# Project SIAN

Using Boolean Network Modeling to Identify Gene Interactions and Intervene in Genetic Diseases

New Mexico Supercomputing Challenge
Final Report
April 4, 2007

Team #44
Las Cruces High School

**Members:**
Jerry Yeh
Chris Smith

**Mentor:**
Dr. Joe Song

**Sponsor:**
Gregory Marez

# Executive Summary

## Simulating and Intervening Abnormal Networks (SIAN)

Modeling a complex system in a global fashion can be realized by representing variables in the system with discrete values and establishing interaction patterns among the variables. A gene may or may not be expressed. This on/off state can be incorporated into a binary model to characterize a genetic network with a large number of genes. The representation of a gene as a binary state is the essence of Boolean network modeling.

The aim of SIAN is to produce genetic network models in cancer by simulating and controlling the regulation of genetic networks as accurately as possible using this form of Boolean network modeling. Most importantly, SIAN intervenes with incongruous cancerous networks to modify networks to a state of normality with the least cost, thus suppressing cancer or any other genetic disease in an efficient manner.

To accomplish this, SIAN implements three separate algorithms in order to model a genetic network and establish effectual intervention techniques. The program is partitioned into the Simulation, Inference, and Intervention algorithms. In addition, a built-in visualization of tumor growth is provided so the user can view a real-life scenario of how the dysfunctional cells can eventually lead to a tumor. Because genetic networks may have an abundant number of genes involved in cancerous activity, computational efficiency in determining the optimal gene therapy is crucial. Since the computations are extremely time-intensive, access to supercomputers was necessary to successfully characterize a network of a realistic size.

Ultimately, once the expression of thousands of genes is collected through advanced biotechnology, our program can be employed to determine the genetic interactions that are responsible for tumor growth and design an effective strategy to intervene in abnormal networks in a proficient manner. Gene therapists around the world can then effectively determine the proper genetic modifications for various genetic mutations from the applied data.

## Problem/Goal Statement

What is an efficient way to assist gene therapists in designing an efficient treatment procedure for patients with genetic diseases?

The objective is to develop a Java program to implement a Boolean network model. Boolean network modeling determines the expression of an individual gene and its relations to the others within a genetic network. This model can identify gene interactions that cause genetic diseases and provide a therapeutic solution for the disease by modifying the expressions of a minimum number of genes to cause the network to return to its normal state. This, in turn, suppresses abnormal genes and reduces the effects of genetic diseases and cancers.

# Background

### Gene Regulatory Networks as a Preface to Gene Therapy

During the reproduction of cells, there are many "checkpoints" that control the phases of the reproduction process. These signals, coordinated by the states of genes, not only control the phases of the cell cycle, but also the rate of cycles performed in the body. Cells that do not heed the regular signals that regulate the cell cycle create cancerous growths or other forms of genetic diseases.

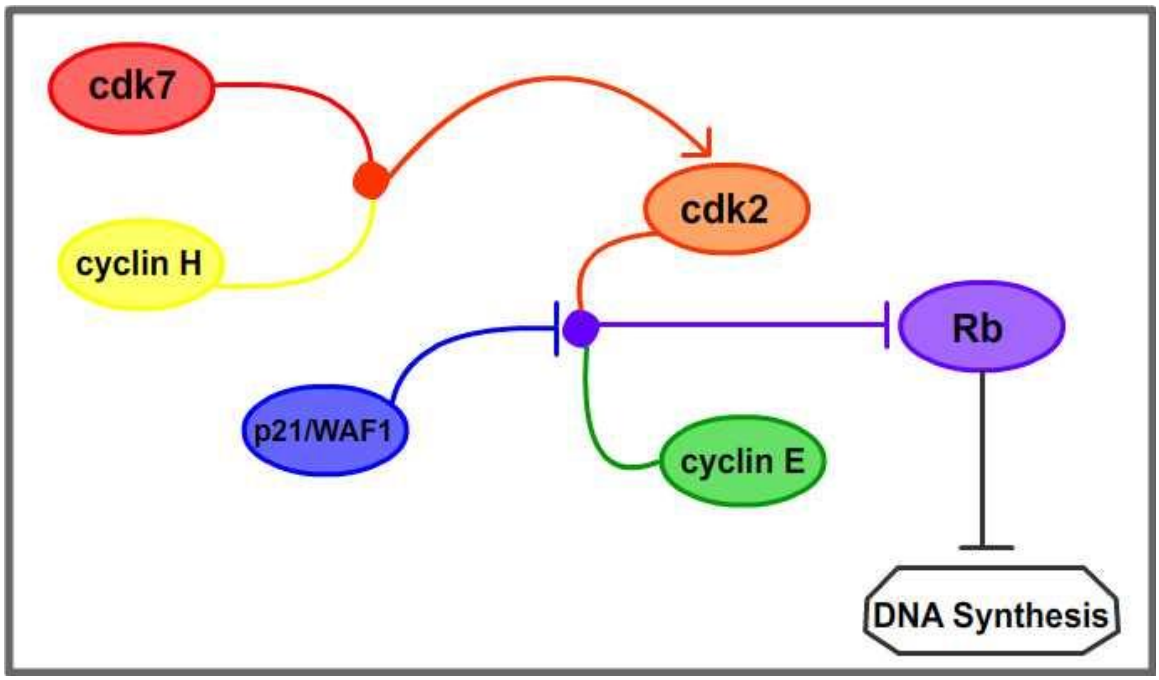



**Fig. 1** A sample genetic network with various proteins acting in relation to DNA synthesis as the ultimate function.

Cancer is the single most damaging and widespread class of disorder, studied and treated for centuries. Due to environmental factors and genetic abnormalities, cancer has become a leading cause of death in society. Although much effort and funding have been spent for the past couple of decades in discovering a "cure" for cancers, there has yet to be a definite understanding of the cancer mechanism at the molecular level, which is believed

to be rooted in the dysfunctions in cell cycles. Thus, there must be investigation of the underlying gene regulations where cancer occurs before an effective treatment is possible.

Forthcoming treatments for cancer and genetic diseases pertain to the most up to date genetic therapy processes. Broadly defined as revising genes in one's cells to treat hereditary diseases, gene therapy is widely sought by many individuals who wish to find a definite cure for such diseases. Even though gene therapy provides successful methods in manipulating the genes in a cancer cell, it is still a precarious procedure in its infancy. The first approved gene therapy procedure was executed in the early 1990's by cloning functional proteins for people who were unable to produce those proteins in their body.

There are two broad types of gene therapy: *ex vivo* and *in vivo*. *Ex vivo* is the process of modifying cells outside the host body and then transplanting them back to their original locations. *In vivo*, on the other hand, is the process in which the abnormal cell is modified inside the body. Gene therapy *ex vivo* is a more common process because it does not require the highly difficult recombination of DNA sequences.

By pinpointing the gene expressions that need to be altered, methods including selective reverse mutation, which returns the abnormal cell to its normal state, and inhibition, which alters the effects of a growth or death regulator gene, can transform a defective network back to a functioning one. In addition, gene insertion and replacement recombination are methods which are currently available.

# Algorithms in Boolean Network Modeling

### Methodology in Boolean Networks

Boolean networks are directed graphs where each node takes the values of 0 or 1. Each node, is associated with a truth table or Boolean function. A Boolean network reflects how a system of variables interact with each other and how the system evolves over time. Truth tables are used to show the individual value(s) of the parent cells(s) and the resulting value of the child node at time [t] in the interval due to the function of the parent value(s).

For instance, if $x_1$ is a function of $\{x_2, x_3\}$, and $x_1[t]$ is determined by the function ($x_2[t-1]$ or $x_3[t-1]$), then here are the following possibilities:

| $x_2[t-1]$ | $x_3[t-1]$ | $x_1[t]$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Table. 1** Sample truth table of $x_1[t+1]$ as a function of $x_2[t] \cup x_3[t]$

Each node may be a function of any number of parents; the node could also be the parent affecting itself in a ( ) (blank set).

## A. Random Boolean Network Generation

This module generates a Boolean network with user specified number of nodes *N* and maximum number of parents *K*. For each node, the number of parents k is first generated randomly. Then k parents are picked randomly from all the nodes. Once the parents are determined, a truth table is randomly chosen. For k parents, there are $2^k$ truth table entries. Each entry has same probability of being 0 or 1.

This part simulates a real-life gene regulatory network using a random Boolean network. Knowing the true Boolean network model, one can test the entire modeling procedures that we have developed, before applying the program to real gene expression data.

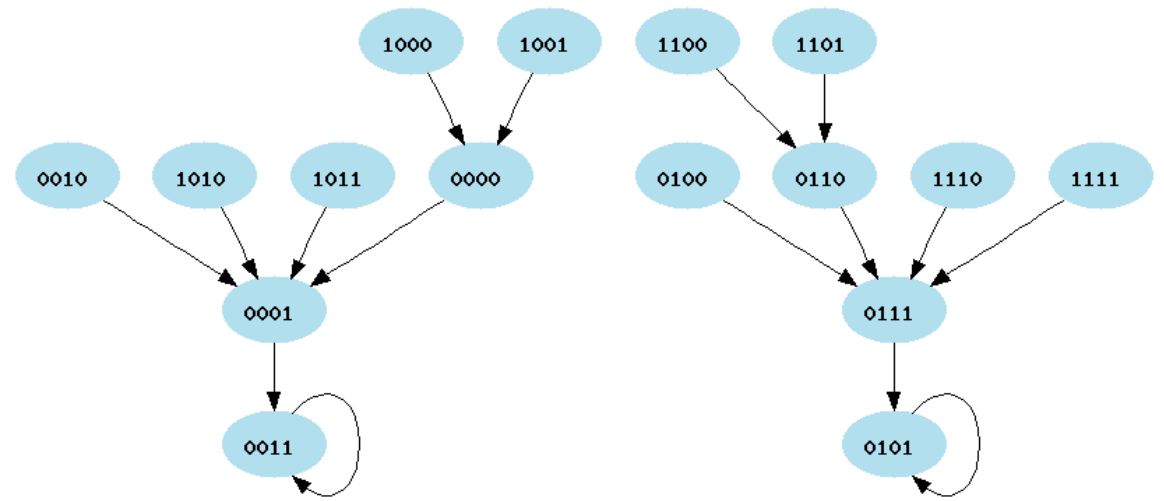



**Fig. 2** A four-node network generated by the Random Boolean Network Generator as illustrated by the graphics program GraphViz

## B. Gene Expression Data

The Boolean network model is built upon temporal gene expression data collected from patients. A gene encodes specific genetic information of an individual. In order for the genetic information to be functional, this gene must be transcribed to messenger RNA and then translated to protein. This information extraction process is known as gene expression. Gene expression is controlled by complex mechanisms of regulation at molecular level that can be considered gene regulatory networks. However, these networks are not typically observable except that the gene expressions can be collected with the micro-array technology widely available today. In our project, such expression data is simulated from a randomly generated Boolean network. The temporal gene expression data constitute the trajectories of all the nodes in a gene regulatory network. The data is stored as tables where the columns represent genes and the rows are the time points. In this table, 0 indicates a gene is expressed at a very low level and 1 indicates the gene is expressed at a high level.

## C. Boolean Network Simulation

This module simulates gene expression data by generating trajectories from a given Boolean network model. The input is a Boolean network model, an initial state, and a length of the trajectory. In return, the output projects a trajectory of the required length starting at the given initial state of the network.

Simulation of trajectories is achieved through evaluation of the truth tables for each individual node in the Boolean network. The value of each node is a Boolean function of its parent value(s). The values are calculated (shown through truth tables) and can be translated into trajectories depicting the values of the child nodes as a function of time [t].

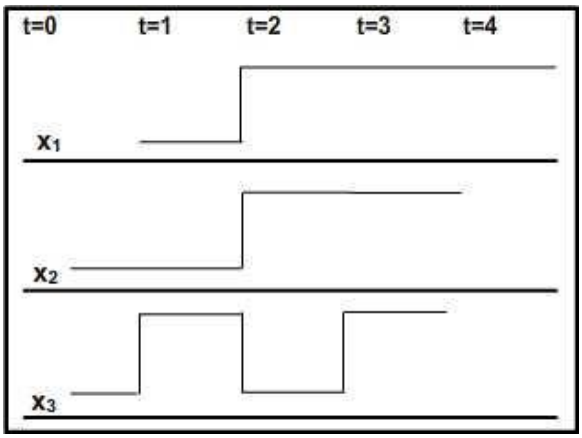



**Fig. 3** The trajectories charted out for the separate nodes according to Table 1.

If the entire set of nodes in a network has the values determined by the calculated truth tables, then the trajectory of each node at every given time interval may be charted separately. The two main functions of the Simulation method are to provide an easy method for checking the validity of the results generated by the Inference method, and to facilitate the functionality of the Intervention method.

## C.1. Monte Carlo Algorithm

The Monte Carlo Algorithm, a reference to the Monaco Casino, is a stochastic technique. In other words, the method is based upon educated trial formulas. Using random numbers and probability statistics, the Monte Carlo Method is used to simulate mathematical systems where unpredictability is essential for accurate results. The algorithm is used in a wide variety of applications including computational sciences, casino development, and video games design because the randomness ensures more accurate results through statistical hypothesis testing.

The Monte Carlo Algorithm is incorporated into the SIAN program to insure a more realistic network, especially when simulating a Boolean network, where genetic mutations are a very real possibility. The added randomness transforms the original version of SIAN from a program implementing deterministic processes to a program employing probabilistic processes. The addition of this algorithm creates “noise” when running the Simulation procedure to better represent a real life situation.

## D. Boolean Network Inference

The goal of the Inference procedure for the program is to find the best truth tables for every node in the network from a given trajectory. The input of this module is one or more trajectories and the output is a Boolean network model.

The Inference method applies the trajectories given by the Random Boolean Network Generation and Simulation, or those taken from real world data, and determines the best parent values for each specific node in a given genetic network.

In order to find the values of the parent nodes, it should be understood that $\{x_1, x_2, x_3 \ldots\}$ at [t+1] are determined by their parent cell(s) at time [t].

For example, if $\{x_2, x_3\}$ act as the parent values for $x_1$, then $x_1$ at [t+1] is determined by $\{x_2,x_3\}$ at [t], and so forth for times [t+2], [t+3], etc. In order to find the correct parent values, the node at [t+1] must generate the values that prove that the combinations of the parent cell(s) are always true in deterministic processes.

In order to accomplish the inverse process to Simulation, the program conjures a transition table. Each transition table illustrates one set of parent value combinations with corresponding counts of either true of false values.

This enumeration consists of binary digits of 1's and 0's that depict the parent values. The 1's represent T's, or True values, and 0's, F's, or False values. To determine that the parent cell(s) are correct, every combination of the parents must always yield the same value for the child node.

Following the previous example in Table 1., if $\{x_2, x_3\}$ is the ordered pair {0,0}, then there must be no counts of 1's enumerated in the transition table. The same goes for the other three possible combinations of $\{x_2, x_3\}$. It is important to note that if there is not a zero count of one certain value for a certain parent combination, then there is no

possibility that the cell value(s) are the parent(s) of the particular node, when using the deterministic model.

The number of possible parent combinations for a single node in a genetic network is given by $2^n$ , where "n" equals the total number of nodes in the network. Since there may be multiple parent combinations, the "best fit" parent combination is the combination with the fewest amounts of parent cells.

If the values for a node at {t, t+1, t+2…} are constant, then the node can be considered as having no parents.

## E. Boolean Network Intervention

This module searches for the most effective way to change a gene node in a Boolean network so as to achieve a desirable state of the system. The input includes a Boolean network model, a current state, and a goal state. The output presents gene perturbations to get to the goal state.

The Intervention procedure for SIAN is defined and founded on a network generated by the Inference procedure that determines the smallest genetic change required to arrive at the target state from the beginning state within a given time period.

As the third and final step of the project that excludes the graphical presentation, the Intervention procedure results from the fact that any given gene expression can only yield one other distinct gene expression at the next time interval, unless an unlikely mutation occurs under the Monte Carlo modeling.

Since the values in the set $\{x_1, x_2, x_3 \ldots\}$ at [t+1] are determined by the parent cell(s) at [t], any distinct state at [t] will always yield a distinct state at [t+1].

In order to determine the optimal modification to the genetic network at its current (faulty) state, each condition that has been altered in one place is checked until the progression:

a.) returns to the original value
b.) returns to a value previously encountered
c.) encounters the desired state, in which, it is a success
d.) has reached the maximum number of steps.

If none of the single gene modifications achieve the goal state, the modifications progress to double gene modifications, and so forth, until the desired result is achieved. If multiple modifications of the same number of genes produce the same desired result, the

modification which produces the desired result in the fewest stages is considered preferable.

Since the number of possible states for a single node of "n" genes in a gene network is $2^n$, a value that increases to extremely large values for moderate values of "n", it is vital that the process of determining a favorable gene therapy procedure take as little time as possible. Consequently, utilizing supercomputing capabilities makes the determination of a preferable therapy much less time intensive.

## F. Visualization of Uncontrolled Cell Growth

In order to simulate the division and death of cells, an animation program (Macromedia Flash 8 Professional) is used to create and depict the activities of cells in a certain system.

The animation effectively displays the anomalies produced by either the overgrowth of cells or the rapid rate of death of cells before cell division. The video shows the consequences of irregular cell activities and shows the causes of such abnormalities in human cells. These frames are all controlled by the Simulation/Inference procedures.

In addition, the graphics program will show that changes could be made to reduce or eliminate the abnormalities in cell functions and gene networks. In order for this to occur, the graphics design will convert information given from the interference procedure to data that can show changes in cell division and the rate of death over time.

Each frame is tantamount to 1/12 of a second.

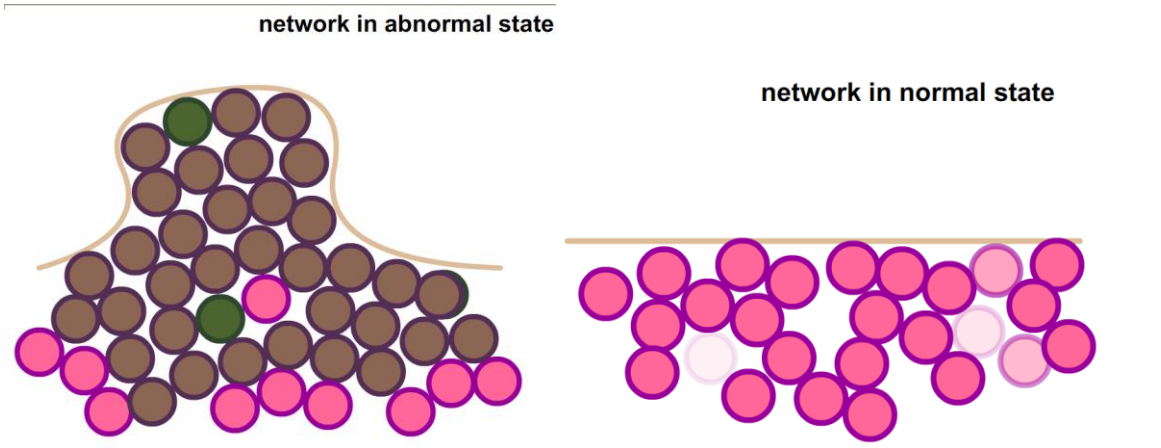
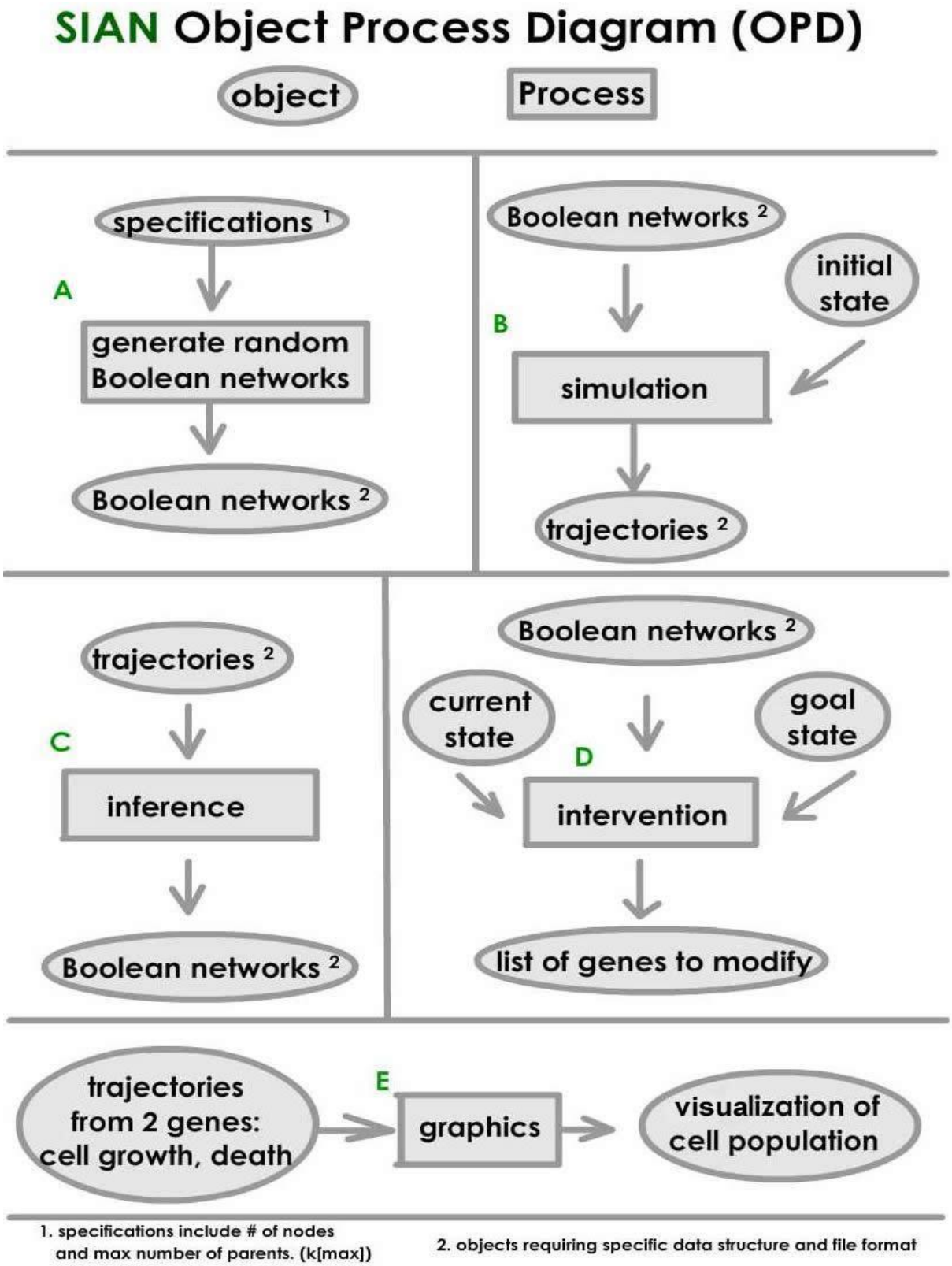



**Fig. 4** Screenshots taken from the Flash animation of a particular skin tumor juxtaposing a treated abnormal genetic network in a before and after state.

In addition to an animated visualization, we have implemented the program GraphViz to produce illustrations of a network. This illustration shows the regulation of all genes within a network and how one node relates to another. The color of individual genes, font, may all be selected and edited by the user.

# Algorithms Summary Flowchart – Chart. 1



# Originality (Higher Order Boolean Networks)

Although nodes represent genes and the connections among them in gene regulatory networks, networks having long term memory cannot be approximated well with a $1^{st}$ order Boolean network. Higher order Boolean networks are employed in SIAN by using recurring representations of nodes in order to realize more life-like regulations among nodes. In other words, higher order Boolean networks can track a single node at various time intervals as opposed to just one. For example, instead of node "x" being a function of its parent(s) at the definite time of [t-1], its parents may be tracked at different periods such as [t+1], [t], or [t-2] to expand to all possibilities in analyzing a Boolean network. Furthermore, higher order levels of Boolean networks act as boundaries for SIAN in order to produce feasible networks

$K^{th}$ -order Boolean networks for modeling gene regulatory networks are first implemented in this project as far as we know from published literature on Boolean networks. In the past, only $1^{st}$-order Boolean networks have been used, mainly due to lack of computational power. SIAN uses higher order Boolean networks to allow a much more complex model to be derived for gene interactions, including time-delayed effects as well as instantaneous ones.

# Results

The results of the program can be interpreted in correspondence to the four major functions of the program: Random Network Generation, Simulation, Inference, and Intervention.

After running the Random Network Generation procedure, the program successfully compiled a sample 5-node network with accepted node values to determine the state of a network.

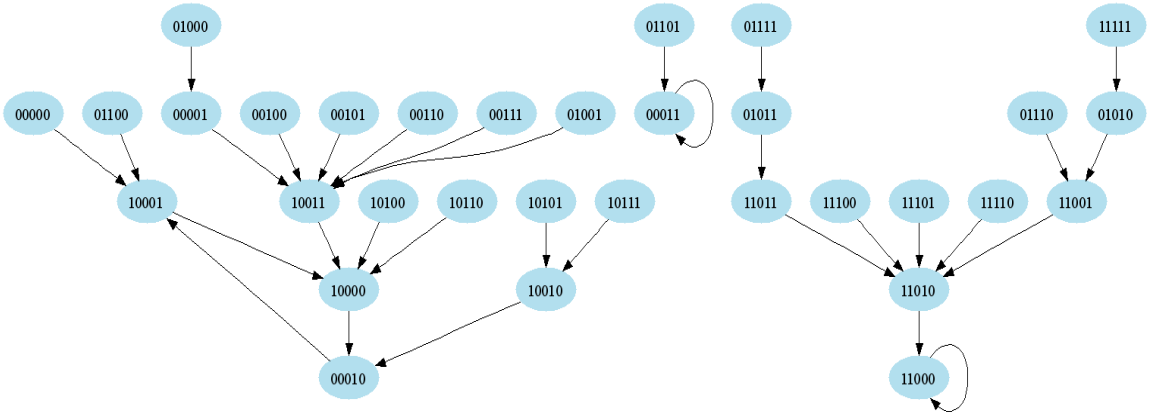



**Fig. 5** A randomly generated 5-node network as illustrated by GraphViz. The maximum number of parents was set to five.

The Boolean network was then translated into a trajectory format. This format, when executed by the Inference method, reversely engineered the trajectory into the original network. When the outputs were compared, both Boolean networks were identical which proved the procedures were accurate.

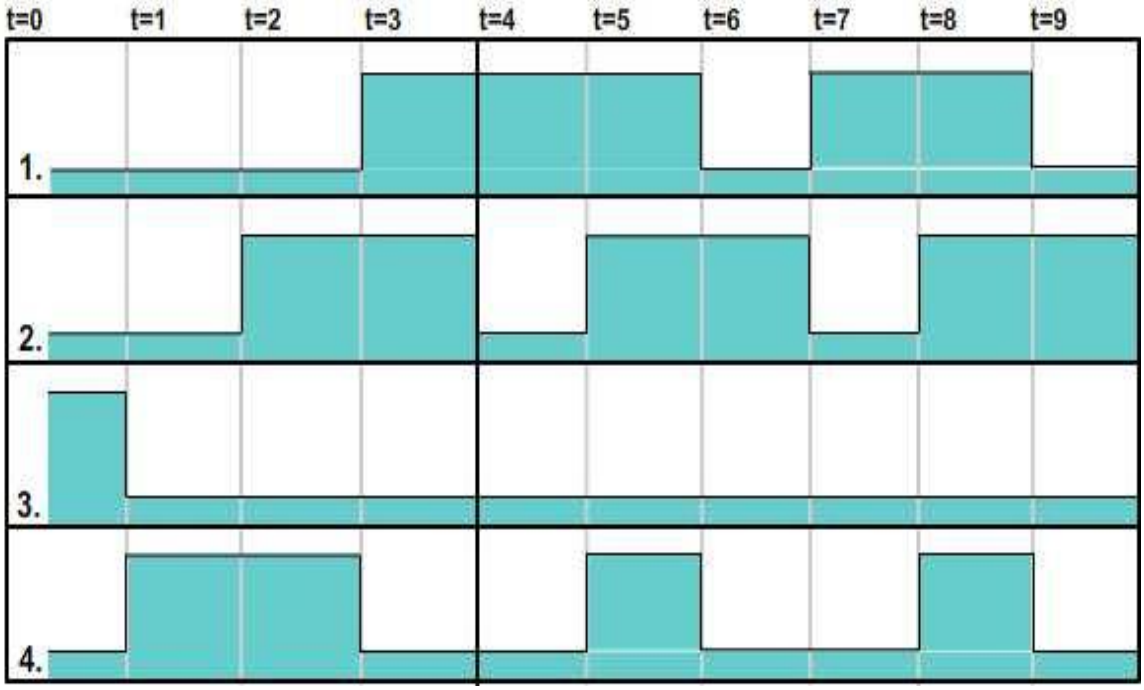



**Fig. 6** Different nodes being tracked through the Simulation process as trajectories. The lower stages represent a 0 value and the higher stage depicts a node obtaining a 1 value.

Finally, when the abnormal Boolean network was generated, it was inputted into the Intervention function of the program. The five-node network was intervened, and the program concluded that a one step adjustment could be made to the program to alter the value of a node and as a result, completely relieve the network of its abnormal node so the network would function properly.

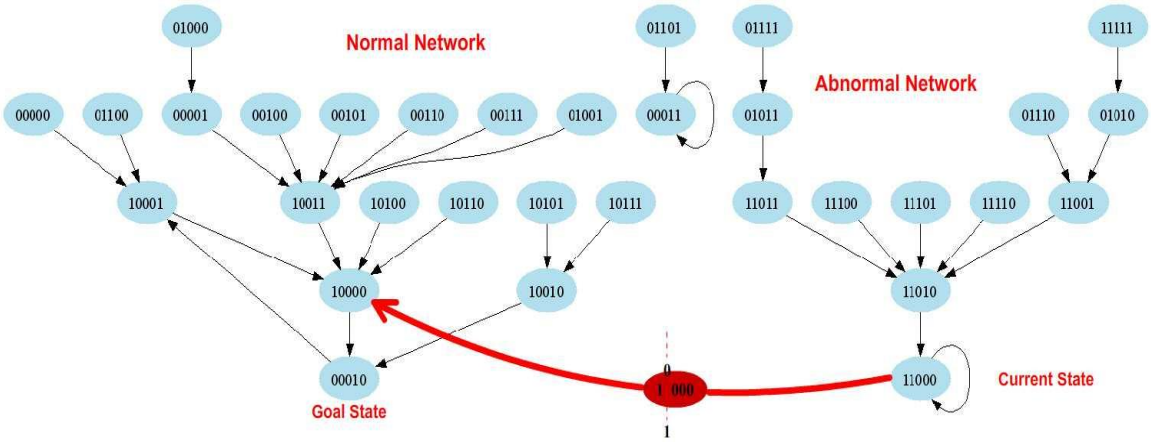



**Fig. 7** The 5-node network in **Fig.5** after the Intervention procedure. The change in the second node of the current state results in a one-step process from reaching the network's goal state.

Networks with six and seven nodes were experimented with as well, and although the program took longer to run, the results of the Intervention process also provided only few changes that would provide a patient with genetic diseases to suppress the abnormal gene in a short amount of time.

Recently, a one-hundred node network was randomly generated and analyzed through the SIAN procedures. Although the network took nearly a day to finish processing, the results were as accurate as the networks containing four, five, and six nodes. This more realistic network proves that real life genetic networks with a large number of nodes can be modeled and investigated with impeccable results.

# Conclusions

Boolean network modeling is an accurate form of representing genetic networks of genetic diseases. Boolean network models help put these complex networks of the twenty thousand genes estimated in the human genome into perspective. Because the results of the first edition of SIAN was discrete and deterministic, we decided to integrate the Monte Carlo Algorithm to create a probabilistic program and add higher order networks to produce increasingly realistic results and analysis. In addition to accurately modeling a genetic network in cancer, the newly developed program (SIAN) is capable of "reverse engineering" the trajectory from a genetic network to yield the original network. It can also determine the most efficient genetic change to bring a network from an abnormal state to a normal state. This economic change can suppress the genes responsible for causing genetic diseases and cancer. The correctness of the code is confirmed through simulation. SIAN not only helps provide efficient, effective therapeutic treatments to cancer patients, but also to patients with other genetic diseases as well. The software enables physicians and gene therapists to pinpoint the cause of the abnormal genetic activities and consequently to design an effective gene therapy procedure for patients with genetic abnormalities.

Project SIAN is subject to further development, including a projected version in the C++ language.

# References

[1]

Lahdesmaki, H., Shmulevich, I., and Yli-Harja, O. "On Learning Gene Regulatory Networks Under the Boolean Network Model." Machine Learning, 52 (2003):147-167.

[2]

Liang, S., S. Fuhrman and R. Somogyi. “REVEAL, a general reverse engineering algorithm for inference of genetic network architecture.” *Pacific Symposium on Biocomputing* 3 (1998): 18–29.

[3]

Pal, R., I. Ivanov, A. Datta, M.L. Bittner and E.R. Dougherty. “Generating Boolean networks with a prescribed attractor structure.” *Bioinformatics* 21 (2005): 4021–4025.

[4]

Shmulevich, I., E.R. Dougherty, S. Kim and W. Zhang. “Probabilistic Boolean networks: rule-based uncertainty model for gene regulatory networks.” *Bioinformatics* 18 *(2002):* 261–274.

[5]

Shmulevich, I., Dougherty, E., and Zhang, W. "From Boolean to Probabilistic Boolean Networks as Models of Genetic Regulatory Networks." IEEE, 90 (2002): 1778-1792.

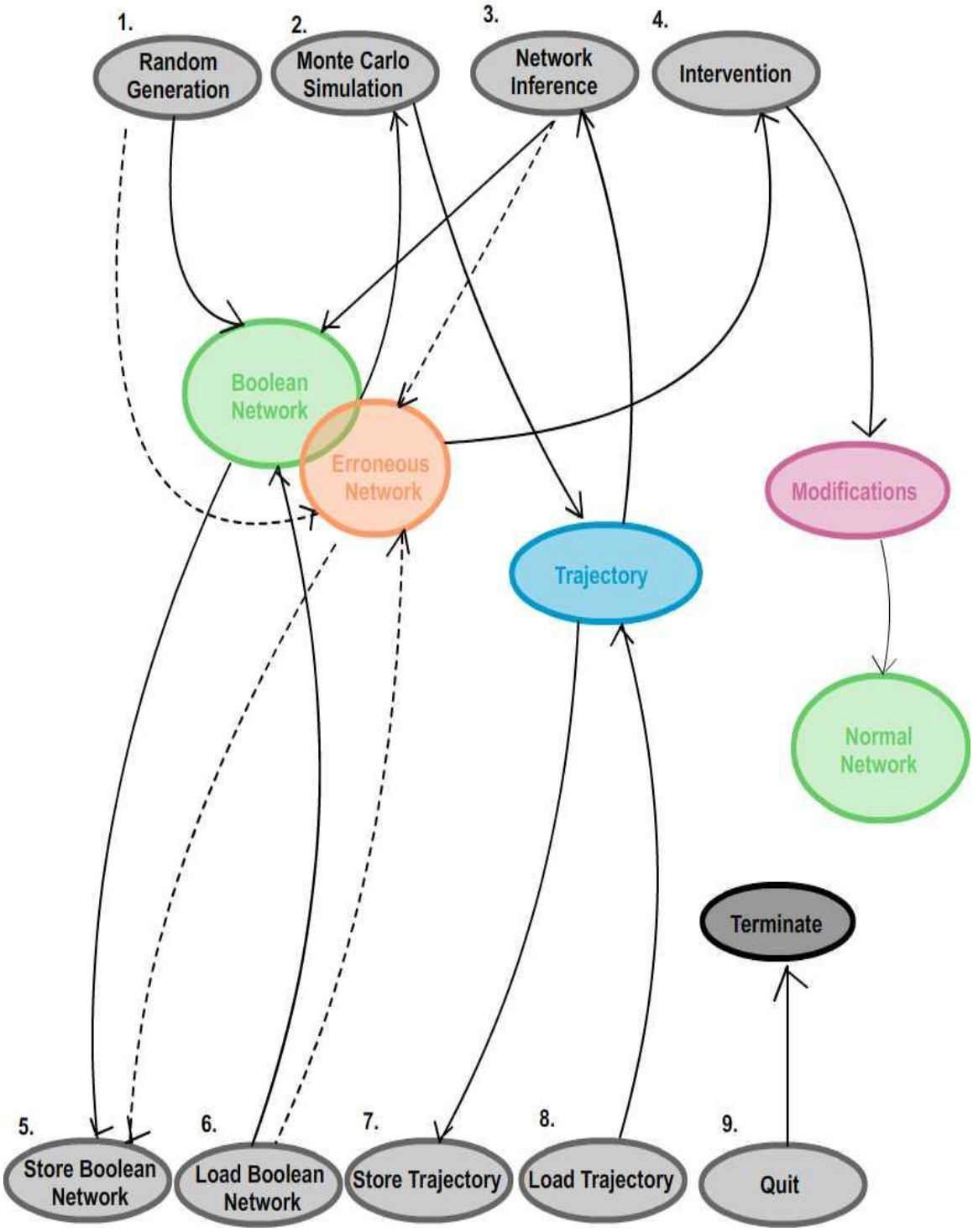
# Acknowledgements & Special Thanks

First, we would like to thank Dr. Joe Song of New Mexico State University, without whom, none of this would be possible. He gave us inspiration and provided valuable ideas and sources that assisted us in our research and programming that has greatly helped us through the various stages of the project. His guidance as been invaluable. In addition we would like to thank Mr. Greg Marez for his unrelenting positive attitude and his sense of "coolness." His humor propelled us through tough, bumpy roads throughout stages of the project. Finally, we would like to thank our parents for constantly providing encouragement and support through the entire research and programming processes; without their support, we may have never gone so far…

# Glossary

| | |
|---|---|
| Boolean network: | Network of nodes with a binary relationship. |
| Gene regulatory network: | The interactions among genes in a cell which govern the rate of replication, transcription, and translation. |
| Node: | Representative of a “gene” in a gene regulatory network. |
| States (for a node): | An “on” or “off” condition which determines whether a node is being expressed or suppressed, respectively. |
| States (for input/output): | An “on” or “off” condition determining the object’s presence. |
| Truth table: | Table representing the binary values of nodes in a network in relation to a node’s parents. |

# Appendix A. User Process Flowchart – Chart. 2



# Appendix B. Code

## i. Boolean Network

```
import javax.swing.*;
import java.util.*;
import java.io.*;

public class BooleanNetwork {
   static int nodeNum;
   static int maxParents;
   static int iterations;
   static int startingPoints;
   static int[][] trajectory;
   static int[][][] trajectories;
   static double mutation;
   static Node[] nodes;
   static Random generator = new Random();
   public static void main(String[] args) {
      int flag = 0;
      do{
      String input = JOptionPane.showInputDialog("Would you like to \n1)
Generate a new random Boolean Network\n2) Run a simulation on the existing
BN\n3) Run a simulation from multiple starting points\n4) Run inference on
an existing trajectory\n5) Run inference on multiple trajectories\n6)
Intervene in the network\n7) Store the BN to a file\n8) Load a BN from a
file\n9) Store the trajectory to a file\n10) Load a trajectory from a
file\n11) Store the trajectories to a file\n12) Load trajectories from a
file\n13) Do a graphical output simulation\n14) Quit");
         int decision = Integer.parseInt(input);
         switch(decision) {
            case 1: Generator.Generate();;break;
            case 2: Simulator.Simulate();break;
            case 3: Simulator.SimulateStartingPoints();break;
            case 4: Inferer.Infer();break;
            case 5: Inferer.InferStartingPoints();break;
            case 6: Intervener.Intervene();break;
            case 7: UserIO.networkOut();break;
            case 8: UserIO.networkIn();break;
            case 9: UserIO.trajectoryOut();break;
            case 10: UserIO.trajectoryIn();break;
            case 11: UserIO.trajectoriesOut();break;
            case 12: UserIO.trajectoriesIn();break;
            case 13: Visualization.Visualize();break;
            case 14: flag = 1;break;
```

```
                default: JOptionPane.showMessageDialog (null,"You didn't
enter a valid number...please re-run the program and start again.  Thank
you");break;
            }
        }while(flag == 0);
    }
}
```

## ii. Generator

```
import javax.swing.*;
import java.util.*;
import java.io.*;

public class Generator extends BooleanNetwork{
   public static void Generate(){
      String nodeNumTemp;
      nodeNumTemp = JOptionPane.showInputDialog("How many nodes does the
Boolean Network have?");
      nodeNum = Integer.parseInt(nodeNumTemp);
      nodes = new Node[nodeNum];
      for(int i=0; i<nodeNum; i++){
         nodes[i] = new Node();
      }
      String parentsTemp;
      parentsTemp = JOptionPane.showInputDialog("How many parents can a
node have?");
      maxParents = Integer.parseInt(parentsTemp);
      int[] numParents = new int[nodeNum];
      for(int i=0; i<nodeNum; i++){
         numParents[i] = generator.nextInt(maxParents+1);
         System.out.println("i = "+i+", i has "+numParents[i]+"
parents");
         nodes[i].setValues(numParents[i]);
         int[] value = new int[numParents[i]];
         int flag;
         for(int j=0; j<numParents[i]; j++){
            flag = 0;
            value[j] = generator.nextInt(nodeNum);
            System.out.println("i = "+i+", j = "+j+", value[j] =
"+value[j]);
//************************************************//
// Ensures that no node is set as a parent twice //
//************************************************//
            if(j>0){
               for(int k=0; k<j; k++){
                  if(value[k] == value[j]){
                     flag = 1;
                  }
               }
            }
            if(flag != 0){
               j--;    // Redoes this parent if it was already present
            }
```

```
            }
            nodes[i].setParents(value);
            for(int k=0;k<(int)Math.pow(2,numParents[i]);k++){
               int truthVal = generator.nextInt(2);
               nodes[i].truth[k] = truthVal;
            }
            nodes[i].Val = generator.nextInt(2);
         }
      }
   }
```

## iii. Simulator

```
import javax.swing.*;
import java.util.*;
import java.io.*;

public class Simulator extends BooleanNetwork{
   public static void Simulate(){
      String getMutation = JOptionPane.showInputDialog("What is the
probability of a mutation occuring (out of 1)?");
      mutation = Double.parseDouble(getMutation);
      String getIterations = JOptionPane.showInputDialog("How many
iterations of the simulation would you like to run?");
      iterations = Integer.parseInt (getIterations);
      String start = JOptionPane.showInputDialog("What state would you
like to start this simulation from?\nThere are "+nodeNum+" nodes in the
network.");
      // Two-dimensional array to store the result of the simulations
      trajectory = new int[nodeNum][iterations+1];
      for(int i=0; i<nodeNum; i++){
         trajectory[i][0] = nodes[i].Val =
Integer.parseInt(start.substring(i,i+1));
      }
      for(int i=0; i<nodeNum; i++){
         trajectory[i][0] = nodes[i].Val;
      }
      for(int i=1; i<=iterations; i++){
         for(int j=0; j<nodeNum; j++){
            trajectory[j][i] = nodes[j].Value();
            if(generator.nextDouble()<mutation){
               if(trajectory[j][i]==0){
                  trajectory[j][i]=1;
               }
               else{
                  trajectory[j][i]=0;
               }
            }
         }
         for(int j=0; j<nodeNum; j++){
            nodes[j].Val = trajectory[j][i];
         }
      }
   }
   public static void SimulateNoInput(){
      // Two-dimensional array to store the result of the simulations
      trajectory = new int[nodeNum][iterations+1];
```

```
        for(int i=0; i<nodeNum; i++){
            trajectory[i][0] = nodes[i].Val;
        }
        for(int i=1; i<=iterations; i++){
            for(int j=0; j<nodeNum; j++){
                trajectory[j][i] = nodes[j].Value();
            }
            for(int j=0; j<nodeNum; j++){
                nodes[j].Val = trajectory[j][i];
            }
        }
    }
    public static void SimulateStartingPoints(){
        String getMutation = JOptionPane.showInputDialog("What is the
probability of a mutation occuring (out of 1)?");
        mutation = Double.parseDouble(getMutation);
        String getIterations = JOptionPane.showInputDialog("How many
iterations of the simulation would you like to run?");
        iterations = Integer.parseInt (getIterations);
        String getStartingPoints = JOptionPane.showInputDialog("How many
different trajectories do you want?");
        startingPoints = Integer.parseInt(getStartingPoints);
        trajectories = new int[nodeNum][iterations+1][startingPoints];
        for(int x=0; x<startingPoints; x++){
            String start = JOptionPane.showInputDialog("What state would you
like to start this simulation from?\nThere are "+nodeNum+" nodes in the
network.");
            for(int i=0; i<nodeNum; i++){
                trajectories[i][0][x] = nodes[i].Val =
Integer.parseInt(start.substring(i,i+1));
            }
            for(int i=1; i<=iterations; i++){
                for(int j=0; j<nodeNum; j++){
                    trajectories[j][i][x] = nodes[j].Value();
                    if(generator.nextDouble()<mutation){
                        if(trajectories[j][i][x]==0){
                            trajectories[j][i][x]=1;
                        }
                        else{
                            trajectories[j][i][x]=0;
                        }
                    }
                }
                for(int j=0; j<nodeNum; j++){
                    nodes[j].Val = trajectories[j][i][x];
                }
```

```
            }
        }
    }
    public static void SimulateStartingPointsNoInput(){
//        String getMutation = JOptionPane.showInputDialog("What is the
probability of a mutation occuring (out of 1)?");
//        mutation = Double.parseDouble(getMutation);
//        String getIterations = JOptionPane.showInputDialog("How many
iterations of the simulation would you like to run?");
        iterations = 1;
        startingPoints = (int)Math.pow(2.,(double)nodeNum);
        trajectories = new int[nodeNum][iterations+1][startingPoints];
        for(int x=0; x<startingPoints; x++){
            String start = "";
            int temp = x;
            char[] tempChar = new char[nodeNum];
            for(int i=0; i<nodeNum; i++){
                if(temp%2==0){
                    start+="0";
                    temp/=2;
                }
                else{
                    start+="1";
                    temp/=2;
                }
            }
            tempChar = start.toCharArray();
            start = "";
            for(int i=nodeNum-1; i>=0; i--){
                start+=tempChar[i];
            }
            for(int i=0; i<nodeNum; i++){
                trajectories[i][0][x] = nodes[i].Val =
Integer.parseInt(start.substring(i,i+1));
            }
            for(int i=1; i<=iterations; i++){
                for(int j=0; j<nodeNum; j++){
                    trajectories[j][i][x] = nodes[j].Value();
                    if(generator.nextDouble()<mutation){
                        if(trajectories[j][i][x]==0){
                            trajectories[j][i][x]=1;
                        }
                        else{
                            trajectories[j][i][x]=0;
                        }
                    }
```

```
            }
            for(int j=0; j<nodeNum; j++){
               nodes[j].Val = trajectories[j][i][x];
            }
         }
      }
   }
}
```

## iv. Inferer

```
import javax.swing.*;
import java.util.*;
import java.io.*;

public class Inferer extends BooleanNetwork {
   static int errors, parents, total;
   static double weightedErrors;
   public static void Infer() {
      String getParents = JOptionPane.showInputDialog("What is the maximum
number of parents a node can have?");
      parents = Integer.parseInt(getParents);
      for(int node=0; node<nodeNum; node++){
         nodes[node].Val = trajectory[node][0];
         int[] bestParents = new int[parents];
         int one=0, zero=0;
         nodes[node].setValues(0);
         /*****************************************/
         /* First does the inference for 0 parents */
         /*****************************************/
         for(int iteration=0; iteration<=iterations; iteration++){
            if(trajectory[node][iteration]==0){
               zero++;
            }
            else{
               one++;
            }
         }
         if(one>zero){
            errors = zero;
            nodes[node].truth[0]=1;
         }
         else{
            errors = one;
            nodes[node].truth[0]=0;
         }
         total = zero + one;
         weightedErrors = (total - errors)/(double)total;
         /*****************************************/
         /*          Now do 1-N parents           */
         /*****************************************/
         for(int parentNum=1; parentNum<=parents; parentNum++){
            errors = 0;
            total = 0;
            double errorWeighted;
```

```
            int counter = 0;
            int flag = 0;                                   //
Flag for when the first parent
            int[] currentParents = new int[parentNum];          //
Current parents
            int[] maxParentValues = new int[parentNum];
            int[] ones = new int[(int)Math.pow(2, parentNum)];
            int[] zeros = new int[(int)Math.pow(2, parentNum)];
            int[] values = new int[(int)Math.pow(2, parentNum)];
            for(int i=0; i<parentNum; i++){
               currentParents[i]=i;
               maxParentValues[i] = nodes.length - parentNum + i;
            }
            do{

/*************************************************************************
*/
               /* Picks a truth table and determines the error for the
current parents */

/*************************************************************************
*/
               for(int iteration=0; iteration<iterations; iteration++){
                  for(int parent=0; parent<parentNum; parent++){

if(trajectory[currentParents[parent]][iteration]==1){
                        counter+=(int)Math.pow(2,parent);
                     }
                  }
                  if(trajectory[node][iteration+1]==1){
                     ones[counter]++;
                  }else{
                     zeros[counter]++;
                  }
                  counter = 0;
               }
               for(int table=0; table<Math.pow(2, parentNum); table++){
                  if(ones[table]>zeros[table]){
                     values[table]=1;
                     errors += zeros[table];
                  }else{
                     values[table]=0;
                     errors += ones[table];
                  }
                  total += zeros[table] + ones[table];
               }
```

```
                errorWeighted = (total - errors)/(double)total;
                errorWeighted = errorWeighted/(parentNum + 1);  //
Change name, it's success not error
                if(errorWeighted>weightedErrors){
                   weightedErrors = errorWeighted;
                   nodes[node].setValues(parentNum);
                   nodes[node].setParents(currentParents);
                   for(int i=0; i<Math.pow(2, parentNum); i++){
                      nodes[node].truth[i] = values[i];
                   }
                }
                /*********************************************/
                /*         Resets counter values          */
                /*********************************************/
                total = 0;
                errors = 0;
                for(int i=0; i<Math.pow(2, parentNum); i++){
                   ones[i] = 0;
                   zeros[i] = 0;
                }
                /*********************************************/
                /*              Enumeration               */
                /*********************************************/
                currentParents[parentNum-1]++;
                for(int overflow = 1; overflow<=parentNum; overflow++){

if(currentParents[parentNum-overflow]>maxParentValues[parentNum-overflow]){
                      if(overflow!=parentNum){
                         currentParents[parentNum-overflow-1]++;

currentParents[parentNum-overflow]=currentParents[parentNum-overflow-1]+1;
                         if(overflow!=1){
                            for(int cycles = overflow-1; cycles > 0;
cycles--){

if(currentParents[parentNum-cycles]>maxParentValues[parentNum-cycles]){
                                     currentParents[parentNum-cycles]
= currentParents[parentNum-cycles-1]+1;
                                  }
                               }
                            }
                         }
                         else{
                            flag = 1;
                         }
                      }
```

```
            }
          }while(flag != 1);
        }
      }
    }
    public static void InferStartingPoints() {
      String getParents = JOptionPane.showInputDialog("What is the maximum
number of parents a node can have?");
      parents = Integer.parseInt(getParents);
      for(int node=0; node<nodeNum; node++){
        nodes[node].Val = trajectories[node][0][0];
        int[] bestParents = new int[parents];
        int one=0, zero=0;
        nodes[node].setValues(0);
        /*****************************************/
        /* First does the inference for 0 parents */
        /*****************************************/
        for(int start=0; start<startingPoints; start++){
          for(int iteration=0; iteration<=iterations; iteration++){
            if(trajectories[node][iteration][start]==0){
              zero++;
            }
            else{
              one++;
            }
          }
        }
        if(one>zero){
          errors = zero;
          nodes[node].truth[0]=1;
        }
        else{
          errors = one;
          nodes[node].truth[0]=0;
        }
        total = zero + one;
        weightedErrors = (total - errors)/(double)total;
        /*****************************************/
        /*          Now do 1-N parents           */
        /*****************************************/
        for(int parentNum=1; parentNum<=parents; parentNum++){
          errors = 0;
          total = 0;
          double errorWeighted;
          int counter = 0;
          int flag = 0;                                  //
```

```
Flag for when the first parent
            int[] currentParents = new int[parentNum];            //
Current parents
            int[] maxParentValues = new int[parentNum];
            int[] ones = new int[(int)Math.pow(2, parentNum)];
            int[] zeros = new int[(int)Math.pow(2, parentNum)];
            int[] values = new int[(int)Math.pow(2, parentNum)];
            for(int i=0; i<parentNum; i++){
               currentParents[i]=i;
               maxParentValues[i] = nodes.length - parentNum + i;
            }
            do{

/**************************************************************************
*/
               /* Picks a truth table and determines the error for the
current parents */

/**************************************************************************
*/
               for(int start=0; start<startingPoints; start++){
                  for(int iteration=0; iteration<iterations;
iteration++){
                     for(int parent=0; parent<parentNum; parent++){

if(trajectories[currentParents[parent]][iteration][start]==1){
                              counter+=(int)Math.pow(2,parent);
                           }
                     }
                     if(trajectories[node][iteration+1][start]==1){
                        ones[counter]++;
                     }else{
                        zeros[counter]++;
                     }
                     counter = 0;
                  }
               }
               for(int table=0; table<Math.pow(2, parentNum); table++){
                  if(ones[table]>zeros[table]){
                     values[table]=1;
                     errors += zeros[table];
                  }else{
                     values[table]=0;
                     errors += ones[table];
                  }
                  total += zeros[table] + ones[table];
```

```
            }
            errorWeighted = (total - errors)/(double)total;
            errorWeighted = errorWeighted/(parentNum + 1);  //
Change name, it's success not error
            if(errorWeighted>weightedErrors){
               weightedErrors = errorWeighted;
               nodes[node].setValues(parentNum);
               nodes[node].setParents(currentParents);
               for(int i=0; i<Math.pow(2, parentNum); i++){
                  nodes[node].truth[i] = values[i];
               }
            }
            /*******************************************/
            /*         Resets counter values           */
            /*******************************************/
            total = 0;
            errors = 0;
            for(int i=0; i<Math.pow(2, parentNum); i++){
               ones[i] = 0;
               zeros[i] = 0;
            }
            /*******************************************/
            /*              Enumeration                */
            /*******************************************/
            currentParents[parentNum-1]++;
            for(int overflow = 1; overflow<=parentNum; overflow++){

if(currentParents[parentNum-overflow]>maxParentValues[parentNum-overflow]){
                  if(overflow!=parentNum){
                     currentParents[parentNum-overflow-1]++;

currentParents[parentNum-overflow]=currentParents[parentNum-overflow-1]+1;
                     if(overflow!=1){
                        for(int cycles = overflow-1; cycles > 0;
cycles--){

if(currentParents[parentNum-cycles]>maxParentValues[parentNum-cycles]){
                                 currentParents[parentNum-cycles]
= currentParents[parentNum-cycles-1]+1;
                              }
                           }
                        }
                     }
                     else{
                        flag = 1;
                     }
```

```
                    }
                }
            }while(flag != 1);
        }
    }
}

}
```

## v. Intervener

```
import javax.swing.*;
import java.util.*;
import java.io.*;

public class Intervener extends BooleanNetwork{
   public static void Intervene(){
      int flag = 1;
      int flag2 = 0;
      int counter = 0;
      String getIterations;
      String getGoal;
      String grammarOutput;
      String changeOutput;
      int Iterations;
      getGoal = JOptionPane.showInputDialog("There are "+nodeNum+" nodes.
Please enter the goal state for the network:");
      int[] goalState = new int[nodeNum];
      System.out.print("The goal state is: ");
      for(int i=0; i<nodeNum; i++){
         goalState[i] = Integer.parseInt(getGoal.substring(i,i+1));
         System.out.print(goalState[i]);
      }
      System.out.println();
      getIterations = JOptionPane.showInputDialog("How many iterations can
take place?");
      iterations = Iterations = Integer.parseInt(getIterations);
      UserIO.networkOut("Intervener.txt");
      Simulator.SimulateNoInput();
      for(int i=0; i<=Iterations; i++){
         for(int j=0; j<nodeNum; j++){
            if(trajectory[j][i] != goalState[j]){
               flag2 = 1;
            }
         }
         if(flag2 != 1){
            flag = 0;
            counter = i;
            i = Iterations + 1;
         }
         flag2 = 0;
      }
      if(flag == 0){
         grammarOutput = " iteration";
         if(counter == 1){
```

```
            grammarOutput+=".";
         }
         else{
            grammarOutput+="s.";
         }
         JOptionPane.showMessageDialog(null, "No Intervention was
necessary.  The goal state was reached in "+counter+grammarOutput);
      }
      else{
         UserIO.networkIn("Intervener.txt");
         for(int changeNum=1; changeNum<=nodeNum; changeNum++){
            int[] currentChanges = new int[changeNum];
            int[] maxChangeValues = new int[changeNum];
            int flag_e = 0;
            for(int i=0; i<changeNum; i++){
               currentChanges[i]=i;
               maxChangeValues[i] = nodes.length - changeNum + i;
            }
            do{
               /*******************************************/
               /*     Tries a node state modification     */
               /*******************************************/
               for(int change = 0; change<changeNum; change++){
                  if(nodes[currentChanges[change]].Val == 0){
                     nodes[currentChanges[change]].Val = 1;
                  }
                  else{
                     nodes[currentChanges[change]].Val = 0;
                  }
               }
               Simulator.SimulateNoInput();
               for(int i=0; i<=Iterations; i++){
                  for(int j=0; j<nodeNum; j++){
                     if(trajectory[j][i] != goalState[j]){
                        flag2 = 1;
                     }
                  }
                  if(flag2 != 1){
                     flag = 0;
                     counter = i;
                     i = Iterations + 1;
                  }
                  flag2 = 0;
               }
               if(flag == 0){
                  grammarOutput = " iteration";
```

```
                if(counter == 1){
                   grammarOutput+=".";
                }
                else{
                   grammarOutput+="s.";
                }
                changeOutput = "Node";
                if(changeNum==1){
                   changeOutput+=" ";
                   changeOutput+=currentChanges[0];
                }
                else if(changeNum==2){
                   changeOutput+="s ";
                   changeOutput+=currentChanges[0];
                   changeOutput+=" and ";
                   changeOutput+=currentChanges[0];
                }
                else{
                   changeOutput+="s ";
                   for(int x=0; x<changeNum; x++){
                      changeOutput+=currentChanges[x];
                      if(x!=changeNum-1){
                         changeOutput+=", ";
                      }
                      if(x==changeNum-2){
                         changeOutput+="and ";
                      }
                   }
                }
                changeOutput += " need to be changed.";
                JOptionPane.showMessageDialog(null,
changeOutput+"The goal state was reached in "+counter+grammarOutput);
                flag_e = 1;
             }
             /*********************************************/
             /*         Resets network values          */
             /*********************************************/
             UserIO.networkIn("Intervener.txt");
             /*********************************************/
             /*              Enumeration               */
             /*********************************************/
             currentChanges[changeNum-1]++;
             for(int overflow = 1; overflow<=changeNum; overflow++){

if(currentChanges[changeNum-overflow]>maxChangeValues[changeNum-overflow]){
                if(overflow!=changeNum){
```

```
                        currentChanges[changeNum-overflow-1]++;

currentChanges[changeNum-overflow]=currentChanges[changeNum-overflow-1]+1;
                        if(overflow!=1){
                           for(int cycles = overflow-1; cycles > 0;
cycles--){

if(currentChanges[changeNum-cycles]>maxChangeValues[changeNum-cycles]){
                                 currentChanges[changeNum-cycles]
= currentChanges[changeNum-cycles-1]+1;
                              }
                           }
                        }
                     }
                     else{
                        flag_e = 1;
                     }
                  }
               }
               if(flag == 0){
                  changeNum=nodeNum;
               }
            }while(flag_e != 1);
         }
      }
   }
}
```

## vi. UserIO

```
import javax.swing.*;
import java.util.*;
import java.io.*;

public class UserIO extends BooleanNetwork{
   public static void networkOut(){
      FileOutputStream out; // declare a file output object
      PrintStream p; // declare a print stream object
      String filename = JOptionPane.showInputDialog("Where do you want to
write the boolean network to?");
      try{
         // Create a new file output stream
         // connected to "myfile.txt"
         out = new FileOutputStream(filename);
         // Connect print stream to the output stream
         p = new PrintStream(out);
         // Prints out the number of nodes in the BN
         p.println(nodeNum);
         for(int i=0; i<nodeNum; i++){
//             p.println("This is node "+i+"'s value");  //Useful for
reading the output as a person
            p.println(nodes[i].Val);
//            p.println("This is how many parents node "+i+" has");
//Useful for reading the output as a person
            p.println(nodes[i].parents.length);
//            p.println("These are node "+i+"'s parents");  //Useful for
reading the output as a person
            for(int j=0; j<nodes[i].parents.length; j++){
               p.print(nodes[i].parentNums[j]+" ");
            }
            p.println();
//            p.println("This is the truth table for node "+i);
//Useful for reading the output as a person
            for(int j=0; j<(int)Math.pow(2,nodes[i].parents.length);
j++){
               p.print(nodes[i].truth[j]);
            }
            p.println();
         }
         p.close();
      }
      catch (Exception e){//Catch exception if any
         System.err.println("Error: " + e.getMessage());
      }
```

```
   }
   public static void networkIn(){
      String filename = JOptionPane.showInputDialog("What file do you want
to read a boolean network in from?");
      try{
         // Open the file that is the first
         // command line parameter
         FileInputStream fstream = new FileInputStream(filename);
         // Get the object of DataInputStream
         DataInputStream in = new DataInputStream(fstream);
         BufferedReader br = new BufferedReader(new
InputStreamReader(in));
         int parentValue = 0, parentNumber, flag = 0;
         String strLine;
         strLine = br.readLine();
         nodeNum = Integer.parseInt(strLine);
         nodes = new Node[nodeNum];
         for(int i=0; i<nodeNum; i++){
            nodes[i] = new Node();
         }
         for(int i=0; i<nodeNum; i++){
            nodes[i].Val = Integer.parseInt(br.readLine());
            parentNumber = Integer.parseInt(br.readLine());
            nodes[i].setValues(parentNumber);
            strLine = br.readLine();
            int[] values = new int[parentNumber];
            for(int j=0; j<parentNumber; j++){
               do{
                  if(strLine.substring(0,1).compareTo(" ")==0){
                     strLine = strLine.substring(1);
                     flag = 1;
                  }else{
                     parentValue = parentValue*10 +
Integer.parseInt(strLine.substring(0,1));
                     strLine = strLine.substring(1);
                  }
               }while(flag==0);
               values[j] = parentValue;
               parentValue = flag = 0;
            }
            nodes[i].setParents(values);
            strLine = br.readLine();
            for(int k=0; k<strLine.length(); k++){
               nodes[i].truth[k] =
Integer.parseInt(strLine.substring(k, k+1));
            }
```

```
            }
            //Close the input stream
            in.close();
            }catch (Exception e){//Catch exception if any
            System.err.println("Error: " + e.getMessage());
        }
    }
    public static void trajectoryOut(){
        FileOutputStream out; // declare a file output object
        PrintStream p; // declare a print stream object
        String filename = JOptionPane.showInputDialog("Where do you want to
write the trajectory to?");
        try{
            // Create a new file output stream
            // connected to "myfile.txt"
            out = new FileOutputStream(filename);
            // Connect print stream to the output stream
            p = new PrintStream(out);
            // Prints out the number of nodes in the BN
            p.println(nodeNum);
            p.println(iterations);
            for(int i=0; i<=iterations; i++){
                for(int j=0; j<nodeNum; j++){
                    p.print(trajectory[j][i]);
                }
                p.println();
            }
            p.close();
        }
        catch (Exception e){//Catch exception if any
            System.err.println("Error: " + e.getMessage());
        }
    }
    public static void trajectoryIn(){
        String filename = JOptionPane.showInputDialog("What file do you want
to read a trajectory in from?");
        try{
            // Open the file that is the first
            // command line parameter
            FileInputStream fstream = new FileInputStream(filename);
            // Get the object of DataInputStream
            DataInputStream in = new DataInputStream(fstream);
            BufferedReader br = new BufferedReader(new
InputStreamReader(in));
            String strLine;
            strLine = br.readLine();
```

```
            nodeNum = Integer.parseInt(strLine);
            nodes = new Node[nodeNum];
            for(int i=0; i<nodeNum; i++){
               nodes[i] = new Node();
            }
            strLine = br.readLine();
            iterations = Integer.parseInt(strLine);
            trajectory = new int[nodeNum][iterations+1];
            for(int i=0; i<=iterations; i++){
               strLine = br.readLine();
               for(int j=0; j<nodeNum; j++){
                  trajectory[j][i] =
Integer.parseInt(strLine.substring(j,j+1));
               }
            }
            //Close the input stream
            in.close();
            }catch (Exception e){//Catch exception if any
            System.err.println("Error: " + e.getMessage());
         }
      }
      public static void networkOut(String filename){
         FileOutputStream out; // declare a file output object
         PrintStream p; // declare a print stream object
         try{
            // Create a new file output stream
            // connected to "myfile.txt"
            out = new FileOutputStream(filename);
            // Connect print stream to the output stream
            p = new PrintStream(out);
            // Prints out the number of nodes in the BN
            p.println(nodeNum);
            for(int i=0; i<nodeNum; i++){
//             p.println("This is node "+i+"'s value");  //Useful for
reading the output as a person
               p.println(nodes[i].Val);
//             p.println("This is how many parents node "+i+" has");
//Useful for reading the output as a person
               p.println(nodes[i].parents.length);
//             p.println("These are node "+i+"'s parents");  //Useful for
reading the output as a person
               for(int j=0; j<nodes[i].parents.length; j++){
                  p.print(nodes[i].parentNums[j]+" ");
               }
               p.println();
//             p.println("This is the truth table for node "+i);
```

```
//Useful for reading the output as a person
            for(int j=0; j<(int)Math.pow(2,nodes[i].parents.length);
j++){
               p.print(nodes[i].truth[j]);
            }
            p.println();
         }
         p.close();
      }
      catch (Exception e){//Catch exception if any
         System.err.println("Error: " + e.getMessage());
      }
   }
   public static void networkIn(String filename){
      try{
         // Open the file that is the first
         // command line parameter
         FileInputStream fstream = new FileInputStream(filename);
         // Get the object of DataInputStream
         DataInputStream in = new DataInputStream(fstream);
         BufferedReader br = new BufferedReader(new
InputStreamReader(in));
         int parentValue = 0, parentNumber, flag = 0;
         String strLine;
         strLine = br.readLine();
         nodeNum = Integer.parseInt(strLine);
         nodes = new Node[nodeNum];
         for(int i=0; i<nodeNum; i++){
            nodes[i] = new Node();
         }
         for(int i=0; i<nodeNum; i++){
            nodes[i].Val = Integer.parseInt(br.readLine());
            parentNumber = Integer.parseInt(br.readLine());
            nodes[i].setValues(parentNumber);
            strLine = br.readLine();
            int[] values = new int[parentNumber];
            for(int j=0; j<parentNumber; j++){
               do{
                  if(strLine.substring(0,1).compareTo(" ")==0){
                     strLine = strLine.substring(1);
                     flag = 1;
                  }else{
                     parentValue = parentValue*10 +
Integer.parseInt(strLine.substring(0,1));
                     strLine = strLine.substring(1);
                  }
```

```
                }while(flag==0);
                values[j] = parentValue;
                parentValue = flag = 0;
            }
            nodes[i].setParents(values);
            strLine = br.readLine();
            for(int k=0; k<strLine.length(); k++){
                nodes[i].truth[k] =
Integer.parseInt(strLine.substring(k, k+1));
            }
        }
        //Close the input stream
        in.close();
        }catch (Exception e){//Catch exception if any
        System.err.println("Error: " + e.getMessage());
    }
  }
  public static void trajectoryOut(String filename){
    FileOutputStream out; // declare a file output object
    PrintStream p; // declare a print stream object
    try{
      // Create a new file output stream
      // connected to "myfile.txt"
      out = new FileOutputStream(filename);
      // Connect print stream to the output stream
      p = new PrintStream(out);
      // Prints out the number of nodes in the BN
      p.println(nodeNum);
      for(int i=1; i<=iterations; i++){
        for(int j=0; j<nodeNum; j++){
          p.print(trajectory[j][i]);
        }
        p.println();
      }
      p.close();
    }
    catch (Exception e){//Catch exception if any
      System.err.println("Error: " + e.getMessage());
    }
  }
  public static void trajectoryIn(String filename){
    try{
      // Open the file that is the first
      // command line parameter
      FileInputStream fstream = new FileInputStream(filename);
      // Get the object of DataInputStream
```

```
            DataInputStream in = new DataInputStream(fstream);
            BufferedReader br = new BufferedReader(new
InputStreamReader(in));
            int parentValue = 0, parentNumber, flag = 0;
            String strLine;
            strLine = br.readLine();
            nodeNum = Integer.parseInt(strLine); // CHECK
            nodes = new Node[nodeNum];
            for(int i=0; i<nodeNum; i++){
               nodes[i] = new Node();
            }
            strLine = br.readLine();
            iterations = Integer.parseInt(strLine);
            trajectory = new int[nodeNum][iterations+1];
            for(int i=0; i<=iterations; i++){
               strLine = br.readLine();
               for(int j=0; j<nodeNum; j++){
                  trajectory[j][i] =
Integer.parseInt(strLine.substring(j,j+1));
               }
            }
            //Close the input stream
            in.close();
            }catch (Exception e){//Catch exception if any
            System.err.println("Error: " + e.getMessage());
         }
      }
      public static void trajectoriesOut(){
         FileOutputStream out; // declare a file output object
         PrintStream p; // declare a print stream object
         String filename = JOptionPane.showInputDialog("Where do you want to
write the trajectories to?");
         try{
            // Create a new file output stream
            // connected to "myfile.txt"
            out = new FileOutputStream(filename);
            // Connect print stream to the output stream
            p = new PrintStream(out);
            // Prints out the number of nodes in the BN
            p.println(nodeNum);
            p.println(iterations);
            p.println(startingPoints);
            for(int x=0; x<startingPoints; x++){
               for(int i=0; i<=iterations; i++){
                  for(int j=0; j<nodeNum; j++){
                     p.print(trajectories[j][i][x]);
```

```
                }
                p.println();
            }
//              p.println("BREAK");
        }
        p.close();
      }
      catch (Exception e){//Catch exception if any
        System.err.println("Error: " + e.getMessage());
      }
   }
   public static void trajectoriesIn(){
      String filename = JOptionPane.showInputDialog("What file do you want
to read a trajectory in from?");
      try{
        // Open the file that is the first
        // command line parameter
        FileInputStream fstream = new FileInputStream(filename);
        // Get the object of DataInputStream
        DataInputStream in = new DataInputStream(fstream);
        BufferedReader br = new BufferedReader(new
InputStreamReader(in));
        String strLine;
        strLine = br.readLine();
        nodeNum = Integer.parseInt(strLine);
        nodes = new Node[nodeNum];
        for(int i=0; i<nodeNum; i++){
          nodes[i] = new Node();
        }
        strLine = br.readLine();
        iterations = Integer.parseInt(strLine);
        strLine = br.readLine();
        startingPoints = Integer.parseInt(strLine);
        trajectories = new int[nodeNum][iterations+1][startingPoints];
        for(int x=0; x<startingPoints; x++){
          for(int i=0; i<=iterations; i++){
            strLine = br.readLine();
//            if(strLine.compareTo("BREAK")==1){
              for(int j=0; j<nodeNum; j++){
                trajectories[j][i][x] =
Integer.parseInt(strLine.substring(j,j+1));
              }
//            }
//            else{
//              i = iterations;
//            }
```

```
            }
         }
         //Close the input stream
         in.close();
         }catch (Exception e){//Catch exception if any
         System.err.println("Error: " + e.getMessage());
      }
   }
   public static void trajOutGV(){
      FileOutputStream out; // declare a file output object
      PrintStream p; // declare a print stream object
      String filename = JOptionPane.showInputDialog("Where do you want to
write the trajectory to?");
      try{
         // Create a new file output stream
         // connected to "myfile.txt"
         out = new FileOutputStream(filename);
         // Connect print stream to the output stream
         p = new PrintStream(out);
         // Prints out the number of nodes in the BN
         p.println("digraph \"results/DBN\" {");
         p.println(" ratio=auto;");
         p.println(" margin=0;");
         p.println("
node[fontname=\"ArialMT\",shape=ellipse,style=filled,fillcolor=lightgray];");
         for(int i=0; i<iterations; i++){
            p.print(" \"");
            for(int j=0; j<nodeNum; j++){
               p.print(trajectory[j][i]);
            }
            p.print("\" -> \"");
            for(int j=0; j<nodeNum; j++){
               p.print(trajectory[j][i+1]);
            }
            p.print("\"");
            p.println();
         }
         p.println("}");
         p.close();
      }
      catch (Exception e){//Catch exception if any
         System.err.println("Error: " + e.getMessage());
      }
   }
}
```

## vii. Visualization

```
import javax.swing.*;
import java.util.*;
import java.io.*;
public class Visualization extends BooleanNetwork{
   public static void Visualize(){
      FileOutputStream out; // declare a file output object
      PrintStream p; // declare a print stream object
      String filename = JOptionPane.showInputDialog("Where do you want to
write the visual data to?");
      try{
         // Create a new file output stream
         out = new FileOutputStream(filename);
         // Connect print stream to the output stream
         p = new PrintStream(out);
         // Prints out the number of nodes in the BN
         p.println("digraph \"results/DBN\" {");
         p.println(" ratio=auto;");
         p.println(" margin=0;");
         p.println("
node[fontname=\"ArialMT\",shape=ellipse,style=filled,fillcolor=lightgray];");
         Simulator.SimulateStartingPointsNoInput();
         System.out.println((int)Math.pow(2.,(double)nodeNum));
         for(int x=0; x<(int)Math.pow(2.,(double)nodeNum); x++){
            for(int i=0; i<iterations; i++){
               p.print(" \"");
               for(int j=0; j<nodeNum; j++){
                  p.print(trajectories[j][i][x]);
               }
               p.print("\" -> \"");
               for(int j=0; j<nodeNum; j++){
                  p.print(trajectories[j][i+1][x]);
               }
               p.print("\"");
               p.println();
            }
         }
         p.println("}");
         p.close();
      }
      catch (Exception e){//Catch exception if any
         System.err.println("Error: " + e.getMessage());
      }
   }
}
```

## viii. Node

```
public class Node {
   Node[] parents;
   int[] parentNums;
   int[] truth;
   int Val;

   public void setValues(int parentNum){
      parents = new Node[parentNum];
      for(int i=0; i<parentNum;i++){
         parents[i] = new Node();
      }
      parentNums = new int[parentNum];
      truth = new int[(int)Math.pow(2,parentNum)];
   }
   public void setParents(int[] value){
      for(int i=0; i<value.length; i++){
         parents[i] = BooleanNetwork.nodes[value[i]];
         parentNums[i] = value[i];
      }
   }
   public int Value(){
      int counter=0;
      for(int i=0;i<parents.length;i++){
         if(parents[i].Val==1){
            counter+=(int)Math.pow(2,i);
         }
      }
      return truth[counter];
   }
}
```