Submitted by

**IDENTIFICATION OF OPTIMUM DRUG COMBINATION FOR CANCER USING BOOLEAN NETWORKS**

ABHISEK KARMAKAR (13000114006)

ARGHA NANDAN (13000114015)

BISHAL SAHA (13000114023)

Under the guidance of Prof. Tapan Chowdhury

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 /she 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 staff for the gracious hospitality they offered us.

Place: Techno India, SaltLake

Date:

………………………………

………………………………

………………………………

Department of Computer Science and Engineering

Techno India, SaltLake

Kolkata – 700 091

West Bengal, India.

## **Approval**

This is to certify that the project report entitled “Identification of Optimum Drug Combination for Cancer using Boolean Networks” prepared under my supervision by*Abhisek Karmakar (13000114006), Argha Nandan (13000114015), and Bishal Saha (13000114023),* be accepted in partial fulfillment for the degree of Bachelor of Technology in Computer Science and Engineering.

It is to be understood that by this approval, the undersigned does not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn thereof, but approves the report only for the purpose for which it has been submitted.

……………………………………….. … ………………………………………

**Mr. Tapan Chowdhury Mrs. Poulomi Dutta**

Asst. Professor In-Charge

Computer Science and Engineering Computer Science and Engineering

Techno India, Salt Lake Techno India, Salt Lake

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SL. No.** | **TOPIC** | **PAGE NO.** |
| **1** | Introduction | 1 |
| **1.1** | Briefing | 1 |
| **1.2** | Problem Domain | 1 |
| **1.3** | Glossary | 1 |
| **2** | Problem Definition | 2 |
| **2.1** | Scope | 2 |
| **2.2** | Exclusions | 2 |
| **2.3** | Assumptions | 2 |
| **3** | Related Study | 2 |
| **3.1** | GRN | 3 |
| **3.2** | Boolean Networks | 3 |
| **3.3** | Faults in BN | 3 |
| **4** | Project Planning | 4 |
| **4.1** | Software Life Cycle Model | 4 |
| **4.2** | Scheduling | 5 |
| **4.2.1** | Project Schedule | 6 |
| **4.2.2** | Gantt Chart | 7 |
| **4.3** | Cost Analysis | 7 |
| **5** | Requirement Analysis | 8 |
| **5.1** | Requirement Matrix | 8 |
| **5.2** | Requirement Elaboration | 8 |
| **5.2.1** | Boolean Network Modelling | 8 |
| **5.2.2** | Unique Input Vector Identification | 8 |
| **5.2.3** | Fault Introduction into Boolean Network | 8 |
| **5.2.4** | Drug Combination Application | 9 |
| **6** | Design | 9 |
| **6.1** | Technical Environment | 9 |
| **6.2** | Hierarchy of Modules | 9 |
| **6.3** | Detailed Design | 10 |
| **6.3.1** | Boolean Network Modelling | 11 |
| **6.3.2** | Unique Input Vector Identification | 12 |
| **6.3.3** | Fault Introduction into Boolean Network | 12 |
| **6.3.4** | Drug Combination Application | 13 |
| **6.4** | Test Plan | 15 |
| **7** | Conclusion | 16 |
| **7.1** | Project Benefits | 16 |
| **7.2** | Future Scope for Improvements | 16 |
| **8** | References | 16 |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **SL. No.** | **List of Tables** | **Page Number** |
| **1.1** | Hardware Specifications | 1 |
|  |  |  |
|  |  |  |
|  | **List of Figures** |  |
| **4.1** | Iterative Waterfall Model | 4 |
| **4.2** | Project Schedule | 5 |
| **4.3.1** | Gantt Chart (November 2017) | 6 |
| **4.3.2** | Gantt Chart (November 2017 – Phase I completion) | 7 |
| **5.1** | Requirement Matrix | 8 |
| **6.2.1** | Hierarchy of Modules | 9 |
| **6.3.1** | Detailed Design | 10 |
| **6.4** | Different types of pathway scenarios | 11 |
|  |  |  |

1. **INTRODUCTION**
   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 GRN, a Boolean Network is modelled 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.

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. **Problem Domain**
     1. Software Specifications
* Python 3.6 Interpreter
* Spyder 3.6 for Python Programming
  + 1. Hardware Specifications

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

Table 1.1: Hardware Specifications

* 1. **Glossary**

GRN: A network that has been inferred from gene expression data a “gene regulatory network,” is briefly denoted as GRN.

Boolean Network(BN): A Boolean Network B (v, f), on n genes / proteins is defined by a set of nodes V=(x1……xn), xi∈ {0, 1} where I = 1… n and a list F = (f1…..fn) of boolean functions, fi: {0, 1} n+m 🡪 {0, 1}, i= 1….n, m>=0 is the number of external inputs e.g. GFs, stresses, metabolic, etc. Each node xi represents the state / expression of the gene I, where xi = 0 means the gene / protein i is OFF (inactive) and if xi = 1, that means gene / protein is ON (active).

Faults: 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.

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

1. **PROBLEM DEFINITION**
   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.

* 1. **Exclusions**

The part where analysis which particular gene are affecting the growth factors to send the messages to the cells to proliferate is excluded.

* 1. **Assumptions**

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

Only Stuck-At faults 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.

1. **RELATED STUDY**
   1. **GRN**

A network that has been inferred from gene expression data a “gene regulatory network,” briefly denoted as GRN. The Gene regulatory networks provide information about regulatory interactions between regulators and their potential targets; gene-gene interactions, and potential protein-protein interactions.

* 1. **Boolean Networks**

A BN (Glass and Kauffman, 1973; Kauffman, 1969, 1993),*B*=(*V,F*), on *n* genes/proteins is defined by a set of nodes (genes/proteins) *V* ={*x*1*,...,xn*}, *xi*∈{0*,*1}, *i*=1*,...,n*, and a list *F* =(*f*1*,...,fn*), of Boolean functions, *fi* :{0*,*1}*n*+*m* →{0*,*1}, *i*=1*,...,n*,*m*≥0 is the number of external inputs e.g. GFs, stresses, metabolites, etc. Each node *xi* represents the state/expression of the gene *i*, where  
*xi* =0 means that gene/protein *i* is OFF (unexpressed or inactive according to their biological significance) and *xi* =1 means that gene/protein *i* is ON (expressed or active). The function *fi* 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. **Faults in BN**

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. 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.

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. **PROJECT PLANNING**
   1. **Software Life Cycle Model**

In our Project we have followed the Iterative Waterfall Model:

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.

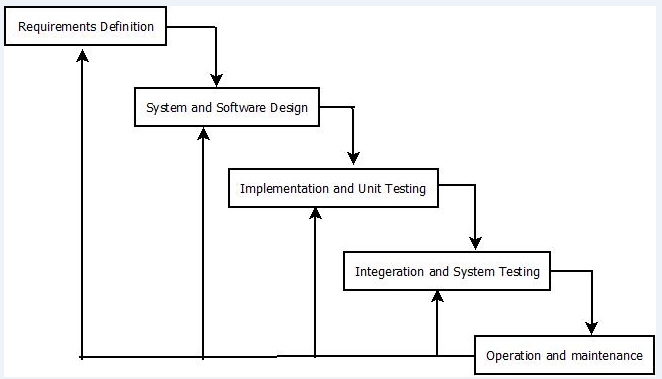
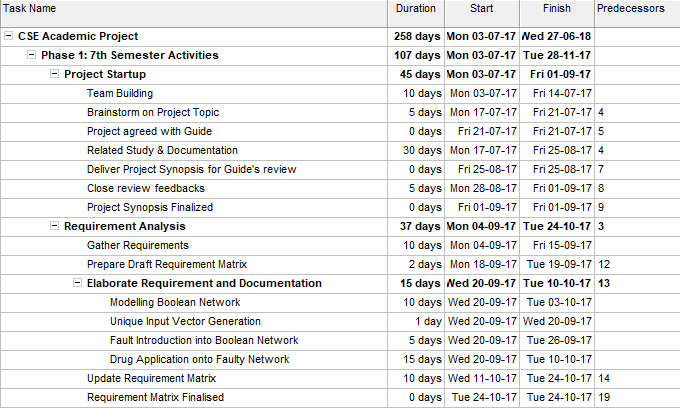


Figure: 4.1 Iterative Waterfall Model

* 1. **Scheduling**
     1. **Project Schedule**



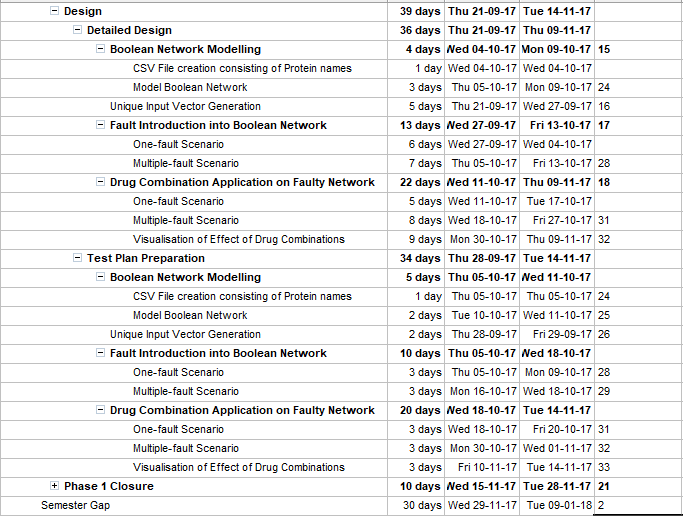


Figure 4.2: Project Schedule

* + 1. **Gantt Chart**

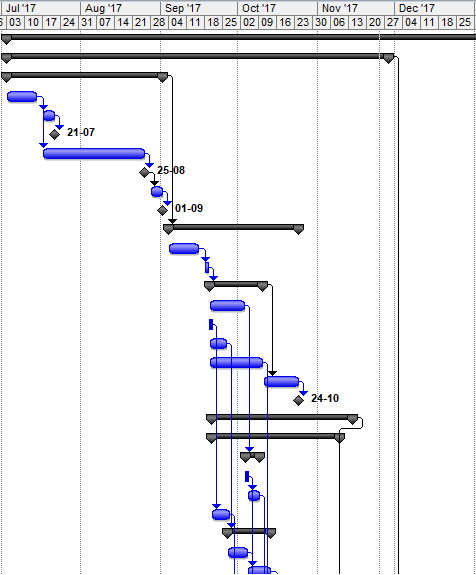


Figure 4.3.1: Gantt chart (November 2017)

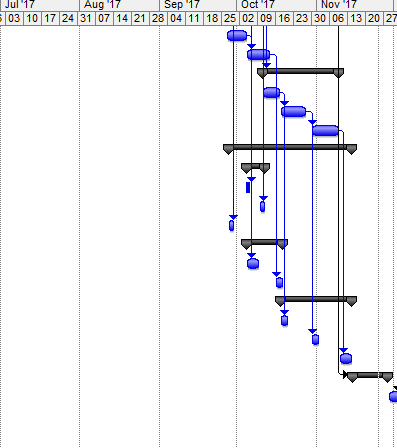


Figure 4.3.2: Gantt chart (November 2017 – Phase I closure)

* 1. **Cost Analysis**

1. **REQUIREMENT ANALYSIS**
   1. **Requirement Matrix**

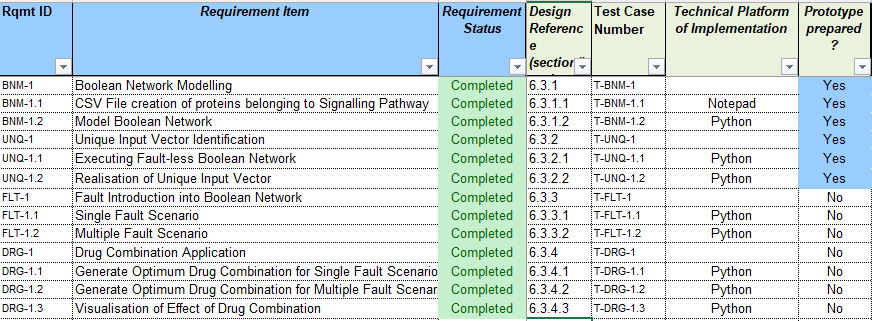


Figure 5.1: Requirement Matrix

* 1. **Requirement Elaboration**
     1. **Boolean Network Modelling**

From a given GRN, a Boolean Network needs to be modelled. 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.

* + 1. **Unique Input Vector Identification**

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

* + 1. **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, and b) more than one faults in the network at any given time.

* + 1. **Drug Combination Application**

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.

1. **DESIGN**
   1. **Technical Environment**

We are using Ubuntu 16.04 as the base Operating System, Intel Core i3 processor, 4 GB RAM, and 500 GB HDD.

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.

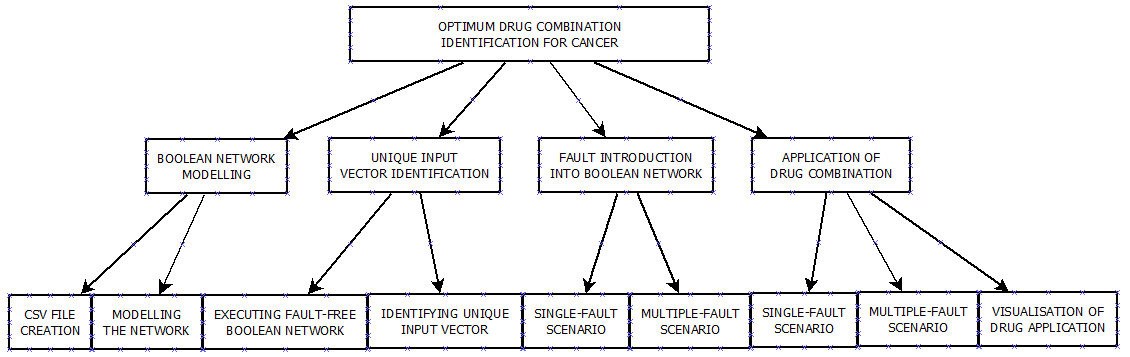
* 1. ** Hierarchy of Modules**

Figure 6.2.1: Hierarchy of Modules

* 1. **Detailed Design**

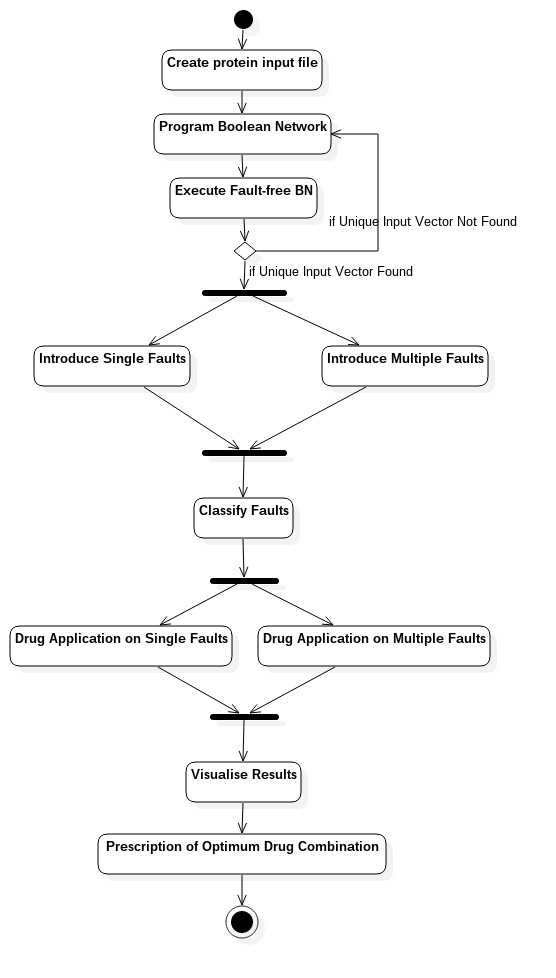
****

Figure 6.3.1: Detailed Design

* + 1. **Boolean Network Modelling**
       1. **CSV file creation for proteins belonging to Signaling Pathway**

CSV(Comma Separated Values) files are appropriate for reading and writing DataFrames, since they are stored in a format consisting of delimiters automatically. Since, our inputs and output are DataFrames, which follow a strict tabular format, CSV is the optimum choice.

The Signaling Pathway we’re using, i.e. Growth Factor Signaling Pathway, consists of 35 proteins. Some of them are growth factors and transcription factors.

We’re constructing the file with integers, starting from index 0 till 34, as the first column. The second column has the names of the corresponding proteins, separated from the 1st column by a comma.

* + - 1. **Model Boolean Network**

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

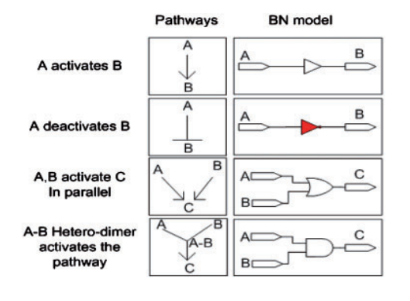


Figure 6.4: Different types of pathway scenarios

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.

* + 1. **Unique Input Vector Identification**
       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.

* + - 1. **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 proliferation is 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 faulty.

* + 1. **Fault Introduction into Boolean Network**
       1. **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 way we have to assign the stuck-at fault value is as follows:

1. Where there is NOT gate involved, we will assign a stuck-at-0 fault.
2. Where there is not a NOT gate involved, we’ll assign a stuck-at-1 fault.

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.

* + - 1. **Multiple Fault Scenario**

Here, we can create a more realistic approach to a cancerous condition, since in reality, more than one fault occur in the signaling pathway. If we consider two faults at any given time, the number of fault combinations would be 24C2=276­.

So, the dimensions of the output DataFrame thus created would be 7×276, i.e. 7 output proteins as rows and 276 faulty conditions as columns.

If required, we can consider three faults at a time, in which case the total number of faulty combinations would be 24C3=2024, which is relatively large, considering that it will be the number of columns in the DataFrame. If we have the computing ability and time, we can work on any scenario of multiple-fault network.

In the end, we will classify these faults as well as to group them into their corresponding types.

* + 1. **Drug Combination Application**
       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 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 4 drugs. So, for each faulty node, there is going to be 24=16 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).

* + - 1. **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 16 (assuming four 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.

* + - 1. **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×S

S=F+S

*f*(Output)=*z*P+(1−*z*)S,

Choosing the value of *z* is important as it will result in the difference between the outputs of the application itself.

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 Data Frame, and look for the greenest entry, and the corresponding drug combination will be the optimum therapy.

* 1. **Test Plan**

1. **CONCLUSION**
   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 nonproliferative 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.
  1. **Future Scope for Improvements**
* Introduction of more than two faults if we can get increased computation power.
* Introduction of bridge faults.

1. **REFERENCES**

* [1] Ritwik Layek, 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, Jijayanagaram Venkatraj, “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, Vijayanagaram Venkatraj, “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