Illinois State University
ISU ReD: Research and eData

Theses and Dissertations

3-15-2016

On the Dynamics of Boolean Gene Regulatory Networks with Stochasticity

Yuezhe Li
Illinois State University, yuezheli@foxmail.com

ON THE DYNAMICS OF BOOLEAN GENE REGULATORY NETWORKS WITH STOCHASTICITY

Yuezhe Li

55 Pages

Genes are responsible for producing proteins that are essential to the construction of complex biological systems. The mechanisms by which this production is regulated have long been the center of widespread research efforts. Deterministic Boolean gene regulatory models have been a particularly effective avenue of research in this field. However, these models fall short of accounting for variations in the gene functionality due to the uncertain internal or external environmental conditions. One of the recent attempts to overcome this weakness is by (Murrugarra, 2012), in which a probabilistic component is introduced as the fixed activation/degradation propensities at the cellular level. This approach still falls short of accounting for cell-to-cell variability as well as the variability at the molecular level. With this study, we introduce an additional stochastic element by modeling the activation/degradation propensities using statistical distributions. This, in turn, allows us to quantify the variability of the different connections within the dynamical system formed by the gene activation/degradation behavior. Finally, we present a converse method of determining the most likely propensity set for a given stochastic gene regulatory network.

KEYWORDS: Boolean Gene Regulatory Network, Engineered Genetic Algorithm, Particle Swarm Optimization, p53-Mdm2 Network, Stochastic Boolean Network

ON THE DYNAMICS OF BOOLEAN GENE REGULATORY NETWORKS WITH STOCHASTICITY

YUEZHE LI

A Thesis Submitted in Partial
Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Mathematics

ILLINOIS STATE UNIVERSITY

2016

ON THE DYNAMICS OF BOOLEAN GENE REGULATORY NETWORKS WITH STOCHASTICITY

YUEZHE LI

COMMITTEE MEMBERS:
Olcay Akman, Chair
Fusun Akman
Allison Harris
Daniel Hrozencik

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisers Dr. Olcay Akman and Dr. Daniel Hrozencik for their continuous support of my research. Their patient guidance, constant encouragement, and immense specialized knowledge helped me throughout my time of research and the writing of this thesis. I could not imagine better advisers and mentors for my master's study.

Besides my advisers, I would like to thank the other members of my thesis committee, Dr. Fusun Akman and Dr. Allison Harris, for their insightful comments and encouragement.

I would like to thank my parents, my aunt, my cousin, and my friends for supporting me spiritually throughout this research and writing. They have been always there for me and have enriched my days in so many ways.

Y. L.

CONTENTS

TABLES

FIGURES

Figure Page

CHAPTER I

INTRODUCTION

There have been many experiments observing how cells with the same genotype exhibit different phenotypes when exposed to the same environmental conditions [13, 40, 2, 22]. Evolution of a cell's phenotype is attributed to the interactions between biological macromolecules inside it. In mathematical models, these macromolecules' amount and/or states are considered as variables. Together, they form dynamical systems. The earliest attempt to use mathematical models to analyze interactions between biological macromolecules traces back to the 1960s [11]. Mathematical modeling has flourished ever since.

Mathematical models of dynamical systems can be broadly divided into two classes: continuous and discrete. The continuous models are based on differential equations, such as ordinary differential equations, partial differential equations, functional differential equations, and stochastic differential equations [34, 3, 14, 46, 15, 7]. These models have applications to a broad range of biological problems such as biochemical oscillations, epidemic outbreaks, and gene regulatory networks.

The second class of models is discrete dynamical models. Unlike continuous models that provide quantitative outcomes on continuous scales, these dynamical systems get updated in discrete steps. Discrete models have one significant advantage: they do not require as many parameters, such as kinetic rates for chemical equilibria, as continuous models. These parametric values are often difficult to obtain.

Boolean networks are discrete dynamical models. They are represented by directed graphs, where nodes represent molecular elements, such as genes, RNAs, or proteins. Edges represent the interactions between these molecules. A function that maps a node's state to its next state is called a *Boolean function*. A collection of Boolean functions that map all nodes in the Boolean network is a *logical regulatory rule*. Together, nodes, edges, and a regulatory rule define the structure of a *regulatory network*.

In particular, in a *gene regulatory network* (GRN), nodes represent genes, which have two states, off (no protein produced) or on (protein produced). Edges represent interactions between genes. These interactions are of two types: *activation*, in which a gene turns on another gene at the next time step, or *inhibition*, in which a gene prohibits the activation of another gene at the next time step. The *update function* defines the rules for updating the dynamical state of each node.

Boolean networks, in general, fail to account for the variability of protein production in different cells that have the same genetic make-up. One way to overcome this drawback is to introduce stochasticity at the update function level [42, 37].

Each gene has an *activation* and a *degradation propensity*. The activation propensity is the probability that a gene activates another gene when the update function dictates that it should. Similarly, the degradation propensity is the probability that a gene deactivates another gene when it should. In [31], a probabilistic structure is introduced to allow for the cases where the gene interactions fail to follow a deterministic update rule.

In this study, we introduce a stochastic propensity structure to account for the variability due to random gene interaction failures. This approach is a way to explain the uncertainty at the cellular and molecular levels.

In our study, we employ *particle swarm optimizations* and *engineered genetic algorithms* to estimate propensities. A *genetic algorithm* (GA) is a method for solving optimization problems based on a natural selection process that mimics biological evolution. An engineered genetic algorithm is a genetic algorithm whose parameters are tuned by a *simulated annealing algorithm*. The simulated annealing algorithm (SAA) is a probabilistic technique for approximating the global optima. Particle swarm optimization (PSO) is a population-based stochastic optimization technique inspired by the social behavior of bird flocking or fish schooling. PSO shares many similarities with evolutionary computation techniques such as GA. Our study uses the global optimization toolbox in MATLAB® 2014b.

The organization of the thesis is as follows: In Chapter 2, we examine deterministic models of gene regulatory networks. In Chapter 3, we introduce stochastic gene regulatory networks, while in Chapter 4, we describe our proposed model that has an additional level of stochasticity. In Chapter 5, we focus on parameter estimation for the statistical distributions from which propensities are drawn. Chapter 6 contains the implementation of the proposed method on the human p53-Mdm2 regulatory network. We analyze advantages and disadvantages of our approach, including possible future directions, in Chapter 7.

CHAPTER II

STOCHASTICITY AND VARIABILITY IN DISCRETE GENE REGULATORY NETWORKS

Deterministic discrete gene regulatory network models assume that at every step, every molecule works steadily and precisely. In other words, the mechanisms with which the nodes and the edges operate are identical at each time step. This is clearly a strong assumption, and it is unlikely to happen in nature. In reality, there is always a chance that regulatory mechanisms may not work as expected. Thus, it is reasonable to introduce stochasticity in gene regulatory networks.

There are two commonly accepted methods of introducing stochasticity into Boolean models. The most frequently used method is to introduce the stochasticity at the update function level. More specifically, at each step, an update function is randomly chosen from a set [23, 25, 24]. Another way of introducing stochasticity is to randomize the success of an update at each node at each step. Both of these methods lead to probabilistic Boolean models. By replacing deterministic dynamics with probabilistic dynamics, the Boolean network gains extra flexibility without altering its structure. The continuous counterpart of this approach is the *Gillespie algorithm* [33].

Probabilistic Boolean networks have an added advantage. They can be viewed as Markov processes [43], where transitions from one state to another are specified by state transition probabilities. This allows us to predict the future states of the networks using the properties of Markov transition matrices.

CHAPTER III

A DETERMINISTIC MODEL OF A DISCRETE GENE REGULATORY NETWORK

A *Boolean network* is defined by a set of nodes, edges, and a list of Boolean functions describing the updating processes. Every node in a Boolean network has two states: *deactivated*, denoted by 0, and *activated*, denoted by 1. Directed edges are used to express the relations between the nodes. One of the early examples is Figure 1 [23], where arrows with solid lines indicate activations and dashed lines indicate inhibitions.

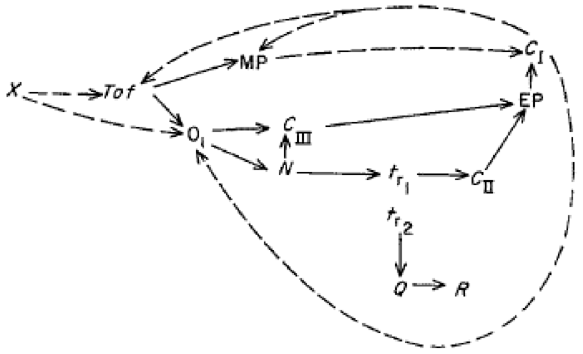

Figure 1: A Model of Lambda Gene Regulatory System

In a Boolean network, a given node transforms its inputs into an output, which is the state or expression of the gene itself at the next time-point. All genes are assumed to update synchronously in accordance with the functions assigned to them, and this process is then repeated. Our goal is to determine the state of any given gene at any given time, which is commonly referred to as the *dynamics* of a network. The dynamics of a synchronous deterministic Boolean network are completely determined by *regulatory* *rules*, which are sets of Boolean functions projecting networks' current states to their next state, and the *initial state*. Examples of this type of dynamical system representation are Boolean networks and *logical models* [8, 44, 18]. We will present the *lac operon* as an example in the rest of this chapter.

Though it is worth considering networks with more than two states for certain systems, the focus of this study is Boolean networks where all nodes have two states.

One of the well-studied gene regulatory networks is the *lac operon* in *E. coli*. A simplified *lac operon* is under the influence of three elements: *Lac I*, *Lac Z*, and lactose. *Lac I* encodes the protein that inhibits RNA polymerases binding to *Lac Z*. *Lac Z*'s final translation product is beta-galactosidase (beta-gal), an enzyme that hydrolyzes lactose. Lactose deactivates *Lac I*'s protein product by allosteric regulation, releasing *Lac Z* from *Lac I*'s inhibition (Figure 2). The regulatory rule of the *lac operon* is in Table 1.

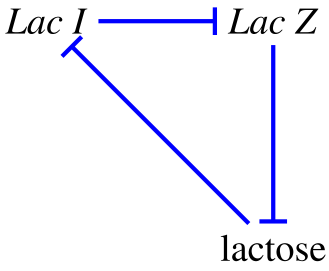

Figure 2: Simplified *Lac Operon* [19, 41]

When there is little lactose around, *Lac Z*'s protein product barely exists. *Lac I* is transcribed and translated. Its protein product is abundant and active. When there is abundant lactose around in *E.coli* cell, the lactose molecule binds to the *Lac I*'s protein product. Thus, the protein is deactivated but not hydrolyzed. When lactose runs low, the lactose molecule dissociates itself from *Lac I*'s protein product. Consequently, *Lac I*'s protein product is automatically activated again. Comparing to a gene's transcription or

Lac I	*Lac Z*	lactose	f_1	f_2	f_3
0	0	0	1	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	0	0
1	0	0	1	0	0
1	0	1	0	0	1
1	1	0	1	0	0
1	1	1	0	0	0

Table 1: *Lac Operon* Regulatory Rule

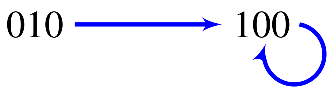

Figure 3: Diagram of Gene States' Change in *E.coli*

RNA's translation, *Lac I*'s activation is finished much faster, or almost spontaneously since this is an allosteric change. In other words, lactose removal and *Lac I*'s activation happen at the same time [20].

We assume that an *E.coli* cell was in a high-lactose and glucose-free culture for some time. Then its *Lac I* was turned off, and *Lac Z* was turned on. At time 0, it is moved to a lactose-free culture. Hence, the initial state is 010, where the three numbers, from left to right, are the states of *Lac I*, *Lac Z*, and lactose, respectively. Following the regulatory rule (Table 1), the gene states change to 100 (Figure 3).

In other words, an *E. coli* cell will stay in a stage of *Lac I* activated, *Lac Z* silenced, and lactose free. This is consistent with reality [41].

Forthwith we discuss a more complicated scenario. We assume that an *E.coli* cell, at time 0, has its Lac I and Lac Z genes on. It stays in a lactose-free culture for some time (state 110) until it evolves to a stable state (100). After that, it is transported to a glucose-free culture with a high concentration of lactose (state 101). Its genes states' changes are as in Figure 4. The thick blue line is the gene states' transition before being moved to the lactose-abundant glucose-free environment, and the cyan dotted lines mark the gene states' transition after being exposed to the high-lactose and glucose-free environment.

The red text marks two critical changes in the culture to which the *E.coli* cell is exposed. The first is when the *E.coli* cell is moved to a lactose-high glucose-free culture. This is clearly a significant change, since the *E.coli* cell has lost its major energy (carbon) source and is forced to use a new one, the lactose. When time passes by, with no further supply, lactose gradually runs low. The second critical moment is when lactose runs out in this new culture. Then the chemical equilibrium moves and lactose molecules disassociate themselves from *Lac I*'s protein products. This dissociation leads to allosteric changes in *Lac I*'s protein product, releasing them from lactose's inhibition. Consequently, *Lac I* is activated again, and *Lac Z* is turned off (Figure 4).

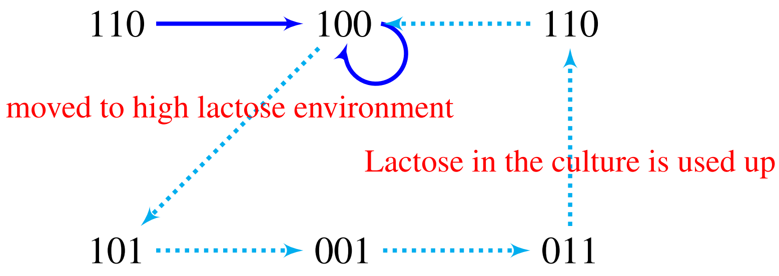
110
100
110
moved to high lactose environment
Lactose in the culture is used up
101
001
011

Figure 4: Diagram of Gene States' Change in *E.coli* While Culture Changes

Deterministic models, such as the simplified *lac operon* described above, are the ideal cases. They, however, may not suit the reality due to the flexibility of macromolecules [45]. By adding probabilities that macromolecules fail to function as they should, *stochastic Boolean networks* are introduced. Stochastic Boolean networks show more flexibility and variability than deterministic Boolean models. Each molecule is represented by a node in the stochastic Boolean network. Probabilities of whether macromolecules function properly are parameters in stochastic Boolean networks. Our study focuses on the inference of parameters from the behavior of the dynamical system.

CHAPTER IV

STOCHASTIC GENE REGULATORY NETWORKS

In this chapter, we introduce stochasiticity to deterministic gene regulatory networks. This goal is achieved by adding a *propensity matrix* in a gene regulatory network.

Model of Stochastic Gene Regulatory Networks

As in [31], we let $G_1, G_2, ..., G_n$ represent genes in a regulatory network. Let $x_1(t), x_2(t), ..., x_n(t)$ be the state of each gene at time t, where $x_i(t) = 0$ if the gene is off at time t, and $x_i(t) = 1$ if the gene is on at time t. Let $x(t) = [x_1(t),\ x_2(t),\ ...,\ x_n(t)]$.

Denote all possible states of gene G_i by a set X_i, $i = 1, 2, ..., n$. In our case, we have $X_i = \{0, 1\}$, $i = 1, 2, ..., n$. Then the state space of this gene regulatory network, X, is the Cartesian product of all potential spaces of the genes. In symbols, we have $X = X_1 \times X_2 \times \cdots \times X_n$.

Let $f_i : X_i \to X_i$ be the *regulatory rule* for gene G_i, $i = 1, 2, ..., n$. Also, let $p_i^{\uparrow}$ and $p_i^{\downarrow}$ ($p_i^{\uparrow}, p_i^{\downarrow} \in [0, 1]$) be the *activation* and *degradation propensities*, respectively, for gene G_i. The activation (degradation) propensity is the probability that the gene G_i will be activated (degraded) according to the update rule.

The *update rule* to calculate transition probabilities, $\pi_{i,x(t)}$, of the Markov process is defined as follows:

$$\pi_{i,x(t)}(x_i(t+1) = f_i(x(t))) = \begin{cases} p_i^{\uparrow} & \text{if } x_i(t+1) < f_i(x(t)) \\ p_i^{\downarrow} & \text{if } x_i(t+1) > f_i(x(t)) \\ 1 & \text{if } x_i(t+1) = f_i(x(t)) \end{cases},$$

and

$$\pi_{i,x(t)}(x_i(t+1) = x_i(t)) = \begin{cases} 1 - p_i^{\uparrow} & \text{if } x_i(t+1) < f_i(x(t)) \\ 1 - p_i^{\downarrow} & \text{if } x_i(t+1) > f_i(x(t)) \\ 1 & \text{if } x_i(t+1) = f_i(x(t)) \end{cases} \cdot$$

Now, denote the propensity matrix by

$$p = \begin{bmatrix} p_1 & \dots & p_{2n-1} \\ p_2 & \dots & p_{2n} \end{bmatrix},$$

where p_{2i-1} is G_i's activation propensity and p_{2i} is the propensity of G_i's deactivation, $i = 1, 2, ..., n$. The dynamics of a gene regulatory network is a Markov process. Once the regulatory rules and the propensities are given, the *transition matrix* of the Markov process is known. We denote the transition matrix by P, which is a $2^n \times 2^n$ matrix.

An Example

Let $n = 2$, $X = \{0, 1\} \times \{0, 1\}$, and $F = \{f_1, f_2\} : X \to X$. Table 2 represents the regulatory rule for G_1 and G_2, and the propensity matrix is

$$p = \begin{bmatrix} 0.1 & 0.5 \\ 0.2 & 0.9 \end{bmatrix},$$

as given in [31]. We have the probabilities

$$Pr(01 \to 10) = 0.1 \times 0.9 = 0.09, Pr(01 \to 00) = (1 - 0.1)(0.9) = 0.81,$$

$$Pr(01 \to 01) = (1 - 0.1)(1 - 0.9) = 0.09, Pr(01 \to 11) = (0.1)(1 - 0.9) = 0.01,$$

$$Pr(10 \to 10) = (1 - 0.2)(1 - 0.5) = 0.4, Pr(10 \to 01) = 0.2 \times 0.5 = 0.1,$$

$$Pr(10 \to 00) = 0.2 \times (1 - 0.5) = 0.1, Pr(10 \to 11) = (1 - 0.2) \times 0.5 = 0.4,$$

$$Pr(11 \to 11) = 1 \times (1 - 0.9) = 0.1, Pr(11 \to 10) = 1 \times 0.9 = 0.9,$$

$$\text{and } Pr(00 \to 00) = 1 \times 1 = 1.$$

The transition matrix is in Table 3.

G_1	G_2	f_1	f_2
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	0

Table 2: Regulatory Rule for the Example

		output			
		00	01	10	11
	00	1	0	0	0
	01	0.81	0.09	0.09	0.01
input	10	0.1	0.1	0.4	0.4
	11	0	0	0.9	0.1

Table 3: Transition Matrix of Two-Gene Regulatory Network Example

The algorithm illustrating the details of these calculations is given in Appendix A.

For a two-gene regulatory network, f_1 and f_2 would have $2^{2^2} = 16$ different combinations, respectively. Thus, in total, there are $16^2 = 256$ different gene regulatory rules.

In this study, we propose that $p_1, p_2, ..., p_{2n}$ be distributed according to a beta distribution, $B(\alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)};\ (0 \le x \le 1)$, and we assume that the regulatory rule for the network is known. Once α and β are known, we can regenerate propensities that follow $B(\alpha, \beta)$. With the propensities and the regulatory rule, we can regenerate the network. Therefore, the network can be modeled by the parameters α and β.

Consequently, we are aiming to find the best α and β in a given network. By best, we mean α and β can generate propensities that vary the least from the real ones.

CHAPTER V

PARAMETER ESTIMATION METHODS

As previously noted, we will consider the $B(\alpha, \beta)$ distribution, from which we generate values of $p_1, p_2, ..., p_{2n}$, and we assume that the regulatory rule for the network is known. Hence, the network can be modeled by the parameters α and β. The beta distribution is appropriate since its domain is $[0, 1]$ and it can exhibit a broad range of shapes accommodating a wide spectrum of propensity behaviors.

Our goal is to measure the variability inherent in a network generated by a given set of propensities. We achieve this by optimizing α and β so that the difference between a network generated by the beta distribution and the network generated by the fixed propensities is minimum. Forthwith a simulated network will be referred to as a *rebuilt network*, and the network generated by the fixed propensities will be referred to as the *fixed network*.

We define the *error of estimation* using the normalized difference between a fixed network and its rebuilt network by

$$E = \sum_{i,j=1}^{2^n} \frac{q_{ij}^2}{2^n},$$

where q_{ij} is the element from the ith row and jth column of $P_0 - P$, P_0 and P being the transition matrices of the fixed and the rebuilt networks, respectively. The value of E is between 0 and 2. The error serves as the criterion for assessing whether or not the estimation is good. Later in this chapter, we use errors as the fitness functions in our evolutionary algorithms. The smaller the error is, the better the estimation.

Furthermore, as an additional aspect of this study, we will compare the performances of different estimation techniques in terms of bias, computational cost, and efficacy. Namely, we will use the method of moment estimation (MME), maximum likelihood estimation (MLE), estimation based on engineered genetic algorithms, and estimation based on particle swarm optimizations. Further details of these algorithms can be found in Appendix B.

We start with a two-gene regulatory network. We consider four cases:

- High Propensities: All propensities are close to 1. This implies that the network is functioning efficiently. Every element does what it is supposed to do most of the time.

- Low Propensities: All propensities are close to 0. This implies that the network is functioning inefficiently. Every element is unlikely to do what it is supposed to do.

- Medium Propensities: All propensities are close to 0.5. This implies that every element's action is as likely to succeed as to fail.

- Spread Propensities: Propensities have a wide range. This implies that some elements are acting efficiently while others are flawed.

For example, a network with high propensities would most likely correspond to a left-skewed beta distribution. Similarly, a network with spread propensities would most likely correspond to a uniform distribution, where all propensity values are equally likely.

Here, we examine the four types of propensities for all possible 256 regulatory rules for a two-gene regulatory network.

Estimation of a Network from a Propensity Matrix

We begin with a fixed propensity matrix. Our goal is to obtain the parameters of the beta distribution that are most likely to generate the given propensities.

We calculate the errors from our simulations. The steps are as follows: we simulate all possible networks for a two-gene regulatory network. To draw a comparison, we generate propensities from a uniform distribution on $[0, 1]$ and use them to rebuild networks and calculate their errors. These are errors from random guesses and will be referred to as *random errors* in the future. A good estimation method should provide errors far less than the random errors. The random errors' calculation procedure is in Figure 5. We list how to calculate estimation errors in Figure 6. In practice, we repeat the whole process multiple times and use the mean of estimation errors to represent the estimation error. This way, we can avoid random outliers.

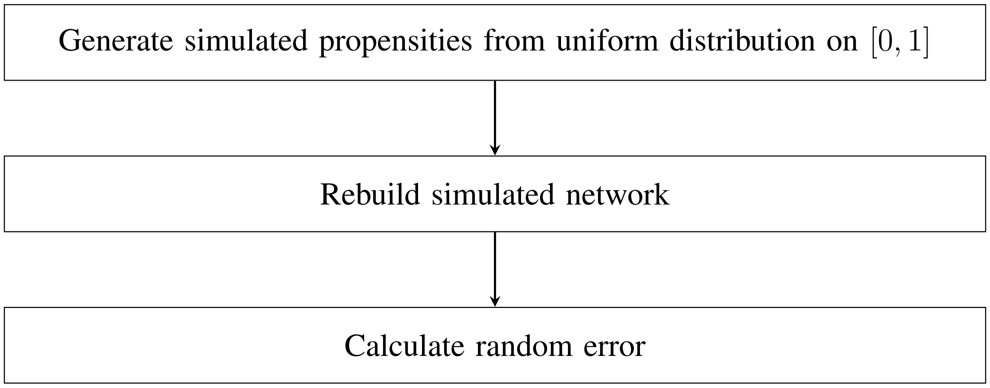
Generate simulated propensities from uniform distribution on $[0, 1]$
Rebuild simulated network
Calculate random error

Figure 5: Calculation of Random Error.
In practice, we do it multiple times and employ their mean value as the error.

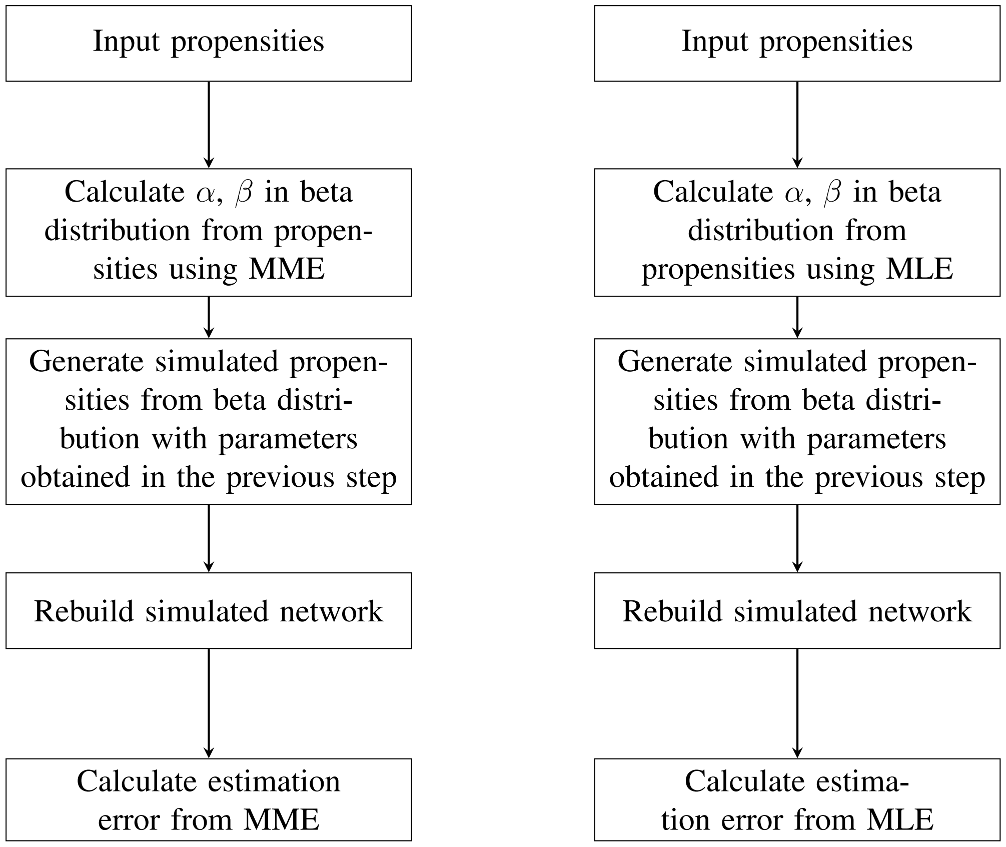
Input propensities
Calculate α, β in beta distribution from propensities using MME
Generate simulated propensities from beta distribution with parameters obtained in the previous step
Rebuild simulated network
Calculate estimation error from MME
Input propensities
Calculate α, β in beta distribution from propensities using MLE
Generate simulated propensities from beta distribution with parameters obtained in the previous step
Rebuild simulated network
Calculate estimation error from MLE

Figure 6: Estimation Error Calculation Procedure for MME (left) and MLE (right) Based on Propensities

In Figure 7, plots A, B, and C show that errors from both MLE and MME are very close to 0, and they overlap a lot, implying these two estimations are precise. Their performance is less satisfying for spread propensities (Figure 7 D). One possible reason is that there are only four elements in each propensity matrix, thus there is not enough information for statistical inference.

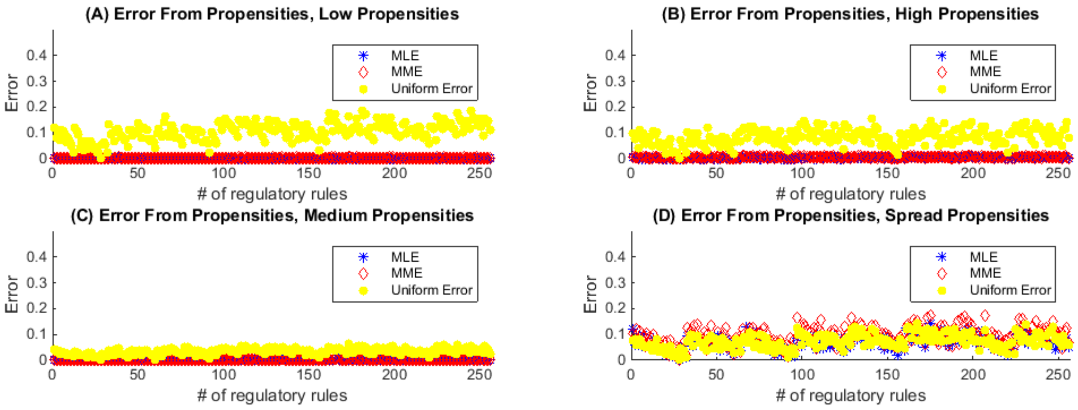

Figure 7: Error From MME and MLE Based on Propensities Compared to Random Error. A: Errors from Low Propensities; B: Errors from High Propensities; C: Errors from Medium Propensities; D: Errors from Spread Propensities. In every subplot of this figure, the filled yellow round dots mark random errors in all possible 256 regulatory networks, each of which corresponds to a regulatory rule labeled by a number between 1 and 256; blue stars mark errors from MLE; red hollow diamonds mark errors from MME.

Estimation of Propensities Based on Transition Matrices

The previous section shows that the method of moment estimation and maximum likelihood estimations based on propensities are precise. However, propensities are hard to measure in wetlab experiments. In contrast, transition probabilities are easier to obtain. Thus, we base our estimation methods on elements from transition matrices. In this section, we focus on a scenario where propensities are unknown, but transition probabilities are known. We will still use random errors defined in the last section for comparison.

It is easy to see in Figure 8 that errors of estimations based on transition probabilities are significantly greater, especially for networks with high propensities (Figure 8 B). This is not surprising, since our hypothesis is that propensities, not transition probabilities, follow a beta distribution. Thus, we conclude that it is inappropriate to estimate beta distribution parameters from transition matrices using maximum likelihood estimations or method of moment estimations.

Optimization Based on Transition Matrices

From previous experiments, we conclude that beta distributions directly estimated from propensities generally work well, while this is not true for estimations based on transition probabilities. Sometimes it is even worse than a random guess. Therefore, in this section, we propose more sophisticated estimation methods. We consider errors as the goal functions that need to be minimized, and α, β are variables.

We use particle swarm optimizations (PSO) and engineered genetic algorithms (engineered GA) to achieve our goal. Figure 9 shows errors of PSO and engineered GA from spread propensities under different regulatory rules, as they are the ones that give the biggest errors in the previous discussion (Figure 7). It is shown that PSO gives errors that are almost the same as errors from MLEs based on propensities in different networks. Engineered genetic algorithms give even better results: the errors are almost zero.

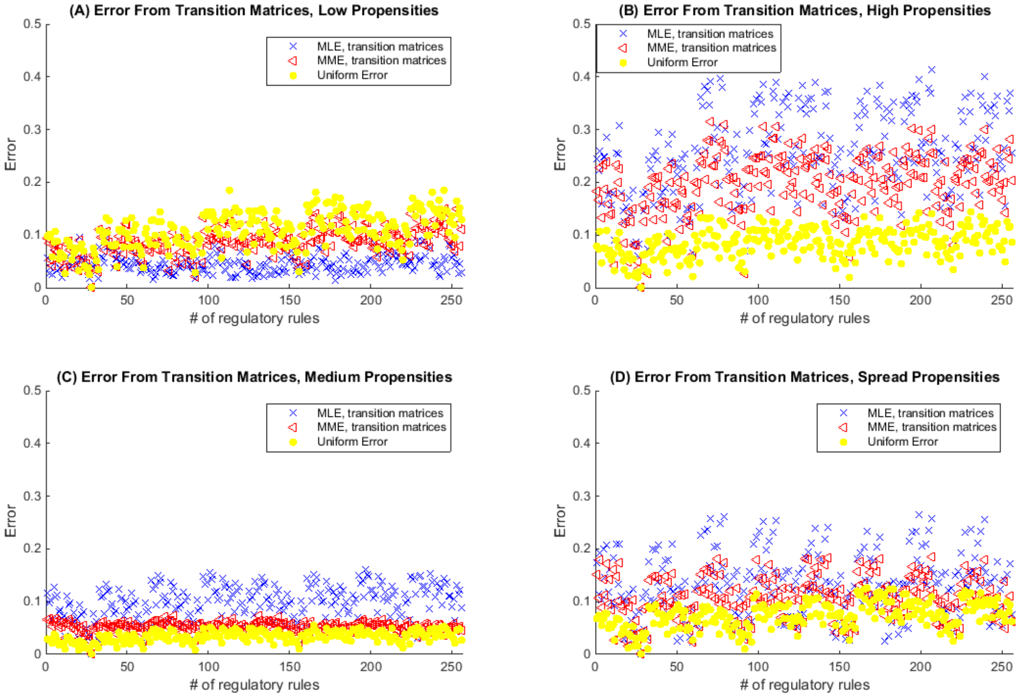

Figure 8: Error From MME and MLE Based on Propensities and Error From MME and MLE Based on Transition Matrix Compared to Random Error.
A: Errors from Low Propensities; B: Errors from High Propensities; C: Errors from Medium Propensities; D: Errors from Spread Propensities. In every subplot of this figure, the yellow filled round dots mark random errors in all possible 256 regulatory networks, each of which corresponds to a regulatory rule labeled by a number between 1 and 256; blue crosses mark errors from MLE; red hollow triangle mark errors from MME.

Engineered genetic algorithms show similar results for low propensities, high propensities, and medium propensities.

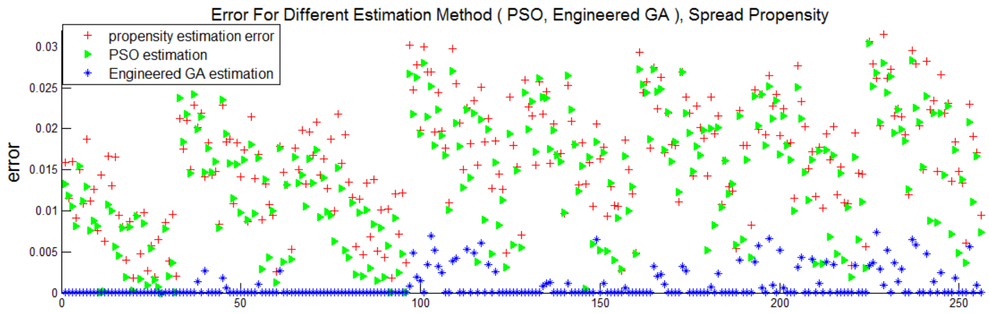

Figure 9: Error in Reconstructed Networks from Three Different Methods, Spread Propensity

Figure 10 shows the probability density functions from different estimation methods for the 256 networks under different regulatory rules based on a spread propensity matrix. The thick yellow dotted lines show the probability density functions estimated based on maximum likelihood estimation of propensities. Figure 11 shows probability density functions from different estimation methods for 256 networks with different regulatory rules with low, high, and medium propensities. The thick black dotted lines show the probability density functions estimated based on maximum likelihood estimation of propensities. It's obvious that they are almost the same, and the engineered genetic algorithm gives even better estimations.

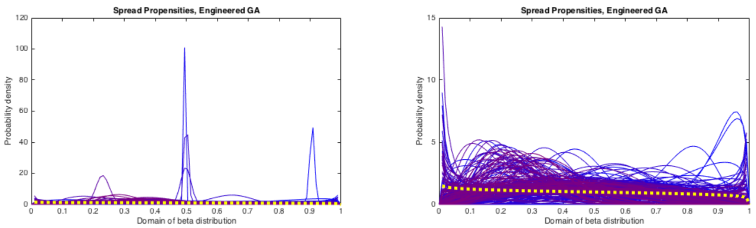

Figure 10: Probability Density Function from PSO and Engineered GA, Spread Propensity

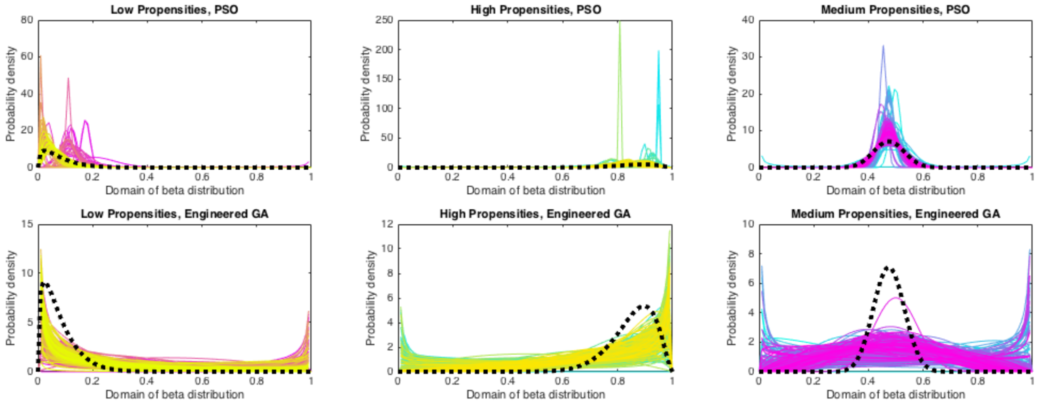

Figure 11: Probability Density Function from PSO and Engineered GA, Low, High, Medium Propensities

Furthermore, we test engineered genetic algorithms on a four-gene network, a five-gene network, and a nine-gene network. Table 4 and Figures 12, 13, 14 show the errors and probability density functions of beta distributions based on maximum likelihood estimation of propensities and engineered genetic algorithms based on transition matrices. With an increasing number of genes, the errors decrease. This is as expected, since more genes in a network mean more propensities, thus we have more information on our beta distribution. Consequently, parameter inference is more precise.

However, the increase in precision comes with increasing computational cost. In a two-gene regulatory network, finding α and β takes only hours, while in a nine-gene regulatory network, finding α and β can take a week. Tuning parameters in genetic algorithms using simulated annealing algorithms is especially problematic. This process sometimes takes more than two weeks.

Type of Propensities	Low	High	Medium	Spread
Four-Gene Network	4.36×10^{-5}	5.98×10^{-5}	1.02×10^{-5}	3.05×10^{-5}
Five-Gene Network	3.07×10^{-5}	1.17×10^{-5}	3.18×10^{-6}	1.21×10^{-5}
Nine-Gene Network	4.77×10^{-7}	7.25×10^{-7}	6.8×10^{-8}	1.24×10^{-6}

Table 4: Errors for Multi-Gene Regulatory Networks from Engineered Genetic Algorithm Based on Transition Matrices

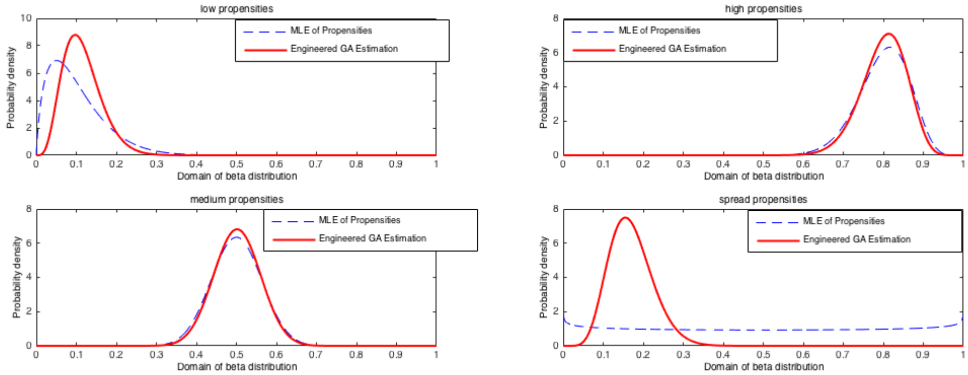

Figure 12: Probability Density Function Estimations of Four-Gene Regulatory Network, Engineered Genetic Algorithm

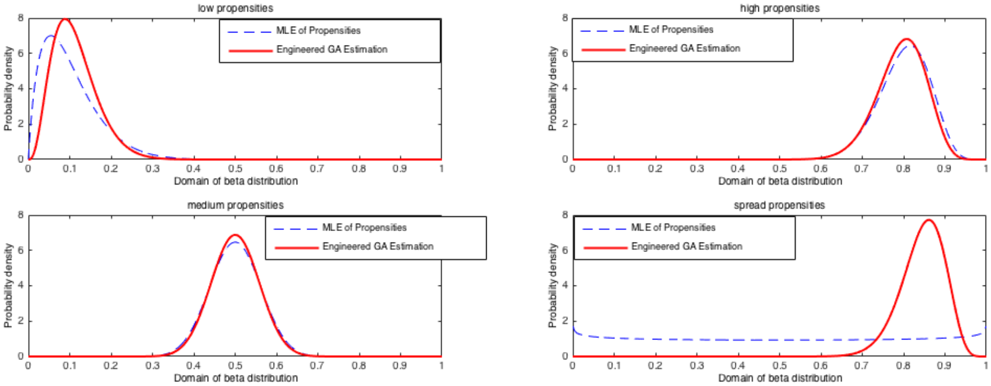

Figure 13: Probability Density Function Estimations of Five-Gene Regulatory Network, Engineered Genetic Algorithm

Among all particle swarm optimization results, regulatory rule 28 (Table 5) gives the minimum error. Its network dynamic corresponds to a Markov process with two absorbing states. Gene regulatory networks like this will eventually evolve to a fixed state. Thus, the initial state of this Markov process has no influence on the final state of any of the networks. Hence, estimations give small errors in every network. On the other hand, regulatory rules 197 (Table 6) and 207 (Table 7) give very big errors. These two Markov processes will converge to periodic solutions. Their oscillatory trajectories make the

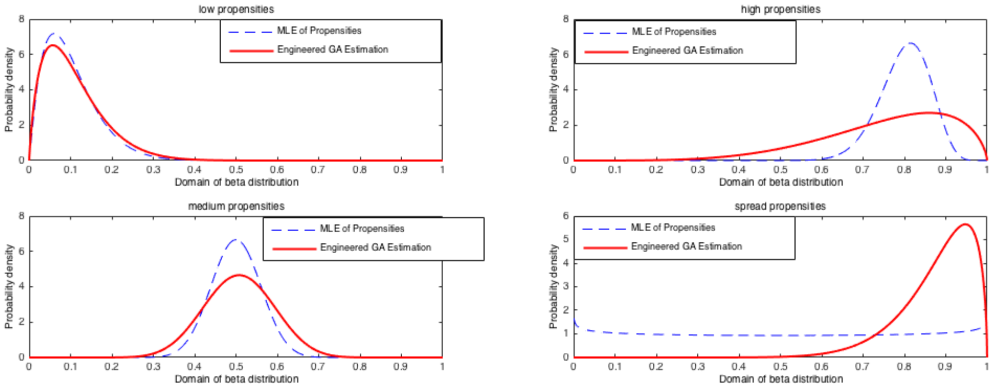

Figure 14: Probability Density Function Estimations of Nine-Gene Regulatory Network, Engineered Genetic Algorithm

errors (*object functions* in optimizations) change sharply when propensities change (Figure 15), which explains why the errors are big. A non-smooth object function is one of the worst scenarios in optimization problems [32]. Regulatory rule 28 gives a perfect object function compared to the regulatory rule 197 or the regulatory rule 207.

G_1	G_2	f_1	f_2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	0	1

Table 5: Regulatory Rule 28

G_1	G_2	f_1	f_2
0	0	1	1
0	1	0	0
1	0	0	1
1	1	0	0

Table 6: Regulatory Rule 197

G_1	G_2	f_1	f_2
0	0	1	1
0	1	0	0
1	0	1	1
1	1	1	0

Table 7: Regulatory Rule 207

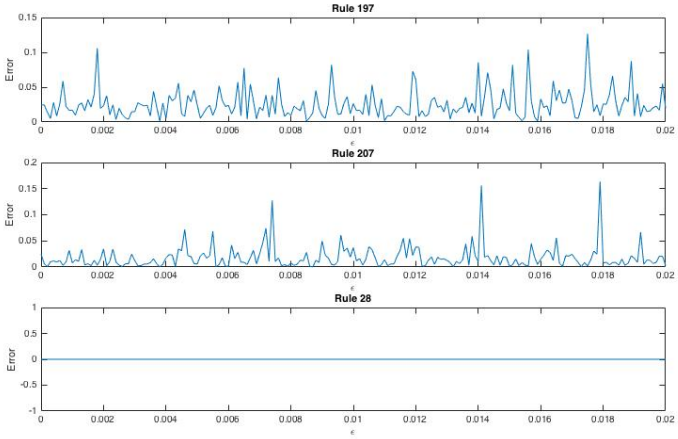

Figure 15: Object Functions Based on Different Regulatory Rules.
Rule 197 and rule 207 Are regulatory rules whose fixed networks converge to oscillatory trajectories. The x-axis shows the disturbance in the transition matrix, and the y-axis shows how the object function (error) changes.

CHAPTER VI

APPLICATION TO THE P53-MDM2 NETWORK

As an application of our method, we study the p53-Mdm2 network in human cells, which describes the dynamics of the tumor suppressor protein, p53, and its negative regulator, Mdm2, when DNA damage occurs, as illustrated in Figure 16 [9, 27]. Many cancer cells show loss of function of p53. Mdm2 is the product of an oncogene, *MDM2*. Mdm2 inactivates p53-mediated transcription [28].

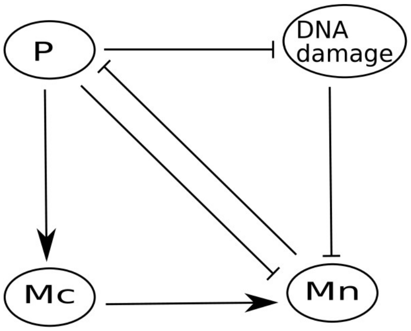

Figure 16: Four-Variable Model for the P53-Mdm2 Regulatory Network.
P, Mc, and Mn Stand For Protein p53, Cytoplasmic Mdm2, and Nuclear Mdm2, Respectively.

Mc and Mn stand for cytoplasmic Mdm2 and nuclear Mdm2, respectively. DNA damage caused by ionic irradiation decreases the level of nucleic Mdm2 that enables p53 to accumulate and to remain active, playing a key role in reducing the effect of the damage. There is a negative feedback loop involving three components: p53 increases the level of cytoplasmic Mdm2 by boosting transcription of *MDM2*, which, in turn, increases the level of nucleic Mdm2. Nucleic Mdm2 reduces p53 activity. This model also contains

P	Dam	Mc	Mn	P	Dam	Mc	Mn
0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	1
0	0	1	0	1	0	0	1
0	0	1	1	0	0	0	1
0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	1	1	0	1
0	1	1	1	0	1	0	1
1	0	0	0	1	0	1	0
1	0	0	1	0	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	0	0	1	1
1	1	0	0	1	1	1	0
1	1	0	1	0	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1

Table 8: Regulatory Rule for p53-Mdm2 Regulatory Network.
P is short for p53, Mc is short for cytoplasmic Mdm2, Mn is short for nuclear Mdm2, and Dam is short for DNA damage.

a positive feedback loop involving two components, where p53 inhibits its negative regulator nucleic Mdm2. Note the dual role of p53, as it positively regulates nucleic Mdm2 through cytoplasmic Mdm2. On the other hand, p53 negatively regulates nucleic Mdm2 by inhibiting its translocation [1]. The logical regulatory rule of this network is shown in Table 8.

Here, we still use low, high, medium, and spread propensities to test engineered GA. Figure 17 depicts beta distributions estimated using MLEs based on propensities and engineered GAs based on transition probabilities, respectively. Here, beta distributions estimated by MLEs based on propensities represent the fixed networks. We establish the reasoning in our last chapter. We measure the difference between estimated beta distributions from engineered genetic algorithms based on transition probabilities and estimated beta distributions from MLEs based on propensities by calculating errors from simulations, which we establish in our last chapter. Errors given by engineered GA

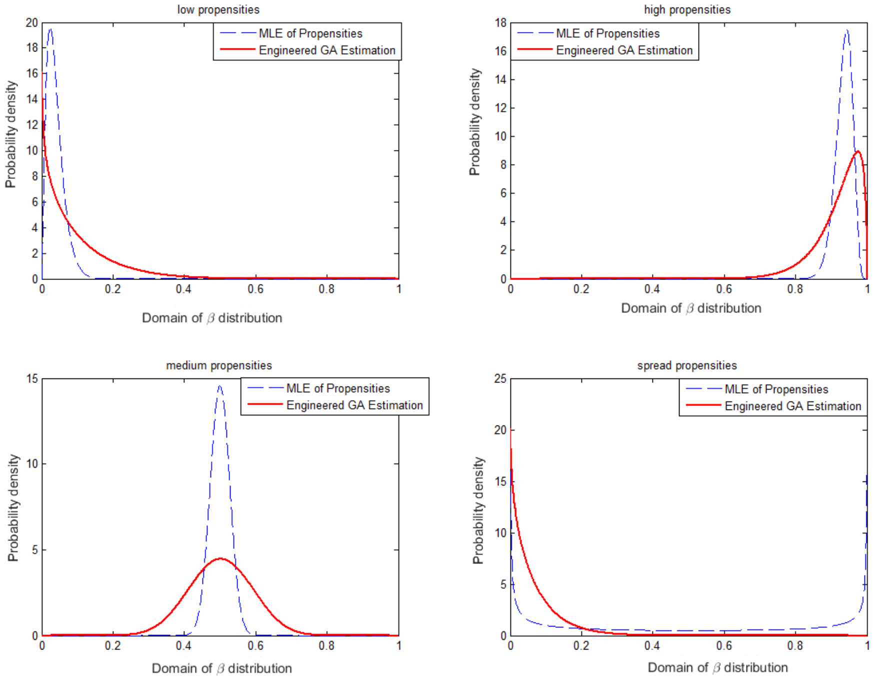

Figure 17: Beta Distribution of Four Genes Regulatory Network, Engineered GA

estimations are 5.6327×10^{-6}, 5.5904×10^{-6}, 5.6787×10^{-6}, and 5.6296×10^{-6} for low, high, medium, and spread propensities, respectively.

Figures 19, 20, 21, 22 show simulation of elements' states of this network using propensities, or propensities follow the estimated beta distributions, from either MLEs or engineered GAs. Every time, our simulation results are plotted separately for four different elements in our networks, and a group of simulations for each network and its estimations is run four times: twice using the original network, once for the rebuilt network from maximum likelihood estimation results, and once from engineered genetic algorithms results. A spike in a graph shows an immediate degradation of an element after its activation. All these graphs show that the rebuilt networks show *qualitatively* similar behavior to the original networks in simulations. By qualitative resemblance, we mean the tendency for an element to stay at a state, or to constantly change states. We see this better when we zoom in (e.g., Figure 18). Here, the generation means how many times we run the simulation. Large generation numbers show an element's behavior in the network in the long run.

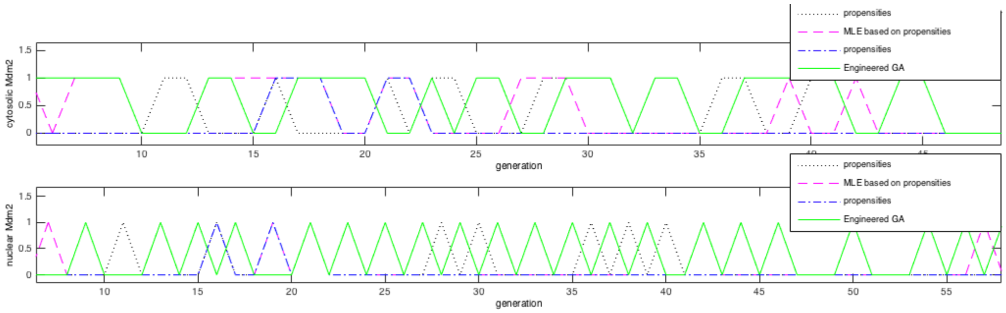

Figure 18: Simulation of Activity of Mdm2 in the p53-Mdm2 Regulatory Network Using Low Propensities, Zoomed in

Figure 18 shows simulations of Mdm2 in the cytoplasm (Mc) and Mdm2 in nuclear (Mn) of the network with low propensities. Cytoplasmic Mdm2 tends to stay at the same activation level for some time while nuclear Mdm2 tends to be deactivated soon after being activated. We consider these as the qualitative resemblance to Mdm2 in the network.

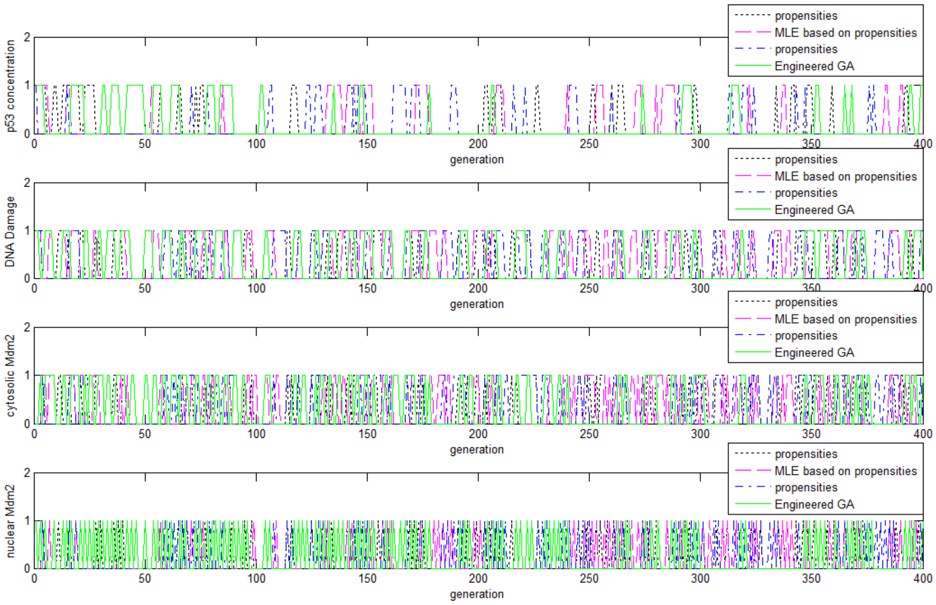

Figure 19: Simulation of Activity of Each Element in the p53-Mdm2 Regulatory Network Using Low Propensities

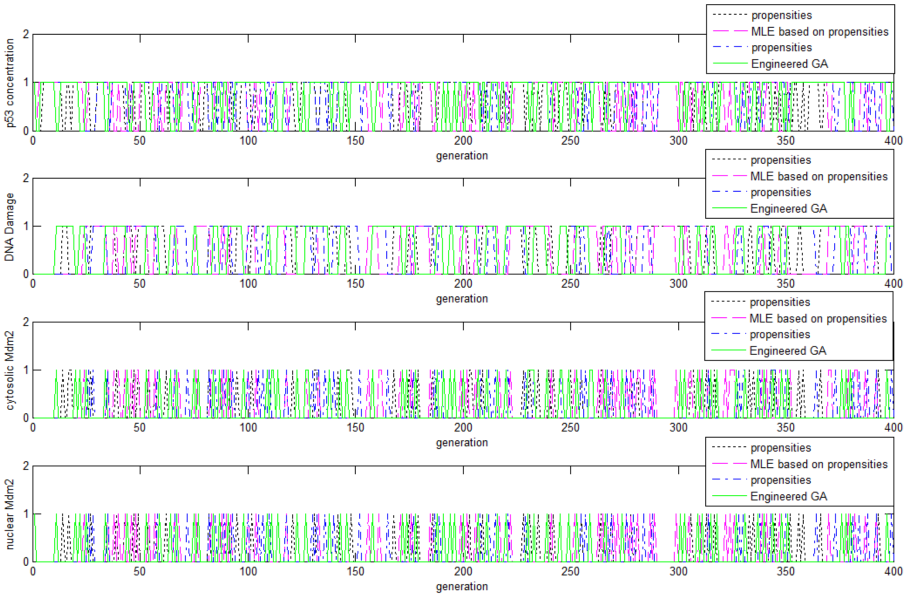

Figure 20: Simulation of Activity of Each Element in the p53-Mdm2 Regulatory Network Using High Propensities

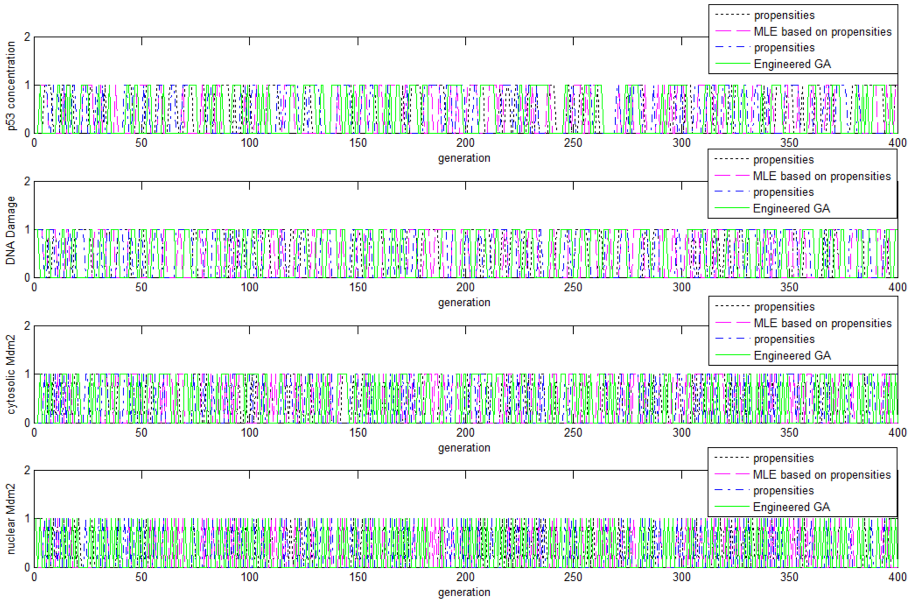

Figure 21: Simulation of Activity of Each Element in the p53-Mdm2 Regulatory Network Using Medium Propensities

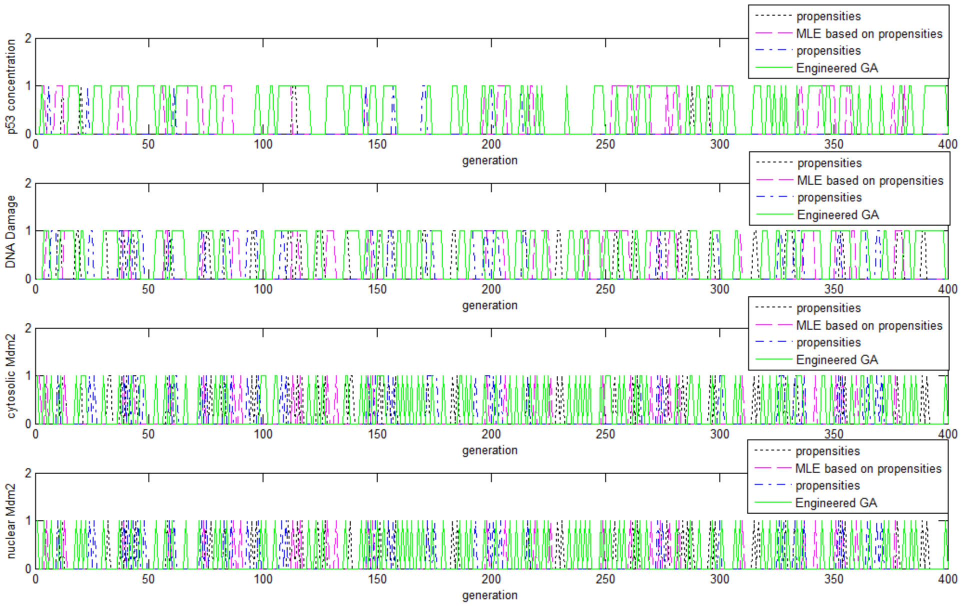

Figure 22: Simulation of Activity of Each Element in the p53-Mdm2 Regulatory Network Using Spread Propensities

CHAPTER VII

CONCLUSIONS AND REMARKS

With this project, we developed a method for estimating the variability on the edges of gene regulatory networks by introducing the stochasticity at the propensity level. We have shown that one can estimate propensities using the knowledge of network edges and regulatory rules. We employed beta distributions to stochasticize both the propensities and the edges. In the process of determining the most likely beta distribution, we utilized the well-known statistical estimation methods, maximum likelihood and method of moments. We also used two novel evolutionary computing methods, engineered genetic algorithm and particle swarm optimization.

Conclusions

As we have shown, one can simulate a network using a fitted beta distribution and obtain realistic variability bounds on the state-to-state probabilities of a network.

We have seen that one can use the edge probabilities and the regulatory rules to estimate the distribution of the propensities, hence yielding the variability of the edges as described above.

It is worth noting the following:

- For small gene networks, due to the limited number of propensities, the maximum likelihood estimates of the beta parameters are not stable. Employing the method of moment estimators is a viable compromise.
- Evolutionary computing methods yield more accurate and precise estimates even if only the regulatory rule and the transition matrix are known.

- Regulatory rules have an impact on the precision of the estimation. Networks that converge to oscillatory behavior, if fixed, tend to have higher estimation errors. However, the engineered genetic algorithm is more robust.

- Generally, for a fixed network and a certain estimation method, the largest error is given by spread propensities.

In our experiments, we noticed that the engineered genetic algorithm is the most robust estimation method. However, the major disadvantage of this algorithm is computational complexity. The algorithm contains two steps: using simulated annealing optimization to tune parameters in genetic algorithms, and using a genetic algorithm to optimize the parameters, α and β, in beta distributions. Both steps are computationally expensive.

Although an engineered genetic algorithm is robust in different optimization situations, it can be sensitive to its parameters. In other words, a bad combination of parameters in an engineered genetic algorithm may lead to a very bad result that gives large error. Since the simulated annealing algorithm is a heuristic algorithm, there is a potential weakness for the engineered genetic algorithm, though we have not observed it in our experiments.

In our particle swarm optimizations, we used *global neighborhoods*. This choice of neighborhoods makes particles converge fast, many of which are premature convergences sinking into local optima. They work well when there are only two nodes in a network, but results worsen when nodes are added.

We can further discuss applying engineered genetic algorithms to asynchronous schemes, such as some genes moving from one state to another, while the rest stay the same. For example, one element is updated three times more frequently than the other elements in the network. In this situation, the lowest minimal multiple of all the genes' update periods is defined as a *common period*, and under this common period, a *network motif* [39] is formed. Consequently, a new Markov process is created. In this situation, based on what we presented before, it is still a good idea to use an engineered genetic algorithm to find out the best parameters for the beta distribution.

Future Work

Future work will focus on the following extensions of this study, from the modeling and algorithm optimization aspects:

- Extending gene states to more than two states. For example, some genes can have three states: 0 for deactivated, 1 for activated, and 2 for hyperactivated.
- Discussing networks with asynchronous updating schemes. We can further discuss whether engineered genetic algorithm is still a robust and efficient parameter estimation method.
- Preventing particle swarm optimization from premature convergence. For example, replacing global neighborhood by local neighborhood in particle swarm optimization. Then we can compare a particle swarm optimization result to an engineered genetic algorithm optimization result and see which one is better.
- Improving computational performance of engineered genetic algorithms. For example, reducing computational complexity in engineered genetic algorithms, or implementing parallel computing in it. As a consequence, we can further analyze more complex gene regulatory networks and see whether our conclusions hold.
- Exploring how to infer network topology if the transition matrices and the beta distribution that the propensities follow is known.

REFERENCES

[1] Wassim Abou-Jaoudé, Djomangan A Ouattara, and Marcelle Kaufman. From structure to dynamics: frequency tuning in the p53–mdm2 network: I. logical approach. *Journal of Theoretical Biology*, 258(4):561–577, 2009.

[2] Murat Acar, Jerome T Mettetal, and Alexander van Oudenaarden. Stochastic switching as a survival strategy in fluctuating environments. *Nature Genetics*, 40(4):471–475, 2008.

[3] Sever Achimescu and Ovidiu Lipan. Signal propagation in nonlinear stochastic gene regulatory networks. *IEEE Proceedings-Systems Biology*, 153(3):120–134, 2006.

[4] RJ Beckman and GL Tiet jen. Maximum likelihood estimation for the beta distribution. *Journal of Statistical Computation and Simulation*, 7(3-4):253–258, 1978.

[5] Dimitris Bertsimas, John Tsitsiklis, et al. Simulated annealing. *Statistical Science*, 8(1):10–15, 1993.

[6] KO Bowman and LR Shenton. Estimation: Method of moments. *Encyclopedia of Statistical Sciences*.

[7] Dmitri Bratsun, Dmitri Volfson, Lev S Tsimring, and Jeff Hasty. Delay-induced stochastic oscillations in gene regulation. *Proceedings of the National Academy of Sciences of the United States of America*, 102(41):14593–14598, 2005.

[8] Claudine Chaouiya, Elisabeth Remy, Brigitte Mossé, and Denis Thieffry. Qualitative analysis of regulatory graphs: a computational tool based on a discrete formal framework. In *Positive Systems*, pages 119–126. Springer, 2003.

[9] Andrea Ciliberto, Béla Novák, and John J Tyson. Steady states and oscillations in the p53/mdm2 network. *Cell Cycle*, 4(3):488–493, 2005.

[10] Maurice Clerc. Stagnation analysis in particle swarm optimization or what happens when nothing happens. *Online at http://clerc. maurice. free. fr/pso*, 2006.

[11] John E Dowd and Douglas S Riggs. A comparison of estimates of michaelis-menten kinetic constants from various linear transformations. *J. Biol. Chem*, 240(2):863–869, 1965.

[12] Wesam Elshamy, Hassan M Emara, and Ahmed Bahgat. Clubs-based particle swarm optimization. In *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 289–296. IEEE, 2007.

[13] Naama Geva-Zatorsky, Nitzan Rosenfeld, Shalev Itzkovitz, Ron Milo, Alex Sigal, Erez Dekel, Talia Yarnitzky, Yuvalal Liron, Paz Polak, Galit Lahav, et al. Oscillations and variability in the p53 system. *Molecular Systems Biology*, 2(1), 2006.

[14] Leon Glass. Classification of biological networks by their qualitative dynamics. *Journal of Theoretical Biology*, 54(1):85–107, 1975.

[15] Sui Huang, Gabriel Eichler, Yaneer Bar-Yam, and Donald E Ingber. Cell fates as high-dimensional attractor states of a complex gene regulatory network. *Physical Review Letters*, 94(12):128701, 2005.

[16] Lester Ingber. Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling*, 18(11):29–57, 1993.

[17] Lester Ingber, Antonio Petraglia, Mariane Rembold Petraglia, Maria Augusta Soares Machado, et al. Adaptive simulated annealing. In *Stochastic Global Optimization and its Applications with Fuzzy Adaptive Simulated Annealing*, pages 33–62. Springer, 2012.

[18] DJ Irons. Logical analysis of the budding yeast cell cycle. *Journal of Theoretical Biology*, 257(4):543–559, 2009.

[19] François Jacob and Jacques Monod. Genetic regulatory mechanisms in the synthesis of proteins. *Journal of Molecular Biology*, 3(3):318–356, 1961.

[20] Alan Jobe and Suzanne Bourgeois. lac repressor-operator interaction: Vi. the natural inducer of the lac operon. *Journal of Molecular Biology*, 69(3):397IN7405–404IN8408, 1972.

[21] Norman L Johnson, Samuel Kotz, and N Balakrishnan. *Continuous Multivariate Distributions, volume 1, Models and Applications*, volume 59. New York: John Wiley & Sons, 2002.

[22] Guy Karlebach and Ron Shamir. Modelling and analysis of gene regulatory networks. *Nature Reviews Molecular Cell Biology*, 9(10):770–780, 2008.

[23] Stuart Kauffman. The large scale structure and dynamics of gene control circuits: an ensemble approach. *Journal of Theoretical Biology*, 44(1):167–190, 1974.

[24] Stuart Kauffman. A proposal for using the ensemble approach to understand genetic regulatory networks. *Journal of Theoretical Biology*, 230(4):581–590, 2004.

[25] Stuart A. Kauffman. *The Origins of Order: Self Organization and Selection in Evolution*. Oxford University Press, 1993.

[26] Scott Kirkpatrick, MP Vecchi, et al. Optimization by simmulated annealing. *Science*, 220(4598):671–680, 1983.

[27] Galit Lahav, Nitzan Rosenfeld, Alex Sigal, Naama Geva-Zatorsky, Arnold J Levine, Michael B Elowitz, and Uri Alon. Dynamics of the p53-mdm2 feedback loop in individual cells. *Nature Genetics*, 36(2):147–150, 2004.

[28] Arnold J Levine. p53, the cellular gatekeeper for growth and division. *Cell*, 88(3):323–331, 1997.

[29] Vladimiro Miranda, Hrvoje Keko, and Alvaro Jaramillo Duque. Stochastic star communication topology in evolutionary particle swarms (epso). *International Journal of Computational Intelligence Research*, 4(2):105–116, 2008.

[30] Melanie Mitchell. *An introduction to genetic algorithms*. MIT Press, 1998.

[31] David Murrugarra, Alan Veliz-Cuba, Boris Aguilar, Seda Arat, and Reinhard Laubenbacher. Modeling stochasticity and variability in gene regulatory networks. *EURASIP Journal on Bioinformatics and Systems Biology*, 2012(1):1–11, 2012.

[32] Yu Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.

[33] Ertugrul M Ozbudak, Mukund Thattai, Iren Kurtser, Alan D Grossman, and Alexander van Oudenaarden. Regulation of noise in the expression of a single gene. *Nature Genetics*, 31(1):69–73, 2002.

[34] Jonathan M Raser and Erin K O'Shea. Noise in gene expression: origins, consequences, and control. *Science*, 309(5743):2010–2013, 2005.

[35] William T Reeves. Particle systemsa technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics (TOG)*, 2(2):91–108, 1983.

[36] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM Siggraph Computer Graphics*, 21(4):25–34, 1987.

[37] Andre S Ribeiro and Stuart A Kauffman. Noisy attractors and ergodic sets in models of gene regulatory networks. *Journal of Theoretical Biology*, 247(4):743–755, 2007.

[38] Miguel Rocha and José Neves. Preventing premature convergence to local optima in genetic algorithms via random offspring generation. In *Multiple Approaches to Intelligent Systems*, pages 127–136. Springer, 1999.

[39] Shai S Shen-Orr, Ron Milo, Shmoolik Mangan, and Uri Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature Genetics*, 31(1):64–68, 2002.

[40] François St-Pierre and Drew Endy. Determination of cell fate selection during phage lambda infection. *Proceedings of the National Academy of Sciences*, 105(52):20705–20710, 2008.

[41] David T Suzuki, Anthony JF Griffiths, Jeffrey H Miller, Richard C Lewontin, et al. *An Introduction to Genetic Analysis.* Number Ed. 3. WH Freeman and Company, 1986.

[42] Shunsuke Teraguchi, Yutaro Kumagai, Alexis Vandenbon, Shizuo Akira, and Daron M Standley. Stochastic binary modeling of cells in continuous time as an alternative to biochemical reaction equations. *Physical Review E*, 84(6):062903, 2011.

[43] René Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42(3):563–585, 1973.

[44] René Thomas and Richard d'Ari. *Biological feedback.* CRC press, 1990.

[45] Francesco Vanzi, Chiara Broggio, Leonardo Sacconi, and Francesco Saverio Pavone. Lac repressor hinge flexibility and dna looping: single molecule kinetics by tethered particle motion. *Nucleic Acids Research*, 34(12):3409–3420, 2006.

[46] Denise M Wolf and Frank H Eeckman. On the relationship between genomic regulatory element organization and gene regulatory dynamics. *Journal of Theoretical Biology*, 195(2):167–186, 1998.

[47] Peng-Yeng Yin, Manuel Laguna, and Jia-Xian Zhu. A complementary cyber swarm algorithm. 2011.

APPENDIX A

EXAMPLE OF THE TWO-GENE REGULATORY NETWORK

For $n = 2$ and $X = \{0,1\} \times \{0,1\}$, let $F = \{f_1, f_2\} : \; X \to X$, with propensity matrix

$$p = \begin{bmatrix} 0.1 & 0.5 \\ 0.2 & 0.9 \end{bmatrix}.$$

x_1	x_2	f_1	f_2
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	0

Table A-1: Regulatory rule in this example

Let us focus on the Boolean functions for the first variable, which we denote by $F_1 = \{h_1, h_2, ..., h_r\}$. For the first variable we have transitions as follows:

$$Pr(00 \to 0) = 1,$$

$$Pr(00 \to 1) = 1 - Pr(00 \to 0) = 0,$$

$$Pr(01 \to 0) = 1 - p_1^{\uparrow} = 0.9,$$

$$Pr(01 \to 1) = p_1^{\uparrow} = 0.1,$$

$$Pr(10 \to 0) = p_1^{\downarrow} = 0.2,$$

$$Pr(10 \to 1) = 1 - p_1^{\downarrow} = 0.8,$$

$$Pr(11 \to 1) = 1,$$

and $Pr(11 \to 0) = 1 - Pr(11 \to 1) = 0.$

It is a similar calculation for the second variable:

$$Pr(00 \to 0) = 1,$$

$$Pr(00 \to 1) = 1 - Pr(00 \to 0) = 0,$$

$$Pr(01 \to 0) = p_2^{\downarrow} = 0.9,$$

$$Pr(01 \to 1) = 1 - p_2^{\downarrow} = 0.1,$$

$$Pr(10 \to 0) = 1 - p_2^{\uparrow} = 0.5,$$

$$Pr(10 \to 1) = p_2^{\uparrow} = 0.5,$$

$$Pr(11 \to 1) = 1 - p_2^{\downarrow} = 0.1,$$

and $Pr(11 \to 0) = p_2^{\downarrow} = 0.9.$

Each transition probability is the joint probability of two independent transitions for two genes take place simultaneously. These joint probabilities are

$$Pr(00 \to 00) = Pr(00 \to 0) \times Pr(00 \to 0) = 1,$$

$$Pr(00 \to 01) = Pr(00 \to 0) \times Pr(00 \to 1) = 0,$$

$$Pr(00 \to 10) = Pr(00 \to 1) \times Pr(00 \to 0) = 0,$$

$$Pr(00 \to 11) = Pr(00 \to 1) \times Pr(00 \to 1) = 0,$$

$$Pr(01 \to 00) = Pr(01 \to 0) \times Pr(01 \to 0) = 0.9 \times 0.9 = 0.81,$$

$$Pr(01 \to 01) = Pr(01 \to 0) \times Pr(01 \to 1) = 0.9 \times 0.1 = 0.09,$$

$$Pr(01 \to 10) = Pr(01 \to 1) \times Pr(01 \to 0) = 0.1 \times 0.9 = 0.09,$$

$$Pr(01 \to 11) = Pr(01 \to 1) \times Pr(01 \to 1) = 0.1 \times 0.1 = 0.01,$$

$$Pr(10 \to 00) = Pr(10 \to 0) \times Pr(10 \to 0) = 0.2 \times 0.5 = 0.1,$$

$$Pr(10 \to 01) = Pr(10 \to 0) \times Pr(10 \to 1) = 0.2 \times 0.5 = 0.1,$$

$$Pr(10 \to 10) = Pr(10 \to 1) \times Pr(10 \to 0) = 0.8 \times 0.5 = 0.4,$$

$$Pr(10 \to 11) = Pr(10 \to 1) \times Pr(10 \to 1) = 0.8 \times 0.1 = 0.4,$$

$$Pr(11 \to 00) = Pr(11 \to 0) \times Pr(11 \to 0) = 0,$$

$$Pr(11 \to 01) = Pr(11 \to 0) \times Pr(11 \to 1) = 0,$$

$$Pr(11 \to 10) = Pr(11 \to 1) \times Pr(11 \to 0) = 1 \times 0.9 = 0.9,$$

and $Pr(11 \to 11) = Pr(11 \to 1) \times Pr(11 \to 1) = 1 \times 0.1 = 0.1.$

Thus, the transition matrix is as follows, and the network looks like Figure A-1.

		output			
		00	01	10	11
	00	1	0	0	0
	01	0.81	0.09	0.09	0.01
input	10	0.1	0.1	0.4	0.4
	11	0	0	0.9	0.1

Here, for more general cases, we give two algorithms: Algorithm 1 is a step-by-step

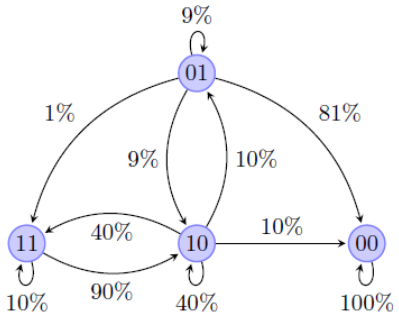

Figure A-1: Example Network from [31]

manual of how to calculate transition matrices given the regulatory rules and propensity matrices. Algorithm 2 is a step-by-step manual of how to simulate the behavior of a given gene regulatory network.

Algorithm 1: Transition probability between two states for stochastic discrete dynamical systems

```
Let $z = F(x) = (f_1(x), ..., f_n(x))$;
for $i = 1$ to $n$ do
    Let $c = 0$ ;
    if $x_i < z_i$ then
        if $y_i = z_i$ then
            $c = p_i^{\uparrow}$
        end
        if $y_i = x_i$ then
            $c = 1 - p_i^{\uparrow}$
        end
    else
        if $x_i < z_i$ then
            if $y_i = z_i$ then
                $c = p_i^{\downarrow}$
            end
            if $y_i = x_i$ then
                $c = 1 - p_i^{\downarrow}$
            end
        else
            if $y_i = x_i$ then
                $c = 1$
            end
        end
    end
    $P_{x,y} = P_{x,y} \times c$
end
```

Algorithm 2: Next state for stochastic discrete dynamical systems

```
input: Initial state $x_0$ and a SDDS $F = \{f_i, p_i^{\uparrow}, p_i^{\downarrow}\}_{i=1}^{n}$ ;
output:$y =$ one of the next states of $x$ Let $z = F(x_0) = (f_1(x_0), ..., f_n(x_0))$;
for $i = 1$ to $n$ do
    Let $r$ be a random number in $[0, 1]$ ;
    if $x_i < z_i$ then
        if $r < p_i^{\uparrow}$ then
            $y_i = z_i$ with probability $p_i^{\uparrow}$
        else
            $y_i = x_i$ with probability $1 - p_i^{\uparrow}$
        end
    else
        if $x_i < z_i$ then
            if $r < p_i^{\downarrow}$ then
                $y_i = z_i$ with probability $p_i^{\downarrow}$
            end
            $y_i = x_i$ with probability $1 - p_i^{\downarrow}$
        else
            $y_i = x_i$ with probability $1$
        end
    end
end
```

APPENDIX B

PARAMETER ESTIMATION METHODS

Method of Moment Estimation

The method of moment is one of the most commonly used methods of point estimations.

Suppose that the problem is to estimate k unknown parameters $\theta_1, \theta_2, \ldots, \theta_k$ characterizing the distribution $f_W(w; \theta)$ of the random variable W. Also suppose that the first k moments of the true distribution can be expressed as functions of the θs:

$$\mu_1 \equiv E[W] = g_1(\theta_1, \theta_2, \ldots, \theta_k),$$

$$\mu_2 \equiv E[W^2] = g_2(\theta_1, \theta_2, \ldots, \theta_k),$$

$$\vdots$$

$$\mu_k \equiv E[W^k] = g_k(\theta_1, \theta_2, \ldots, \theta_k).$$

Let a sample of size n be drawn, resulting in the values $w_1, \ldots, w_n$. For $j = 1, \ldots, k$, let

$$\hat{\mu}_j = \frac{1}{n} \sum_{i=1}^{n} w_i^j$$

be the jth sample moment, an estimate of μ_j. The method of moments estimator for $\theta_1, \theta_2, \ldots, \theta_k$, denoted by $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_k$, is defined as the solution (if there is one) to the equations:

$$\hat{\mu}_1 = g_1(\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_k),$$

$$\hat{\mu}_2 = g_2(\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_k),$$

$$\vdots$$

$$\hat{\mu}_k = g_k(\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_k).$$

By solving the equations above, we can utilize the method of moments estimators [6].

For a beta distribution, two moments are enough to estimate the parameters α and β. We have $\mu = \frac{\alpha}{\alpha+\beta}$ and $\sigma^2 = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$, thus

$$\alpha = \bar{x}\big(\frac{\bar{x}(1-\bar{x})}{s^2} - 1\big),$$

$$\text{and } \beta = (1-\bar{x})\big(\frac{\bar{x}(1-\bar{x})}{s^2} - 1\big).$$

Here $\bar{x}$ and s^2 are the mean and variance of the sample, respectively [21].

Maximum Likelihood Estimation

For the beta distribution, there is no closed form of its maximum likelihood estimation, as the differentiation of the beta function is written using a digamma function. The secant method is one of the numerical approaches to calculate the maximum likelihood estimation of parameters in a beta distribution [4].

Denote the probability function of beta distribution by

$$f(y) = \frac{(y-a)^{p-1}(b-y)^{q-1}}{B(p,q)(b-a)^{p+q-1}};\ (a \le y \le b).$$

The maximum likelihood equations for the estimators $\hat{p}$ and $\hat{q}$ are as follows:

$$\psi(\hat{p}) - \psi(\hat{p}+\hat{q}) = \text{ln}G_1,$$

$$\text{and } \psi(\hat{q}) - \psi(\hat{p} + \hat{q}) = \text{ln}G_2,$$

where

$$G_1 = \prod_{i=1}^{n} \sqrt[n]{\frac{Y_i - a}{b - a}},\ G_2 = \prod_{i=1}^{n} \sqrt[n]{\frac{b - Y_i}{b - a}},$$

$\psi(x)$ is the digamma function (the derivative of the logarithm of the gamma function), and Y_i $(i = 1, 2, ..., n)$ are observed values for y in the beta distribution.

From previous functions, it is easy to see that $\psi(\hat{p}) = \text{ln}(G_1) + \psi(\hat{p} + \hat{q})$, $\psi(\hat{q}) - \text{ln}(G_1) = \psi(\hat{p} + \hat{q})$, and $\psi(\hat{p}) = \text{ln}G_1 - \text{ln}G_2 + \psi(\hat{q})$, thus

$$\hat{p} = \psi^{-1}(\text{ln}G_1 - \text{ln}G_2 + \psi(\hat{q})).$$

Here, we have

$$\psi(\hat{q}) - \psi\{[\psi^{-1}(\text{ln}G_1 - \text{ln}G_2 + \psi(\hat{q}))] + \hat{q}\} - \text{ln}G_2 = 0.$$

Evaluating $\psi^{-1}(z)$ is equivalent to finding the root c of the equation $\psi(c) - z = 0$, and this is accomplished by using the approximation

$$\psi(z) \sim \text{ln}z - \frac{1}{2}z - \frac{1}{12}z^2 + \frac{1}{120}z^4 - \frac{1}{252}z^6$$

for $z \geq 3$ and the recurrence formula

$$\psi(z) = \psi(z + 1) - \frac{1}{z}$$

for $z < 3$. These approximations generally yield 6 decimal places accuracy [4].

Genetic Algorithm

Genetic algorithms simulate natural selection. Every parameter is considered a gene. Every individual has a genotype, which is encoded into a chromosome and corresponds to a phenotype. A fitness function serves as the criterion of an individual's fitness. Natural selection is based on fitness. Figure B-1 is an example of how parameters are encoded into chromosomes [30].

The simplest form of genetic algorithm involves three types of operators, selection, crossover, and mutation [30]:

- Selection: The operator selects chromosomes in the population for reproduction. There is a higher chance for individuals with fitter phenotypes to survive and reproduce.
- Crossover: The operator randomly chooses a locus, and exchanges the subsequences before and after that locus between two chromosomes, to create two offspring. For example, the strings 10000100 and 11111111 could be crossed over after the third locus in each to produce the two offspring: 10011111 and 11100100. The crossover operator roughly mimics biological recombination between two sister chromosomes.
- Mutation: The operator randomly flips some of the bits in a chromosome. For example, the string 00000100 might be mutated in its second position to yield 01000100. Mutation can occur at each bit position in a string with some probability, usually very small.

A fitness function projects phenotype onto fitness. In our case, the smaller the fitness, the better is the phenotype. Between two generations, the old ones are referred to as *parents*, and the new ones are referred to as *children*. Parents with smaller fitness have a better chance of passing on their genotypes to children, which are commonly referred as *elite children*.

There are several different established methods to avoid local optima [38]. The easiest one is to accept non-elite children with a certain probability. This gives genetic algorithms

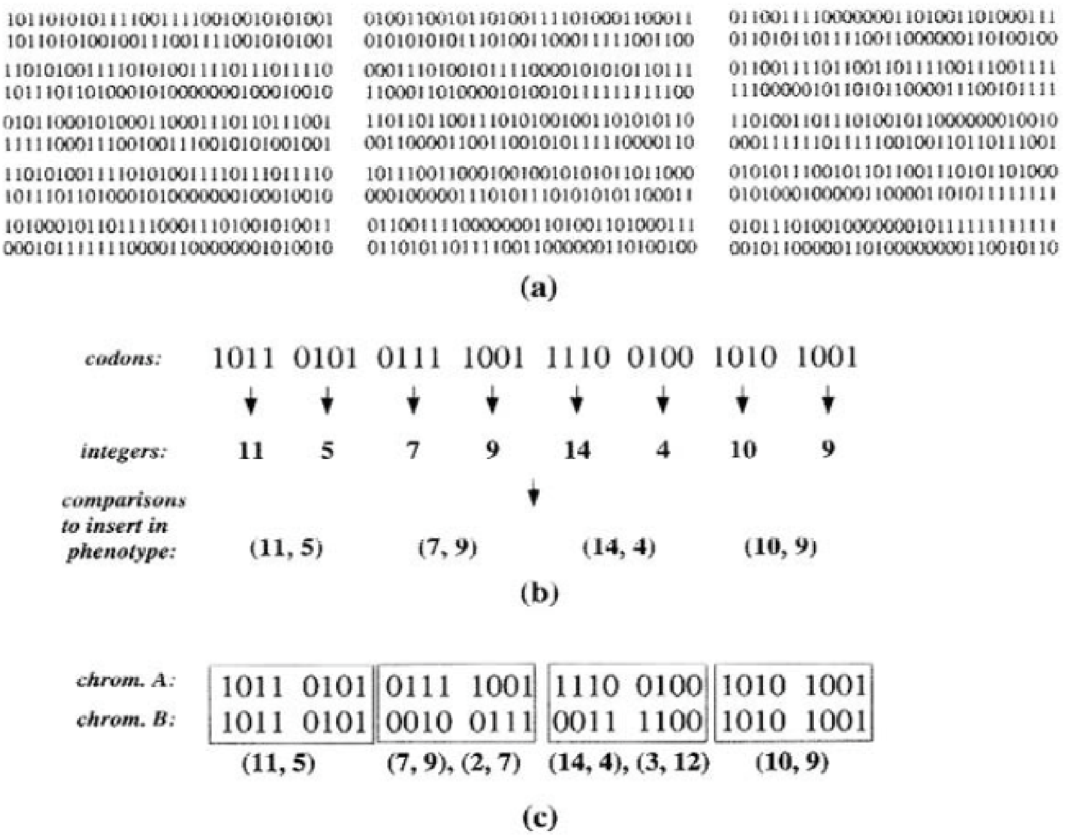

Figure B-1: Transform Parameters into Chromosomes extra flexibility and a better chance to find the global optima.

Simulated Annealing Algorithm

Simulated annealing is a probabilistic method proposed in 1983 [26] to find the global minimum of a cost function that may possess several local minima. It works by emulating the physical process where a solid is slowly cooled, so that when eventually its structure is ”frozen”, it is at its minimum energy configuration.

Basic elements of simulated annealing (SA) include [5]:

- Domain: a finite set S;
- Cost Function: a real-valued function J defined on S;
- Sets of neighbors: $\forall i \in S, S(i) \subset S - \{i\}$;
- A collection of positive coefficients: $\forall i,\ \exists q_{ij}, j \in S(i)$ such that $\sum_{j\in S(i)} q_{ij} = 1$, and $j \in S(i)$ if and only if $i \in S(j)$;
- Cooling Schedule: a nonincreasing function $T : N \rightarrow (0, \infty)$, where $T(t)$ denotes temperature at time t;
- An initial state $x(0) \in S$.

A simulated annealing algorithm is a homogeneous Markov chain if the temperature $T(t)$ has a constant value, T. Assuming that this Markov chain is irreducible and periodic with $q_{ij} = q_{ji}\ \forall i, j$, then the invariant probability distribution is given by

$$\pi_T(i) = \frac{1}{Z_T}\text{exp}\Big[-\frac{J(i)}{T}\Big],\ i \in S,$$

where Z_T is a normalized constant. For $\pi_T(i) > r$, where r is a uniformly distributed random number in $[0, 1]$, the relatively bad result is accepted.

The physical analogy that is used to justify the simulated annealing algorithm is that the cooling rate is low enough for the probability distribution of the current state to be near thermodynamic equilibrium at all times. The ideal cooling rate, however, could be difficult to determine beforehand, as too fast cooling leads to premature convergence, but too slow cooling takes too much time. There are several improvements, such as adaptive simulated annealing algorithms, that have been proposed to solve this problem [17, 16].

Engineered Genetic Algorithm

The difference between an engineered genetic algorithm (engineered GA) and genetic algorithm is that the simulated annealing algorithm is applied to tune the parameters in the genetic algorithm before the genetic algorithm is employed. Tunning parameters makes sure that we use the most efficient genetic algorithm to find the best parameters in the beta distributions.

Particle Swarm Optimization

Particle swarm optimization (PSO) is a population-based stochastic approach for solving both continuous and discrete optimization problems. It simulates a flock of birds.

In particle swarm optimization, every agent, called a *particle*, emulates the behavior of a bird moving in the search space of an optimization problem. The position of a particle represents a candidate solution. Each particle searches for better positions in the search space by changing its velocity according to rules originally inspired by behavioral models of bird flocking [35, 36].

Let us denote a swarm by $P = \{p_1, p_2, ..., p_k\}$. There are several attributes in a given time step:

- A position inside the search space;
- The fitness value at this position;
- A velocity, which will be used to compute the position for the next time;

- An individual memory, which is the fitness value of the previous time step;
- A group memory, which is the best position of the particle's neighbors found so far; it is also referred as the precious best.

The object of optimization is minimizing the fitness function $f : \Theta \to \mathbb{R}, \Theta \in \mathbb{R}^n$, with

$$\Theta^* = \text{arg min} f(\vec{\theta}) = \{\vec{\theta^*} \in \Theta : f(\vec{\theta^*}) \leq f(\vec{\theta})\},$$

where $\vec{\theta}$ is an n-dimensional vector that belongs to the set Θ, the search space for all feasible solutions.

In every time step for each particle, there are updates for both velocity and position. At any time step t, p_i has a position $\vec{x_i}^t$ and velocity $\vec{v_i}^t$, and the individual memory is $\vec{b_i}^t$. The group memory, which is the best position obtained from the particle's neighbors, is denoted by $\vec{l_i}^t$. The update rule for velocity is given by

$$\vec{v_i}^{t+1} = w\vec{v_i}^t + \phi_1 \vec{U_1}^t(\vec{b_i}^t - \vec{x_i}^t) + \phi_2 \vec{U_2}^t(\vec{l_i}^t - \vec{x_i}^t),$$

and the update rule of position is

$$\vec{x_i}^{t+1} = \vec{x_i}^t + \vec{v_i}^{t+1},$$

where w is the inertia weight, and ϕ_1, ϕ_2 are two parameters that function as acceleration coefficients. $\vec{U_1}^t, \vec{U_2}^t$ are two $n \times n$ diagonal matrices of random numbers uniformly distributed in the interval $[0, 1)$. At each iteration, these matrices are regenerated.

In some situations, parameter values are set as

$$w = \frac{1}{2\ln 2} \simeq 0.721, c = \frac{1}{2} + \ln 2 \simeq 1.193$$

[10].

There are different types of neighborhoods in particle swarm optimization, and each of them provides different features in the results and performance to particle swarm optimization. The basic particle swarm optimization uses global neighborhood. In other words, the group memory is the current best location for all the particles. In this version, the algorithm convergences fast, but most of the time it converges to local optima. The other types of the neighborhood have swarms divided into several sub-swarms. One of them is divided based on distance, m-nearest neighborhood. There are other types of division inspired by social structures. If we assume that there is an information link between each particle and its neighbors, then the set of these links builds a graph, or a communications network, which is called the *topology* of the PSO variant. A commonly used social topology is the *ring*, in which each particle has just two neighbors, but there are many other social topology types. The topology varies, and can be adaptive (SPSO, stochastic star, TRIBES, Cyber Swarm, C-PSO) [47, 29, 12]. Figure B-2 shows some topologies of neighborhoods.

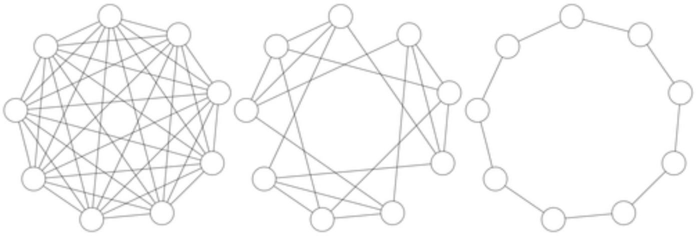

Figure B-2: Different topologies for PSO neighborhoods