 11-10-17	Wed 18-10-17	35	100%



Test Plan Preparation	**97 days**	**Mon 03-07-17**	**Tue 14-11-17**		**100%**
Boolean Network Modelling	**72 days**	**Mon 03-07-17**	**Tue 10-10-17**		**100%**
CSV File creation consisting of Protein names	1 day	Mon 03-07-17	Mon 03-07-17		100%
Model Boolean Network	2 days	Mon 09-10-17	Tue 10-10-17	25	100%
Unique Input Vector Generation	2 days	Thu 28-09-17	Fri 29-09-17	26	100%
Fault Introduction into Boolean Network	**10 days**	**Thu 05-10-17**	**Wed 18-10-17**		**100%**
One-fault Scenario	3 days	Thu 05-10-17	Mon 09-10-17	28	100%
Multiple-fault Scenario	3 days	Mon 16-10-17	Wed 18-10-17	29	100%
Drug Combination Application on Faulty Network	**20 days**	**Wed 18-10-17**	**Tue 14-11-17**		**100%**
One-fault Scenario	3 days	Wed 18-10-17	Fri 20-10-17	31	100%
Multiple-fault Scenario	3 days	Mon 30-10-17	Wed 01-11-17	32	100%
Visualisation of Effect of Drug Combinations	3 days	Fri 10-11-17	Tue 14-11-17	33	100%
Search for More Efficient Drug Points	**8 days**	**Wed 11-10-17**	**Mon 23-10-17**		**100%**
Simulating Binodal Drugs on Single Faults	2 days	Wed 11-10-17	Fri 13-10-17	35	100%
Visualisation and Identification of More Efficient Drugs	2 days	Thu 19-10-17	Mon 23-10-17	36	100%
Phase 1 Closure	**10 days**	**Wed 15-11-17**	**Tue 28-11-17**	**22**	**100%**
Prepare 7th Semester Project Report	8 days	Wed 15-11-17	Fri 24-11-17		100%
Update Requirement Matrix	1 day	Mon 27-11-17	Mon 27-11-17	53	100%
Update Project Plan	1 day	Tue 28-11-17	Tue 28-11-17	54	100%
Project Viva	0 days	Tue 28-11-17	Tue 28-11-17	55	100%
Approved Project Report - 7th Semester	0 days	Tue 28-11-17	Tue 28-11-17	56	100%
Phase 2: 8th Semester Activities	**89.56 days**	**Wed 10-01-18**	**Tue 15-05-18**	**58**	**99%**
Coding and Unit Testing	**70 days**	**Wed 10-01-18**	**Tue 17-04-18**		**100%**
Coding	**30 days**	**Wed 10-01-18**	**Tue 20-02-18**		**100%**
Boolean network modeling	6 days	Wed 10-01-18	Wed 17-01-18		100%
Unique input vector identification	6 days	Thu 18-01-18	Thu 25-01-18	62	100%
Fault introduction into Boolean network	6 days	Fri 26-01-18	Fri 02-02-18	63	100%
Drug combination application	6 days	Mon 05-02-18	Mon 12-02-18	64	100%
Custom drug application	6 days	Tue 13-02-18	Tue 20-02-18	65	100%
Unit Testing	**40 days**	**Wed 21-02-18**	**Tue 17-04-18**	**61**	**100%**
Boolean network modeling	8 days	Wed 21-02-18	Fri 02-03-18		100%
Unique input vector identification	8 days	Mon 05-03-18	Wed 14-03-18	68	100%
Fault introduction into Boolean network	8 days	Thu 15-03-18	Mon 26-03-18	69	100%
Drug combinations application	8 days	Tue 27-03-18	Thu 05-04-18	70	100%
Custom drug application	8 days	Fri 06-04-18	Tue 17-04-18	71	100%
Sysem Integration Testing	**14 days**	**Wed 18-04-18**	**Mon 07-05-18**	**60**	**100%**
System Testing	**14 days**	**Wed 18-04-18**	**Mon 07-05-18**		**100%**
Indentification of optimum drug therpy for cancer using Boole	14 days	Wed 18-04-18	Mon 07-05-18		100%
Project Closure	**5 days**	**Tue 08-05-18**	**Tue 15-05-18**	**73**	**99%**
Prepare 8th Semester Project Report	3 days	Tue 08-05-18	Fri 11-05-18		100%
Updated Requirement Matrix	1 day	Fri 11-05-18	Mon 14-05-18	77	100%
Updated Project Plan	1 day	Mon 14-05-18	Tue 15-05-18	78	100%
Review by Faculties	0 days	Tue 15-05-18	Tue 15-05-18	79	0%
Approved Project Report - 8th Semester	0 days	Tue 15-05-18	Tue 15-05-18	80	0%

Figure 3.2: Project Schedule

3.2.2. Gantt Chart



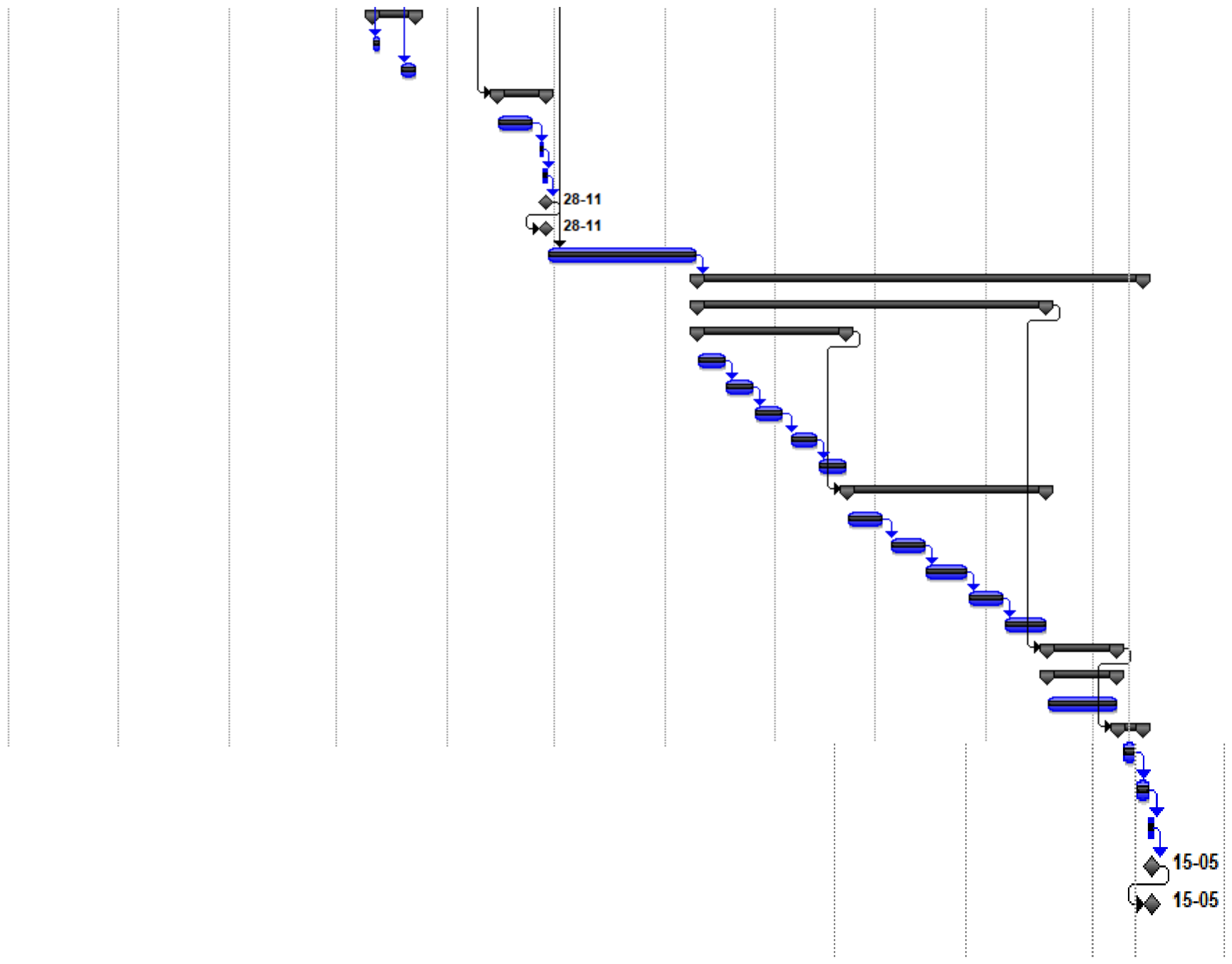


Figure 4.3: Gantt chart

3.3.Cost Analysis

Lines of Code= 7201.76 = 7.202 KLOC

Effort= $2.4*7.202^{1.05}$= 19.07752

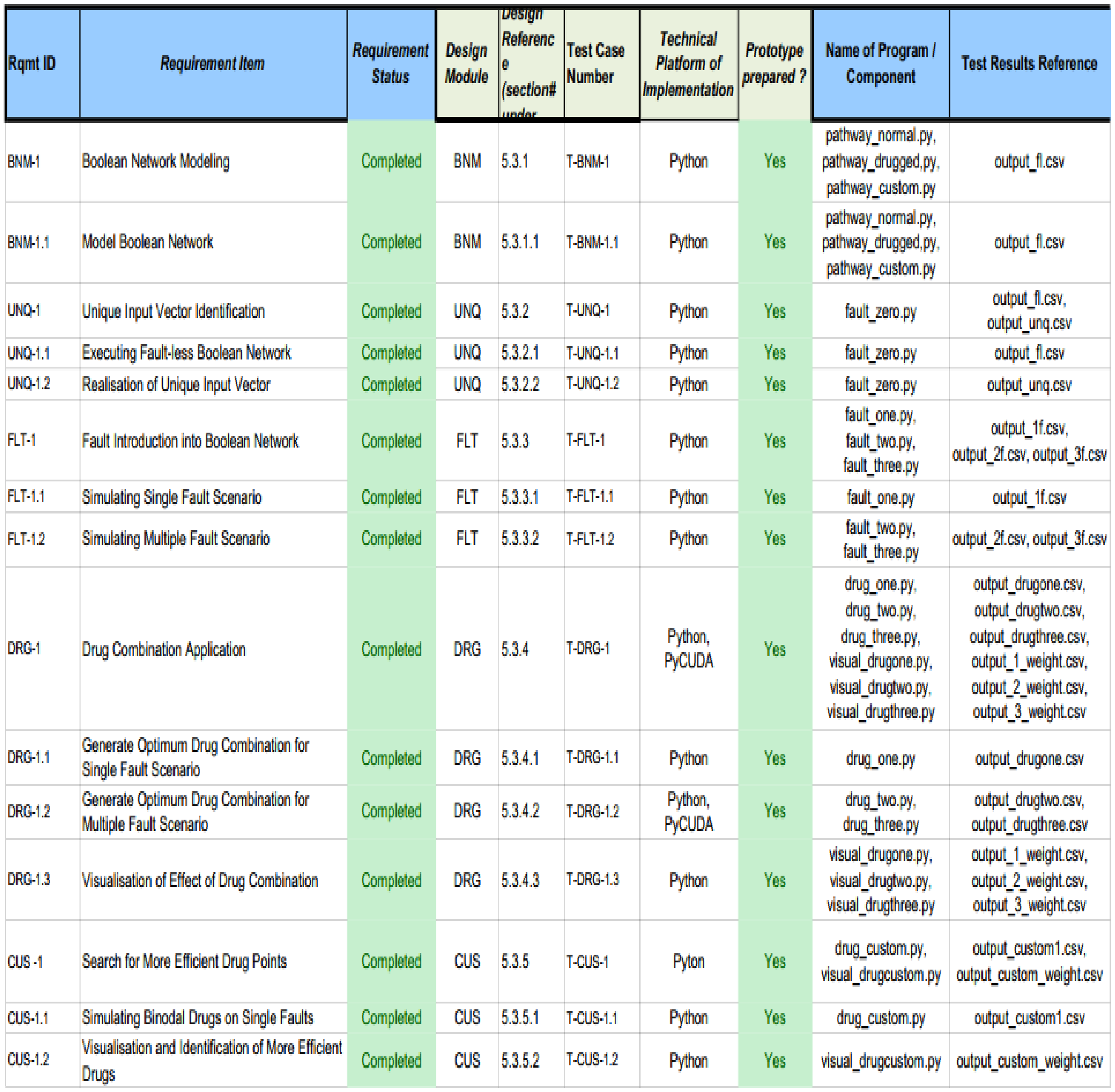
Duration= $2.5*\text{Effort}^{0.38}$ = 7.67 months

Cost= ₹183971.1

Men Required= Effort / Duration = 2.488 ~ 3

4. REQUIREMENT ANALYSIS

4.1. Requirement Matrix



Rqmt ID	Requirement Item	Requirement Status	Design Module	Design Referenc e (section# under	Test Case Number	Technical Platform of Implementation	Prototype prepared ?	Name of Program / Component	Test Results Reference
BNM-1	Boolean Network Modeling	Completed	BNM	5.3.1	T-BNM-1	Python	Yes	pathway_normal.py, pathway_drugged.py, pathway_custom.py	output_fl.csv
BNM-1.1	Model Boolean Network	Completed	BNM	5.3.1.1	T-BNM-1.1	Python	Yes	pathway_normal.py, pathway_drugged.py, pathway_custom.py	output_fl.csv
UNQ-1	Unique Input Vector Identification	Completed	UNQ	5.3.2	T-UNQ-1	Python	Yes	fault_zero.py	output_fl.csv, output_unq.csv
UNQ-1.1	Executing Fault-less Boolean Network	Completed	UNQ	5.3.2.1	T-UNQ-1.1	Python	Yes	fault_zero.py	output_fl.csv
UNQ-1.2	Realisation of Unique Input Vector	Completed	UNQ	5.3.2.2	T-UNQ-1.2	Python	Yes	fault_zero.py	output_unq.csv
FLT-1	Fault Introduction into Boolean Network	Completed	FLT	5.3.3	T-FLT-1	Python	Yes	fault_one.py, fault_two.py, fault_three.py	output_1f.csv, output_2f.csv, output_3f.csv
FLT-1.1	Simulating Single Fault Scenario	Completed	FLT	5.3.3.1	T-FLT-1.1	Python	Yes	fault_one.py	output_1f.csv
FLT-1.2	Simulating Multiple Fault Scenario	Completed	FLT	5.3.3.2	T-FLT-1.2	Python	Yes	fault_two.py, fault_three.py	output_2f.csv, output_3f.csv
DRG-1	Drug Combination Application	Completed	DRG	5.3.4	T-DRG-1	Python, PyCUDA	Yes	drug_one.py, drug_two.py, drug_three.py, visual_drugone.py, visual_drugtwo.py, visual_drugthree.py	output_drugone.csv, output_drugtwo.csv, output_drugthree.csv, output_1_weight.csv, output_2_weight.csv, output_3_weight.csv
DRG-1.1	Generate Optimum Drug Combination for Single Fault Scenario	Completed	DRG	5.3.4.1	T-DRG-1.1	Python	Yes	drug_one.py	output_drugone.csv
DRG-1.2	Generate Optimum Drug Combination for Multiple Fault Scenario	Completed	DRG	5.3.4.2	T-DRG-1.2	Python, PyCUDA	Yes	drug_two.py, drug_three.py	output_drugtwo.csv, output_drugthree.csv
DRG-1.3	Visualisation of Effect of Drug Combination	Completed	DRG	5.3.4.3	T-DRG-1.3	Python	Yes	visual_drugone.py, visual_drugtwo.py, visual_drugthree.py	output_1_weight.csv, output_2_weight.csv, output_3_weight.csv
CUS -1	Search for More Efficient Drug Points	Completed	CUS	5.3.5	T-CUS-1	Pyton	Yes	drug_custom.py, visual_drugcustom.py	output_custom1.csv, output_custom_weight.csv
CUS-1.1	Simulating Binodal Drugs on Single Faults	Completed	CUS	5.3.5.1	T-CUS-1.1	Python	Yes	drug_custom.py	output_custom1.csv
CUS-1.2	Visualisation and Identification of More Efficient Drugs	Completed	CUS	5.3.5.2	T-CUS-1.2	Python	Yes	visual_drugcustom.py	output_custom_weight.csv

Figure 4.1: Requirement Matrix

4.2.Requirement Elaboration

4.2.1. Boolean Network Modeling

From a given GRN, a Boolean Network needs to be modeled. We generate a CSV file containing all the proteins belonging to a Growth Factor Signaling pathway and using that data we establish their connections in the form of the Boolean Network. Some of the proteins are inputs known as growth factors; some are the nodes inside the pathway, while the rest are outputs as transcription factors and residual proteins which trigger proliferation if activated.

4.2.2. Unique input vector identification

A unique vector containing a combination of 0's and 1's is to be obtained from the modeled Boolean network. That input vector would further help to analyze the network after the introduction of the fault.

4.2.3. Fault introduction into Boolean network

Since we're only considering stuck-at faults, we'll program the faults in the network such that instead of computing the values of nodes from data taken from adjacent nodes, some particular nodes will bear only one value, 0 or 1, no matter what it gets as input. Those nodes will transfer those same stuck-at faults to the next adjacent nodes as input.
We'll consider two scenarios:
a) Only one fault in the network at any given time
b) More than one faults in the network at any given time.

4.2.4. Application of drug combinations

Having found the unique input, and classified the number of faults in the network, we'll apply a combination of drugs which attempts to nullify the effect of faults on the output vector.
We'll apply each of the combinations, and identify the one, which generates an output vector closest to the output generated in a fault-less Boolean Network, as the optimum therapy for that cancerous condition.

4.2.5. Search for more efficient drug Points

Having worked with known drugs in the previous module, we will look for new drug inhibition points, which may produce better results than what achieved with known drugs. Upon generation, the nodes where the new drugs are attacking may serve as a guideline to manufacture drugs which inhibit those points.

5. DESIGN

5.1.Technical Environment

We are using Ubuntu 16.04 as the base Operating System, Intel Core i3 processor, 4 GB RAM, 500 GB HDD, NVIDIA 820 M GPU.
We need Python v3.6 for interpreting our programs and for the programming we're using Spyder 3.6 IDE, which is a part of the Anaconda software package. Spyder comes pre-built with all the necessary packages and libraries, like pandaS, numpy, matplotlib, etc., which we need to construct our DataFrames and visualize our results in the end.
We need CUDA enabled GPU for parallel computation and CUDA drivers provided by NVIDIA. We need PyCUDA which will help to implement the CUDA kernel code from Python.

5.1.1. Software Specifications

- Python 3.6 Interpreter
- Spyder 3.2 for Python Programming

5.1.2. Hardware Specifications

TABLE 2
HARDWARE SPECIFICATIONS

Operating System	Ubuntu 16.04 LTS
Memory	4 GB
HDD	500 GB
Processor	Intel Core i3

5.2.Hierarchy of Modules

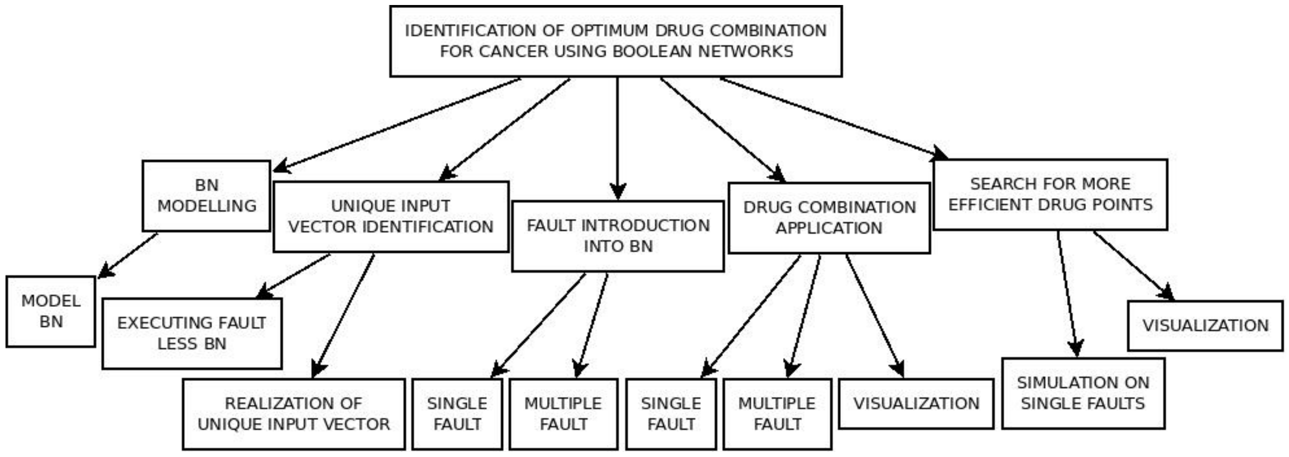
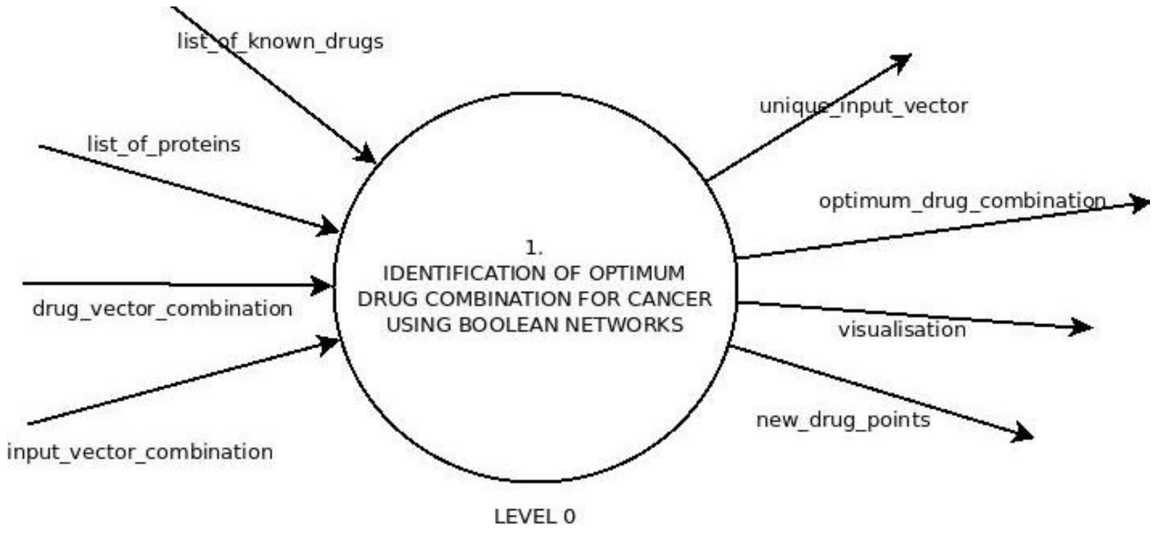

IDENTIFICATION OF OPTIMUM DRUG COMBINATION FOR CANCER USING BOOLEAN NETWORKS
BN MODELLING
UNIQUE INPUT VECTOR IDENTIFICATION
FAULT INTRODUCTION INTO BN
DRUG COMBINATION APPLICATION
SEARCH FOR MORE EFFICIENT DRUG POINTS
MODEL BN
EXECUTING FAULT LESS BN
REALIZATION OF UNIQUE INPUT VECTOR
SINGLE FAULT
MULTIPLE FAULT
SINGLE FAULT
MULTIPLE FAULT
VISUALIZATION
SIMULATION ON SINGLE FAULTS
VISUALIZATION

Figure 5.1: Hierarchy of Modules

5.3.Detailed Design



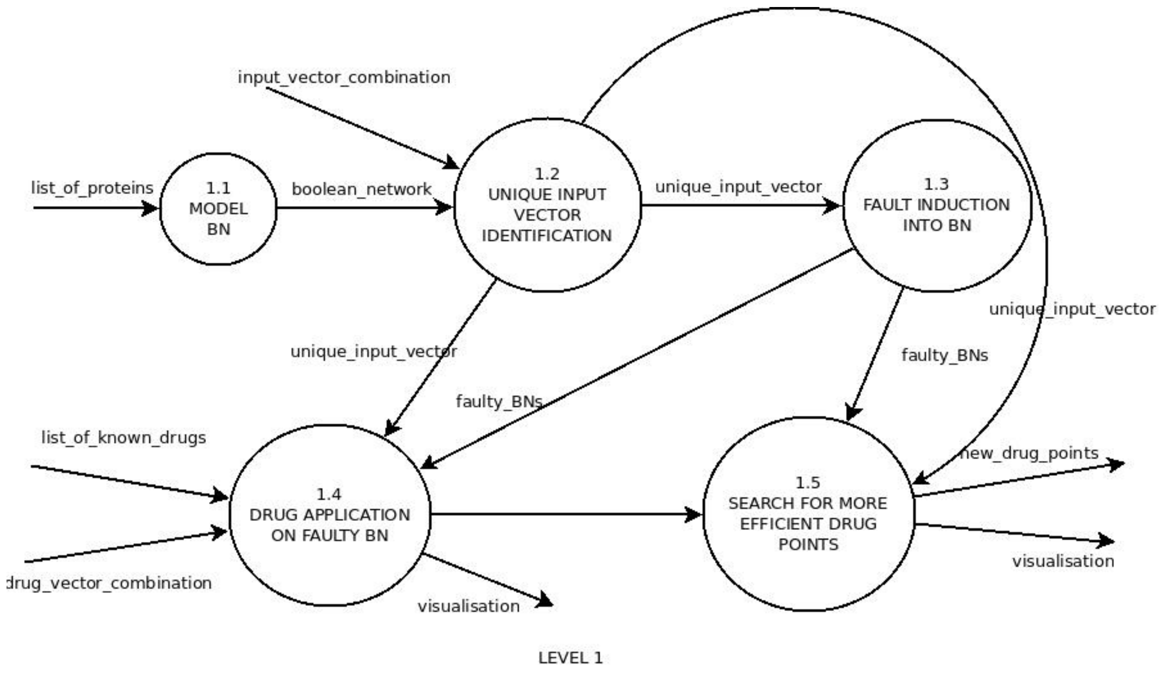


Figure 5.2: Data Flow Diagram

5.3.1. Boolean Network Modeling

5.3.1.1.Model Boolean Network

From the given signaling pathway [1], we'll construct the pathway logic, by replacing the proteins with nodes which only bear Boolean values. There are four types of pathway logic [2], which use logic gates to interpret the given scenario. The scenarios are as follows:

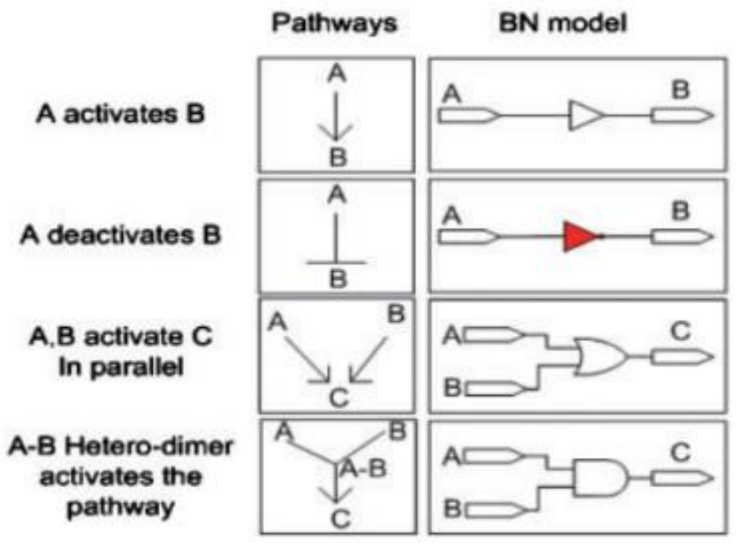


Figure 5.3: Different types of pathway scenarios

In scenario 1, the value of one node is transferred to the next as it is. This corresponds to a situation where a protein is activated only if another is also active.
In scenario 2, the value of one node is transferred to the next through a NOT gate. This corresponds to a situation where an active protein deactivates the next.
In scenario 3, the values of two different nodes are passed to the next through an OR gate. This corresponds to a situation where a protein is activated if at least one of the previous proteins is active.
In scenario 4, the values of two different nodes are passed on through an AND gate. This corresponds to a situation when a protein is activated only if the previous two proteins are activated.
We use this strategy to program the entire pathway logic into a Boolean Network, with 5 input proteins, 23 proteins in the pathway and 7 output proteins.

All three modeling would be programmed; one with faults, one with faults and known drugs, and one with faults and custom drugs.

5.3.2. Unique Input Vector Identification

5.3.2.1. Executing fault-less Boolean Network

As of now, the Boolean Network doesn't have any induced faults. Keeping the 5 protein input vector in mind, we know that a Boolean number consisting of 5 bits, generates 32 combination.
So, we execute the Boolean Network using each combination as the input vector, and observe the output vectors.

5.3.2.2. Identifying Unique Input Vector

We get 32 output vectors corresponding to the 32 input vectors. If the Boolean Network we've designed is correct, i.e. in accordance with the signaling pathway which it is modeled from, we expect to get at least one input vector, which corresponds to an output where all the bits are 0's.
This goes along with the biology of signaling pathway, as there should be at least one combination which stops the cells from dividing, i.e. the output vector which triggers apoptosis, 0000000.
If we find the input vector, we use that vector in our future programs. If we don't, it is certain that the modeled Boolean Network is not modeled properly.

5.3.3. Fault Introduction into Boolean Network

5.3.3.1. Testing Single Fault Scenario

Having found the unique input vector, we'll use this vector for all the operations coming next. In this section, out of all the 24 potentially faulty gates in the network, we will consider only one fault at a time. We will run the Boolean Network (now faulty) with the same input vector, but each time we would change the fault location.
Now, there are two types of gates in the network, one where a NOT (inverter) is involved, which is shown in red, and some do not involve a NOT gate, i.e. AND, OR. The guide to inducing stuck-at faults biologically is given in reference [1].
We will observe all the output vector generated, a total of 24 for each fault, and classify them according to their resemblance to other faulty outputs.
A special type of fault is the **undetectable fault**, where the output vector is the same as that of the fault-free Boolean Network. In this situation, the fault cannot be detected because it generates the output vector=0000000.

5.3.3.2. Testing Multiple Fault Scenario

We have created a double fault scenario in which we are applying two faults at a time. There are 24 nodes and considering 2 faults, the total number of combinations are $^{24}C_2 = 276$. Thus the dimensions of the output DataFrame would be 7×276, i.e. 7 output proteins as rows and 276 faulty conditions as columns. Now, we have created a triple fault scenario in which we are applying three faults at a time. Considering that, the total number of combinations would be ${}^{24}C_3$ = 2024 which is relatively large and will take much more computation time.

5.3.4. Drug Combination Application

5.3.4.1. Single Fault Scenario

Say, we have four drugs, and we know where in the Boolean Network they usually interfere. Here is how a drugs works in accordance with pathway logic:

For example, let's consider a stuck-at-1 fault [1] in the network, meaning the gate is conveying the value 1 to the following nodes no matter what the function and input values are for that particular node. If we apply a drug, i.e. the logical value of drug is 1, and pass it through an inverter to an AND gate, whose other input is the faulty path itself, the inverted value of the drug, i.e. 0, will generate the value 0 from the AND gate, thus changing the value of that particular signal. In simple words, the drug inhibits the effect of the faulty protein.

Now, in our case, there are 24 potential faults, and 6 drugs. So, for each faulty node, there is going to be 2^6=64 drug combination that we can use to generate the output vectors and observe how they change in contrast to the faulty output, and how close they are to the non-proliferative output (0000000) and how far they are from the proliferative output (1111111).

5.3.4.2. Multiple Fault Scenario

In the previous section, we considered only one fault at a given time. Here, we will consider at least two faults. So, the number of rows, i.e. drug combinations will be 64 (assuming six drugs in the vector), and the number of columns is going to be, as we found out in the previous module, 276. Again, we can increase the number of faults if we have the computing time.

5.3.4.3. Visualization of Effect of Drug Combination

Since, we have 7 output bits in the output vector, and say, for example, that a particular output is 1111111, this does not mean that the output is 127. The output vector doesn't convey a value, but the states of the output proteins in that particular input and faulty condition. Out of the seven outputs, 4 are transcription factors, which finally trigger the proliferation if required, and the rest are residual proteins. So, we have to separate the output vector into two parts and develop an expression that converts each of them into a meaningful number.

Let, output = [a,b,c,d,e,f ,g]
$F = a+b+c+d$
$S = e+f+g$
$P = F \times S$
$S = F+S$
$f(\text{Output}) = zP+(1-z)S$,

Choosing the value of *z* is important as it will result in the difference between the outputs of the application itself. Furthermore, encoding operations take more time to execute than generating faulty networks and their outputs themselves. Therefore, encoding is parallelized with the use of GPU instead of CPU to engage more than a thousand parallel threads at once, providing maximum speed for execution. GPU run kernel codes which are written in PyCUDA, which can access the CUDA APIs developed by Nvidia.

Now, for visualizing the results, the closest the value of ***f*** is to zero, the greener it will be displayed in the table. The far it gets from zero, the redder it will appear. In the end, the row, i.e. the combination of drugs, with the most number of values closer to zero (the most greenish number of entries), will be the optimum therapy if the fault location is unknown. In the location in known, we would isolate that particular column from the DataFrame, and look for the greenest entry, and the corresponding drug combination will be the optimum therapy.

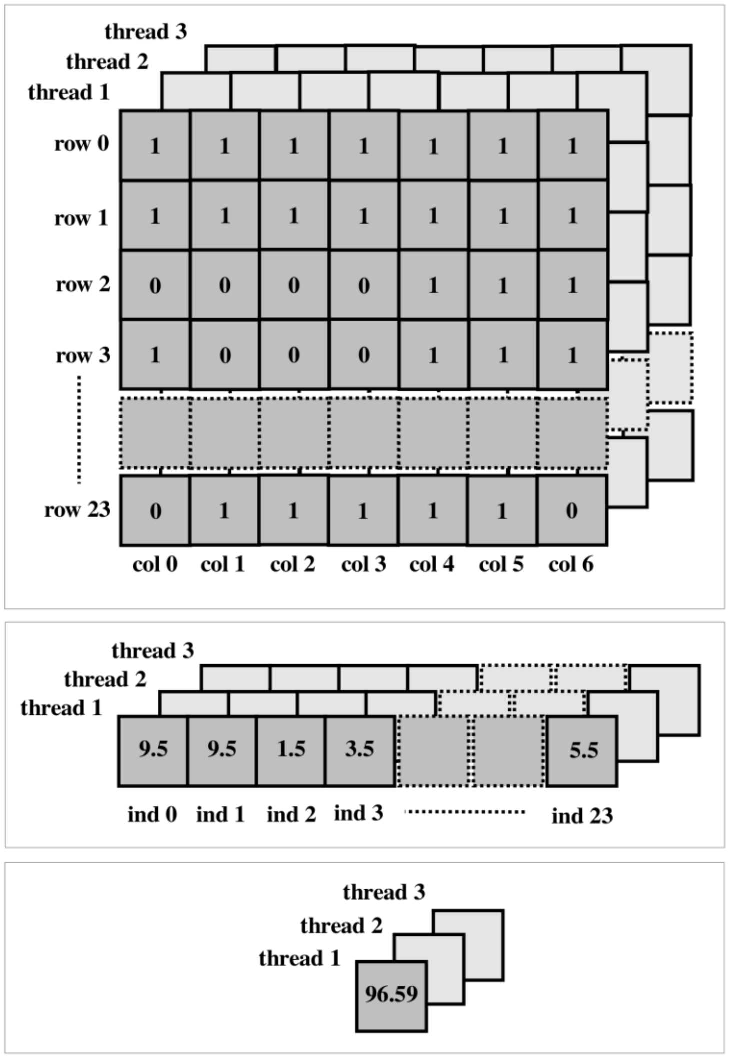


Figure 5.4: Conversion of output vectors into encoded weights, and further into condensed weights

For multiple fault and drug application simulation, the data has to be further condensed into condensed weights, where each of the condensing operations would generate one value for each drug combination.

Let, *encv*$_i$ be a list of encoded weights evaluated from an array containing output vectors. Here, *encv*$_i$ contains the encoded weights corresponding to a particular drug vector *drugv*$_i$, which belongs to the set of all drug combinations. Now, let *max*$_0$ be a constant variable which stores the summation of all encoded weights, belonging to a drug-less drug combinations, i.e. where all drug bits are 0's. This particular drug combination is the first combination in the drug set. If *faultn* is the number of fault combinations enumerated, then:

$$max_0 = \sum_{i=0}^{faultn} enc_0[i]$$

To evaluate the relative effect of each drug combination, the sum of the differences between each of the elements *encv*$_0$ and *encv*$_i$ lists corresponding to every drug combination has to be compared with *max*$_0$. Let each of the condensed weight be *cond*$_i$.

$$cond_i = \frac{100}{max_0} \sum_{j=0}^{faultn} \{encv_0[j] - encv_i[j]\}$$

The sum of differences between drug-less encoded list and given encoded list for particular drug combination would be compared with the value obtained from the drug combination where all bits are 0, i.e. no drugs are present. This would provide an insight into the relative change the drugs are bringing about.

5.3.5. Search for more efficient drug points

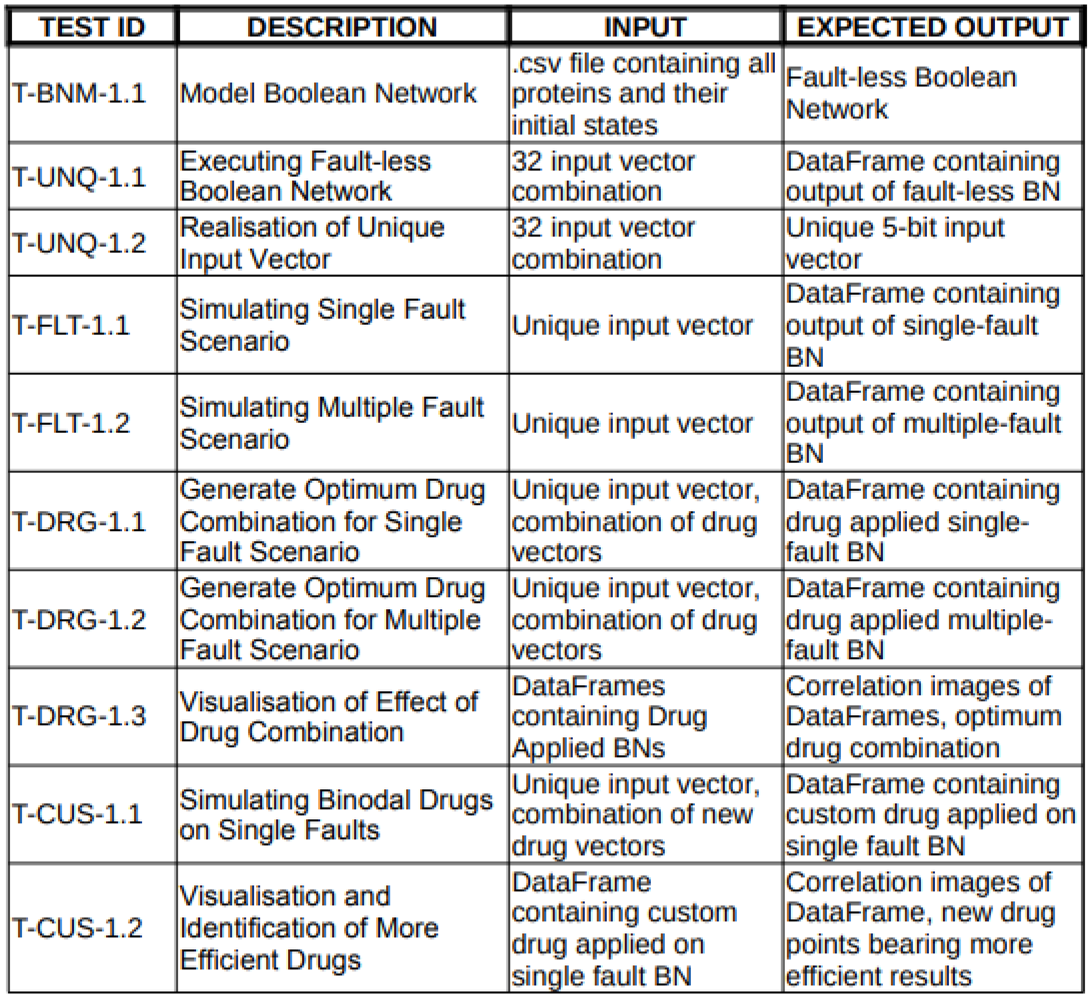
5.3.5.1.Simulation of binodal drugs on single faults

After getting some new drug points which are considered to be more efficient than the previously available points, we simulate that condition to generate the output matrix in a single fault scenario.

5.3.5.2.Visualization

We visualize the matrix as an image. The drug combination with the output more greenish than what achieved by the optimum drug combination in the previous module, is going to be our desired result.

5.4.Test Plan



TEST ID	DESCRIPTION	INPUT	EXPECTED OUTPUT
T-BNM-1.1	Model Boolean Network	.csv file containing all proteins and their initial states	Fault-less Boolean Network
T-UNQ-1.1	Executing Fault-less Boolean Network	32 input vector combination	DataFrame containing output of fault-less BN
T-UNQ-1.2	Realisation of Unique Input Vector	32 input vector combination	Unique 5-bit input vector
T-FLT-1.1	Simulating Single Fault Scenario	Unique input vector	DataFrame containing output of single-fault BN
T-FLT-1.2	Simulating Multiple Fault Scenario	Unique input vector	DataFrame containing output of multiple-fault BN
T-DRG-1.1	Generate Optimum Drug Combination for Single Fault Scenario	Unique input vector, combination of drug vectors	DataFrame containing drug applied single-fault BN
T-DRG-1.2	Generate Optimum Drug Combination for Multiple Fault Scenario	Unique input vector, combination of drug vectors	DataFrame containing drug applied multiple-fault BN
T-DRG-1.3	Visualisation of Effect of Drug Combination	DataFrames containing Drug Applied BNs	Correlation images of DataFrames, optimum drug combination
T-CUS-1.1	Simulating Binodal Drugs on Single Faults	Unique input vector, combination of new drug vectors	DataFrame containing custom drug applied on single fault BN
T-CUS-1.2	Visualisation and Identification of More Efficient Drugs	DataFrame containing custom drug applied on single fault BN	Correlation images of DataFrame, new drug points bearing more efficient results

6. Implementation

6.1.Implementation Details

6.1.1. Boolean Network Modeling

pathway_normal.py contains our network flow, which takes faults, input, pathway, outputs as lists. Since, in this module, the network doesn’t have any fault, we would pass [0] as the fault list, so that it doesn’t flow throught any faulty statement.
pathway_drugged.py contains the network with faults along with the inhibition points of the known drugs.
pathway_custom.py contains the network with faults and the custom inhibition points of new drug combinations generated in the last module.

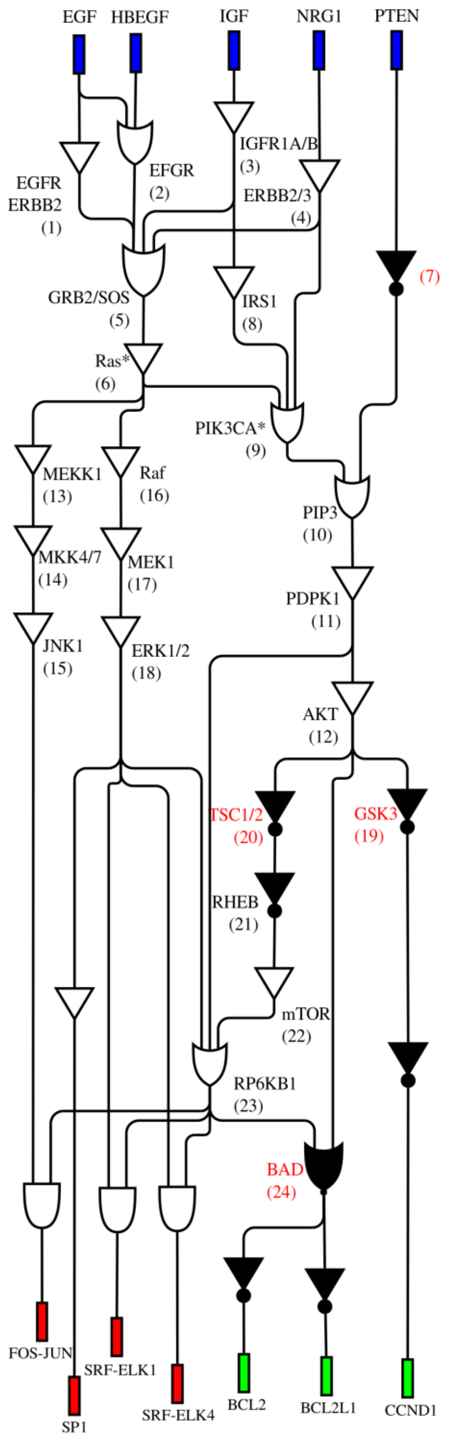


Figure 6.1: Boolean network converted from a growth factor “GF” signaling pathway

6.1.2. Unique Input Vector Identification

6.1.2.1. Executing Faultless Boolean Network

The Fault Less network, i.e. pathway_normal.py with fault list=[0], is traversed and all possible input vectors consisting of 0's and 1's are passed as input vector, and output vectors are generated.

6.1.2.2. Realization of Unique Input Vector

From all the output vectors obtained by traversing the network, we take a unique input value, corresponding to a unique output vector, which will act as the input vector for all the subsequent phases.
In our case the Input Vector obtained is 00001, which is the only input combination that produces the output vector 0000000.

6.1.3. Fault Introduction into Boolean Network

6.1.3.1. Simulating Single Fault Scenario

We introduce one fault at a node and traverse throughout the network. With every iteration, we introduce one fault to every node of the network.
The function pathway_normal.py takes in parameters as follows:def pathway(faultv,inpv,pathv,outv)
Here, only one fault location is sent in the faultv list, and inpv contains the unique input vector generated in the previous module.

6.1.3.2. Simulating Multiple Fault Scenario

Two Fault Scenario: Unlike the single fault scenario, in the two fault scenario, we are considering two nodes which are faulty for each iteration throughout the network.

Three Fault Scenario: In this case we are considering three faulty nodes at a time during a single iteration of the network.

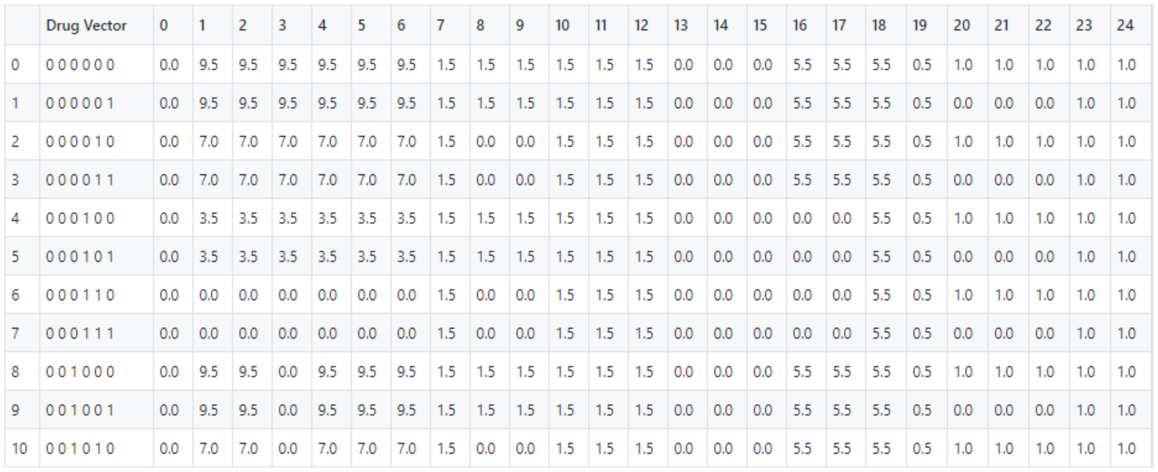
Here, faultv takes in two and three faultv locations for a given iteration, for double and triple fault simulations respectively.

6.1.4. Drug Combination Application

6.1.4.1. Generate optimum drug for single Fault Scenario

The python function pathway_drugged.py models the Boolean network along with the inhibition points of six known drugs. Here, along with the fault location list, a drugv list would be passed in the parameter list, containing the drug combination for that particular network flow. faultv contains only one fault location per network iteration. pathway_drugged.py takes parameters as:

```
    def pathway(faultv,drugv,inpv,pathv,outv)
```



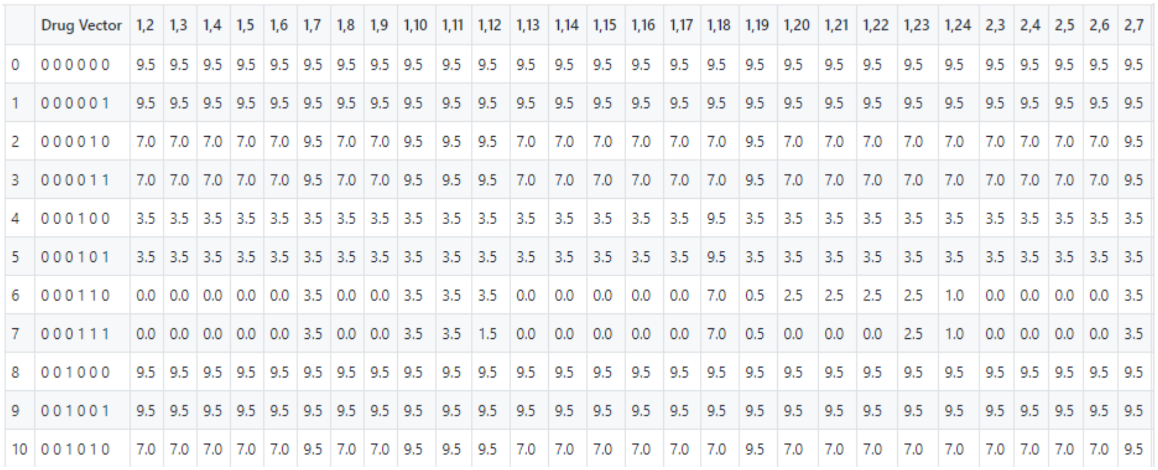
	Drug Vector	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	0 0 0 0 0 0	0.0	9.5	9.5	9.5	9.5	9.5	9.5	1.5	1.5	1.5	1.5	1.5	1.5	0.0	0.0	0.0	5.5	5.5	5.5	0.5	1.0	1.0	1.0	1.0	1.0
1	0 0 0 0 0 1	0.0	9.5	9.5	9.5	9.5	9.5	9.5	1.5	1.5	1.5	1.5	1.5	1.5	0.0	0.0	0.0	5.5	5.5	5.5	0.5	0.0	0.0	0.0	1.0	1.0
2	0 0 0 0 1 0	0.0	7.0	7.0	7.0	7.0	7.0	7.0	1.5	0.0	0.0	1.5	1.5	1.5	0.0	0.0	0.0	5.5	5.5	5.5	0.5	1.0	1.0	1.0	1.0	1.0
3	0 0 0 0 1 1	0.0	7.0	7.0	7.0	7.0	7.0	7.0	1.5	0.0	0.0	1.5	1.5	1.5	0.0	0.0	0.0	5.5	5.5	5.5	0.5	0.0	0.0	0.0	1.0	1.0
4	0 0 0 1 0 0	0.0	3.5	3.5	3.5	3.5	3.5	3.5	1.5	1.5	1.5	1.5	1.5	1.5	0.0	0.0	0.0	0.0	0.0	5.5	0.5	1.0	1.0	1.0	1.0	1.0
5	0 0 0 1 0 1	0.0	3.5	3.5	3.5	3.5	3.5	3.5	1.5	1.5	1.5	1.5	1.5	1.5	0.0	0.0	0.0	0.0	0.0	5.5	0.5	0.0	0.0	0.0	1.0	1.0
6	0 0 0 1 1 0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.5	0.0	0.0	1.5	1.5	1.5	0.0	0.0	0.0	0.0	0.0	5.5	0.5	1.0	1.0	1.0	1.0	1.0
7	0 0 0 1 1 1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.5	0.0	0.0	1.5	1.5	1.5	0.0	0.0	0.0	0.0	0.0	5.5	0.5	0.0	0.0	0.0	1.0	1.0
8	0 0 1 0 0 0	0.0	9.5	9.5	0.0	9.5	9.5	9.5	1.5	1.5	1.5	1.5	1.5	1.5	0.0	0.0	0.0	5.5	5.5	5.5	0.5	1.0	1.0	1.0	1.0	1.0
9	0 0 1 0 0 1	0.0	9.5	9.5	0.0	9.5	9.5	9.5	1.5	1.5	1.5	1.5	1.5	1.5	0.0	0.0	0.0	5.5	5.5	5.5	0.5	0.0	0.0	0.0	1.0	1.0
10	0 0 1 0 1 0	0.0	7.0	7.0	0.0	7.0	7.0	7.0	1.5	0.0	0.0	1.5	1.5	1.5	0.0	0.0	0.0	5.5	5.5	5.5	0.5	1.0	1.0	1.0	1.0	1.0

Figure 6.2: DataFrame of Single Fault Scenario

6.1.4.2.Generate optimum drug for multiple Fault Scenario

The same drugv contents are sent from the previous sub-module. But the faultv list would contain the list of alllocations which will be faulty for that iteration.

Since, the three-fault scenario is going to generate is large DataFrame, which will take more than 6 minutes of computing time, if run on the CPU, we will use the machine's GPU to compute the results in parallel threads. This approach will reduce the computing time down from 6 minutes to about 30 seconds.



	Drug Vector	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,10	1,11	1,12	1,13	1,14	1,15	1,16	1,17	1,18	1,19	1,20	1,21	1,22	1,23	1,24	2,3	2,4	2,5	2,6	2,7
0	0 0 0 0 0 0	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5
1	0 0 0 0 0 1	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5
2	0 0 0 0 1 0	7.0	7.0	7.0	7.0	7.0	9.5	7.0	7.0	9.5	9.5	9.5	7.0	7.0	7.0	7.0	7.0	7.0	9.5	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	9.5
3	0 0 0 0 1 1	7.0	7.0	7.0	7.0	7.0	9.5	7.0	7.0	9.5	9.5	9.5	7.0	7.0	7.0	7.0	7.0	7.0	9.5	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	9.5
4	0 0 0 1 0 0	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	9.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5
5	0 0 0 1 0 1	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	9.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5
6	0 0 0 1 1 0	0.0	0.0	0.0	0.0	0.0	3.5	0.0	0.0	3.5	3.5	3.5	0.0	0.0	0.0	0.0	0.0	7.0	0.5	2.5	2.5	2.5	2.5	1.0	0.0	0.0	0.0	0.0	3.5
7	0 0 0 1 1 1	0.0	0.0	0.0	0.0	0.0	3.5	0.0	0.0	3.5	3.5	1.5	0.0	0.0	0.0	0.0	0.0	7.0	0.5	0.0	0.0	0.0	2.5	1.0	0.0	0.0	0.0	0.0	3.5
8	0 0 1 0 0 0	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5
9	0 0 1 0 0 1	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5
10	0 0 1 0 1 0	7.0	7.0	7.0	7.0	7.0	9.5	7.0	7.0	9.5	9.5	9.5	7.0	7.0	7.0	7.0	7.0	7.0	9.5	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	9.5

Figure 6.3: DataFrame of Two Fault Scenario

6.1.4.3.Visualization of Effect of Drug Combination

The DataFrames generated from the previous two sub-modules are taken as input to a matplotlib function called imshow(). The x-axis maps the fault locations, and the y-axis maps the different drug combinations.

On a relatively large input, like what contains in the three-fault output, the image generated will be hard to understand, since each of the results will take less space than a pixel, a program condenser.py would return a single condensed value for each drug combination. This will increase the readability of the image, to an extent where a simple scatter plot, a function of the matplotlib package, can be used to map the values returned from each drug combination.

The drug vectors corresponding to the lowest points in the scatter plot is the optimum drug combination generated.

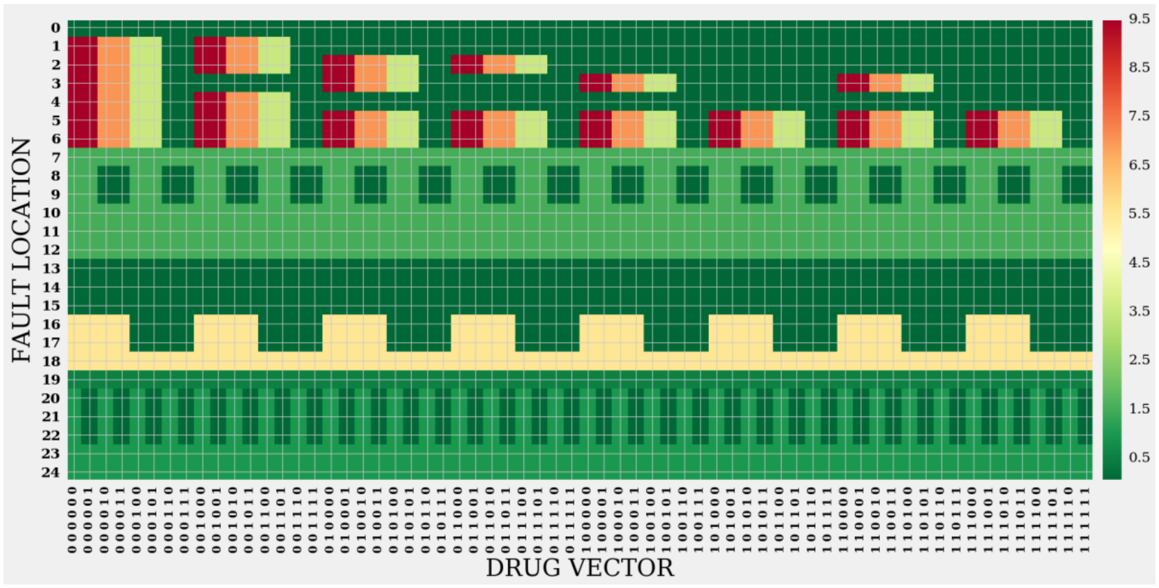


Figure 6.4: Image Show of Drug Application on single fault

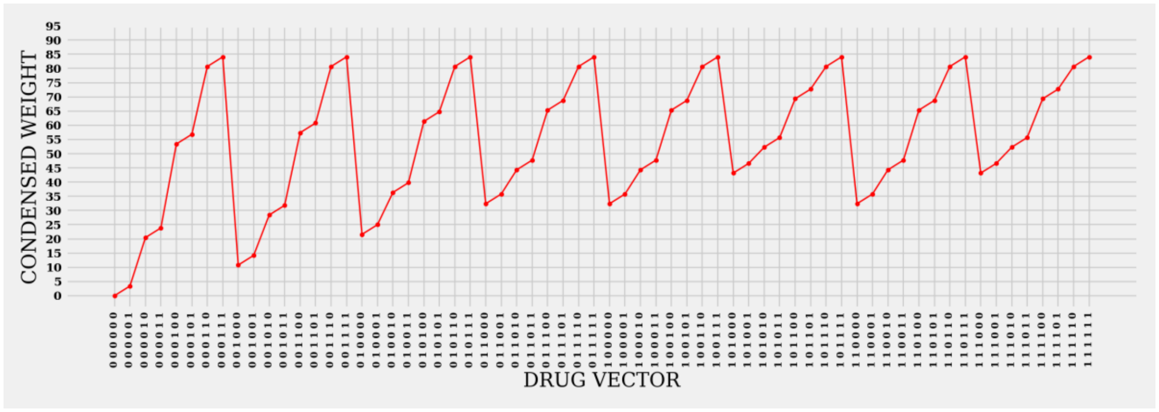


Figure 6.5: Scatter Plot of Drug Application on single fault

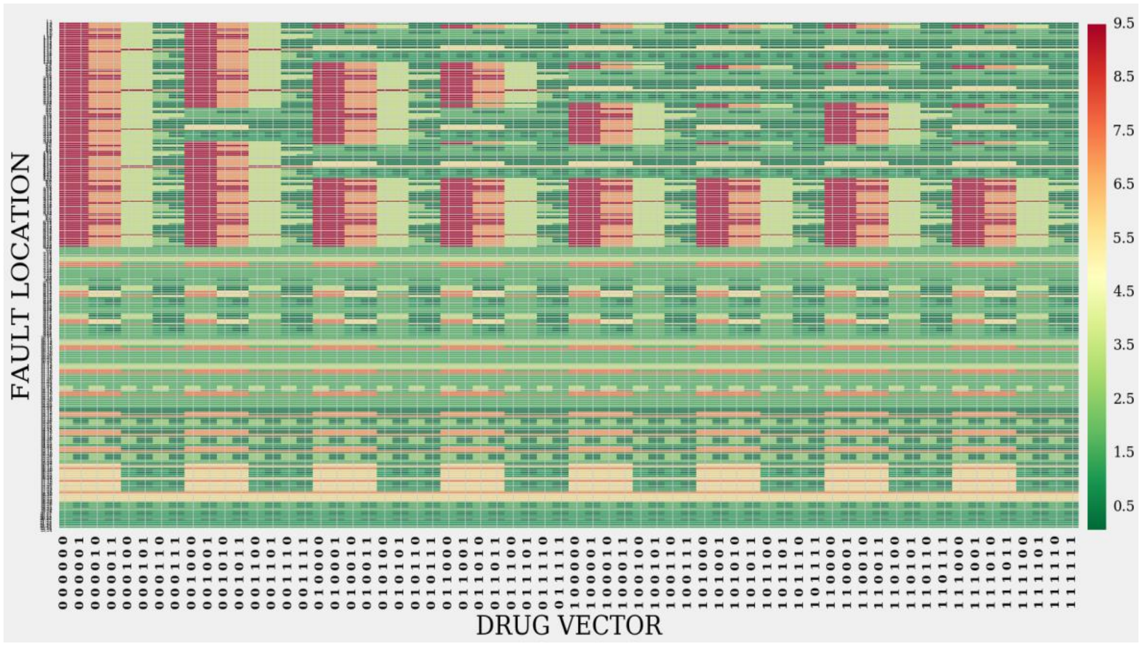


Figure 6.6: Image Show of Drug Application on two faults

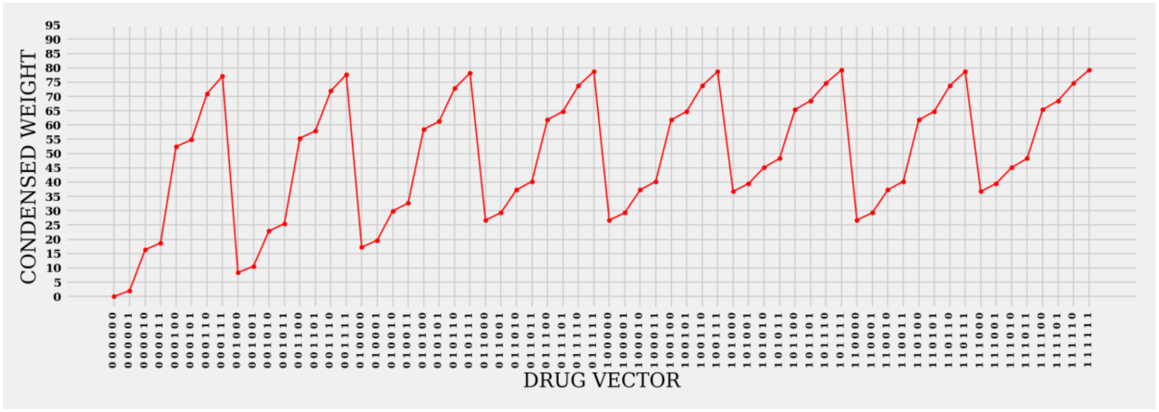


Figure 6.7: Scatter Plot of Drug Application on two faults

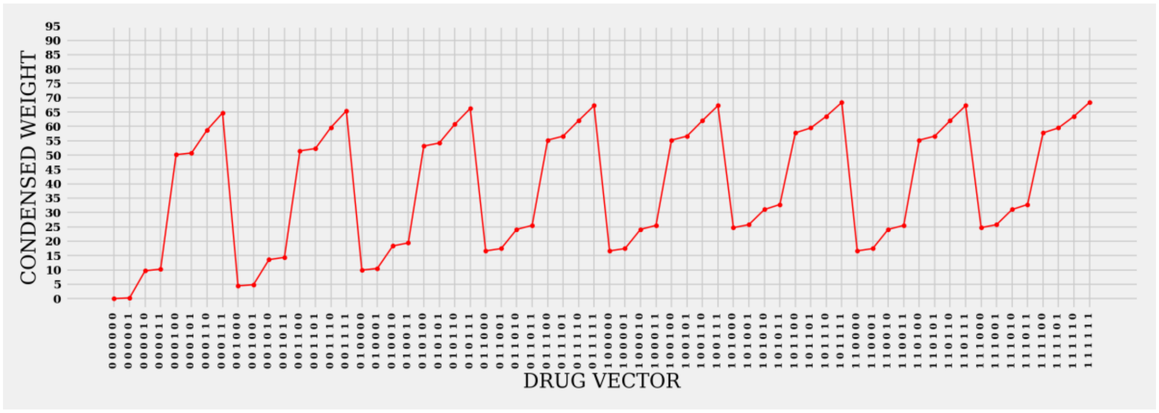


Figure 6.8: Scatter Plot of Drug Application on three faults

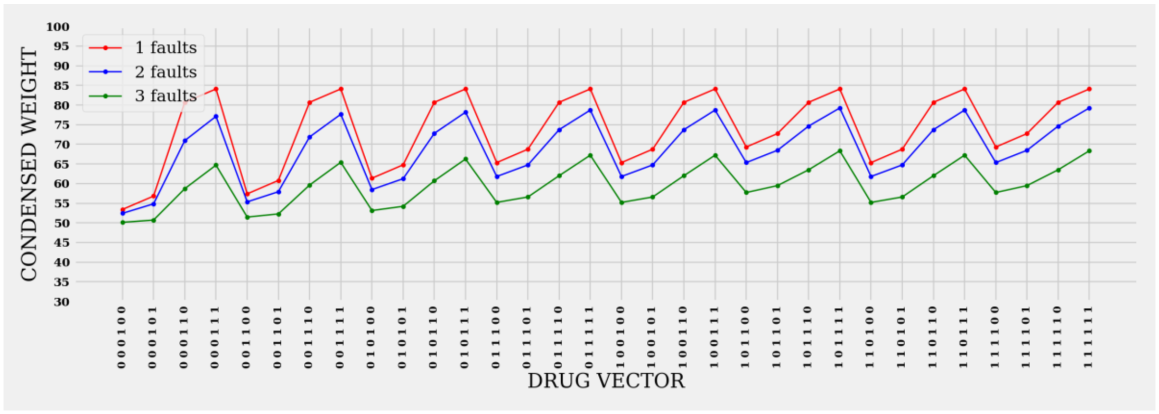


Figure 6.9: Scatter Plot of Drug Application on all fault scenarios combined

Execution time: Both sequential and parallel executions of encoding operations are carried out to contrast between their respective execution time. The details are mentioned in the following table and also visualized in figure 6.10.

TABLE 3
EXECUTION TIME CORRESPONDING TO EACH FAULTY SCENARIO

Parameters	Concurrent fault scenario F			
	F = 1	F = 2	F = 3	F = 4
Operations performed	1,536	17,664	129,536	680,064
GPU time	1.01 s	3.23 s	18.52 s	≈ 2 m
CPU time	0.1 s	6.08 s	≈ 3 m	≈ 3 hr

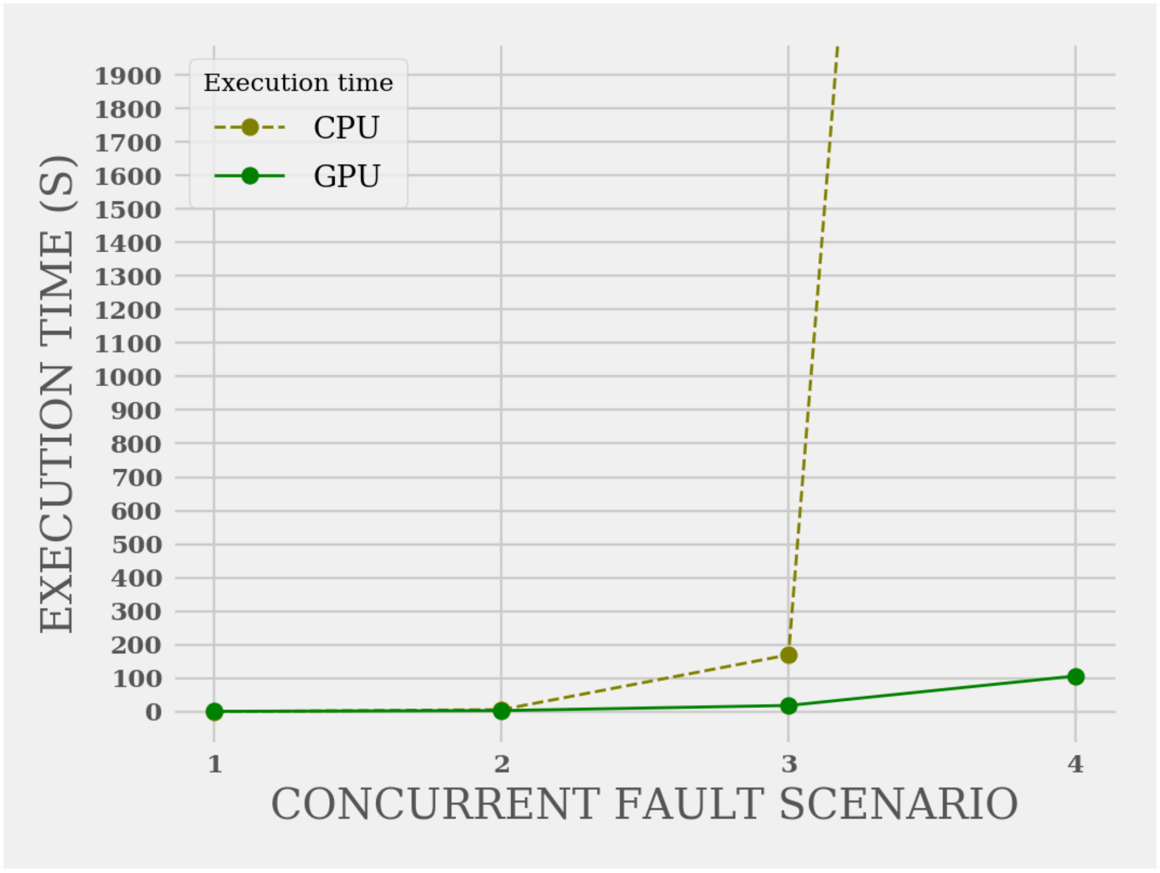


Figure 6.10: Execution time corresponding to CPU and GPU

6.1.5. Search for More efficient drug points

6.1.5.1.Simulating binodal drugs on single faults

A combination of different drugs would be generated, each of which affects two nodes in the network at a time, just like what we did in the two-fault simulation. Those set of combinations will be applied to a network containing single faults. The results would be stored in the DataFrame.

6.1.5.2.Visualization and Identification of more efficient drugs

matplotlib's imshow() and scatter() would be used to visualize the results produced in the previous sub-module. Conclusions would be drawn as done in module 6.1.4.3. If results are better than what produced using known drugs, we'd conclude that we've generated the desired results.

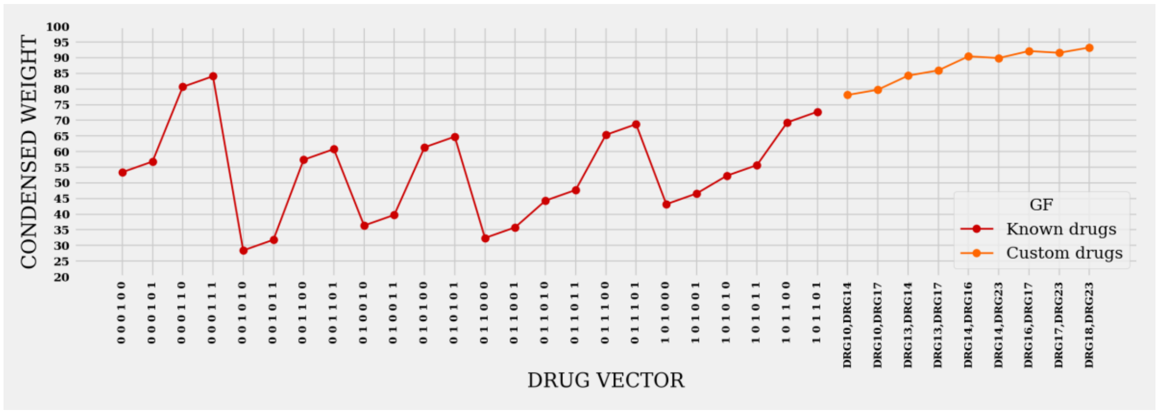


Figure 6.11: Comparison of known drugs and custom drugs on single faults

In figure 6.11, known drugs on the left hand side, are listed as simple 7-bit drug vectors, while unknown drugs are mentioned as a list of DRGi separated by commas, where i belongs to the set of all potentially faulty nodes. For example, the label “DRG10, DRG14” names a drug which affects node 10 and 14 simultaneously. There are a few custom drug combinations which generate higher scores than what achieved highest by a known drug vector 0001110 (score = 84). The custom drug vector which produced the highest score is “DRG18, DRG23”, implying the inhibition of nodes 18 and 23, with a score of 93.

7. CONCLUSION

7.1.Project Benefits

- Laying out a Boolean Network from a Gene Regulatory Network will help us understand the workings of proteins in a simple manner, stemming from the fact that every protein has an ON-OFF feature.
- Identification of unique input vector will produce output vectors which signify both proliferative and non-proliferative situations.
- Fault location will convey that every gate in the Boolean Network is a potential fault in the system.
- Assumption of more than one fault at a given time will produce a more realistic solution to cancerous conditions, while figuring out the drug combination for the faulty network.

7.2.Future Scope for Improvements

- Introduction of more than two faults if we can get increased computation power.
- Introduction of bridge faults.

8. REFERENCES

[1] RitwikLayek, Aniruddha Datta, Michael Bittner and Edward R. Dougherty, "Cancer therapy design based on pathway logic", Bioinformatics, Vol. 27, Advance Access 30 December 2010

[2] Osama A. Arshad, Priyadarshini S Venkatasubramani, Aniruddha Datta, JijayanagaramVenkatraj, "Using Boolean Logic Modeling of Gene Regulatory Networks to Exploit the Links Between Cancer and Metabolism for Therapeutic Purposes", IEEE Journal of Biomedical and Health Informatics, Vol. 20, No. 1, January 2016

[3] Frank Emmert-Streib, Matthias Dehmer, Benjamin Haibe-Kains, "Gene regulatory networks and their applications: understanding biological and medical problems in terms of networks", Frontiers in Cell and Developmental Biology, Mini Review Article, 19 August 2014

[4] Handbook of Computational Molecular Biology, "Chapter 27: Identifying Gene Regulatory Networks from Gene Expression Data"

[5] Anwoy Kumar Mohanty, Student Member, Aniruddha Datta, VijayanagaramVenkatraj, "A Model for Cancer Tissue Heterogeneity", IEEE Transactions on Biomedical Engineering, Vol. 61, No. 3, March 2014

[6] https://www.cancer.gov/about-cancer/understanding/statistics

[7] Fundamentals of Software Engineering (Rajib Mall), 4th Edition, PHI Learning Pvt. Ltd.

[8] Cuda By Example, NVIDIA by Jason Sanders and Edward Kandrot

[9] https://developer.nvidia.com/pycuda