AN ALGORITHM TO RECOGNIZE MULTI-STABLE BEHAVIOR FROM AN ENSEMBLE OF STOCHASTIC SIMULATION RUNS

by

Eduardo Monzon

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Dr. Chris Winstead Major Professor	Dr. Reyhan Baktur Committee Member
Dr. Charles Miller Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2013

Abstract

An Algorithm to Recognize Multi-Stable Behavior from an Ensemble of Stochastic Simulation Runs

by

Eduardo Monzon, Master of Science
Utah State University, 2013

Major Professor: Dr. Chris Winstead
Department: Electrical and Computer Engineering

A myriad of methods exist to simulate the time evolution of genetic circuits. The assessment process to verify the behavior of these systems usually involves performing many stochastic simulation runs and performing statistical analysis on the ensemble of simulated paths. Inferring information from this sea of random data is not always easy and the designer usually needs to be trained in stochastic processes to make correct interpretations. To help the biological designer in this duty, this thesis presents a new method to visualize the typical behavior of genetic circuits when they exhibit more than one path. The method is shown to produce correct results with the simulation of a genetic toggle switch circuit.

(79 pages)

Public Abstract

An Algorithm to Recognize Multi-Stable Behavior from an Ensemble of Stochastic Simulation Runs

by

Eduardo Monzon, Master of Science
Utah State University, 2013

Major Professor: Dr. Chris Winstead
Department: Electrical and Computer Engineering

Synthetic biological designers are demanding tools to help with the design and verification process of new biological models. Some of the most common tools available aggregate multiple simulation results into one "clean" trajectory that hopefully is representative of the system's behavior. However, for systems exhibiting multiple stable states, these techniques fail to show all the possible trajectories of the system. This work introduces a method capable of detecting the presence of more than one "typical" trajectory in a system, which can also be integrated with other available simulation tools.

To my family, friends, and my advisor, Dr. Chris Winstead.

Acknowledgments

Many thanks go to my advisor, Dr. Chris Winstead, for his advising and guidance on this research; his support and dedication to my work are very much appreciated. Also, I would like to thank my co-worker and friend, Abiezer Tejeda, for all his help, support, and interesting conversations that sparked many ideas for this research. Also, without his help I would have not been able to finish this work on time. I would also like to thank Dr. Chris Myers and Curtis Kendall at the University of Utah, for sharing very useful information and their software, iBioSim, which was used to create many of the simulation results shown on this thesis. I would also like to thank Mireya Marte Pepen for providing support during difficult moments. There are other people who helped revising and polishing this thesis and to whom I am thankful: my graduate committee for taking the time to review and evaluate this work as well as sharing their expertise, Mary Lee Anderson for proof-reading and correcting this work, and Gopalakrishnan Sundararjan for helping me with the whole process of revising and printing the thesis. Finally, I would also like to thank the NSF for providing the funds to support this work.

Eduardo Monzon

Contents

List of Tables

List of Figures

Acronyms

CLE	Chemical Langevin Equation
CME	Chemical Master Equation
DNA	Deoxyribonucleic Acid
EDA	Electronic Design Automation
GDA	Genetic Design Automation
iGEM	international Genetically Engineered Machine
ODE	Ordinary Differential Equation
RNA	Ribonucleic Acid
SCK	Stochastic Chemical Kinetics
SSA	Stochastic Simulation Algorithm

Chapter 1

Introduction

Synthetic biology is a relatively new field of research and engineering with the goal of designing and synthesizing biological components and systems that do not exist in nature. According to Ingalls [1], the design and construction of synthetic gene networks, as an area of synthetic biology, has grown very rapidly after the first engineered gene circuits were announced in the year 2000.

Synthetic biology promises many applications in medicine and pharmaceutical. For instance, bacteria can be manipulated to synthesize proteins that produce drugs [2]. Also, bacteria can potentially be engineered to target cancer cells and destroy them before they are pernicious to the human body [3], to clean toxic wastes, such as oil spills, and detect the presence of toxins in the environment [4]. The potential that can be reached by programming organisms to do specific tasks is yet unknown. However, like in any engineering field, there is the need of automated tools to aid in the design and constructions process.

Following the success of electronic design automation (EDA) tools in the electronics industry, scientists and engineers in synthetic biology have created genetic design automation (GDA) tools to speed up the progress in this field. One way this is achieved, is by accelerating the design process and reducing the skill-sets needed to achieve successful biological designs. Up to some extent this is possible if the designer is presented with summarized and useful information that is easy interpret and does not lead to ambiguities. However, this is not an easy task since ambiguity and randomness are inherent in these biological systems.

To reveal the highly random behavior in genetic circuits due to small molecule counts and sporadic gene expression, stochastic simulation algorithms are necessary in any GDA tool. Statistics generated from stochastic simulation runs, however, are usually very noisy and can be difficult to analyze. Furthermore, targeted users of GDA software include practicing biochemists who are not necessarily trained in the analysis and interpretation of stochastic systems. This imposes a fundamental difficulty to the use of GDA tools: the ability to distilled functional behavior from noisy simulation results. Masking noise from the "typical" behavior of a genetic circuit helps to speed up the design and verification process of these biological systems and and helps the designer to make informed decisions easier and quicker.

Designers typically intend for their genetic circuits to behave nearly-deterministically, despite the growing argument that noise is an integral part of a reaction system's behavior. Deterministic behaviors are desirable because predictions correspond faithfully to real simulation runs and reveal the functional details intended by the designers. Unfortunately, this is rarely the case and simulations often reveal unintended behaviors such as spurious oscillations or multiple stable states that were not anticipated in the design process.

Computer simulations have become an important part of functional verification allowing the designer to assess the likelihood of a design's success before proceeding with manufacturing and experimental testing. Hence, there is a growing interest in general-purpose techniques for modeling genetic circuits and predicting their behavior. Visualization of the behavior of genetic circuits is one way computer simulations help biological designers assess their models. Being able to visualize properly a genetic circuit model provides feedback on what things are working properly and what things may not. On the extreme, poor visualization methods may cause confusion and even lead the designer to infer an erroneous behavior from the circuit. Hence, proper visualization is of utmost importance in synthetic biology.

The information presented on this thesis is organized as follows: Chapter 2 presents background information about synthetic gene networks (a.k.a genetic circuits) and the processes of transcription and translation. Some examples of genetic circuits are also presented. Chapter 3 introduces some common stochastic simulation algorithms. Chapter 4 describes the multi-path visualization algorithm as a contribution of this thesis, as well as the outline of a new Multi-Path Detection iSSA (MPD-iSSA) method to address some challenges of current GDA tools, like the detection and visualization of multi-stable behavior. Chapter 5 presents some discussions and future direction of these methods.

Chapter 2

Background

The behavior of any organism, at the molecular level, is determined by the information contained in its DNA. This information instructs the organism how to perform functions like reproduction, communication with the environment, and production of certain proteins and other cell components for survival. Hence, a clear understanding of how an organism makes use of this information is vital to any biological designer intending to modify an organism's behavior. The term *behavior* is being used loosely here, but primarily refers to any possible response or interaction of the organism with its environment.

This chapter introduces how the genetic machinery of the cell makes use of the information encoded in its DNA, and how this process can be harnessed to modify an organism's behavior by changing this information. Some very common genetic constructs that have been used to this purpose are also introduced. The information on this chapter is organized as follows: Section 2.1 gives an overview of gene expression and the processes of transcription and translation. Section 2.2 introduces synthetic gene networks and its various components. Section 2.3 presents some synthetic gene networks that have been constructed and tested successfully *in vivo* (i.e., inside living organisms).

2.1 Gene Expression

Gene expression is a complex process, which occurs in two stages. In the first stage, the DNA of the gene is transcribed into messenger RNA (mRNA) by the enzyme RNA polymerase (i.e., the information stored in the nucleotide order on the DNA is copied into information stored by the nucleotide order on the mRNA). In the second stage, the mRNA is translated into protein by enzymes called ribosomes (i.e., the information stored in nucleotides on the mRNA is translated into the amino acids sequence of the protein).

The first stage is called *transcription* and the second stage is called *translation*, and both processes are illustrated in detail in Figures 2.1 and 2.2.

Figure 2.1 shows the transcription process step by step. This process is initiated when the enzyme RNA polymerase (RNAP) recognizes a specific part of the DNA, which marks the beginning of a coding region. This is shown in step 1. In step 2, RNAP moves along the DNA coding region and produces a complimentary copy of every DNA base pair it reads, forming a messenger RNA (mRNA) molecule. This transcription process terminates in step 3, when RNAP reads a DNA sequence signaling the end of the coding region. The information contained in the mRNA molecule just produced can now be used by the ribosomes during the translation process to create proteins or any other molecules encoded.

The translation process looks simple from Figure 2.2, but it is no less complex than transcription. Here, the ribosome recognizes and binds to a specific sequence in the mRNA known as the *ribosome binding site*. After that, the ribosome reads a sequence of three nucleotides at a time, which code for a specific *amino acid*. There is a total of 20 different kinds of amino acids found in living organisms and their specific order determines the protein's shape and function. Proteins are constructed one amino acid at a time in the order specified by the *codons* in the mRNA. A codon is a group of three bases which specifies a particular amino acid using the genetic code shown in Table 2.1. As it can be seen in the table, some amino acids are associated with more than one codon. This redundancy provides robustness in the face of *mutations* (i.e., small random changes) that occur naturally when DNA is replicated during cell divisions [5].

The information encoded in genes includes not only coding sequences for the specific order of amino acids in a protein, but also regulatory sequences that control the rate that a gene is transcribed. Hence, transcription can be either *activated* (i.e., turned on) or *repressed* (i.e., turned off) if certain proteins bind to this regulatory sequences. Transcription can also be regulated through *post-transcriptional modifications*, *DNA folding*, and other feedback mechanisms [5]. This regulation is analogous to electrical circuits in which multiple input signals are processed to produce multiple output signal and the reason why these regulatory

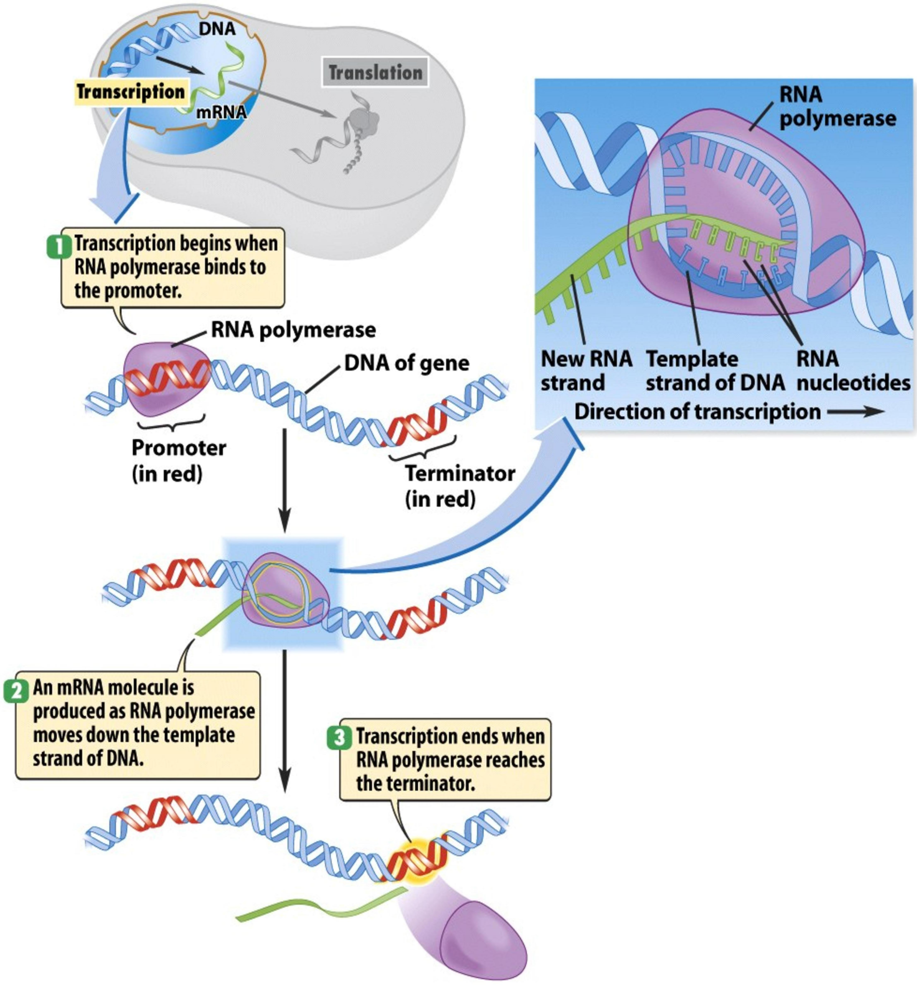
DNA
Transcription
mRNA
Translation
RNA polymerase
New RNA strand
Template strand of DNA
RNA nucleotides
Direction of transcription
1 Transcription begins when RNA polymerase binds to the promoter.
RNA polymerase
DNA of gene
Promoter (in red)
Terminator (in red)
2 An mRNA molecule is produced as RNA polymerase moves down the template strand of DNA.
3 Transcription ends when RNA polymerase reaches the terminator.

Figure 13-3 Discover Biology 3/e
© 2006 W. W. Norton & Company, Inc.

Fig. 2.1: An overview of the transcription process.

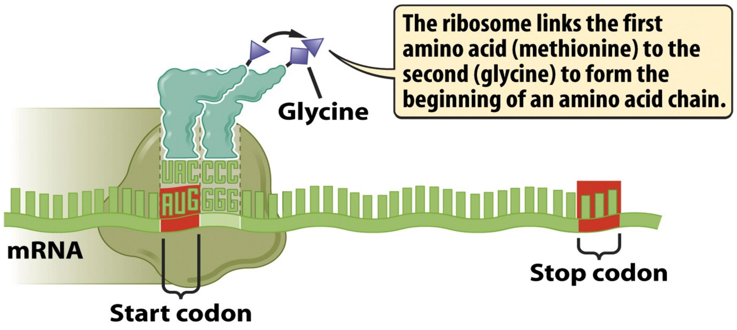
The ribosome links the first amino acid (methionine) to the second (glycine) to form the beginning of an amino acid chain.
Glycine
UAC CCC
AUG GGG
mRNA
Start codon
Stop codon

Figure 13-8 part 3 Discover Biology 3/e
© 2006 W. W. Norton & Company, Inc.

Fig. 2.2: An overview of the translation process.

networks are sometimes referred to as *genetic circuits*.

2.2 Genetic Circuits

Current technology in genetic engineering has made it possible to synthesize fragments of DNA consisting of almost any gene sequence. Since genes are responsible for the behavior of organisms at the molecular level, these DNA fragments can be synthesized to contain gene sequences capable of altering the organism's behavior to certain internal or environmental conditions. These genetic components are inserted into the organism's genome through a variety of *transformation* techniques in which the organism accepts this exogenous genetic material and makes it part of its genome. These constructed gene sequences are known as *synthetic gene networks* and sometimes are called *genetic circuits*. Throughout this thesis the terms genetic circuit and synthetic gene network will be used interchangeably.

A genetic circuit, in the context of synthetic biology, can be considered as a group of genes forming a network of interaction, which resembles an electrical circuit. This analogy with electrical circuits abstracts away the complexities of the DNA transcription and trans-

Table 2.1: The genetic code for the different kinds of amino acids.

	U	C	A	G
U	UUU Phenylalanine	UCU Serine	UAU Tyrosine	UGU Cysteine
	UUC Phenylalanine	UCC Serine	UAC Tyrosine	UGC Cysteine
	UUA Leucine	UCA Serine	UAA Stop	UGA Stop
	UUG Leucine	UCG Serine	UAG Stop	UGG Tryptophan
C	CUU Leucine	CCU Proline	CAU Histidine	CGU Arginine
	CUC Leucine	CCC Proline	CAC Histidine	CGC Arginine
	CUA Leucine	CCA Proline	CAA Glutamine	CGA Arginine
	CUG Leucine	CCG Proline	CAG Glutamine	CGG Arginine
A	AUU Isoleucine	ACU Threonine	AAU Asparagine	AGU Serineine
	AUC Isoleucine	ACC Threonine	AAC Asparagine	AGC Serineine
	AUA Isoleucine	ACA Threonine	AAA Lysine	AGA Arginine
	AUG Methionine	ACG Threonine	AAG Lysine	AGG Arginine
G	GUU Valine	GCU Alanine	GAU Aspartate	GGU Glycine
	GUC Valine	GCC Alanine	GAC Aspartate	GGC Glycine
	GUA Valine	GCA Alanine	GAA Glutamate	GGA Glycine
	GUG Valine	GCG Alanine	GAG Glutamate	GGG Glycine

lation processes, and helps with the idea of components that can be manipulated to achieve a desired behavior. Even though this abstract view of genetic circuits simplifies the process of designing and working with these systems, it is important to keep in mind the intrinsic difference with their electrical counterpart. In genetic circuits, instead of electrical signals representing information through a sequence of ones and zeros, the chemical concentrations of specific DNA-binding proteins and inducer molecules act as the input and output signals of the system. These molecules are able to interact with other proteins, bind to specific DNA sites, and regulate the expression of other proteins within the cell. It is this regulatory activity which can be exploited to construct genetic systems able to process chemical signals in a way similar to what digital logic functions do, as well as some analog electrical circuits.

Gene expression, in a genetic circuit, is controlled by a region of DNA called the *promoter*. Transcription of the gene is initiated when the RNA polymerase recognizes and binds to this promoter sequence. This sequence instructs RNAP both where to start synthesis of the mRNA transcript and in which direction. The transcription process terminates when the RNAP reaches a transcriptional stop signal sequence.

Regulation is mediated by proteins, called *transcription factors*. These proteins recognize portions of the DNA sequence near the promoter region, known as *operator sites*. Once bound, they either hinder the binding of RNAP to the promoter and thus repress gene expression (the transcription factors are then called *repressors*) or they enhance the binding of RNAP to the promoter and activate gene expression (the transcription factors are then called *activators*). According to Swain and Longtin [6], nearly all genes *in vivo* are regulated. However, unregulated genes also exist and they are said to be *constitutively expressed*.

All the terms mentioned above (i.e., genes, promoters, transcription factors, and operator binding sites) are the basic building blocks of any genetic circuit. These building blocks can be observed in Figure 2.3. This figure shows a simple genetic circuit found in the *phage* λ virus and it is described in more details by Myers [5]. It is common among the community to represent the promoters with arrows pointing either left or right, the genes with filled rectangles, ribosome binding sites with empty squares, and the proteins and transcription factors with round shapes.

When inserted into an organism's genome, genetic circuits are able to change this organism's behavior. This DNA sequence contains information that the cell machinery inside the organism is able to interpret. As noted by Weiss et al. [7], the resulting behavior of these synthetic constructs in an organism is not always easy to predict, but this has not stopped the community of scientists and engineers to assemble a component library of genetic circuit building blocks.

2.3 Component Library of Genetic Gates

The first step to be able to build more complex systems consists in establishing a library of well-defined components. The integration of these components enables cells to perform sophisticated digital and analog computation, both as individual entities and as part of larger cell communities. The simplest digital gate constructed out of biological parts is probably the NOT gate or biochemical inverter. In the following subsections we will see how this simple component can be used to form other genetic circuits like the NAND gate or even a genetic oscillator.

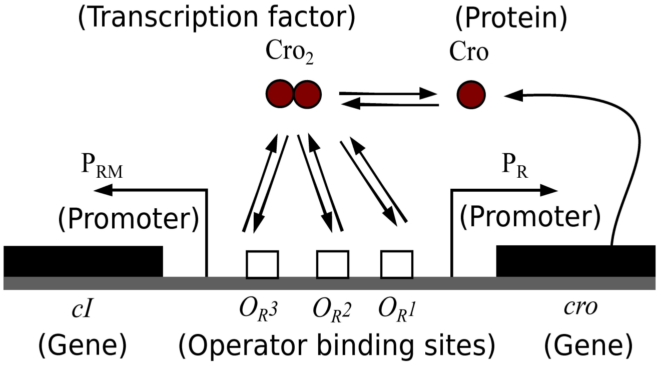

Fig. 2.3: Circuit found in the *phage* λ virus showing the different components of a genetic circuit.

2.3.1 Biochemical Inverter

The biochemical inverter or biochemical NOT gate follows the same working principle as the digital NOT gate shown in Figure 2.4(a). In digital electronics, this logic gate inverts the binary state of its input. So, if the input is low (i.e., binary 0) the output is high, and if the input is high (i.e., binary 1) the output is low. In the case of the biological inverter, when the input signal is high (i.e., a high concentration of repressor molecules) the output signal is low (i.e., low gene expression) and vice-versa.

The graphical representation of the biochemical inverter is shown in Figure 2.4(b). It shows a protein called *TetR* acting as the repressor signal for the expression of the *Green Fluorescent Protein* (GFP), which acts as the output signal of the circuit. Reporter genes like GFP are very important in synthetic biology because they help assessing if a constructed circuit is working properly by providing a visible output.

2.3.2 Biochemical NAND Gate

The biochemical NAND gate follows the same principle of operation of the logical NAND gate in digital electronics. The logical NAND gate has two inputs, which can be high or low independently providing four possible combinations. The output is low only when the two inputs are high. Any other combination of input logic states will produce a high output signal. This is summarized in Table 2.2.

The symbol used in digital electronics to represent the digital NAND gate is shown in Figure 2.5(a). Similar to the digital NOT gate symbol, it has a circle at the output indicating that it is actually an AND gate with an inverted output. The biochemical representation of this logic gate is shown in Figure 2.5(b). In this case, the NAND gate is formed by putting in parallel two biochemical NOT gates with the same output (GFP), but different inputs (LacI and TetR). So, according to the truth Table 2.2, only when the concentration of the two repressors is high will the gene expression of GFP be low. It is important to note at this point that although the two components of the NAND gate are drawn separate, they may be together, side by side, on the same strand of DNA.

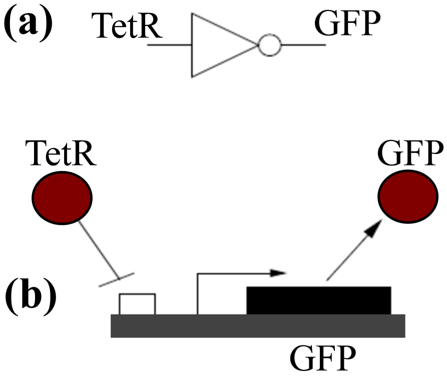

Fig. 2.4: (a) Symbol representing digital NOT gate. (b) Biochemical representation of NOT gate.

Table 2.2: Truth table for the logical NAND gate.

LacI	TetR	GFP
0	0	1
0	1	1
1	0	1
1	1	0

2.3.3 Biochemical Ring Oscillator

An oscillator is a circuit or device that produces a repetitive signal, often a sine wave or a square wave [8]. They are widely used in electronic communications to generate carrier signals and in digital systems to generate clock signals that regulate computers and quartz clocks. However, electronic oscillators are not the only class of oscillators in existence. There are also biological oscillators and they exist in many living organisms. They appear in functions ranging from cell-division cycle [9] to helping organisms keep track of the time of day [10]. In these oscillators, obviously, the repetitive signal is produced by an enhanced production and inhibition of gene expression.

According to Friesen and Block [11], there are two essential elements of any biological oscillator: 1) an inhibitory feedback loop, which includes one or more oscillating variables, and 2) a source of delay in this feedback loop, which allows an oscillating variable to

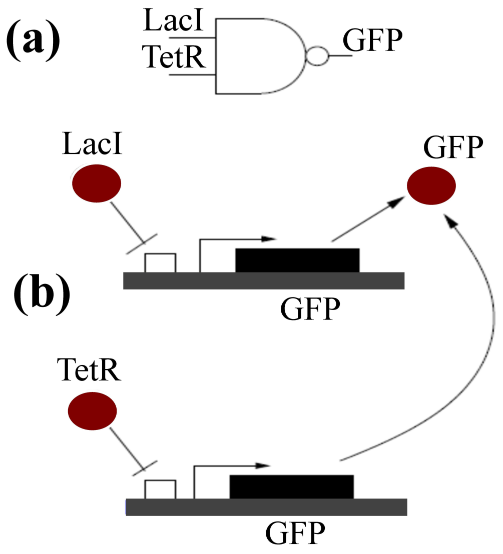

Fig. 2.5: (a) Symbol representing digital NAND gate. (b) Biochemical representation of NAND gate.

overshoot a steady-state value before the feedback inhibition is fully effective.

Figure 2.6 shows both the digital (on the top) and the biological representation (on the bottom) of an oscillator. Each node in the digital version has the name of the corresponding gene expressed in the biological counterpart. We can observe that this is a ring oscillator formed with three inverters connected in a feedback loop. The propagation delay of the inverters provides the source of delay required to produce the oscillations. The same manner, in the biological oscillator in Figure 2.6, the source of delay is the time required by the transcription and translation process, plus the time required by these proteins to affect the promoter of the gene downstream. This time, of course, is random due to the diffusion process involved and hence the periods and amplitude of the oscillations are also stochastic.

The genetic circuits presented above are just examples of some of the simplest biological systems that can be achieved. More complex circuits can also be constructed by connecting simple parts, however, it is important to remember that signals in genetic circuits are carried out by molecules that diffuse and interact with other chemical species, and whenever a new part is introduced, attention must be paid to how this may affect the rest of the circuit.

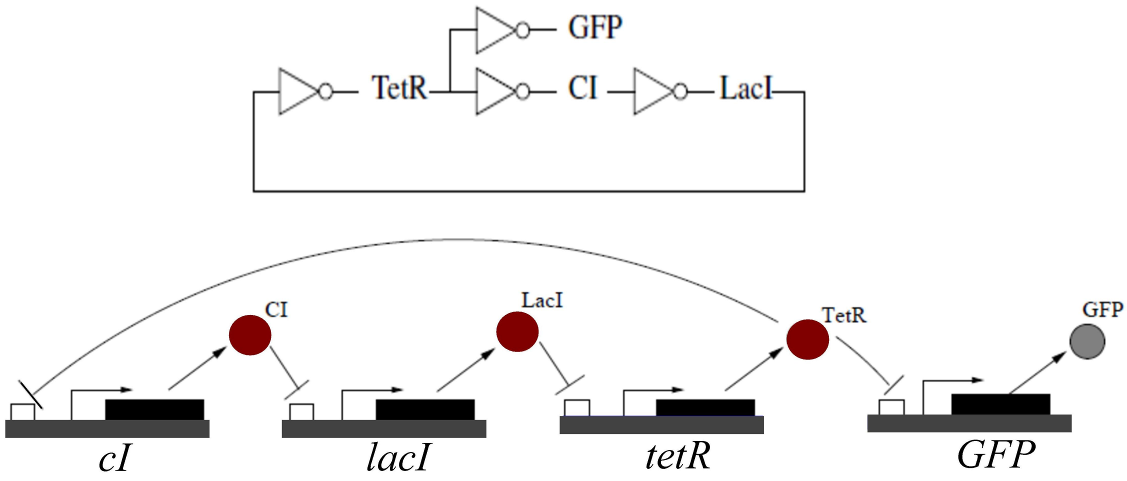

Fig. 2.6: Digital and biochemical representation of a ring oscillator.

2.3.4 Genetic Toggle Switch

Another interesting genetic circuit worthwhile examining is the genetic toggle switch. The toggle switch model, shown in Figure 2.7(b) is a genetic circuit implementation of the well-known set-reset latch from traditional electronics, shown in Figure 2.7(a). The circuit's inputs are aTc and IPTG, and it is designed such that the molecular species TetR and LacI mutually repress each other. The TetR gene is also associated with a gene that codes for green fluorescent protein (GFP), which serves as a detectable output signal similar to the previously shown genetic circuit examples. Due to the mutual repression between LacI and TetR, only one of these species persists under normal conditions. This creates a bi-stable situation in which the circuit has two possible states: one state in which LacI is present but TetR is absent (the "off" state), and another state in which TetR is present but LacI is absent (the "on" state) [12].

The circuit's state can be controlled by temporarily adding one of the input species, either aTc or IPTG. When aTc is added, the circuit is expected to switch *off*. Similarly, when IPTG is added, the circuit is expected to switch *on*. If both input species are added, the circuit's behavior is undefined. When both input species are initially absent, a race condition situation is created in which one species is produced slightly faster than the other, leading eventually to a latched *on* or *off* state.

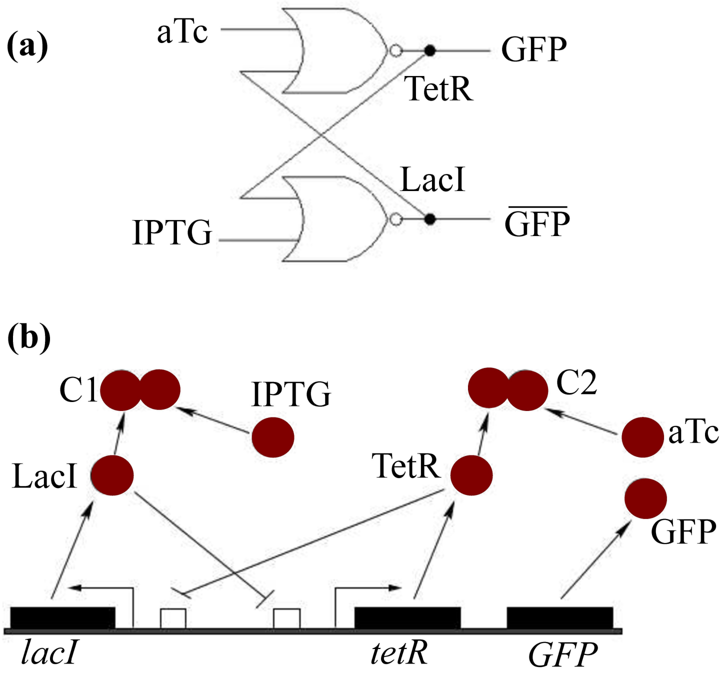

Fig. 2.7: Genetic circuit for the toggle switch.

Once the circuit is latched, it is expected to hold its state indefinitely via the negative feedback mechanism. By adjusting the kinetic parameters of the model, the race can be made fair so that either outcome is equally likely, or biased towards one of the two states. However, when the circuit is initially set to one of the two states and the repressor input is increased, the circuit is more likely to switch to the other state.

Chapter 3

Stochastic Simulations

The solution to Ordinary Differential Equation (ODE) models of a chemical reaction system provides expressions for the concentration of each species as a function of time. These models assume concentrations vary continuously and deterministically. However, the molecule counts in genetic circuits is generally small (often tens or hundreds of molecules of each transcription factor and one strand of DNA) and the discrete and stochastic nature of these systems may influence significantly the observable behavior. Therefore, a stochastic description of the genetic circuit is required to perform more accurate simulations.

3.1 Stochastic Chemical Kinetic Model

Chemical reaction network models are composed by n chemical species $\{S_1, S_2, ..., S_n\}$ interacting through m chemical reaction channels $\{R_1, R_2, ..., R_n\}$. Stochastic Chemical Kinetic (SCK) models assume that the molecular species are contained in a constant volume Ω, like the volume of a cell, and the system is *well-stirred* [5], which means the molecules are equally distributed throughout the volume. This assumption, however, is not always true and methods that account for spatial effects must be used for more accurate simulations. Another assumption in SCK models is that the system maintains a constant temperature T, which is referred as *thermal equilibrium.*

The state of the system at time t is represented by the vector $\mathbf{X}\left(t\right) = (X_i(t), ..., X_n(t))$, where $\mathbf{X}_i(t)$ is the number of molecules of species S_i at time t. The initial state of the system (i.e., the initial number of molecules at some initial time t_0) is $\mathbf{X}(t_0) = \mathbf{x}_0$. When a reaction R_μ occurs, the system state is updated by adding the *state-change vector* v_μ to the current system state (i.e. $x' = \mathbf{x} + \mathbf{v}_\mu$). The elements of $v_\mu = (\mathbf{v}_{1\mu}, ..., \mathbf{v}_{n\mu})$ contain the change in molecule count to S_i due to reaction R_μ. The two-dimensional array formed by $\{\mathbf{v}_{i,\mu}\}$ is also known as the *stoichiometry matrix*.

The stoichiometry matrix is the mathematical representation of the biochemical reaction network as shown in the matrix below. Its elements indicate the amount of molecules that are lost or gained when a particular reaction occurs. The reactions are aligned in the columns of the matrix, while the rows indicate which species participate in a particular reaction. For instance, element $a_{1,2}$ in the matrix below indicates that species 1 participates in reaction 2. The sign of matrix entry indicates whether the species is a reactant, with negative sign, or a product, with positive sign. If the species does not participate at all in the reaction then a zero entry is placed in the matrix.

$$\mathbf{N}_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

Every reaction channel R_μ is associated with a specific *probability rate constant* c_μ, which is selected such that $c_\mu dt$ can be defined to be the probability that a random combination of reactant molecules react inside volume Ω in the next infinitesimal time interval $[t, t+dt]$, as defined by reaction R_μ. This value multiplied by the total number of possible combinations of reactant molecules for R_μ produces the *propensity function*, a_μ. Since the number of molecules of each species may change according to the state $\mathbf{x}$ of the system, this function must be recalculated as the system evolves. So, put in precise words, $a_\mu(\mathbf{x})\, dt$ is defined to be the probability that R_μ occurs in the state $\mathbf{x}$ within Ω in the next infinitesimal time interval $[t, t+dt]$.

Let us illustrate this with the following example. Consider the following reaction system:

$$A + B \xrightarrow{c_\mu} C. \tag{3.1}$$

In order to determine c_μ for this reaction, it is necessary to find the probability that an

A molecule and a B molecule collide and react within the next dt time units. We assume that each molecule of A and B are hard spheres of mass m_a and m_b with radius r_a and r_b, respectively. The previous assumption of thermal equilibrium means that a randomly selected molecule of A and B can be found uniformly distributed within the volume Ω. Also, it means that the average relative speed in which A and B see each other moving is given by $\vec{v}_{ab} = \sqrt{8k_BT/\pi m_{ab}}$ where k_B is Boltzmann's constant and $m_{ab} = m_a m_b/(m_a + m_b)$. A B molecule then should sweep in the next dt time units a *collision cylinder* relative to a molecule A, which has height $\vec{v}_{ab}dt$ and base area $\pi(r_a + r_b)^2$. Since the molecules are uniformly distributed within Ω, the probability of one molecule finding itself within the collision cylinder of a molecule of the other reactant is the ratio of $\pi(r_a + r_b)^2\vec{v}_{ab}dt$ to the volume Ω. The specific probability rate constant for this reaction is given by the following formula:

$$c_\mu = \Omega^{-1}\pi(r_a + r_b)^2\vec{v}_{ab}p_\mu, \tag{3.2}$$

where p_μ is the probability that A and B react when they collide. If we assume that they collide only when their kinetic energy exceeds the activation energy, ϵ_μ, then c_μ can be expressed by this formula:

$$c_\mu = \Omega^{-1}\pi(r_a + r_b)^2\left(\frac{8k_BT}{\pi m_{ab}}\right)^{1/2} exp(-\epsilon_\mu/k_BT). \tag{3.3}$$

Since the number of possible combinations of A and B that can react is $a \times b$, the propensity function for R_μ is $a_\mu(\mathbf{x}) = c_\mu ab$.

3.2 Chemical Master Equation

The stochastic model described above is a *jump Markov process*, where the state updates occur in discrete amounts and the next state of the system is only dependent on the present state and not the past history. It is not possible to know the exact state $\mathbf{X}(t)$ of the system due to its stochastic nature, but we can calculate the probability of being in a specific state at time t starting from a state $\mathbf{X}(t_0) = \mathbf{x}_0$ (i.e., $\mathcal{P}(\mathbf{x}, t|\mathbf{x}_0, t_0)$). This probability can be described using a time-evolution of step dt as shown below:

$$\begin{aligned}\mathcal{P}\left(\mathbf{x}, t+dt|\mathbf{x}_0, t_0\right) = \mathcal{P}\left(\mathbf{x}, t|\mathbf{x}_0, t_0\right) \times &\left[1 - \sum_{j=1}^{m}\left(a_j\left(\mathbf{x}\right)dt\right)\right] \\ &+ \sum_{j=1}^{m}\mathcal{P}\left(\mathbf{x}-\mathbf{v}_j, t|\mathbf{x}_0, t_0\right) \times \left(a_j\left(\mathbf{x}-\mathbf{v}_j\right)dt\right).\end{aligned} \tag{3.4}$$

On the right-hand side we have two terms adding together. The first term is the probability that the system is already in state $\mathbf{x}$ at time t, and there are no reaction in the time period $[t, t+dt]$. The second term indicates that the state $\mathbf{x}$ is $\mathbf{v}_j$ away at time t, and the reaction $\mathcal{R}_j$ occurs in the time period $[t, t+dt]$. Let us make the observation that dt is chosen small enough that at most one reaction can occur during this time period. Then, the time evolution of state probabilities $\mathcal{P}\left(\mathbf{x}, t|\mathbf{x}_0, t_0\right)$ of the Chemical Master Equation (CME) are the result of performing the limit as shown next.

$$\begin{aligned}\frac{\partial\mathcal{P}\left(\mathbf{x}, t|\mathbf{x}_0, t_0\right)}{\partial t} &= \lim_{dt\to 0}\frac{\mathcal{P}\left(\mathbf{x}, t+dt|\mathbf{x}_0, t_0\right) - \mathcal{P}\left(\mathbf{x}, t|\mathbf{x}_0, t_0\right)}{dt} \\ &= \sum_{j=1}^{m}\left[a_j\left(\mathbf{x}-\mathbf{v}_j\right)\mathcal{P}\left(\mathbf{x}-\mathbf{v}_j, t|\mathbf{x}_0, t_0\right) - a_j\left(\mathbf{x}\right)\mathcal{P}\left(\mathbf{x}, t|\mathbf{x}_0, t_0\right)\right]\end{aligned} \tag{3.5}$$

This differential equation, however, cannot be solved analytically or numerically except in very simple situations because it represents a set of equations that is nearly as large as the number of molecules in the system. Hence, other methods like the ones described next are used instead to solve these systems.

3.3 Gillespie's Stochastic Simulation Algorithm

Like the master equation, this stochastic simulation algorithm correctly accounts for the inherent fluctuations and correlations that are necessarily ignored in the deterministic formulation, but it is not based directly on the CME. Hence, the key to generating simulated trajectories of $\mathbf{X}\left(t\right)$ is not the function $P\left(\mathbf{x}, t|\mathbf{x}_0, t_0\right)$, but rather a new probability function $p\left(\tau, \mu|\mathbf{x}, t\right)$, which is defined as follows:

$$
\begin{aligned}
p\left(\tau,\mu|\mathbf{x},t\right)d\tau \triangleq\ & \text{the probability, given } \mathbf{X}\left(t\right)=\mathbf{x}, \text{ that the next reaction} \\
& \text{in the system will occur in the infinitesimal time interval} \\
& \left[t+\tau,t+\tau+d\tau\right), \text{ and will be an } R_\mu \text{ reaction.}
\end{aligned} \tag{3.6}
$$

Therefore, this function is a joint probability density function of two random variables, the time to the next reaction, τ, and the index of the next reaction, μ. The advantage of this formulation is that the simulation is able to advance from one reaction to the next skipping over times in which no reaction occurs.

Let us derive an analytical expression for $p\left(\tau,\mu|\mathbf{x},t\right)$ to understand better how this algorithm works. First, let us represent the probability that there is no reaction in the time interval $[t,t+\tau)$ with a new function $\mathcal{P}_0\left(\tau|\mathbf{x},t\right)$. Now, let us express the function $p\left(\tau,\mu|\mathbf{x},t\right)$ as follows:

$$
p\left(\tau,\mu|\mathbf{x},t\right)=\mathcal{P}_0\left(\tau|\mathbf{x},t\right)\times\left(a_\mu\left(\mathbf{x}\right)d\tau\right). \tag{3.7}
$$

Explicitly, this means that no reactions occur in the interval $[t,t+\tau)$, but the R_μ reaction can occur in the interval $[t+\tau,t+\tau+d\tau)$. However, the following must be satisfied:

$$
\mathcal{P}_0\left(\tau+d\tau|\mathbf{x},t\right)=\mathcal{P}_0\left(\tau|\mathbf{x},t\right)\times\left[1-\sum_{j=1}^{m}\left(a_j(\mathbf{x})d\tau\right)\right]. \tag{3.8}
$$

Manipulating this formula and denoting $a_0(\mathbf{x})=\sum_{j=1}^{m}a_j(\mathbf{x})$, we can derive the following differential equation:

$$
\begin{aligned}
\mathcal{P}_0\left(\tau+d\tau|\mathbf{x},t\right) &= \mathcal{P}_0\left(\tau|\mathbf{x},t\right)-a_0\left(\mathbf{x}\right)\mathcal{P}_0\left(\tau|\mathbf{x},t\right)d\tau \\
\frac{\mathcal{P}_0\left(\tau+d\tau|\mathbf{x},t\right)-\mathcal{P}_0\left(\tau|\mathbf{x},t\right)}{d\tau} &= -a_0(\mathbf{x})\mathcal{P}_0\left(\tau|\mathbf{x},t\right) \\
\frac{\mathcal{P}_0\left(\tau,\mathbf{x},t\right)}{d\tau} &= -a_0\left(\mathbf{x}\right)\mathcal{P}_0(\tau|\mathbf{x},t).
\end{aligned} \tag{3.9}
$$

With initial condition $\mathcal{P}_0(\tau = 0|\mathbf{x}, t) = 1$, this differential equation has solution:

$$\mathcal{P}_0(\tau|\mathbf{x}, t) = exp(-a_0(\mathbf{x})\tau). \tag{3.10}$$

If we now combine equations 3.7 and 3.10 and cancel $d\tau$ we obtain:

$$p\left(\tau, \mu|\mathbf{x}, t\right) = exp\left(-a_0\left(\mathbf{x}\right)\tau\right) \times a_\mu\left(\mathbf{x}\right), \tag{3.11}$$

which can be rewritten as:

$$p\left(\tau, \mu|\mathbf{x}, t\right) = a_0\left(\mathbf{x}\right) exp\left(-a_0\left(\mathbf{x}\right)\tau\right) \frac{\times a_\mu\left(\mathbf{x}\right)}{a_0\left(\mathbf{x}\right)}. \tag{3.12}$$

Equation (3.12) shows that $p\left(\tau, \mu|\mathbf{x}, t\right)$ can be divided into a probability density function for τ and another for μ. The variable τ is an exponential random variable with mean and standard deviation of $1/a_0\left(\mathbf{x}\right)$. The variable μ, on the other hand, is an integer random variable with point probabilities $a_\mu\left(\mathbf{x}\right)/a_0\left(\mathbf{x}\right)$.

Gillespie's SSA is built upon these previous observations and its steps are outlined in Algorithm 1. In step 1, the starting time t_0 and the initial state of the system $\mathbf{x}_0$ are set. In step 2, all the propensity functions $a_j(\mathbf{x})$ and their sum a_0 are recalculated every iteration since their values depend on the current state $\mathbf{x}$, which may change due to the last reaction. In step 3, two uniform random numbers, r_1 and r_2, are selected between $[0, 1]$. These random numbers are used in the next two steps to determine τ and μ. In step 6, the next state is determined by updating the molecule count using the stoichiometry information for the reaction, $\mathbf{v}_\mu$, and moving time forward τ time units.

Finally, in step 7 the current time t is checked to verify whether the simulation time has not been exceeded. If it has, the trajectory is complete and the simulation halts, otherwise, in step 8 the new state is recorded and the simulation continues back at step 2.

3.4 Next Reaction Method

Since the algorithm's introduction, numerous SSA variations have been developed. One

Algorithm 1 Gillespie's stochastic simulation algorithm (SSA).

```
1: Initialize: $t = t_0$ and $\mathbf{x} = \mathbf{x}_0$.
2: Evaluate propensity functions $a_j(\mathbf{x})$ at state $\mathbf{x}$, and their sum $a_0 = \sum_{j=1}^{m} a_j(\mathbf{x})$.
3: Draw two unit uniform random numbers, $r_1$ and $r_2$.
4: Calculate the time $\tau$, until the next reaction:
       $$r = \frac{1}{a_0(\mathbf{x})} \ln\left(\frac{1}{r_1}\right)$$
5: Determine the next reaction, $R_\mu$:
       $$\mu = \text{ the smallest integer satisfying } \sum_{j=1}^{\mu} a_j(\mathbf{x}) > r_2 a_0(\mathbf{x})$$
6: Determine the new state after reaction $\mu : t = t + \tau$ and $\mathbf{x} = \mathbf{x} + v_\mu$.
7: If $t$ is greater than the desired simulation time then halt.
8: Record $(\mathbf{x}, t)$ and go to step 2.
```

of its variants is the Next Reaction Method developed by Gibson and Bruck [13]. Similar to Gillespie's SSA, this is an exact algorithm to simulate coupled chemical reactions, but it is computationally more efficient because it uses only a single random number per simulation event, and it takes time proportional to the logarithm of the number of reactions, instead of the number of reactions itself. One main idea of this algorithm comes from recognizing that when a reaction fires, the molecule count only changes for the chemical species involved in that reaction, hence only reaction channels having those species as reactants need to update their propensities $a_j(\mathbf{x})$. In the Gillespie's SSA all propensities are updated in every iteration. Also, by storing the times τ_j when each reaction is likely to happen, and not just a_j, it is possible to save calculating one random number every iteration. This may not seem a huge save, but it becomes significant for many iterations.

The statement of recalculating a_j (and τ_j) only if it changes may seem circular because to know whether a_j has changed one normally would calculate the new value $a_{j,new}$ and compare it to the old value $a_{j,old}$. However, one can analyze the set of reactions beforehand and determine which reactions change which a_j. This is achieved introducing a data structure, called a *dependency graph.* The dependency graph for a set of reactions R is a directed graph $\mathbf{G}(V, E)$ with vertex set $V = R$ and with a directed edge from v_i to v_j if and only if $\mathit{Affects}(v_i) \bigcap \mathit{DependsOn}(a_{v_j}) \neq \varnothing$. In other words, a dependency graph is a data structure that tells precisely which a_j to change when a given reaction is executed. Using the dependency graph allows one to recalculate only the minimum number of a_j.

With respect to be able to re-use τ_j where appropriate, it is known that, in general, Monte Carlo simulations assume statistically independent random numbers and it is usually not legitimate to re-use random numbers. However, in this particular special case, it is legitimate because the τ_j values are changed to use absolute time rather than relative time between reactions and they are re-normalized whenever its propensity has changed. Updating a_is and τ_js is made efficient thanks to the use of another data structure, called an *indexed priority queue*. An indexed priority queue consists of *(a)* a tree structure of ordered pairs of the form (j, τ_j), where j is the number of a reaction and τ_j is the putative time when reaction j occurs, and *(b)* an index structure whose j^{th} element is a pointer to the position in the tree that contains (j, τ_j).

The next reaction algorithm is outlined in Algorithm 2.

3.5 Tau-Leaping Method

The next reaction method provides some improvements over the exact SSA method for systems with many species and many reaction channels; however, reactions are still simulated one at a time and this is a real bottleneck for speeding up simulation time. Sometimes it is necessary to give up the exactness of an algorithm to improve the simulation speed. *Tau-leaping* is an approximate way of accelerating the SSA in which each time step τ advances the system through possibly many reaction events, instead of just one by one as has been seen before. With the system in state $\mathbf{x}$ at time t, let us suppose there exists a $\tau > 0$ that satisfies the leap condition: During $[t, t + \tau)$ no propensity function is likely to change its value by a significant amount. With $a_j(\mathbf{x})$ remaining essentially constant during $[t, t + \tau)$, it then follows that the number of times reaction channel R_j fires in $[t, t + \tau)$ is a Poisson random variable with mean (and variance) $a_j(\mathbf{x})\tau$ [14]. In other words, instead of jumping precisely to the next reaction, many reactions are allowed to fire at once in the time interval $[t, t + \tau)$.

Algorithm 2 Gibson and Bruck's next reaction method.

```
1: Initialize:
      a) $t = t_0$ and $\mathbf{x} = \mathbf{x}_0$.
      b) Generate a dependency graph, $\mathbf{G}(V, E)$.
      c) Evaluate propensity functions $a_j(\mathbf{x})$ at state $\mathbf{x}$.
      d) For each $j$, determine the time, $\tau_j$, until the next $R_j$ reaction:
            $$\tau_j = t + \frac{1}{a_j(\mathbf{x})} \ln\left(\frac{1}{r_j}\right)$$
         where each $r_j$ is a unit uniform random number.
      e) Store the $\tau_j$ values in an indexed priority queue $\mathbf{Q}$.
2: Let $R_\mu$ be the reaction whose $\tau_\mu$ is the smallest stored in $\mathbf{Q}$.
3: Let $\tau$ be $\tau_\mu$.
4: Determine the new state after reaction $R_\mu : t = \tau$ and $\mathbf{x} = \mathbf{x} + \mathbf{v}_\mu$.
5: For each edge $(\mu, \alpha)$ in the dependency graph $\mathbf{G}$,
      a) Set $a_{\alpha,old} = a_\alpha$ and update $a_\alpha$.
      b) If $\alpha \neq \mu$, set $\tau_\alpha = (a_{\alpha,old}/a_\alpha)(\tau_\alpha - t) + t$.
      c) If $\alpha = \mu$, generate a random number, $r_\mu$, and
            $$\tau_\mu = t + \frac{1}{a_\mu(\mathbf{x})} \ln\left(\frac{1}{r_\mu}\right)$$
6: If $t$ is greater than the desired simulation time then halt.
7: Record $(\mathbf{x}, t)$ and go to step 2.
```

To the degree that the leap condition is satisfied, this change is accomplished by introducing m random functions, $\mathcal{K}_j(\tau, \mathbf{x}, t)$, each one returning the number of times that the reaction channel, R_j, fires in the interval $[t, t+\tau)$ assuming the system is currently in state $\mathbf{X}(t) = \mathbf{x}$. After this τ-leap is determined, the next state of the system is given by:

$$\mathbf{X}(t+\tau) = \mathbf{x} + \sum_{j=1}^{m} \mathcal{K}_j(\tau, \mathbf{x}, t)\mathbf{v}_j. \tag{3.13}$$

Equation (3.13) is the basic tau-leaping formula. How this formula can be used in an algorithm to perform faster stochastic simulations will be discussed later. For the moment, let us suppose that τ is not only small enough to satisfy the leap condition, but also large enough that the expected number of firings of each reaction channel R_j during τ is $\gg 1$:

$$a_j(\mathbf{x})\tau \gg 1 \text{ for all } j = 1, ..., m. \tag{3.14}$$

Then, denoting the normal (Gaussian) random variable with mean m and variance σ^2 by $\mathcal{N}(m, \sigma^2)$, and recalling the fact that a Poisson random variable with a mean and variance that is $\gg 1$ can be approximated as a normal random variable with that same mean and variance, we can approximate Equation (3.13) as

$$\mathbf{X}(t+\tau) \approx \mathbf{x} + \sum_{j=1}^{m} \mathcal{N}_j(a_j(\mathbf{x})\tau, a_j((x)\tau)\mathbf{v}_j = x + \sum_{j=1}^{m} \left[a_j(\mathbf{x})\tau + \sqrt{a_j(\mathbf{x})\tau} N_j(0,1) \right]_j. \tag{3.15}$$

In the last step the well-known property of the normal random variable that $\mathcal{N}(m, \sigma^2) = m + \sigma\mathcal{N}(0,1)$, is invoked. Collecting terms and assuming τ is a *macroscopically infinitesimal* time increment dt produces what is known as the chemical Langevin equation (CLE) or Langevin leaping formula [14],

$$\mathbf{X}(t+dt) \approx \mathbf{X}(t) + \sum_{j=1}^{m} \mathbf{v}_j a_j(\mathbf{X}(t))dt + \sum_{j=1}^{m} \mathbf{v}_j \sqrt{a_j(\mathbf{X}(t))} N_j(t)\sqrt{dt}, \tag{3.16}$$

where $N_j(t)$ are m statistically independent and temporally uncorrelated normal random variables with mean 0 and variance 1. This equation has a deterministic component that grows linearly with respect to the propensity functions and a stochastic component that grows proportional to the square root of the propensity functions. Since the propensity functions grow in direct proportion with the system size (volume and species population), the stochastic component scales as the inverse square root of the system size. Thus, as the size of the system increases, the magnitude of the stochastic fluctuations diminishes. At some point, due to the system size, the fluctuations will become so insignificant that Equation (3.16) can be approximated to:

$$\mathbf{X}(t+dt) \approx \mathbf{X}(t) + \sum_{j=1}^{m} \mathbf{v}_j a_j(\mathbf{X}(t))dt, \tag{3.17}$$

which can be rearranged in the following:

$$\frac{\mathbf{X}(t+dt) - \mathbf{X}(t)}{dt} = \frac{d\mathbf{X}(t)}{dt} = \sum_{j=1}^{m} \mathbf{v}_j a_j(\mathbf{X}(t)). \tag{3.18}$$

This equation is the reaction rate equation, which has been derived from stochastic chemical kinetics.

Let us now go back to the formulation of the tau-leaping algorithm using all this knowledge. As it has been shown, the number of reactions of a reaction channel, R_j, is dependent on its propensity, $a_j(\mathbf{x})$, which is dependent on the state which in turn is dependent on the number of all other reactions. Since all these functions are dependent on each other, they are not so easy to compute. The first thing that needs to be done when the leap condition is satisfied, is to approximate the value of $\mathcal{K}_j(\tau, \mathbf{x}, t)$ for each reaction, R_j, to be a statistically independent Poisson random variable:

$$\mathcal{K}_j(\tau, \mathbf{x}, t) \approx \mathcal{P}_j(a_j(\mathbf{x}), \tau) \ (j = 1, ..., m). \tag{3.19}$$

Here, $\mathcal{P}_j(a_j(\mathbf{x}), \tau)$ returns the number of events k in the interval $[t, t+\tau)$ such that the probability of each k value is given by:

$$\mathcal{P}[k \text{ events}] = \frac{e^{-a_j(\mathbf{x})\tau}(a_j(\mathbf{x})\tau)^k}{k!}. \tag{3.20}$$

The core part of the tau-leaping algorithm is then to find a value for τ that is small enough to satisfy the leap condition, but large enough to fire a number of events able to speed up the simulation time significantly. This value for τ can be found using the following equation:

$$\tau = \min_{i \in I_{rs}} \left\{ \frac{max\{\epsilon_i x_i, 1\}}{|\sum_{j \in J_n cr} v_{ij} a_j(\mathbf{x})|}, \frac{max\{\epsilon_i x_i, 1\}^2}{\sum_{j \in J_{ncr}} v_{ij}^2 a_j(\mathbf{x})} \right\}, \tag{3.21}$$

where I_{rs} are the chemical species that appear as reactants in reactions and J_{ncr} are the *non-critical reactions*, meaning reactions that can be fired nc times without causing the species count to become negative. Using this equation it is ensured that no propensity function is likely to change by more than $\epsilon a_j(\mathbf{x})$, where ϵ is an *accuracy control parameter* satisfying $0 < \epsilon \ll 1$. The value of ϵ provides a mean of trading off accuracy of the algorithm for simulation time. The greater the value for ϵ, the greater speed up can be achieved at the cost of accuracy. However, care has to be taken to avoid any species count being made negative. The steps involved in this algorithm are detailed in Algorithm 3.

3.6 Incremental Stochastic Simulation Algorithm

The previous simulation methods presented up to now show only a single trajectory of the time evolution of the genetic circuit. However, a single path is usually not sufficient to verify a design's non-rare behavior from SSA data, and researchers commonly execute many repetitions of the SSA simulation. This yields a bundle of sample paths, which can then be analyzed using statistical measures. Some publications report the typical behavior of a genetic circuit by computing the mean over all SSA sample paths. This approach, however, becomes less useful with systems showing dynamic behavior, like state-holding or oscillating circuits, because dynamic behaviors do not necessarily occur at the same time in different simulated trajectories. So, averaging mis-aligned events tends to distort the underlying patterns.

To address this situation, Winstead et al. [15] and Kuwahara et al. [16] have presented

Algorithm 3 Tau-leaping simulation algorithm.

```
1: Initialize: $t = t_0$ and $\mathbf{x} = \mathbf{x}_0$.
2: Evaluate propensity functions $a_j(\mathbf{x})$ at state $\mathbf{x}$.
3: Determine $J_{ncr}$.
4: If $J_{ncr} = 0$ then $\tau' = \infty$ else determine value for $\tau'$ using Equation 3.21.
5: If $J_{ncr}$ includes all reactions then $\tau'' = \infty$ else use SSA to compute $\tau''$ and $j_c$, the next
   critical reaction.
6: $\tau = min(\tau', \tau'')$ and $t = t + \tau$.
7: $\mathbf{x} = \mathbf{x} + \sum_{j \in J_{ncr}} \mathcal{P}_j(a_j(\mathbf{x})\tau)\mathbf{v}_j)$
8: If $t$ is greater than the desired simulation time then halt.
9: Record $(\mathbf{x}, t)$ and go to step 2.
```

the *incremental stochastic simulation algorithm* (iSSA), for visualizing stochastic simulation results. This method selects representative results and excludes outliers from among a collection of stochastic simulations. By selecting results from traditional SSA simulations, the iSSA method avoids possible distortions that may arise from statistical processing. The iSSA method returns a single trajectory that is representative of the typical system's behavior. The reason why a single-trace result is desirable is because they help improve productivity in early-stage design exploration by allowing rapid verification for a complex system.

The incremental approach in iSSA ensures that patterns between different sample paths are aligned in time, so that statistical measures are appropriate and meaningful. This is achieved by bundling stochastic simulation runs in small time increments, instead of complete sample paths, and averaging over all simulation runs at the end of each time increment to determine the mean state. A single state is then selected from these statistics which in turn is used to constrain the initial condition of each run in the next time increment. By performing simulation runs in this manner, this algorithm is able to follow the dominant SSA trajectory on a genetic circuit, rejecting outliers that occur in a minority of SSA trajectories.

The basic idea of the iSSA is depicted in Algorithm 4. It takes as parameters a maximum number of simulation runs (*maxRuns*), a simulation time limit (*timeLimit*), a simulation time increment (*increment*), and an initial system state-vector $\mathbf{x_0}$. At the start of the k^{th} increment, the run number, i, is reset to 1, and the global time is set to $t' = (k-1) \times increment$. Within each increment, the SSA is executed over the local time-interval $t \in (t, t + increment]$. Once the local time t has exceeded this interval, t is reset to the start of the increment, and the SSA is repeated until *maxRuns* is reached. At this point, the global time is advanced to the next increment until the *timeLimit* has been exceeded.

As seen in line 2, the iSSA also requires a function to select a state in which to start the simulation run, and another to record the simulation data and statistics (line 5). These two functions can be defined in a number of alternative ways to produce specialized forms of the iSSA. Each of these specialized iSSA methods delivers different statistical information. For example, the iSSA reduces to the SSA when *increment* is set equal to *timeLimit*, the *select* function sets $\mathbf{x}$ to $\mathbf{x}_0$, and the record function tracks raw simulation data. Therefore, the essence of the iSSA actually lies upon these functions.

Even though the term iSSA could refer to any of the specialized forms as determined by the *record* and *select* functions, throughout this thesis it is used to refer to the incremental approach in general.

Algorithm 4 Incremental stochastic simulation algorithm (iSSA).

```
1: Initialize: $k = 1$ and $X^{(0)} = \text{init}\ (\mathbf{x}_0)$.
2: Set $i = 1$, $t' = (k - 1) \times increment$, and $limit = t' + increment$.
3: Set $t = t'$ and $\mathbf{x} = select(\mathbf{X}^{(k-1)})$.
4: Execute a Gillespie SSA step:
        a) Evaluate propensity functions $a_j(\mathbf{x})$ at state $\mathbf{x}$, and also their sum
           $a_0(\mathbf{x}) = \sum_{j=1} a_j(\mathbf{x})$.
        b) Draw two unit uniform random numbers, $r_1$, $r_2$.
        c) Determine the time, $\tau$, until the next reaction:
               $$\tau = \frac{1}{a_0(\mathbf{x})} \ln \left( \frac{1}{r_1} \right)$$
        d) Determine the next reaction, $R_\mu$, where $\mu$ is the smallest integer satisfying
               $$\sum_{j=1}^{\mu} a_j(\mathbf{x}) > r_2 a_0(\mathbf{x})$$
        e) Determine the new state: $t = t + \tau$ and $\mathbf{x} = \mathbf{x} + \mathbf{v}_\mu$ .
5: If $t < limit$ then $record(X^{(k)}, \mathbf{x}, i)$, go to step 4.
6: If $i < maxRuns$ then $i = i + 1$, go to step 3.
7: If $t < timeLimit$ then $k = k + 1$, go to step 2.
```

Chapter 4

Recognizing and Visualizing Multiple Paths

Visualization is an important part of the design process because it provides the designer feedback on how changes in the system's parameters affect the simulation results. Noisy results are generally undesirable because they distract the designer from the normal behavior of the system and may occasionally lead to incorrect interpretation of the typical behavior. We cannot completely eliminate noise from stochastic simulations because it is inherent in the system itself, but we can reduce it and produce "clean" simulation paths that would be easier to interpret and less likely to be misunderstood.

It was previously mentioned that a common practice among researchers is to perform many stochastic simulations runs of a biochemical system and then average the results to get a clean path, which hopefully is representative of the system's behavior. By averaging many paths, the noise is reduced and outliers are masked to show only a single path containing the "expected value" states at every time step. However, this technique has not shown satisfying results with complex systems like genetic oscillators and toggle switches. The iSSA method, shown in the previous section, has been shown as a promising alternative; however, it lacks the ability to show multiple paths if they are available.

Biochemical systems showing bi-stable or multi-stable behavior are prevalent in nature. For example, in the context of disease networks, it is believed that bi-stable or multi-stable circuits may drive transitions from one locked-in state (healthy state) to another (disease state) [17]. When studying those kind of systems is important to have tools capable of detecting any possible paths and showing correct results of the system's behavior. Performing this detection automatically is a great challenge, especially when noise can lead to mis-detection of states in the system.

This chapter presents the contribution of this thesis: a visualization method capable of detecting multi-stable behavior of a biochemical system by analyzing an ensemble of stochastic simulation runs. This method can be used both as a replacement of the current averaging of an ensemble of paths, or integrated with incremental methods like iSSA. The chapter is organized as follows: Section 4.1 presents the multi-path visualization algorithm as a method to detect and show multiple stable paths from an ensemble of runs using *Kernel Density Estimators.* This algorithm is applied to both artificially generated data as well as simulation results from a genetic toggle switch model. Section 4.2 describes how this algorithm can be integrated within methods like iSSA and the challenges of this integration.

4.1 The Multi-Path Visualization Algorithm

The main problem when averaging sample paths from systems showing more than one stable behavior is that the result of this operation is one path that may not be representative of any of the system's states. To make this clearer, let us assume the genetic toggle switch described in subsection 2.3.4 is simulated 20 different times using the Gillespie SSA algorithm, and 50% of the runs are "high" for molecular species TetR, while the other 50% of the runs are "low." The definitions of "high" and "low" can be somewhat arbitrary depending on the system's characteristics. For this example, "high" represents a molecule count of 50 or more molecules, and "low" a molecule count below 10. When these sample paths are averaged, the result is a new path showing TetR with a molecule count around 30 molecules, which is not representative of the real system's behavior. A more accurate result should be able to show two paths: one with a molecule count around 50 molecules, and the other with a molecule count close to zero. To achieve this, instead of blindly aggregating all the sample path into one average path, it is required a detection step that will identify groupings of these sample paths and will show a path that is representative of each group.

4.1.1 Reasoning Behind the Algorithm

When averaging an ensemble of sample paths, the molecule count for all the runs of each chemical species at time t is contained in a vector $x_{i,t}\left(r_j\right)$, where i represents the species and r_j represents the run number. These arrays of values $x_{i,t}$ provide a snapshot of the system at time t and contain all the information needed to perform any detection of multiple states. These states correspond to some range of values for the molecule count of species i, which can be seen as clusters or groups of molecules around some values in $x_{i,t}\left(r_j\right)$. These groups can be detected using many different techniques, like clustering algorithms. However, not every technique provides the required flexibility or desired running time complexity, which can become quite important for large systems running large simulations.

Clustering refers to finding groups of similar objects in a set, according to some measure. This is different to classification, in which objects are labeled and placed into categories according to some *prior* information. In machine learning terminology, clustering techniques are usually referred as *unsupervised learning*, while classification techniques are known as *supervised learning*. Grouping sample stochastic simulation paths requires unsupervised learning because there are no categories to identify or prior information to include in the decision step of the algorithm. Instead, groups need to be formed according to some similarity (e.g., molecule count) and without any prior assumptions.

One common clustering algorithm in machine learning is the *k-means* clustering algorithm. According to Jain [18], the k-means algorithm was first proposed over 50 years ago, and still continues to be widely used due to its simplicity and popularity. K-means works by partitioning n observations into k clusters, in which each observation belongs to the cluster with the nearest mean. The k-means clustering algorithm can be used to group paths when performing the average operation; however, there is one fundamental problem: the k-means requires a k parameter value to correctly identify k clusters, but this information is not known beforehand. Hence, a separate method must be used to detect the possible groups, and then, k-means can be used to perform this separation.

4.1.2 Kernel Density Estimation

Density estimation is a very well-known technique used to reconstruct the probability density function of a group of observations. One way to reconstruct this density is using *kernel density estimators* (*kde*). Kernel density estimators can be considered as a generalization of histograms in the sense that instead of summing up “boxes” centered at the observations, symmetric probability density functions (pdf) are added together. These symmetric pdf are called the *kernel* of the estimator and hence the name of kernel density estimator. The kernel density estimator is defined as

$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h} K \left(\frac{x - x_i}{h} \right), \tag{4.1}$$

where K is a kernel function such as the Gaussian distribution, and h is the bandwidth of the kernel. The bandwidth, is a value that indicates how wide or narrow is the shape of the kernel function. This value is important because as it is shown in Figure 4.1, if the kernel shape is narrow (width is small) the density estimate may be under-smoothed, or if the kernel shape is too wide (large width) the density estimation may be over-smoothed. Usually, an optimal value for the right amount of smoothing can be automatically obtained from the data itself. Some methods to find this optimal value are presented in the works of Stone and others [19–21].

Density estimation has been widely used to investigate the properties of a given set of data, such as skewness and multi-modality [22]. By reconstructing the density of an ensemble of sample paths at specific points in time t, and detecting the modes of this density, it is possible to achieve the detection that is not provided by the k-means algorithm, when analyzing an ensemble of stochastic simulation paths.

4.1.3 Description of the Algorithm

The contribution of this thesis is an algorithm able to “summarize” an ensemble of sample paths into one or more paths that are representative of the behavior of the system. This algorithm is similar to the method of averaging, but instead of calculating the average at every point in time, the probability density function is reconstructed using kernel density estimators, and the peaks of this density are the values used as the “average” to reconstruct the resulting paths. To illustrate this idea, let us refer to Figure 4.2. In this figure, six sample paths are shown representing the bi-stable behavior of some system. At time $t = 5$ the pdf of the paths is estimated and it is shown to have one mode because the paths are close

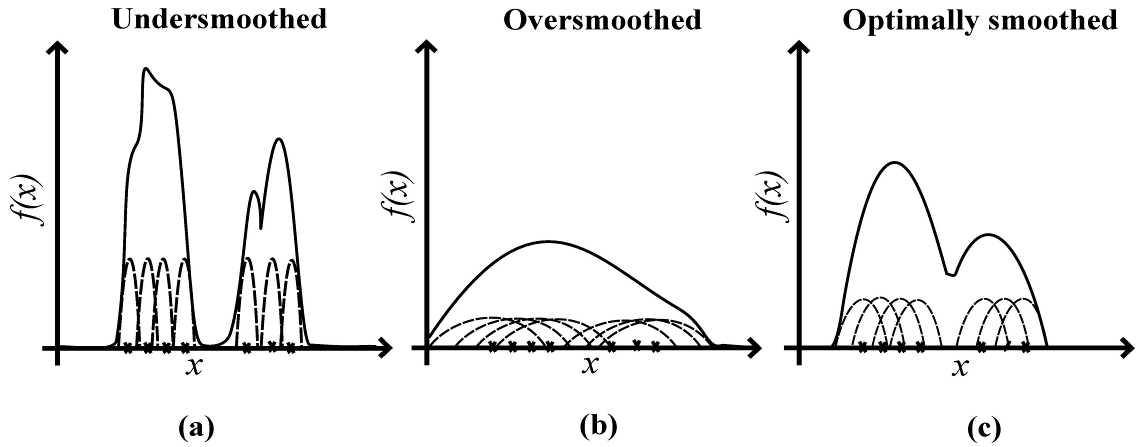

Fig. 4.1: Effect of three different bandwidths. Figure (a) has a very narrow bandwidth and the kernel density estimate is said to under-smoothed as the bandwidth is too small. Figure (b) has a very large bandwidth and the estimate is said to be over-smoothed. Figure (c) shows an estimate with an optimal bandwidth for the given data.

together forming one group. Hence, only one peak is detected in this distribution, which is marked with a red x. At time $t = 50$, however, the paths have taken two different directions. The pdf of the ensemble of paths at this point in time now shows two modes indicating the two different states of the system. These two peaks marked at time $t = 50$ will be recorded to reconstruct the two final "summarized" trajectories shown to the biological designer.

It is important to note that calculating the kde of the ensemble of runs at every time step is significantly more computationally expensive than just calculating the average. To ameliorate this situation it is possible to trade speed for accuracy by calculating the kde at time intervals greater than the time steps in the data. In other words, if the ensemble of simulation paths contains values for some number points in time, instead of calculating the kde at every time increment, it is possible to do it every λ time increments, and assume there is not a significant change in the behavior of the system during the time in-between. Of course, this may not always hold true for systems changing very rapidly, hence this is a trade-off that needs to be carefully considered.

The steps of the multi-path visualization algorithm, applied to a ensemble of stochastic simulation runs, are outlined in Algorithm 5. Steps 1 through 8 are basically "data col-

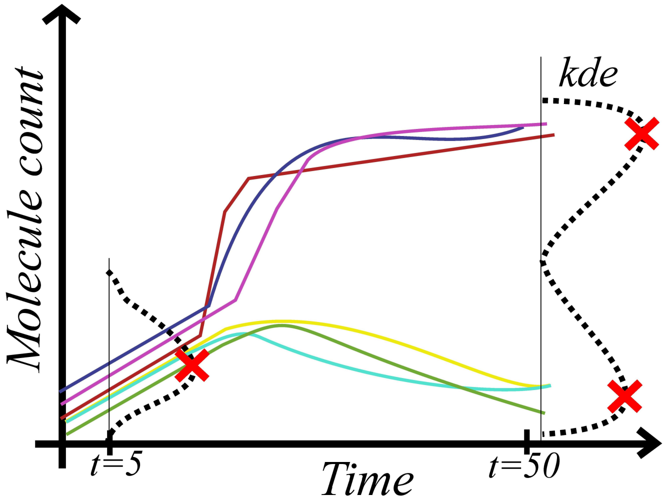

Fig. 4.2: Detection of bi-stable behavior of a system using kernel density estimators.

lection" steps to ensure that all the paths are aligned in time according to the same time increment. Step 1 initializes the simulation runs counter with $j = 1$. Step 2 initializes the n matrices $X^n_{j,k}$ to contain the initial molecule count for species $1 < i < N$, at the discrete time increment $k = 0$ for run j. The next two steps, 3 and 4, are very similar to the steps in iSSA, where the limit to the next discrete time increment is calculated. After the execution of a Gillespie SSA in step 5, the simulation time t is checked in step 6 until it has reached the time step limit. If this limit has bee reached, the molecule count for all species i is recorded for discrete time k. Otherwise, another Gillespie SSA step is performed. Step 7 checks that one simulation runs has been completed once t has reached the total $timeLimit$ for the simulation, and then another simulation run is started in step 8. As it can be noted, these are serialized steps to fill in the X^n matrices with the ensemble of runs, but since one run is independent of another, they can also be executed in parallel as long as their discrete time increments are the same. Also, to this end, the matrices can be organized in different ways to take advantage of how information is stored in computer memory, or to gain better parallelized performance. The important part is to have an ensemble of of path aligned in time to proceed with the next part of the algorithm.

The next part of the algorithm (steps 9 through 13) evaluates the ensemble of runs to reconstruct a “summarized” path or set of paths that will be shown to the biological designer. Step 9 resets the species counter and the discrete time counter. Step 10 calculates the pdf of all the runs in the ensemble at time k using kernel density estimation. The peaks of this density are calculated and recorded in step 11. It will be shown later that recording these peaks requires further analysis and computation to make sure the “summarized” paths are reconstructed appropriately. This is due to the fact that noise and outliers in the ensemble may cause spurious modes to appear in the density, which may produce a wrong number of peaks to be stored at different points in time. Step 12 is just iterates the previous step for all species in the system, while step 13 completes the analysis for the whole simulation time. One important thing to note here is parameter *Jump*, which can skip over some time steps to speed the simulation up, at the expense of accuracy.

This algorithm was tested and verified by feeding the input with different test cases involving multi-stable behavior. The test cases comprised both artificially generated data, as well as simulation results from a genetic toggle switch model. Ideally, the input data would be generated from a variety of real biochemical systems; however, finding real system models exhibiting multi-stable behavior proved to be very hard. The only real biochemical system simulated was the genetic toggle switch. Other tests involved data artificially generated in MATLAB simulating the required behavior.

Algorithm 5 Multi-path visualization algorithm.

```
1: Set $j = 1$.
2: Initialize: $X^{(i)}_{j,0}$ = init $(\mathbf{x}_i)$, and set $k = 1$.
3: Set $t' = (k - 1) \times \mathit{increment}$, and $\mathit{limit} = t' + \mathit{increment}$.
4: Set $t = t'$.
5: Execute a Gillespie SSA step.
6: If $t < \mathit{limit}$ then $\mathit{record}(X^{(i)}_{j,k})$, go to step 5.
7: If $t < \mathit{timeLimit}$ then $k = k + 1$, go to step 3.
8: If $j < \mathit{maxRuns}$ then $j = j + 1$, go to step 2.
9: Set $i = 1$, $k = 1$.
10: Calculate KDE$(X^{(i)}_{k,all})$ (i.e. $k^{th}$ column vector of $X^{(i)}$).
11: Calculate and save the peaks of the KDE for $i$.
12: If $i < N$ (number of species) then $i = i + 1$, go to step 10.
13: If $k < \mathit{timeLimit}$ then $k = k + \mathit{Jump}$, go to step 10.
```

4.1.4 Tests with Artificially Generated Data

A test bench was created in MATLAB with artificial data simulating a system exhibiting bifurcating behavior. Figure 4.3 shows this artificially generated data, which consists of one hundred sigmoid functions with randomly varied parameters and Gaussian noise added on top. The sigmoid function is defined as

$$S(t) = \frac{1}{1 + e^{-t}}. \tag{4.2}$$

The MATLAB code used to generate the data in Figure 4.3 can be found in Appendix A.1. It simulates the values for a single chemical species, but to simulate more than one species would be trivial. The data for other species would just have to be placed in their own matrices following the same steps. However, it is important to note that not every species in a system undergo bifurcation. By generating data this way, we effectively skip steps $1 - 8$ of the visualization algorithm. The rest of the algorithm only makes use of the data generated in the previous steps.

The two paths shown in Figure 4.4 were reconstructed from the generated data shown in Figure 4.3 with $Jump = 20$. The MATLAB code used to generate the data and the plots is available in Appendix A.2 and Appendix A.3.

Another example was performed generating artificial data to simulate a system undergoing two other bifurcations, after an initial bifurcation, for a total of four different system states. This may be an extreme case, which may or may not exist in nature, but it is helpful to verify the multi-path visualization algorithm. The data generated in MATLAB is shown in Figure 4.5. Again, this data is created manipulating the different parameters of the sigmoid function and tweaking the vertical and horizontal displacement. This data is placed into a matrix, where the sigmoid functions representing an individual simulation run form the rows of the matrix. This is similar to the matrix of a chemical species containing the ensemble of values of its system simulation. The MATLAB code to generate this example is also available in Appendix A.4. It is important to note that these functions are generated randomly and may look different than in Figure 4.5.

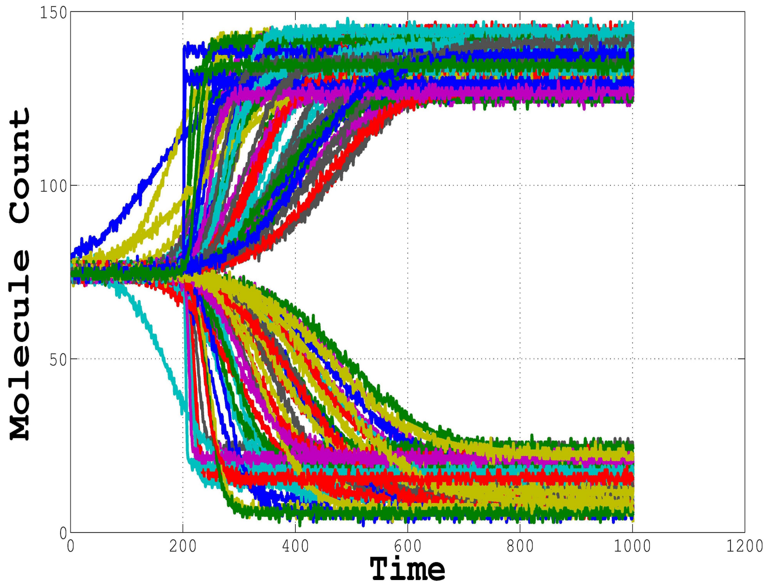

Fig. 4.3: Plot of 100 randomly generated sigmoid functions.

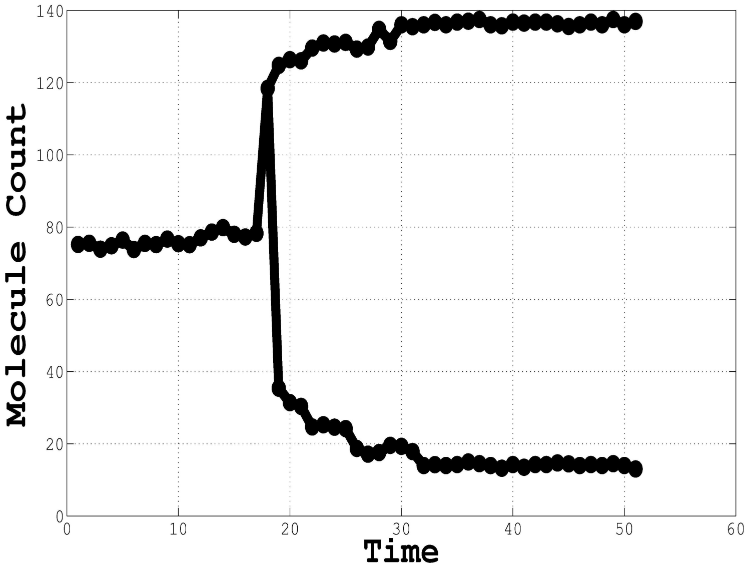

Fig. 4.4: Results generated from the multi-path visualization method when applied randomly generated data.

When the data in Figure 4.5 is provided to the multi-path simulation algorithm, the result is shown in Figure 4.6. To demonstrate the algorithm actual results, this figure shows dots every time a peak is detected in the mode of the pdf generated by the kde. The dots are not connected on purpose to demonstrate the actual results of the method and it is the reason why the paths appear with discontinuities. Connecting the dots is a trivial if the values are stored in a vector. It is necessary to mention at this point that the MATLAB script in Appendix A.2 is a simple proof of concept of the algorithm and the kernel width for the kde was given a fixed value. If the reader wants to replicate these results, small changes on this value may be necessary to get the desired result if the input data changes. This exposes a real drawback of this algorithm: the dependency on the kernel width of the kde. Although methods exist to find optimal values for the kernel width, the underlying data may vary greatly in unexpected ways and it is unclear what "optimal" means under these circumstances.

4.1.5 Test with Data from a Real Model

The method has also been tested using data from simulations of the genetic toggle switch model described in Section 2.3.4. The toggle switch model was simulated using iBioSim with 20 independent runs and a time limit of $2,000$, using the Gillespie SSA simulator option. The results of these simulations were then exported to MATLAB for processing. Figure 4.7 plots the 20 sample paths generated for molecular species TetR and Figure 4.8 shows the pdf of the ensemble, generated by the kde at at time $t = 2,000$. Again, this is only to demonstrate the algorithm in action and what happens under the hood before the "summarized" paths are shown.

The results of applying the method to the data in Figure 4.7 are shown in Figure 4.9. The results are considerably noisier than before and this is in part due to the small number of sample paths. The greater this number, the more accurate can the pdf be reconstructed at the expense of a larger running time of the algorithm. The MATLAB code for these tests can be found in Appendix A.5 and Appendix A.6.

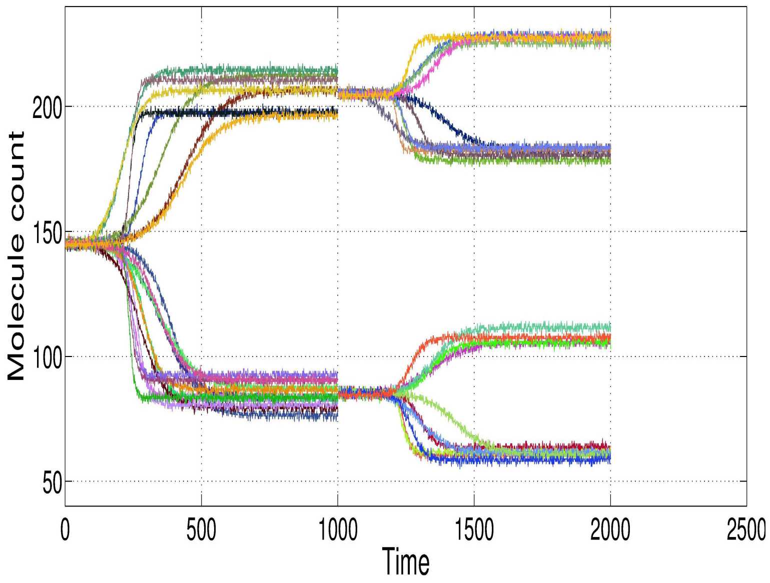

Fig. 4.5: Artificial data system undergoing multiple bifurcations.

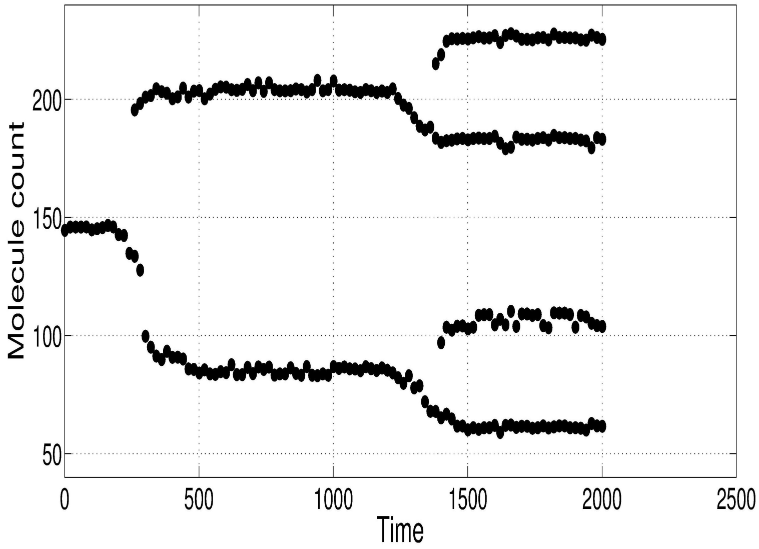

Fig. 4.6: Results of the method when data undergoes multiple bifurcations.

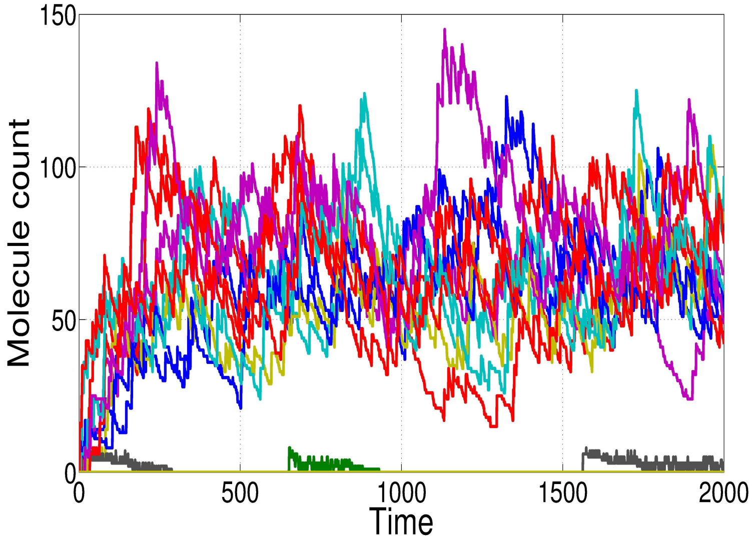

Fig. 4.7: Plots of TetR for 20 SSA runs of the toggle switch.

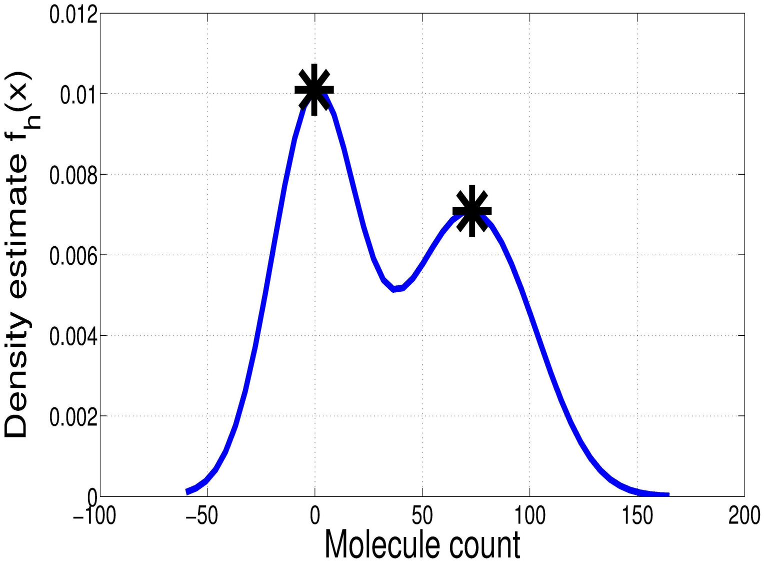

Fig. 4.8: Density distribution estimate for TetR at time $t = 2,000$.

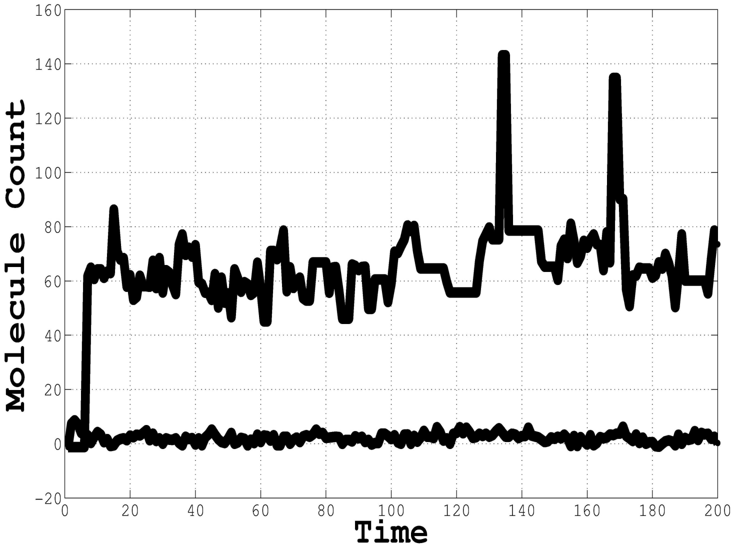

Fig. 4.9: Two stable states paths of the toggle switch recovered from the multi-path visualization algorithm.

4.1.6 Summary

An alternative method to averaging an ensemble of stochastic simulation runs has been presented as a way to distinguish possible multi-stable behavior in the system. As opposed to averaging the sample paths, this method makes use of kernel density estimators to reconstruct the pdf of the samples at specific points in time and recognizes groups of paths as indicators of possible system states. The peaks of this pdf are then stored and used to reconstruct the "summarized" path or paths representing the system's behavior.

4.2 Integration with iSSA

The multi-path visualization algorithm, as it was presented in the previous section, is capable of identifying multiple "typical" paths from an ensemble of runs. Although this method showed correct results when applied to a system with oscillating behavior, it is unclear whether it will produce correct results for all cases. So, it is desirable to integrate this method with other more robust methods like iSSA, which can to produce correct results for such systems. This integration, however, has some caveats because calculating the kde and the peaks of this density will now have an effect on the starting point of the next time increment. If more than one value are chosen as starting points for the next time increment in the regular iSSA method, then, it is unclear how many runs should be performed for each new starting state. Let us note that the word *state* here is different than the *system state* mentioned previously. In the context of of iSSA, state refers to the molecule count of some chemical special at the start of the next time increment.

One solution to the problem of how many runs to execute at the next time increment is to always perform the same number of runs (*maxRuns*) for each starting state, regardless of how many there are. However, the disadvantages of doing that are obvious. As possible bifurcations are detected, the number of performed simulation keeps adding as well as the time the algorithm takes to execute. Consider, for example, that iSSA receives as parameter $maxRuns = 50$, and after some time, four different paths are detected. The next time step will then have to perform 200 simulation runs ($4 \times maxRuns$), and this is not counting false detections that may occur due to a very noisy system or the presence of outliers. One solution to avoid increasing the number of runs considerably, could be to divide *maxRuns* by the number of newly discovered starting points and execute that number of runs for the next time increment. However, this brings up another issue, which is that simulation runs may not be evenly distributed among the different points, or the number of runs for each starting state may be significantly reduced to the point that statistics are no longer producing satisfying results.

One more consideration when referring to the integration of the multi-path visualization method with iSSA is that the internal data structure to store information in iSSA is typically a matrix, and matrices are not flexible enough to handle situations where one path bifurcates into two paths and then reintegrates into a single path. When the ensemble of sample paths has been completely generated it is easier to spot and remove these spurious peaks from the final reconstructed paths, however, when performing this processing every time increment it is more challenging to filter out unwanted peaks. Since it is impossible to predict exactly what will happen next, every peak, whether spurious or not, must be recorded and considered valid until future data can disprove it. To work with this unknown number of possible paths it may be necessary to use an unconventional data structure.

4.3 The Multi-Path Detection iSSA (MPD-iSSA) Method

Another contribution of this work is an iSSA implementation with multi-path detection capabilities. This method, called the Multi-Path Detection iSSA (MPD-iSSA), applies kernel density estimation and peak detection to incremental SSA methods to make them capable of handling multiple paths. This integration could be one step forward to the goal of creating one general visualization tool in synthetic biology to aid with the design and verification of genetic circuits. This section presents a description of the method and results obtained from the simulation.

4.3.1 Description of the Algorithm

MPD-iSSA is in essence a combination of the incremental idea of iSSA with the use of kernel density estimators to detect bifurcations and multiple stable states in a biological system. Similar to the previous multi-path visualization method applied to an ensemble of paths, a parameter $Jump$ indicates how many incremental steps to skip before performing the kde on the generated trajectories at time t. In general, we want a value that would allow the system sufficient time to show a clear separation of the paths. Otherwise, any bifurcation or multi-stable behavior could go undetected.

The steps performed by the MPD-iSSA algorithm are outlined in Algorithm 6. These steps are presented at a higher level with less details than before because it is in essence the same iSSA described in Algorithm 4 with different *select* and *record* functions that calculate the kde of the trajectories at time t and then select the trajectories closest to the peaks of the modes of this distribution. Selecting actual trajectories is important to make sure the simulation does not violate any conservation constraints in the system. Also, recording the system's state at t requires a flexible data structure that can be post-processed once the simulation to eliminate spurious paths, if necessary.

Algorithm 6 MPD-iSSA method.

```
1: Simulation is set to some initial state.
2: Stochastic simulations are run to obtain several end-states over some interval.
3: The end-states are fit to a distribution using kernel density estimation for each independent species.
4: From the density estimate, the peaks of the modes of the distribution are chosen to represent the molecule count.
5: The trajectories closest to the peaks are chosen.
6: New nodes are created in the linked structure and the molecule count values are stored.
7: Any link from a previous node is created to the new nodes.
8: Back to step 2, until simulation time is completed.
9: Nodes are traversed and cycles are eliminated, if necessary.
```

4.3.2 Conservation Constraints

Since iSSA updates the system state every time increment from the trajectories generated, a selection of the new state for the next time increment must respect any conservation laws in the system. Conservation laws regulate that physical conditions are appropriately expressed in the mathematical model. For example, let us denote B to be a promoter bounded by another chemical species like a transcription factor or RNA polymerase, and U to be the same unbounded promoter. The same promoter cannot be bounded and unbounded at the same time. The mathematical constraint in this case would be something like $B+U=1$, where the promoter is either bounded (i.e. $B=1$ and $U=0$) or unbounded (i.e. $B=0$ and $U=1$). In MPD-iSSA violations to conservation laws can occur if a new state is calculated for two or more molecular species that are dependent on each other. To illustrate this, let's use the fictitious example shown in Figure 4.10. Assume molecular species A and B are dependent on each other with the system constraint $A+B=100$. After some processing, if the new calculated states for A and B are $A=95$ and $B=6$, the conservation constraint is violated and the results are no longer valid.

Conservation constraints can appear in many networks as conserved groups of molecules called *moieties* [23]. These groups can be spotted by analyzing the network's topology, which is embedded in the stoichiometry matrix, as explained by Sauro and Ingalls [24]. Essentially, when conservation constraints are imposed on the system, there will be linear dependencies among the rows (species) of the stoichiometry matrix. MPD-iSSA avoids violating conser-

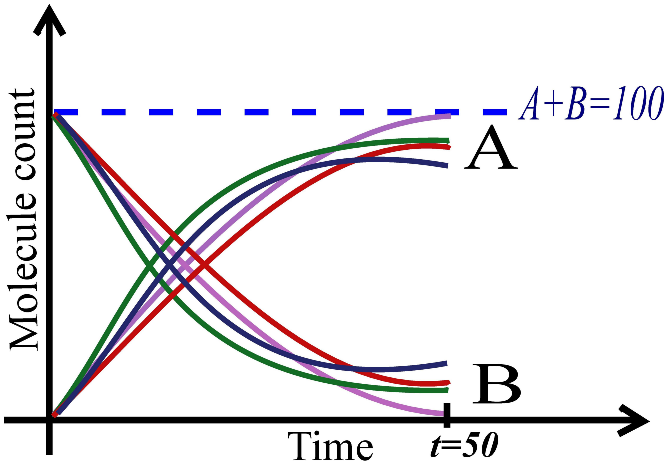

Fig. 4.10: Example of a conservation constraint.

vation constraints in the system by choosing real trajectories closest to the peaks in the modes of the distribution. By choosing real real trajectories it is guaranteed that the new starting states for the next time increment will abide the conservations constraints of the system. Also, choosing one trajectory from *maxRuns* is computationally less expensive than evaluating every chemical species in the systems and calculating new values for the dependent ones to ensure the constraints are not violated.

4.3.3 A Flexible Data Structure

In general, storing information in memory arrays is very efficient because information is placed in contiguous spaces in memory, which helps for fast retrieval. However, once the memory for these arrays has been allocated and information has been placed in them, it is cumbersome to change their dimensions. For a method like MPD-iSSA, which can have outliers appearing and disappear spontaneously at different time increments, this can be burden. The *record* function for MPD-iSSA requires a more flexible data structure that is able to accommodate any number of starting trajectories, whether spurious or not.

A data structure similar to a linked-list can be used to allow MPD-iSSA fast adaptation to changes in the underlying behavior. A linked-list is a data structure where nodes are linked together in a linear order. In the regular linked-list structure every node contains only one pointer to the next node, however, by changing this property and allowing multiple pointers to other nodes it is possible to effectively achieve the desired characteristics. This idea is illustrated in Figure 4.11. Every block represents a node containing two parts: a *value*, and *pointers* to others nodes. The value holds the actual molecule count found of the trajectories closest to the peaks of the modes in the pdf generated by the kde, and the pointers are the links to other nodes in the path. One advantage of this structure is that cycles formed by spurious nodes can be detected when reconstructing the final path in the post-processing stage. When the linked structured is traversed, if any set of nodes leads to an already visited node, these nodes can be deleted from the final path.

If the information stored in the structure shown in Figure 4.11 were to be plotted, the designer would see something like Figure 4.12. If necessary, the cycle marked in red, representing the false detection of a third path at time $t = 3$ could be filtered and only two paths would be effectively shown to the biological designer.

4.3.4 Computational Complexity Analysis

The computational complexity of this method is the same of the original iSSA method, plus the added complexity of the new *record* and *select* functions. In Big-O notation, the complexity of a naive implementation of these two functions is $O(N^2)$, which is the complexity of calculating the density estimation. However, according to Elgammal et al. [25], this can be optimized to $O(M + N)$, where M is the length of the Gaussian kernel and

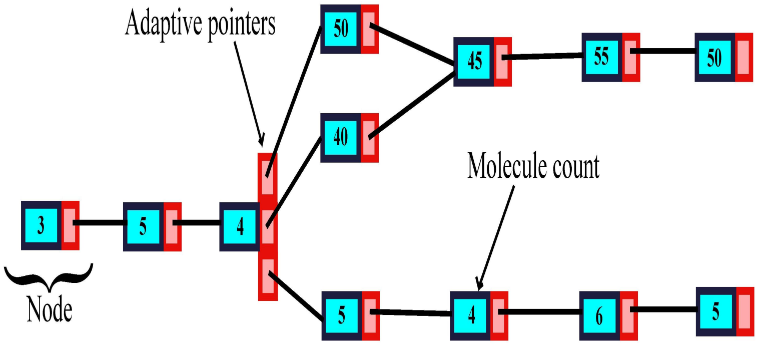

Fig. 4.11: Flexible data structure to store MPD-iSSA information.

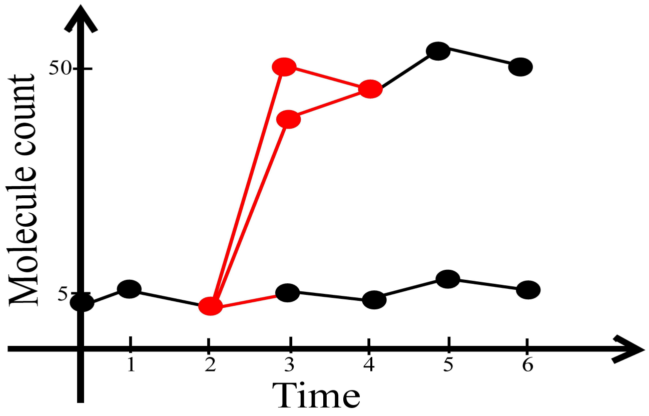

Fig. 4.12: Plot of the information contained in data structure of Figure 4.11.

N is the number of evaluation points. Besides the calculation of the density, the remaining parts of the two functions can all be computed in linear time $O(N)$ because they only require scanning through the N sample points in the density a constant number of times.

If instead of choosing the trajectories closest to peaks in the distribution, this method calculated the values of the dependent species from the values of the independent species, the computational complexity could easily reach $O(N^3)$ because it would be necessary to perform some matrix multiplications. This can become prohibitively expensive for very large systems or very large simulations. The fact that functions *record* and *select* can be optimized to almost linear time complexity in MPD-iSSA is very significant for the practicality of this method.

4.3.5 Simulation Results

The MPD-iSSA method has been implemented in MATLAB simulating the genetic toggle switch described by Lepzelter et al. [26]. This model is simpler than the one simulated in iBioSim with fewer species and reactions. It is described by the following eight reactions:

$$O_A^{on} + 2B \overset{h_A}{\rightarrow} O_A^{off}, \tag{4.3}$$

$$O_A^{off} \overset{f_A}{\rightarrow} O_A^{on} + 2B, \tag{4.4}$$

$$O_B^{on} + 2A \overset{h_B}{\to} O_B^{off}, \tag{4.5}$$

$$O_B^{off} \overset{f_B}{\to} O_B^{on} + 2A, \tag{4.6}$$

$$O_A^{on} \overset{g_A}{\to} O_A^{on} + b_A A, \tag{4.7}$$

$$O_B^{on} \overset{g_B}{\to} O_B^{on} + b_B B, \tag{4.8}$$

$$A \overset{k}{\to}, \tag{4.9}$$

$$B \overset{k}{\to} . \tag{4.10}$$

The simulation had parameters $maxRuns = 30$, $Jump = 20$, and $timeLimit = 400$. The trajectories generated by the Gillespie SSA steps are shown in Figure 4.13, while the results of the MPD-iSSA method are shown in Figure 4.14. It can be noted that the trajectories in Figure 4.11 are relatively less noisier than before. This may be due to the fact that all trajectories are brought back to one of the two common points when starting the next time increment. Also, let us note that the generated paths resulting from the algorithm are shorter in time with respect to the original Gillespie SSA trajectories due to the $Jump$ value. In this sense, these generated paths should be regarded as paths showing the "typical behavior" of the system and not trajectories with real values the specific points in time, although the time axis could be scaled to meet this criteria.

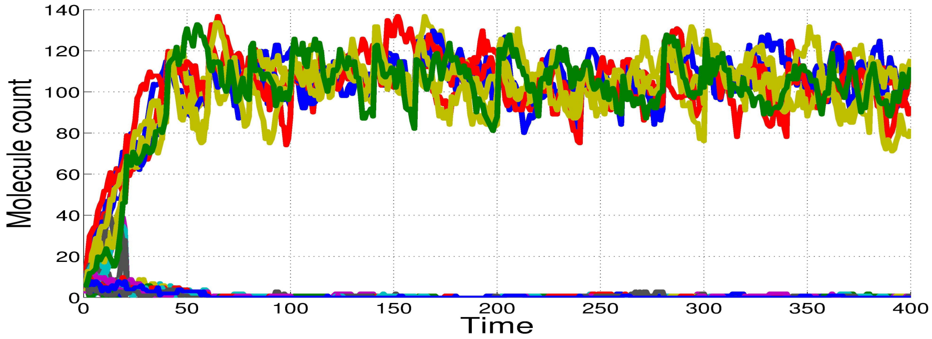

Fig. 4.13: Gillespie SSA trajectories of a genetic toggle switch simulated with MPD-iSSA.

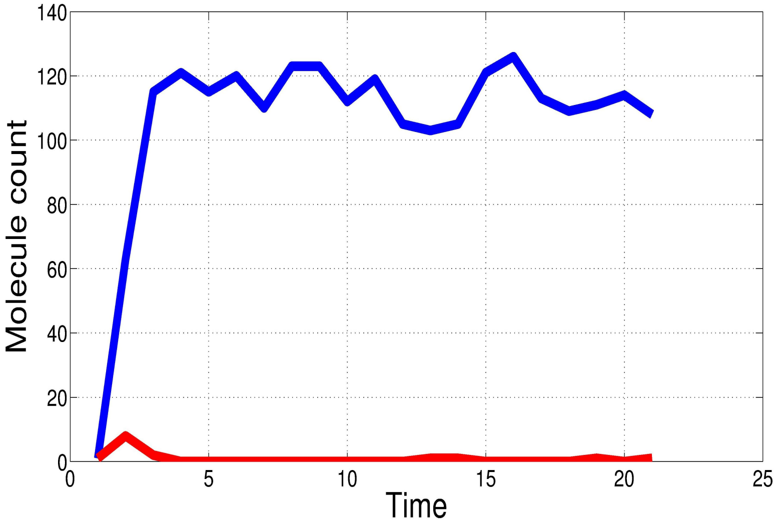

Fig. 4.14: Simulation results of a genetic toggle switch with MPD-iSSA.

4.3.6 Summary

The Multi-Path Detection iSSA (MPD-iSSA) method has been presented as an improvement of the original iSSA method to provide the capability of simulating systems with multi-stable behavior. The main difference rely in the *record* and *select* functions. The *record* function incorporates an internal data structure with flexibility to handle spurious trajectories that can be filtered out in a post-processing stage after the simulation has completed. This eliminates the burden on the designer of having to carefully select an optimal kernel width for the density estimation. The *select* function makes use of this density and a peak detection algorithm to find the trajectories closest to the peaks, which will be the starting points of the next time increments.

Chapter 5

Discussion

The dynamics of genetics circuits is very complex and sometimes their behavior varies greatly from one simulation run to another. To capture the “typical” behavior of the system, researchers and biochemical designers often execute many simulation runs and perform statistical analysis on the ensemble of trajectories. Usually, this statistical analysis consists of averaging the ensemble of trajectories, which tends to mask important system dynamics and hide multi-stable behavior. This thesis provides a method to detect and visualize this multi-stable behavior from an ensemble of sample paths. The method makes use of kernel density estimators to generate the probability density function of the trajectories at time t to detect possible clustering of paths around some number of molecules, which could mean that this is one of the “typical” behavior of the system. We say “typical” because that is a behavior trajectories tend to follow more frequently.

When the density of an ensemble of trajectories is reconstructed using kde, the clusters of paths around some molecule number appear as modes of the distribution. So, identifying the paths lying in specific modes of the density is essentially a clustering operation, and finding the trajectory closest to the peak of one mode is equivalent to finding the centroid of that cluster.

The multi-path visualization method applied to an ensemble of stochastic simulations helps designers of biological systems to view a “summary” of the ensemble, which shows more correct results than averaging, when the system exhibits multi-stable behavior. However, it is unclear whether this method will always produce correct results if the system exhibits oscillatory behavior. The iSSA method, on the other hand, has shown to produce correct results for these type of systems, but fails to account more than one trajectory for systems with multi-stable behavior. A new method with hybrid characteristics from the multi-path visualization method and iSSA is then desirable to get one step closer of a general simulation tool of genetic circuits.

The Multi-Path Detection iSSA (MPD-iSSA) method has been proposed and simulated with promising results. This method makes some changes in the *record* and *select* functions of the original iSSA to provide the capability of detecting and handling multiple trajectories for systems exhibiting multi-stable behavior. The *record* function has been changed to store the state of system at each time increment in a more flexible data structure, which can later be analyzed after the simulation has completed to filter out unwanted trajectories.

However, it remains to be shown the physical meaning of an MPD-iSSA result compared to an actual Gillespie SSA trajectory. As it was shown in the previous chapter, an MPD-iSSA trajectory if formed by "snapshots" of the system state at specific points in time determined by the *Jump* parameter. Even if we align in time this resulting trajectory with an actual Gillespie SSA path and extrapolate the points in-between, should biological designers regard it as a real path or just an example of what the behavior of the system looks like? Extrapolating values from the system state at one point time to another creates fictitious values that were not generated by the system's dynamics. Could these values violate the system's conservation laws at some point?

Besides the previous questions, there are some other questions that remain unanswered. For example, how do biological designers choose an appropriate *Jump* value for every system that would allow the multi-stable behavior to be detected by MPD-iSSA? Is there a way to automatically calculate this number from the reactions based on how fast or slow changes the dynamics of the system? Further research in this area is required to answer these questions.

With the new data structure proposed for MPD-iSSA to store the snapshots of the system at different points in time and reconstruct the typical behavior from them, there is a variety of features that can be added to the method. For example, the nodes can contain not only the molecule count for each species at a specific point in time, but also the number of trajectories that fall in the same cluster or mode of the distribution. This could be use calculate a percentage of trajectories that follow one path or another. This is especially useful to determine if there is one preferred path among all possible. Also, based on the percentage of trajectories going different ways, the designer may choose to show or hide paths with lower probabilities. This is also another way to filter out unwanted paths from the results.

More work needs to be done to test these methods with systems showing not only multi-stable behavior, but also oscillating, and possibly other system behaviors not currently considered. Even though these methods were designed to solve the problem of detecting multiple paths and multi-stable behavior in biological systems, it may be interesting to know if they can also solve other kinds of problems. Also, one future goal is to embed these methods within tools like iBioSim that support simulation and verification of experimental designs.

References

[1] B. Ingalls, *Mathematical modeling in systems biology: An introduction.* Cambridge, MA: The MIT Press, 2013.

[2] N.-P. Nguyen, C. Myers, H. Kuwahara, C. Winstead, and J. Keener, "Design and analysis of a robust genetic muller c-element," *Journal of Theoretical Biology*, 2009.

[3] T. K. Lu, A. S. Khalil, and J. J. Collins, "Next-generation synthetic gene networks," *Nature Biotechnology*, vol. 27, pp. 1139–1150, 2009.

[4] J. Hasty, D. McMillen, and J. J. Collins, "Engineered gene circuits," *Nature*, vol. 420(6912), pp. 224–230, 2002.

[5] C. J. Myers, *Engineering Genetic Circuits.* Salk Lake City, UT: Chapman & Hall/CRC, 2009.

[6] P. S. Swain and A. Longtin, "Noise in genetic and neural networks," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 16, no. 2, 2006.

[7] R. Weiss, S. Basu, S. Hooshangi, A. Kalmbach, D. Karig, R. Mehreja, and I. Netravali, "Genetic circuit building blocks for cellular computation, communications, and signal processing," *Natural Computing*, pp. 47–84, 2003.

[8] M. Snelgrove, *Oscillator.* Ontario, CA: AccessScience, McGraw Hill Education, 2012.

[9] M. B. Elowitz and S. Leibler, "A synthetic oscillatory network of transcriptional regulators," *Nature*, vol. 403, pp. 335–338, 2000.

[10] M. J. Gardner, K. E. Hubbard, C. T. Hotta, A. N. Dodd, and A. A. R. Webb, "How plants tell the time," *The Biochemical Journal*, vol. 397, no. 1, pp. 15–24, 2006.

[11] W. O. Friesen and G. D. Block, "What is a biological oscillator?" *American Journal of Physiology – Regulatory, Integrative and Comparative Physiology*, vol. 246, no. 6, pp. R847–R853, 1984.

[12] T. S. Gardner, C. R. Cantor, and J. J. Collins, "Construction of a genetic toggle switch in *escherichia coli*," *Nature*, vol. 403, pp. 339–342, 2000.

[13] M. Gibson and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels," *Journal of Physical Chemistry*, vol. A 104, pp. 1876–1889, 2000.

[14] D. T. Gillespie, "The chemical langevin equation," *The Journal of Chemical Physics*, vol. 113, pp. 297–306, 2000.

[15] C. Winstead, C. Madsen, and C. Myers, "issa: An incremental stochastic simulation algorithm for genetic circuits," *Proceedings of 2010 IEEE International Symposium in Circuits and Systems (ISCAS)*, pp. 553–556, 2010.

[16] H. Kuwahara, C. Madsen, I. Mura, C. Myers, A. Tejada, and C. Winstead, *Efficient stochastic simulation to analyze targeted properties of biological systems*, C. Myers, Ed. Sciyo, 2010.

[17] T. Shiraishi, S. Matsuyama, and H. Kitano, "Large-scale analysis of network bistability for human cancers," *PLoS computational biology*, vol. 6, no. 7, p. e1000851, Jan. 2010.

[18] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[19] C. J. Stone, "An asymptotically optimal window selection rule for kernel density estimates," *The Annals of Statistics*, vol. 12(4), pp. 1285–1297, 1984.

[20] V. C. Raykar and R. Duraiswami, "Fast optimal bandwidth selection for kernel density estimation," *Proceedings of the sixth SIAM International Conference on Data Mining*, pp. 524–528, 2006.

[21] S. Sheather and M. Jones, "A reliable data-based bandwidth selection method for kernel density estimation," *Journal of the Royal Statistical Society*, vol. 53, pp. 683–690, 1991.

[22] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. London, UK: Chapman and Hall/CRC, 1986.

[23] J. G. Reich, *Energy Metabolism of the Cell: A Theoretical Treatise*. Waltham, MA: Academic Press, 1981.

[24] H. M. Sauro and B. Ingalls, "Conservation analysis in biochemical networks: computational issues for software writers," *Biophysical chemistry*, vol. 109, no. 1, pp. 1–15, Apr. 2004.

[25] A. Elgammal, R. Duraiswami, and L. Davis, "Efficient kernel density estimation using the fast gauss transform with applications to color modeling and tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 11, pp. 1499–1504, 2003.

[26] D. Lepzelter, K.-Y. Kim, and J. Wang, "Dynamics and intrinsic statistical fluctuations of a gene switch," *The Journal of Physical Chemistry B*, vol. 111, no. 34, pp. 10 239–10 247, 2007.

Appendices

Appendix A

MATLAB Scripts

A.1 Artificial Data Simulating Bifurcating Behavior

```

clear all

% Variance of molecules
sigma = 5;
a = 50; b = 70;
runs = 100;
t_start = -200; t_end = 800; step = 1;
len = (t_end - t_start)/step + 1;

mydata = zeros(runs,len);

figure(1)
for idx = 1:runs

    m_mean = ceil((b - a) * rand + a);

    t = [t_start:step:t_end];
    p(idx,:) = (m_mean./(1 + (50 * rand)*exp(-t/(80 * rand)))) + ...
        floor(randn(1, len));

    if (rand < 0.5)
        p(idx,:) = -p(idx,:);
    end

    plot(t - t_start, p(idx,:) + b+sigma, 'color', [rand rand rand])
    mydata(idx,:) = p(idx,:) + b+sigma;
    hold on

end

xlabel('Time','FontSize',16)
ylabel('Molecule count','FontSize',16)
title([num2str(runs) ' independent runs'],'FontSize',18)
grid on
```

A.2 Testing Algorithm With Artificial Data

```
TetR = mydata.';
[m n] = size(TetR);

figure(1)
xlabel('Molecule count','FontSize',22)
ylabel('Probability P(x)','FontSize',22)
title('PDF at time t = 1','FontSize',22)

theplot = zeros(2,length(1:20:m));
jdx = 1;

for idx=1:20:m
   [f,xi] = ksdensity(TetR(idx,:),'npoints',50,'width',20);

   figure(1)
   plot(xi,f,'LineWidth',3.5);
   title(['PDF at time t = ' num2str(idx)],'FontSize',22)
   grid on;

   [maxtab, mintab] = peakdet(f, 0.00001, xi);
   hold on;
```

```
    plot(maxtab(:,1), maxtab(:,2), 'r*', 'MarkerSize', 25);
    hold off;
    pause(.1)

    if ( length(maxtab(:,1)) == 1 )
        theplot(1,jdx) =  maxtab(:,1);
        theplot(2,jdx) =  maxtab(:,1);
    else
        if ( length(maxtab(:,1)) > 1 )
         theplot(:,jdx) =  maxtab(1:2,1);
        end
    end
    jdx = jdx + 1;

    figure(2)
    plot(idx.*ones(size(maxtab(:,1)),1), maxtab(:,1),'k.', 'MarkerSize', 26)
    grid on;
    ylabel('Molecule count','FontSize',16)
    xlabel('Time','FontSize',16)
    title('Peaks of the Distributions','FontSize',22)
    hold on;

end

figure(1)
xlabel('Molecule count','FontSize',22)
ylabel('Probability P(x)','FontSize',22)
```

A.3 Peak Detection Algorithm

```
function [maxv]=peakdet(v, ∆, x)

maxv = [];

```

```
mn = Inf; mx = -Inf;

for i=1:length(v)
  this = v(i);
  if this > mx
      mx = this; mxpos = x(i);
  end
  if this < mn
      mn = this;
  end
  if this < mx-Δ
      maxv = [maxv ; mxpos mx];
      mn = this;
  end
end
```

A.4 Artificial Data Simulating Multiple Bifurcating Behavior

```
% Variance of molecules
sigma = 5;
a = 50; b = 70;
runs = 20;
t_start = -200; t_end = 800; step = 1;
len = (t_end - t_start)/step + 1;
len2 = 2*len;

mydata = zeros(runs,len2);

figure(1)

for idx = 1:runs

    m_mean = ceil((b - a) * rand + a);

    t = [t_start:step:t_end];
    p(idx,:) = (m_mean./(1 + (50 * rand)*exp(-t/(70 * rand)))) + ...
        floor(randn(1, len));

    if (rand < 0.5)
        p(idx,:) = -p(idx,:);
    end

    plot(t - t_start, p(idx,:) + 2*b+sigma, 'color', [rand rand rand])
    mydata(idx,1:len) = p(idx,:) + 2*b+sigma;
    hold on
end

xlabel('Time','FontSize',16)
ylabel('Molecule count','FontSize',16)
title([num2str(runs) ' independent runs'],'FontSize',18)
grid on

a = 20; b = 30;
runs = 10;
for idx = 1:runs

    m_mean = ceil((b - a) * rand + a);

    t = [t_start:step:t_end];
    p(idx,:) = (m_mean./(1 + (50 * rand)*exp(-t/(70 * rand)))) + ...
        floor(randn(1, len));

    if (rand < 0.5)
        p(idx,:) = -p(idx,:);
    end

    mydata(idx,(len+1):len2) = p(idx,:) + 80+sigma;
    plot(t - t_start + len, p(idx,:) + 80+sigma, 'color', [rand rand rand])
    hold on
```

```
end

for idx = 1:runs

    m_mean = ceil((b - a) * rand + a);

    t = [t_start:step:t_end];
    p(idx,:) = (m_mean./(1 + (50 * rand)*exp(-t/(70 * rand)))) + ...
        floor(randn(1, len));

    if (rand < 0.5)
        p(idx,:) = -p(idx,:);
    end

    mydata(idx+10,(len+1):len2) = p(idx,:) + 200+sigma;
    plot(t - t_start + len, p(idx,:) + 200+sigma, 'color', [rand rand rand])
    hold on
end
```

A.5 Gillespie SSA Algorithm for Multiple Runs

```
function data = mgillespie(state, parms, tau)

X0 = state.X;
Nruns = parms.Nruns;
multiplicity = parms.multiplicity;
Δ = parms.Δ;
c = parms.c;

[M N] = size(Δ);

%///////////////// INITIALIZE THE SIMULATION  //////////////////////
h = zeros(1,M);
a = zeros(1,M);
```

```
acc = zeros(1,M);

data = zeros(Nruns, N);

%///////////////   MAIN SIMULATION LOOP   /////////////////////////

for rundx = 1:1:Nruns
    t = 0;
    X = X0(rundx,:);

    while (t < tau)
        a0 = 0;

        %COMPUTE THE PROPENSITIES //
        for idx = 1:1:M

            h(idx) = 1;

            for jdx=1:1:N
                k = multiplicity(idx,jdx);
                if (k>0)
                    n = X(jdx);
                    if (k == 1)
                        h(idx) = h(idx) * n;
                    elseif (n ≥ k)
                        h(idx) = h(idx) * nchoosek(n,k);
                    else
                        h(idx) = 0;
                    end
                end
            end

            a(idx) = h(idx) * c(idx);

            a0 = a0 + a(idx);
            acc(idx) = a0;
        end

        r1 = rand;
        r2 = rand;

        %// GENERATE THE NEXT REACTION TIME //
        dt = (1/a0)*log(1/r1);
        t = t + dt;

        %// GENERATE THE NEXT REACTION EVENT //
        r2a0 = r2*a0;
        mudx = 1;

        for idx=1:1:M
            testval = acc(idx);
            if (r2a0 > testval)
                mudx = mudx + 1;
            end
        end

        %// UPDATE THE SYSTEM'S STATE //
        for jdx=1:1:N
            X(jdx) = X(jdx) + Δ(mudx,jdx);
            if (X(jdx) < 0)
                X(jdx) = 0;
            end
        end
    end


    for jdx=1:1:N
        data(rundx,jdx) = X(jdx);
    end
end
```

A.6 Multi-Path Visualization of a Genetic Switch Model

```
function data = mgillespie(state, parms, tau)
clear all;

%%%%%% Genetic Toggle Switch Version %%%%%

%      OonA   OnffA   OonB   OoffB    A    B
X0 = [  1       1       1       1      1    1 ];

△ = [−1  1  0  0  0 −2 ;  % OonA + 2B −> OnffA
           1 −1  0  0  0  2 ;  % OnffA −> OonA + 2B
           0  0 −1  1 −2  0 ;  % OonB + 2A −> OoffB
           0  0  1 −1  2  0 ;  % OoffB −> OonB + 2A
           0  0  0  0  1  0 ;  % OonA −> OonA + bA∗A
           0  0  0  0  0  1 ;  % OonB −> OonB + bB∗B
           0  0  0  0 −1  0 ;  % A −> empty
           0  0  0  0  0 −1 ]; % B −> empty

% Reaction rates
c = [0.0005  0.005  0.0005  0.005  0.05  0.05  1e−3  1e−3];

%  [OonA OnffA OonB OoffB  A   B ]
multiplicity = [
      1    0     0    0    0   2 ;
      0    1     0    0    0   0 ;
      0    0     1    0    2   0 ;
      0    0     0    1    0   0 ;
      1    0     0    0    0   0 ;
      0    0     1    0    0   0 ;
      0    0     0    0    1   0 ;
      0    0     0    0    0   1 ];

tau = 100;
```

```
Nruns = 30;

% Repeat states for Nruns
state.X = repmat(X0,Nruns,length(X0));
parms.Nruns = Nruns;
parms.multiplicity = multiplicity;
parms.Δ = Δ;
parms.c = c;

Tsim = 15;
Tsteps = 1000;

figure(1)
count = 1;
spA_PathUp(count) = X0(5);
spA_PathDown(count) = X0(5);

for tdx = 1:Tsteps
  data = mgillespie(state,parms,tau);
  state.X = data;

  % Get molecule count for all runs of species A
  trace_sp5(count,:) = data(:,5).';
  % Get molecule count for all runs of species B
  trace_sp6(count,:) = data(:,6).';

  % Calculate KDE every 5 steps
  if (mod(tdx,50)==0)
      [bandwidth density nmolecules] = kde(trace_sp5(count,:),20);
      [maxv minv] = peakdet(density,0.000001);
      [pm pn] = size(maxv);
      % If there is more than one path

      if ( pm > 1)

```

```
          [group med] = getmeans(trace_sp5(count,:),nmolecules(maxv(:,1)));

          % Get paths closer to the bottom peak
          paths_down = find(group == 1);
          meanp_down= data(med(1),:);

          % Get paths closer to the top peak
          paths_up = find(group == 2);
          meanp_up = data(med(2),:);

          % Replace corresponding paths with the mean path
          for idx=1:length(paths_up)
            data(paths_up(idx),:) = meanp_up;
          end

          for idx=1:length(paths_down)
            data(paths_down(idx),:) = meanp_down;
          end

          spA_PathDown(count+1) = trace_sp5(count,med(1));
          spA_PathUp(count+1) = trace_sp5(count,med(2));

          state.X = data;
          count = count+1;
      end
  end
end

%%
figure(1)
plot(1:length(spA_PathUp),spA_PathUp,'b','LineWidth',5)
hold on;
plot(1:length(spA_PathDown),spA_PathDown,'r','LineWidth',5)
grid on;
hold off;
```