Implicit Methods for probabilistic modeling of Gene Regulatory Networks

Abhishek Garg, Debasree Banerjee and Giovanni De Micheli

***Abstract*— In silico modeling of Gene Regulatory Networks (GRN) has recently aroused a lot of interest in the biological community for modeling and understanding complex pathways. Boolean Networks (BN) are a common modeling tool for in silico dynamic analysis of such pathways. Although they are known to have effectively modeled many real and complex regulatory networks, they are deterministic in nature and have shortcomings in modeling non-determinism that is inherent in biological systems. Probabilistic Boolean Networks (PBN) have been proposed to counter these shortcomings. The capabilities of PBNs have been so far under-utilised because of the lack of an efficient PBN toolbox. This work addresses some issues associated with traditional methods of PBN representation and proposes efficient algorithms to model gene regulatory networks using PBNs.**

I. Introduction

There has been a renewed interest in modeling gene regulatory networks as Boolean networks because of their capability to efficiently model complex networks in the absence of knowledge on the expression dynamics of the genes involved. Although Boolean Networks have been used successfully in the past to model various real gene regulatory networks [4, 9, 10], their inherent deterministic nature and Boolean logic have been a central issue of criticism. *Non-determinism* in gene regulatory networks may exist for various reasons. For instance, a protein may not bind to its operation site (leading to loss of functionality represented by the GRN) or, multiple functions may exist to explain the behaviour of a gene in similar circumstances. Such non-deterministic behaviour is difficult to accommodate in Boolean Networks. This prompted researchers in [12] to propose a probabilistic extension of Boolean networks termed as Probabilistic Boolean Networks.

In a PBN, behaviour of a gene can be described with multiple Boolean functions. Each function has a probability associated with it and there is at least one function corresponding to each gene that can predict its expression as a function of the expressions of the input genes. If all the genes have only one function then a PBN is similar to a BN. Alternatively, a PBN can be seen as a set of BNs. In that case each BN has a probability equal to the product of the probabilities associated with the Boolean functions of which it is composed. Although most analyses on PBNs are done by looking at the latter description of a PBN, in this paper we look at the equivalent former decription and propose a more suitable mathematical model for efficient analyses of probabilistic gene regulatory networks.

In this paper, we present algorithms for efficient implicit representation of PBNs which enable analyses of large GRNs that were not feasible earlier due to the correponding computational complexity. The new algorithms are available in `genYsis-P` (an extended version of our `genYsis` [8]) toolbox. The software binaries are available on `http://si2.epfl.ch/~garg/genysisP.html`.

Abhishek Garg and Prof. Giovanni De Micheli are with the Laboratory of System Integrated, Faculty of Information and Communication Sciences, Ecole Polytechnique Fédérale de Lausanne, Station 14, 1015 Lausanne, Switzerland `abhishek.garg@epfl.ch`, `giovanni.demicheli@epfl.ch`

Debasree Banerjee is in the Faculty of Information and Communication Sciences, Ecole Polytechnique Fédérale de Lausanne, Station 14, 1015 Lausanne, Switzerland `debasree.banerjee@epfl.ch`

II. Probabilistic Boolean Networks

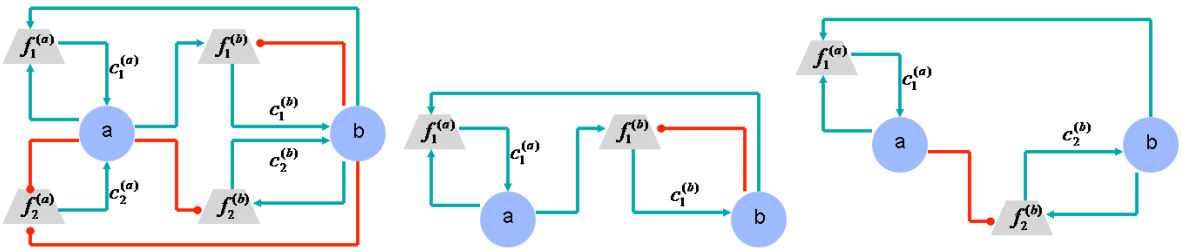
A probabilistic Boolean network (PBN) is defined by the triplet (V, F, C), where $V = \{v_1, v_2, ..., v_n\}$ is the set of variables in the network. Each variable v_i is described by a set of Boolean functions, $F_i = \{f_1^{(i)}, f_2^{(i)}, ..., f_{l_i}^{(i)}\}$. Each function $f_i^{(j)}$ has a probability (or chance of selection) associated with it, which is given by the real number $c_i^{(j)}$. Using this terminology, $F = \{F_1, F_2, ..., F_n\}$ and $C = \{C_1, C_2, ..., C_n\}$, where $C_i = \{c_1^{(i)}, c_2^{(i)}, ..., c_{l_i}^{(i)}\}$ such that $\sum_{l_i}^{j=1} c_j^{(i)} = 1$. A sample PBN is shown in Figure 1(a). In this figure, each variable can be described by two functions.

When PBNs are used as a representation for gene regulatory networks, variables v_i correspond to the genes in the regulatory network. For each v_i, the corresponding set of Boolean functions F_i represent the Boolean relationships between v_i and the genes that can have an influence on this gene. And the probability values $c_i^{(j)}$ represent the confidence on using the function $f_i^{(j)}$ to explain the dynamics of v_j. PBN of Figure 1(a) correspond to a gene regulatory network of Figure 2.

A particular realization of the PBN is given by the vector $\mathbf{f} = \{f^{(1)}, f^{(2)}, ..., f^{(n)}\}$ taking values in $F_1 \times F_2 \times ..... \times F_n$. By this definition, the number of all possible realizations (*N*) of a PBN is given by Equation 1 :

$$\begin{aligned} N &= \prod_{i=1}^{n} l_i \\ \text{where } l_i &= |F_i| \text{ for } i = 1, 2, ..., n \end{aligned} \tag{1}$$

Two of the four possible realizations of the PBN in Figure 1(a) are as shown in Figures 1(b) and 1(c). These two realizations correspond to $\mathbf{f} = \{f_1^{(a)}, f_1^{(b)}\}$ and $\mathbf{f} = \{f_1^{(a)}, f_2^{(b)}\}$ respectively. Each realization of a PBN represents a Boolean network. All possible realizations of a PBN can be represented by using an $N \times n$ matrix $\mathbf{K}$ where the row elements k_{ij} $\{\forall j = 1, 2, ..., n\}$ represent the network



(a) Probabilistic Boolean Network (b) Boolean Network 1 (c) Boolean Network 2

Fig. 1. (a) A sample PBN. There are two variables a and b and four functions $f_1^{(a)} = a \wedge b$, $f_2^{(a)} = \neg a \wedge \neg b$, $f_1^{(b)} = a \wedge \neg b$ and $f_2^{(b)} = \neg a \wedge b$. Arrow headed edges represent activation and circle headed edges represent inhibition. (b) A Boolean network formed by selecting only $f_1^{(a)}$ and $f_1^{(b)}$. (b) A Boolean network formed by selecting only $f_1^{(a)}$ and $f_2^{(b)}$.

Fig. 2. A Gene Regulatory Network corresponding to PBN in Figure 1(a).

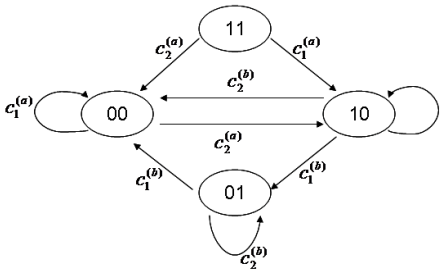


Fig. 3. State Transition Diagram corresponding to Figure 1(a). States 00,01 and 10 form an attractor. Labels on the edges represent the probability of transition

realization i $\{\forall i = 1, 2, ..., N\}$ given by a vector $\mathbf{f}$. Then the probability P_i that a network realization i is selected is given by Equation 2, where $c_{K_{ij}}^{(j)}$ defines the probability of choosing the function defined by k_{ij} for the gene v_j.

$$P_i = \prod_{j=1}^{n} c_{k_{ij}}^{(j)} \quad (2)$$

Let us define a Boolean vector $\mathbf{x}^t$ of size n, that represents the state of the network at time t. Gene v_i is ON or active at time t if $x_i^t = 1$ and the gene v_i is OFF or inactive if $x_i^t = 0$. The probability of making a transition from a state $\mathbf{x}^t$ to state $\mathbf{x}^{t+1}$ can be computed by using Equation 3 [12].

$$P(\mathbf{x}^t, \mathbf{x}^{t+1}) = \sum_{i: f_{K_{i1}}^{(1)} = x_1^{t+1}, ..., f_{K_{in}}^{(n)} = x_n^{t+1}} P_i \quad (3)$$

The computational complexity of Equation 3 is $O(N)$, where N is the number of possible PBN realizations. If each of the n genes in the network has two possible functions, then Equation 1 implies $N = 2^n$. This in turn implies that Equation 3 can not be used for networks that have a very large number of genes (i.e. n) even if the number of alternate functions for each gene is as small as 2. The state transition diagram corresponding to Figure 1(a) is as shown in Figure 3. The values on the edges in Figure 3 represent the probability of transition computed using Equation 3.

III. Biological Motivation

Dynamic analysis of gene regulatory networks (GRN) can be a powerful tool in understanding a cell differentiation process or the progression of a disease. One of the central features in such analyses is the identification of steady states (or the attractors) of the GRN. If a given GRN represents the interactions between the genes/proteins participating in a cell differentiation process, then steady states may correspond to the cell states. Each cell state has a gene expression profile (i.e. a set of uniquely activated genes) that distinguishes it from other cell states. Cell states may correspond to different functions of the cell such as proliferation, metabolism, apoptopsis, etc. If a given GRN represents the pathways responsible for some diseases like cancer, then the cell states may even correspond to tumors.

Based on the glioma gene expression data set of [5], a small PBN of 14 genes was proposed in [13]. The 14 genes selected in the Glioma Network play a very important role in the formation of blood vessels. Presence or absence of some of them can be used to differentiate between a healthy cell and a tumor cell. For example, Tyrosine Kinase receptors (Tie-2) along with angiopoietins plays an important role in vasculogenesis (formation of blood vessels), the excision repair cross-complementing (ERCC1) gene helps in DNA damage repair, while Nuclear Factor-Kappa B (NFkB) has been linked to the presence of tumors. On modeling this network, we find a single attractor consisting of 4450 states. However, when the gene NFkB is knocked-out (i.e. NFkB is always inactive) two small attractors are found (Table I). This behaviour could be biologically interesting as blocking NF-kB has been known to cause tumor cells to stop proliferating, to die, or to become more sensitive to the action of anti-tumor agents. For these reasons NF-kB is the subject of much active research among pharmaceutical companies as a target for anti-cancer therapy[3].

We envision to use PBNs for more advanced applications like experimental data analysis and on-demand drug therapies. For example, given a PBN that is known to represent a biological phenomenon and an experimental dataset emerging from a new experiment on the same biological system, it

would be interesting to associate a confidence measure with the data. This would be particularly interesting if their is time series data as the dynamics of PBN can then be matched with the dynamics of the genes in the dataset.

Alternatively, given a PBN with unknown probabilities on the Boolean functions, one could learn the probabilities from the experimental data. This could be useful in highlighting the gene functions that are active in a given cell. Such analysis could be helpful for on-demand drug therapies where treatment can be personalised to the patient under study.

Central to all these analyses is: computation of steady states, computation of the probability of a path from one state of the network to another state and identification of key genes that should be perturbed to have a desired impact on the system. All these computational tasks have been addressed in the literature [12, 13, 15]. However, current tools for PBNs use an explicit representation and computation of the networks, restricting their application to networks having less than ten genes. Even for small networks, these explicit techniques can not detect the presence of multiple attractors. For instance for the glioma network, the existing PBN toolbox could detect only one attractor on knocking the NFkB gene [15].

IV. PBN Functionalities

In this section we describe some of the functionalities of PBNs. To efficiently perform these functionalities, we provide implicit representation and traversal techniques based on Reduced Ordered Binary Decision Diagrams (ROBDDs or in short BDDs)[2] and Algebraic Decision Diagrams (ADDs)[1]. For readers not familiar with BDDs and ADDs, a small introduction is available on the software webpage. Any Boolean function can be represented as BDDs. Henceforth, we will describe all the algorithms in terms of Boolean functions, although they are implemented using BDDs in practice. ADDs are an extension of BDDs for arbitrary finite domain values for terminal nodes and are very efficient for matrix representation and manipulation.

A. Steady State Distribution

The state transition diagram of PBNs can be modeled as Markov Chains consisting of 2^n states and the corresponding state transition matrix $\mathbf{A}$ is given by Equation 4 [12].

$$A(\mathbf{x}^t, \mathbf{x}^{t+1}) \quad = \quad P(\mathbf{x}^t, \mathbf{x}^{t+1}) \tag{4}$$

Using the transition matrix $\mathbf{A}$, the *Power method* can be used to compute the steady state probability distribution. In this method, given an initial probability distribution vector $\mathbf{y}^{(0)}$, Equation 5 is iterated until the condition in Equation 6 is satisfied for some tolerance ε.

$$\mathbf{y}^{(k)} \quad = \quad \mathbf{A}\mathbf{y}^{(k-1)} \tag{5}$$

$$\|\mathbf{y}^{(k)} - \mathbf{y}^{(k-1)}\|_\infty \quad < \quad \varepsilon \tag{6}$$

This method has been observed to converge in a few iterations [15], once the matrix $\mathbf{A}$ has already been constructed. In the Boolean domain, construction of the state transition matrix as defined by [12] has the computational complexity of $O(N \cdot 2^n \cdot 2^n)$. Under synchronous transition assumption the matrix $\mathbf{A}$ may have at most $N \cdot 2^n$ non-zero entries [13]. An improved algorithm was proposed in [15] that computes only those transitions that have non-zero probabilities and hence reduces the complexity of the construction of the matrix $\mathbf{A}$ to $O(N \cdot 2^n)$. Even with this reduced complexity, the factor N can still be very large for even simple networks as we described in Section II. Even if N is small, an explicit construction of the transition diagram still has an exponential complexity. We will see later in this section, how we counter this problem by using the implicit representation and traversal techniques based on BDDs and ADDs.

Computing a steady state distribution using Equations 5 and 6 depends upon the starting distribution $\mathbf{y}^{(0)}$. In the case of multiple attractors, different steady state probability distributions may be found depending upon the chosen $\mathbf{y}^{(0)}$. It is not possible to make a claim on the number of attractors for a given network using this method. Since it is often interesting to study gene regulatory networks with multiple steady states (or cell states), the current methodology prevents the user from exploiting the full functionality of PBNs. Now we describe how these issues are addressed with our BDD based techniques.

First we introduce a modified form of Equation 3 to reduce the complexity of computing the probability of transition from $O(N)$ to $O(\tilde{N})$, where $\tilde{N}$ is given by Equation 7. $\tilde{N}$ is the number of Boolean functions in a given PBN.

$$\begin{aligned} \tilde{N} \quad &= \quad \sum_{i=1}^{n} l_i \\ \text{where } l_i \quad &= \quad |F_i| \text{ for } i = 1, 2, ..., n \end{aligned} \tag{7}$$

The modified formula for computing the probability of transition is then given by Equations 8 and 9:

$$P(x_i^t, x_i^{t+1}) \quad = \quad \sum_{j:f_j^{(i)}=x_i^{t+1}} c_j^{(i)} \tag{8}$$

$$P(\mathbf{x}^t, \mathbf{x}^{t+1}) \quad = \quad \prod_{i=1}^{n} P(\mathbf{x}_i^t, \mathbf{x}_i^{t+1}) \tag{9}$$

In Equation 8, $0 \leq P(x_i^t, x_i^{t+1}) \leq 1$ and is equal to the sum of the probabilities associated with all those functions $f_j^{(i)}$ for which the expression of gene v_i at time $t+1$ matches the function evaluation (i.e. $f_j^{(i)} = x_i^{t+1}$). The probability of transition of the state of the gene regulatory network from $\mathbf{x}^t$ to $\mathbf{x}^{t+1}$ is given by the product of the transition probabilities of all the genes and is defined by Equation 9. The computational complexity of Equation 9 is $O(\tilde{N})$. To get an idea of the improvement in complexity with this modified equation, if each gene can have k functions then $\tilde{N} = k \cdot n$, whereas $N = k^n$ in Equation 1. Mathematically, Equations 3 and 9 are equivalent.

However, using this modified equation may not help in reducing the complexity of constructing the transition matrix $\mathbf{A}$ as the number of non-zero elements in the matrix are

$O(N \cdot 2^n)$ and using this modified equation for each non-zero entry would only increase the computational complexity to $O(\tilde{N} \cdot N \cdot 2^n)$. If the size of the transition matrix $\mathbf{A}$ could be reduced so that the number of nonzero entries are significantly smaller than $O(N \cdot 2^n)$, then using this modified equation could be helpful. This is the central idea behind our BDD based technique. We compute the steady states (or attractors) using an implicit representation based on BDDs and compute the transition matrix implcitly (using ADDs) only for the states restricted to the attractors. This way the size of the matrix $\mathbf{A}$ is proportional to the size of the attractor. Although it is difficult to put a bound on the number of states in an attractor, it has been observed that it is often a small number [15]. Dividing the problem of computing the steady state distribution into two parts ensures that in the event of a situation in which the size of an attractor becomes very large to be represented as ADDs, our algorithm is at least able to detect the states in the attractor. However, we may not be able to compute the probability distribution of these states in that situation.

We start by giving an implicit representation of PBNs as Boolean functions which is suitable for the BDD representation. Given a PBN, a gene v_i can have only two states '0' and '1'. Since there are multiple functions $f_j^{(i)}$ acting on the gene v_i at the same time, there is a possibility that some of the functions set the expression of the gene v_i to '0' and some others may set the expression to '1'. Let us define the expression of the gene v_i by using the function defined in Equation 10 :

$$x_i(t+1) = \underbrace{\{x_i^{t+1} \Leftrightarrow \bigvee_{j=1}^{l_i} f_j^{(i)}\}}_{I} \vee \underbrace{\{x_i^{t+1} \Leftrightarrow \bigwedge_{j=1}^{l_i} f_j^{(i)}\}}_{II} \quad (10)$$

Part (I) of Equation 10 represents the situation when one of the input functions $f_j^{(i)}$ can set the expression of gene v_i at time $t+1$ (given by x_i^{t+1}) to 1. Part (II) of Equation 10 represents the case when one of the input functions $f_j^{(i)}$ can set x_i^{t+1} to 0.

If all the genes in the network are assumed to make a synchronous transition, i.e. all the genes change their expression at the same time, then the transition function representing the state of the network between consecutive time steps can be given by Equation 11 :

$$T(\mathbf{x}^t, \mathbf{x}^{t+1}) = \bigwedge_{i=1}^{n} x_i(t+1) \quad (11)$$

Equation 11 is similar to Equation 2, except that it evaluates to a binary value.

We proposed, in [8], algorithms to compute steady states using the transition function $T(\mathbf{x}^t, \mathbf{x}^{t+1})$ (of a different form but similar structure). The same algorithms can be used here with a modified $T(\mathbf{x}^t, \mathbf{x}^{t+1})$. We will only give the theorems [14] on which these algorithms are based and leave the details of the algorithms which can be found in [14, 8].

Definition 1: Forward reachable states $FR(S_0)$ from the states set S_0 are all the states that can be reached from the states in the set S_0 by iteratively computing the forward image (states reachable in one step) on the transition function $T(\mathbf{x}^t, \mathbf{x}^{t+1})$ until no new states are reachable.

Definition 2: Backward reachable states, $BR(S_0)$, are all the states $\mathbf{x}^t$ whose forward reachable set contains at least one state in S_0.

Definition 3: A *Steady State (or attractor)* is the set of states $SS(\mathbf{x}^t)$ such that for all the states $s \in SS(\mathbf{x}^t)$, the forward reachable set $FR(s)$ is the same as $SS(\mathbf{x}^t)$.

Theorem 1: A state $i \in S$ is a steady state if and only if $FR(i) \subseteq BR(i)$. State i is transient otherwise.

Theorem 2: If state $i \in S$ is transient, then states in $BR(i)$ are all transient. If state i is steady, then all the states in $FR(i)$ are steady. In the latter case set $\{BR(i) - FR(i)\}$ has all the transient states.

Based on Theorems 1 and 2, we can find all the attractors that may exist in a given state transition diagram. For every attractor, we can construct the matrix $A(\mathbf{x}^t, \mathbf{x}^{t+1})$ restricted to the states in that attractor and compute the steady state probability distribution for those states. Equation 8 can also be computed symbolically using Equations 12 and 13.

$$q_j^{(i)} = (x_i^{t+1} \Leftrightarrow f_j^{(i)}) \wedge c_j^{(i)} \quad (12)$$

$$\tilde{P}(x_i^t, x_i^{t+1}) = \bigvee_{j=1}^{l_i} (q_j^{(i)} \wedge \mathbf{x}^t \wedge \mathbf{x}^{t+1}) \quad (13)$$

We remove the variables x_i^t and x_i^{t+1} from Equation 13 by applying the *Existential Quantification* (i.e. the operator $\exists$) [2]. The outcome of the $\exists$ operator on Equation 13 is the logic OR of Boolean variables $c_j^{(i)}$ for which $f_j^{(i)} = x_i^{t+1}$. On substituting Boolean variables $c_j^{(i)}$ with probabilities $c_j^{(i)}$ and logic OR with the sum operator, we get the probability of transition of gene v_i from x_i^t to x_i^{t+1}. The probability of transition of all the genes v_i $(\forall i = 1, 2, ..., n)$ is then used together as in Equation 9 to compute the probability of transition from the state $\mathbf{x}^t$ to $\mathbf{x}^{t+1}$. Equations 9 and 13 can be computed implicitly using ADDs.

Algorithm 1 formally describes the procedure for computing the probability of a transition. In this algorithm, Equation 13 is implemented in lines 7-10. Equation 9 is implemented in line 11. The number of times the for loop in line 7 is iterated is given by Equation 7 and explains the $O(\tilde{N})$ complexity of this computation.

Using both Equations 11 and 13, we can construct the sparse transition matrix $\mathbf{A}$. The matrix $\mathbf{A}$ now only represents the transitions among the states restricted to an attractor (given by a set S). The column entries A_{ij} gives the probability of state j making a transition into state i in one step. Equation 11 can be used to compute all the states j that can transition into the state i in a single step. The probabilities for these transitions can then be computed using Equation 13. The resulting matrix $\mathbf{A}$ can then be used as in Equations 5 and 6 to find the steady state probability distribution.

Given a set of states S, Algorithm 2 can be used to construct the state transition matrix for transitions among

Algorithm 1: Computing the probability of a transition $P(s^{src}, s^{dest})$

```
1  prob_tran(Q, s^src, s^dest)
2  begin
3      P(s^src, s^dest) = 1.0
4      s^dest = s^dest(x^t ← x^{t+1})
5      for i = 1 to n do
6          p_i = 0.0;
7          for j = 1 to l_i do
8              p̃_i^sym = (q_j^(i) ∧ s^src ∧ s^dest)
9              p_i^sym = ∃_{x∈{x^t,x^{t+1}}} p̃_i^sym
10             p_i = p_i + real(p_i^sym)
11         P(s^src, s^dest) = p_i × P(s^src, s^dest)
12     return P(s^src, s^dest)
13 end
```

these states. In this algorithm, first all the states in the set S are mapped to a unique id in line 3. Then for every state S_i in the set S, we compute the states that can reach S_i in one step using the image function $I^b(S_i)$ in line 5. Now for all the states in this set we compute the probability of transition in line 9 using the prob_tran() function described in Algorithm 1. Algorithm 2 is implemented using ADDs in practice.

B. Probability of a path

Given a path $p(i \rightsquigarrow^1 j) = (i, s_1, s_2, ..., s_n, j)$ from state i to state j, the probability of this path, $P(i \rightsquigarrow^1 j)$ is given by Equation 14 :

$$P(i \rightsquigarrow^1 j) \quad = \quad P(i, s_1)P(s_1, s_2)....P(s_n, j) \quad (14)$$

There can be multiple paths between any two states and the probability $P(i \rightsquigarrow j)$ to go from a state i to state j is given by the sum of the probability of all these paths. Summation of the probability of all the paths l maintains the probability constraint, i.e. $0 \leq \sum_l P(i \rightsquigarrow^l j) \leq 1$. To avoid making too many $P(i \rightsquigarrow^l j)$ computations, we apply a constraint that only the shortest paths (or paths with minimum number of steps) are enumerated between any two states. This gives a lower bound on the probability $P(i \rightsquigarrow j)$.

Given the state transition matrix $\mathbf{A}$, a row vector $\mathbf{y}^{(0)}$ of size equal to the number of states in the state transition diagram is constructed. In this vector $\mathbf{y}^{(0)}$ all the entries

Algorithm 2: Constructing the transition matrix for a given set of states S

```
1  const_tran_matrix(T, S, Q)
2  begin
3      ID{1, ..., M} = map_states(S)
4      for i = 1 to S.size() do
5          S^next = I_b(S_i) ∧ S
6          src = find_ID(ID, S_i)
7          for j = 1 to S^next.size() do
8              dest = find_ID(ID, S_j^next)
9              A_{src,dest} = prob_tran(Q, S_i, S_j^next)
10     return A
11 end
```

Algorithm 3: Computing Forward and Backward reachable sets

```
1  forward_set(S_0, T, S^target)
2  /* backward_set(S^0, T, S^target) */
3  begin
4      RS^(0) ⟵ ∅, FS^(0) ⟵ {S^0}
5      k ⟵ 0
6      while FS^(k) ≠ ∅ do
7          FS^(k+1) = I_f(FS^(k))(x^{t+1} ← x^t) ∧ ¬RS^(k)
8          /* FS^(k+1) = I_b(FS^(k))(x^t ← x^{t+1}) ∧ ¬RS^(k) */
9          RS^(k+1) = RS^(k) ∨ FS^(k+1)
10         if RS^(k+1) ∧ S^target ≠ ∅ then
11             return (RS^(k+1))
12         k ⟵ k + 1
13     /* Target is not reachable from source. Return Empty Set */
14     return ∅
15 end
```

Algorithm 4: Computing set of states that lie on all shortest paths between a source and destination

```
1  compute_states_path(T, S^src, S^dest)
2  begin
3      FR = forward_set(S^src, T, S^target)
4      S^src_tmp = S^src(x^t ← x^{t+1}))
5      S^dest_tmp = S^dest(x^t ← x^{t+1}))
6      BR = backward_set(S^dest_tmp, T, S^src_tmp)
7      S^Reduced = FR ∧ BR(x^t ← x^{t+1})
8      return S^Reduced
9  end
```

are 0, except the entry corresponding to the source state which is 1. Then Equation 15 is iterated until the condition in Equation 16 is satisfied.

$$\mathbf{y}^{(k)} \quad = \quad \tilde{\mathbf{A}}\mathbf{y}^{(k-1)} \quad (15)$$

$$y^k[dest] > 0 \quad (16)$$

Intuitively, the iteration of Equation 15 computes the probability of reaching a state i in k iterations. The probability values are stored in the vector $\mathbf{y}^{(k)}$. Equation 16 ensures that we stop the iteration the first time we reach the destination state.

This computation has similar computational issues as the probability distribution computation using Equations 5 and 6 in Section IV-A. But again, if a smaller transition matrix $\mathbf{A}$ can be used then $P(src \rightsquigarrow dest)$ can be computed very efficiently. Using the implicit representation we described earlier, here we propose algorithms that first compute the states that lie on all the shortest paths between the source and the destination and then compute $\tilde{\mathbf{A}}$ restricted to these states.

Algorithms 3, 4 and 5, along with Equations 15 and 16 can be used to compute $P(i \rightsquigarrow j)$. Algorithms 3 and 4 compute the states that lie on the shortest paths between the source and the destination states. First, the forward reachable states from the source state are computed in line 3 of Algorithm 4. Then the backward reachable states from the destination state are computed in line 6. The intersection of the forward and

Algorithm 5: Computing transition matrix for the states lying on shortest paths between the given source and destination states.

```
1  tran_matrix_path(T, Q, s^src, s^dest)
2  begin
3      S = compute_states_path(T, s^src, s^dest)
4      if S = ∅ then
5          /* Target is not reachable from source. Return NULL */
6          return NULL
7      A = const_tran_matrix(T, S, Q)
8      return A
9  end
```

Algorithm 6: Algorithm to compute Influence Matrix

```
1  compute_influence_matrix(F, C)
2  begin
3      for i = 1 to n do
4          for j = 1 to n do
5              for k = 1 to l_i do
6                  ∂f/∂x = ∃_{x_i}(f_k^(j) ∧ ¬x_i) ⊕ ∃_{x_i}(f_k^(j) ∧ x_i)
7                  cubes_drv = compute_Cubes(∂f/∂x)
8                  cnt = 0
9                  for p = 1 to cubes_drv.size() do
10                     cnt += 2^(n − num_var_support(cubes_drv[p]))
11                 Γ_ij += cnt/2^n × c_k^(j)
12 end
```

the backward reachable states gives all the states that may lie on all the paths between the source and the destination. To ensure that we include only the shortest paths while computing the forward and backward reachable states in Algorithm 3, we compute reachable states upto the point when the target state is first seen (in line 10). The reduced state space along with Algorithm 2 is then used in Algorithm 5 to construct the state transition matrix $\tilde{\mathbf{A}}$.

C. Sensitivity Analysis

Another interesting functionality of PBNs is their ability to represent the influence of a gene on other genes in a network. This functionality can be very useful in deciding the genes that should be perturbed to have the maximum impact on the state space of the network. This impact could be in terms of the size of the attractor, number of attractors or the probability of the reachability from one state to another. The influence of a variable x_i on a function $f(x_1, x_2, ..., x_n)$ is given by Equations 17-19 [12], where $\frac{\partial f(\mathbf{x})}{\partial x_i}$ is the Boolean difference of Function $f(\mathbf{x})$ with respect to the variable x_i.

$$I_i(f) = Pr\left\{\frac{\partial f(\mathbf{x})}{\partial x_i} = 1\right\} \quad (17)$$

$$= Pr\left\{f(x_i = 0) \neq f(x_i = 1)\right\} \quad (18)$$

$$= \frac{\text{no. of } \mathbf{x} \text{ such that } \frac{\partial f(\mathbf{x})}{\partial x_i} = 1}{2^{\text{ size of vector } \mathbf{x}}} \quad (19)$$

Equation 20 can then be used to compute the influence of a gene v_i on the gene v_j.

$$I_i(v_j) = \sum_{k=1}^{l_i} I_i(f_k^{(j)}) \cdot c_k^{(j)} \quad (20)$$

This way an $n \times n$ *influence matrix* Γ can be constructed to represent the influence of all the genes on all the other genes.

In our implicit representation of PBNs, the influence matrix can be computed trivially as explained below. Given a Boolean function $f(a, b, c)$ in Equation 21, it can be rewritten in an SOP form as in Equation 22.

$$f = (a \bar{\oplus} b)c + bc \quad (21)$$

$$f = \bar{a}\bar{b}c + bc \quad (22)$$

In Equation 22, a three variable function f consists of two cubes $\bar{a}\bar{b}c$ and bc. Cube $\bar{a}\bar{b}c$ has all the three variables in support and evaluates to true for only one $(= 2^{3-3})$ Boolean vector (i.e. $a = 0, b = 0, c = 1$). Cube bc has two variables in support and evaluates to true for two $(= 2^{3-1})$ Boolean vectors $\{111, 110\}$. Union of these two cubes gives three Boolean vectors $\{001, 111, 110\}$ for which function f is true.

In a BDD representation, any path from root node to **1**-leaf node represents a cube. So computing cubes is very efficient in BDDs and the number of cubes is always less than the number of possible boolean vectors. This method for computing the influence matrix is formally described in Algorithm 6. In this algorithm, an $n \times n$ influence matrix Γ is constructed in lines 3-11. The influence of gene v_i on v_j (i.e. Γ_{ij}) is computed in lines 5-11. A for loop is iterated over all the functions that may influence the gene v_j. For each function, the boolean difference is computed in line 6. Then the cubes of the boolean difference are stored in an array in line 7. In lines 9-10, the number of boolean vectors for all the cubes are computed and the corresponding influence for this function is finally added to Γ_{ij} in line 11.

V. Results

In this section, we present the results obtained from the simulations on the glioma network [5, 13]. The PBN network is not shown in this paper due to space constraints. The network with the corresponding truth tables can be found on the software website.

Table I gives the number and size of attractors when

TABLE I

Influence of Genes on Attractors.

Gene Perturbed	Number Of		Time (sec)
	attractors	states	
Un-perturbed	1	4450	50
ERCC1 = 1	1	768	6
SCYB10 = 0	2	1920 ; 1662	33
NFkB1 = 0	2	2180 ; 2145	38

TABLE II

Influence of Functions on Attractors.

Inactive Function	Number Of		Time (sec)
	attractors	states	
f_2^1	1	4424	38
f_1^3	1	1633	27
f_2^{14}	1	3815	36

TABLE III

COMPUTATIONAL RESULTS ON SYNTHETIC DATA.

Number of					Maximum	Time (sec)	
	Alt. Functions				Attractor	Steady	Probability
Genes	1	2	3	Attractors	size	States	Distribution
20	10	10	0	2	844	0.6	8
20	9	8	3	3	288	0.8	3
40	30	10	0	8	1536	2	7
40	23	15	2	3	2684	3	30
60	41	19	0	12	4096	39	40
60	46	10	4	18	1536	120	20

different genes are perturbed in the glioma network. A gene is knocked out when it is constantly inactive (i.e. level 0) and it is over-expressed when it is constantly active (i.e. level 1). Table II represents the situation when some of the functions are inactive in the glioma network. Function f_j^i in Table II represents the j^{th} function of gene v_i. Further details of Functions f_j^i are available on the software webpage.

All the results obtained from simulations on the glioma network match the results shown in [15]. The computation time of our algorithm was, at maximum, 50 seconds on a 1.8 GHz Dual Core Pentium machine with 1GB of RAM running on Linux Fedora Core 5. The algorithms are implemented in C++ using the CUDD package for BDDs. It is difficult to compare the run time of our algorithms with the one achieved by the authors in [15] since their program is not available in the PBN toolbox [11]. However, it has been mentioned in their paper that it takes them 20 minutes to compute the steady state distribution on a CPU Pentium 4 machine with 1GB RAM. Moreover, their software is written in MATLAB. The PBN toolbox in [11] can not run networks that consist of more than 10 nodes as it quickly runs out of memory. Even with a network of 10 nodes and two functions per gene, it takes more than 1 hour to compute the state transition matrix. Some results to benchmark our algorithm are given in Table III. Columns 2, 3 and 4 represent the number of genes in the network with 1, 2 and 3 alternate functions per gene respectively.

VI. CONCLUSION

In this work, we have provided a framework for implementing a PBN toolbox using implicit representation and traversal techniques. The software binaries of the new toolbox `genYsis-P` are freely available on our website. In the presence of limited computational resources, `genYsis-P` can handle biological networks that were earlier not possible to simulate using the PBN toolbox of [11].

Due to space constraints we do not show the extension of the algorithms proposed in this paper for Multiple valued networks and Asynchronous transition models. In Multiple Valued Networks, a gene can have more than two levels of expressions. All the equations and methods in this paper can be easily extended to multiple valued logic on the same lines as in our previous work in [7]. A synchronous model assumes that all the genes take an equal amount of time in changing their expression levels. For example, in the state transition diagram of Figure 3, transition from state 10 to 01 implies that gene a and gene b move concurrently from 1 to 0 and 0 to 1 respectively. however, for some biological phenomena it might be interesting to model asynchronous transition. We have proposed, in the past [6], methods for asynchronous modeling of boolean networks. Similar modeling approaches can be extended to asynchronous modeling of PBNs.

In the future, we will incorporate multiple valued logic and asynchronous modeling in `genYsis-P` and add more functionalities for advanced applications of PBNs.

REFERENCES

[1] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *In Proc. of IEEE/ACM ICCAD'93*, pages 188–191, 1993.

[2] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[3] R.O. Escárcega, S Fuentes-Alexandro, M Garc'ia-Carrasco, A Gatica, and A Zamora. The transcription factor nuclear factor-kappa b and cancer. *Clinical Oncology*, 19(2):154–161, 2007.

[4] C. Espinosa-Soto, P. Padilla-Longoria, and E.R. Alvarez-Buylla. A gene regulatory network model for cell fate determination during arabidopsis thalina flower development. *Plant Cell*, (16):2923–2939, 2004.

[5] G. N. Fuller, C.H. Rhee, K.R. Hess, L.S. Caskey, R. Wang, J.M. Bruner, W.K.A. Yung, and W. Zhang. Reactivation of insulin-like growth factor binding protein 2 expression in glioblastoma multiforme. *Cancer Research*, 59(17):4228–4232, 1999.

[6] A. Garg, A. Di Cara, I. Xenarios, L. Mendoza, and G. DeMicheli. Synchronous vs. asynchronous modeling of gene regulatory networks. *Under Review. A copy available on http://si2.epfl.ch/∼garg/genysisP.html*.

[7] A. Garg, L. Mendoza, I. Xenarios, and G. DeMicheli. Modeling of multiple valued gene regulatory networks. *In Proc. of 29th Annual IEEE EMB Conference*, pages 1398–1404, 2007.

[8] A. Garg, I. Xenarios, L. Mendoza, and G. DeMicheli. Efficient methods for dynamic analysis of genetic networks and in silico gene perturbation experiments. *Lecture Notes in Bioinformatics*, (4453): 62–76, 2007.

[9] L. Mendoza and I. Xenarios. A method for the generation of standardized qualitative dynamical systems of regulatory networks. *Theoretical Biology and Medical Modelling*, 3, 2006.

[10] J. Saez-Rodriguez, L. Simeoni, J. A Lindquist, R. Hemenway, U. Bommhardt, B. Arndt, U.U. Haus, R. Weismantel, E. D. Gilles, S. Klamt, and B. Schraven. A logical model provides insights into t cell receptor signaling. *PLoS Comput Biol*, 3(8):e163, 2007.

[11] I. Shmulevich and H. Lahdesmaki. Pbn matlab toolbox. URL `personal.systemsbiology.net/ilya/`.

[12] I. Shmulevich, E.R. Dougherty, S. Kim, and W. Zhang. Probabilistic boolean networks: a rule-based uncertainty model for gene regulatory network. In *Bioinformatics*, volume 18, pages 261–274, 2002.

[13] I. Shmulevich, I. Gluhovsky, R.F. Hashimoto, E.R. Dougherty, and W. Zhang. Steady-state analysis of genetic regulatory networks modelled by probabilistic boolean networks. In *Comparative and Functional Genomics*, volume 4, pages 601–608, 2003.

[14] A. Xie and P.A. Beerel. Efficient state classification of finite state markov chains. In *Design Automation Conference*, pages 605–610, 1998.

[15] S-Q. Zhang, W-K. Ching, M.K. Ng, and T. Akutsu. Simulation study in probabilistic boolean network models for genetic regulatory networks. In *Int. J. Data Mining and Bioinformatics*, volume 1, pages 217–240, 2007.