

HIERARCHICAL STOCHASTIC SIMULATION OF GENETIC CIRCUITS

by

Leandro Hikiji Watanabe

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Computer Science

School of Computing

The University of Utah

May 2014

Copyright © Leandro Hikiji Watanabe 2014

All Rights Reserved

ABSTRACT

This thesis describes a hierarchical stochastic simulation algorithm which has been implemented within `iBioSim`, a tool used to model, analyze, and visualize genetic circuits. Many biological analysis tools flatten out hierarchy before simulation, but there are many disadvantages associated with this approach. First, the memory required to represent the model can quickly expand in the process. Second, the flattening process is computationally expensive. Finally, when modeling a dynamic cellular population within `iBioSim`, inlining the hierarchy of the model is inefficient since models must grow dynamically over time. This paper discusses a new approach to handle hierarchy on the fly to make the tool faster and more memory-efficient. This approach yields significant performance improvements as compared to the former flat analysis method.

CONTENTS

ABSTRACT	i
LIST OF FIGURES	iii
LIST OF ALGORITHMS	v
ACKNOWLEDGMENTS	vi
CHAPTERS	
1. INTRODUCTION	1
1.1 Thesis Contributions	2
1.2 Thesis Outline	2
2. BACKGROUND	3
2.1 Genetic Circuit Models	3
2.2 Chemical Reaction Models	4
2.3 ODE Simulation	5
2.4 Stochastic Simulation	6
2.5 Cellular Population Models	8
3. HIERARCHICAL SIMULATION	12
3.1 hSSA Algorithm Overview	13
3.2 Example	16
3.3 Results	22
4. CONCLUSIONS	28
4.1 Summary	28
4.2 Future Work	28
REFERENCES	30

LIST OF FIGURES

2.1	Repressilator circuit modeled in iBioSim.	4
2.2	Repressilator modeled as chemical reactions.	5
2.3	Repressilator ODE simulation results.	7
2.4	Repressilator SSA simulation results.	9
2.5	Cellular population modeling in iBioSim.	10
2.6	Cellular population visualization in iBioSim.	11
2.7	Diffusion in an iBioSim grid model [21].	11
3.1	Simple chemical reaction network.	16
3.2	Hierarchical model example.	17
3.3	Initial amount for all species in the hierarchical model.	18
3.4	Example of replacement.	18
3.5	Example of replacement.	19
3.6	Propensity for each reaction.	19
3.7	Computed next reaction time and next reaction selected.	19
3.8	Amount for all species in the hierarchical model after first iteration.	20
3.9	Propensity for each reaction for the second iteration.	20
3.10	Amount for all species in the hierarchical model after second iteration.	21
3.11	Propensity for each reaction for the third iteration.	21
3.12	Amount for all species in the hierarchical model after third iteration.	22
3.13	Propensity for each reaction for the fourth iteration.	23
3.14	Comparison of performance of the SSA using flattening and our hierarchical approach.	23
3.15	Comparison of simulation time of the SSA using flattening and our hierarchical approach.	24
3.16	25 Repressilator circuits in which GFP degradation is deleted and has the local GFP species replaced with the top-level one.	25
3.17	Total amount of GFP for 25 instances of the repressilator circuit.	25
3.18	Comparison of performance of the SSA using flattening and our hierarchical approach for a model that includes replacements and deletions.	26

3.19 Comparison of simulation time of the SSA using flattening and our hierarchical approach for a model that includes replacements and deletions. 27

LIST OF ALGORITHMS

2.1 Gillespie's SSA	8
3.1 Hierarchical SSA	13
3.2 <i>initialize</i> (M)	14
3.3 <i>computePropensities</i> (M, \mathbf{x})	14
3.4 <i>selectNextReaction</i> (\mathbf{a}, a_0)	15
3.5 <i>updateState</i> ($M, t, \tau, \mathbf{x}, \nu, \mu$)	15

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Prof. Chris Myers for giving me the opportunity to contribute to the development of iBioSim. Prof. Chris Myers is definitely a top-notch professor and advisor. Since I started working in the research lab, I learned many things I would have never thought I would be able to learn. Furthermore, my experiences since I joined the research group have been phenomenal.

While my advisor, Chris Myers, has allowed all of these to happen, my experiences as a research assistant could not have been successful without good colleagues in the lab. I would like to thank everyone in the research lab for helping me with all sorts of things. Not only they helped me with questions I had about the research or school related topics, but they have given me many advices. In addition, I would like to thank the past students who have contributed to the development of iBioSim.

I would like to thank Tramy Nguyen for helping me innumerous times. She helped me practice for my presentations, which I definitely needed help with. She listened to me practice my talk so many times and I still wonder how she managed to do that. I appreciate the fact she supported my research. In addition, she was always cheering me up when I was feeling overwhelmed and stressed out. I will always be in debt.

Most importantly, I would like to thank my parents, Egi and Suzana, and brother, Alex. I appreciate all the effort from my parents for giving me everything I needed and more. I know how much they sacrificed in order to give me everything I have today. I thank my brother for helping me whenever I needed and also for answering all the questions related to chemistry and biology I had while I was writing my thesis. His feedback on my thesis was very helpful.

The authors of this work are supported by the National Science Foundation under Grant No. CCF-1218095. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We would also like to thank Jason Stevens who developed the original non-hierarchical simulator.

CHAPTER 1

INTRODUCTION

Genetic engineering and other forms of biotechnology are having a substantial impact on the world economy, and this trend is likely to continue [12]. Although genetic engineers have had success on the development of some pharmaceuticals, the field has encountered many challenges. One of the challenges is to gain control over the cells by predicting their behavior [12]. This motivated the creation of the synthetic biology field, which is a subset of bioengineering developed by biologists and engineers for the systematic design of *genetic circuits*. Genetic circuits are used to regulate gene expression at many molecular levels and can potentially be used to consume toxic spills and waste [2, 4], destroy tumors [1, 19], and produce drugs or bio-fuels more efficiently [18, 20].

Since *electronic design automation* (EDA) software tools have had success in the electrical engineering field in the construction of complex circuits, the synthetic biology field has introduced the development of *genetic design automation* (GDA) tools [15]. `iBioSim`, being developed at the University of Utah, is one example of such a tool [13]. The `iBioSim` tool can be used to model, analyze, and visualize genetic circuits. Models in `iBioSim` are represented using the *systems biology markup language* (SBML), which is a standard representation format for chemical reaction network models in systems biology [9].

`iBioSim` has had positive results with static models [10, 17, 13]. However, many biological models are complex and rely on the communication and cooperation of cells. Recently, `iBioSim` has been extended to support such dynamic modeling of bacterial populations [21]. However, improvements in simulation performance are needed. Currently, the bottleneck of population models in `iBioSim` is the routine that flattens out the hierarchical constructs. This routine is very expensive and causes the state space of the model to grow quickly.

This fact motivated the development of the *hierarchical stochastic simulation algorithm* (hSSA) described in this paper. This method handles hierarchy at runtime rather than

compiling the hierarchical model into a flat model before simulation. The *discrete event system specification* (DEVS) formalism developed by Zeigler [6] has been widely used for the modeling and simulation of discrete event systems. The DEVS formalism allows one to build systems in a modular and hierarchical fashion. Extensions to DEVS have been made to support stochastic methods [5]. In addition, DEVS-based applications include modeling and simulation of systems biology models [14]. The paper adapts hierarchical stochastic simulation to the requirements necessary for chemical reaction networks specified in SBML.

1.1 Thesis Contributions

Hierarchy is syntactic sugar to SBML. Hierarchical models in SBML can be constructed without any notion of hierarchy. However, constructing such models would be very tedious. Not only tedious, constructing hierarchical models in a flat manner is difficult to reason about.

Despite the fact that the extension of SBML to allow hierarchical modeling is a step forward in the standard, the analysis of hierarchical models is stagnated. Analysis tools flatten out the hierarchy to an intermediate model representation, but this approach is not fully taking advantage of the capabilities of hierarchical constructs. This thesis adapts an algorithm used for the analysis of flat models to incorporate the support of hierarchical models. This approach has resulted in positive results and can potentially be used in many applications such as the dynamic modeling of genetic circuits.

1.2 Thesis Outline

After giving some background in genetic circuit modeling and simulation in Chapter 2, this thesis presents our new hierarchical simulation method which is described in Chapter 3. This method is shown to produce promising improvements in simulation performance as compared with the existing flat simulation method. This method also improves memory consumption as only one copy of each submodel needs to be stored while methods that flatten the hierarchy require one copy for each instance of each submodel. Finally, Chapter 4 concludes the thesis by summarizing the work and future directions.

CHAPTER 2

BACKGROUND

In order to understand the work presented in this thesis, some background in biology and simulation are needed. Section 2.1 briefly explains the biological processes needed to understand what is a genetic circuit. Section 2.2 describes how genetic circuits can be turned into a chemical reaction network. Section 2.3 explains how chemical reaction models can be simulated using ODE methods. Section 2.4 describes a stochastic simulation method to analyze genetic circuits represented as chemical reaction models. Finally, Section 2.5 explains how to model a population of cells hierarchically.

2.1 Genetic Circuit Models

All organisms are made up of cells. Some organisms are composed of a single cell (e.g. bacteria) and some are composed of many cells (e.g. humans). Within each cell, a *deoxyribonucleic acid* (DNA) molecule includes *coding sequence* (known as *genes*) that provide instructions on how to construct *proteins*. Proteins are macromolecules made from chains of amino acids that serve many important functions in all organisms. Protein synthesis begins with a process known as *transcription* in which an enzyme called *RNA polymerase* (RNAP) binds to a specific sequence in the DNA called a *promoter*. RNAP walks the DNA to produce a single-stranded *messenger RNA* (mRNA). The resulting mRNA sequence is converted into a sequence of amino acids that folds into a protein by a *ribosome* using a process known as *translation*. The rate of this protein synthesis process can be regulated through the binding of proteins known as *transcription factors* to regions on the DNA called *operator sites*. That is, transcription factors can facilitate or block the binding of RNAP to certain promoters. The interaction of all these elements can be used to create networks that control the rate of transcription of the genes. These regulatory networks are known as genetic circuits.

One well-known genetic circuit is the *repressilator*, which was constructed in *Escherichia coli* (E. coli) by Elowitz and Leibler in 2000 [7]. In the repressilator, there are

three proteins produced from three promoters in which each protein acts as a transcription factor for one promoter creating a loop that forms an oscillator. Namely, the first protein, LacI, inhibits the transcription of the production of the second, TetR, which inhibits the production of the third protein, CI, which inhibits the production of LacI. Figure 2.1 depicts a graphical model of this genetic circuit in iBioSim in which the vertices are proteins, the edges represent repression relationships, and the edge labels are the promoter names. Note that a fourth protein, *green fluorescent protein* (GFP), is included in this genetic circuit to be produced at the same time as CI. The purpose of this protein is to make the cells glow green when CI is high, allowing an observer to see the oscillation.

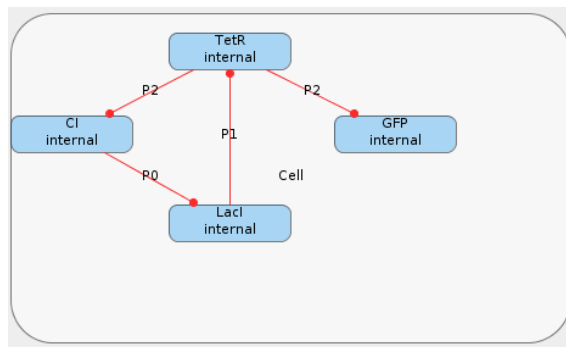


Figure 2.1: Repressilator circuit modeled in iBioSim.

2.2 Chemical Reaction Models

In order to simulate the repressilator, the model must be converted into a set of *chemical reactions* [16]. Chemical reactions combine *species* (DNA, RNA, protein molecules, etc.) to form new species. The species and chemical reactions for the repressilator circuit in Figure 2.1 are shown in Figure 2.2. Note that reactions can be shown explicitly as circles or implicitly as labels on edges between species. Also note that edges from species to reactions indicate that a species is a *reactant* (i.e., consumed by the reaction), edges from reactions to species indicate that a species is a *product* (i.e., produced by the reaction), and edges with no direction indicate that the species is a *modifier* (i.e., is neither produced or consumed). Finally, bi-directional edges indicate that a reaction is reversible, meaning that it can run in either direction. The number of molecules produced or consumed by a reaction is known as its *stoichiometry*. The edge is labeled with the stoichiometry when it is not one.

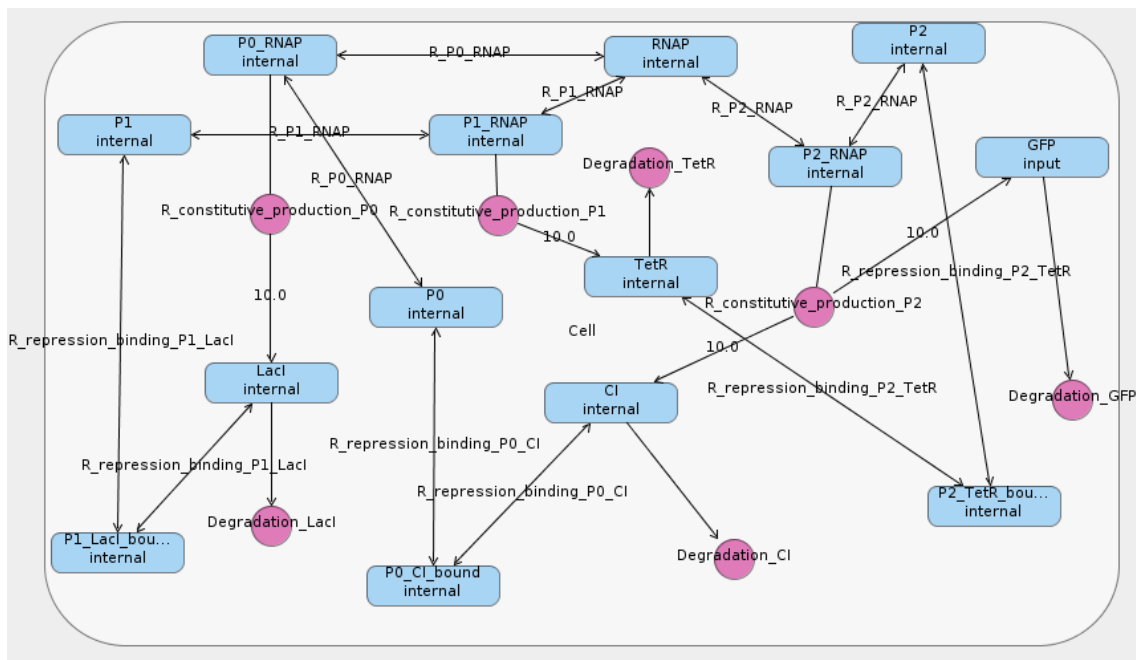
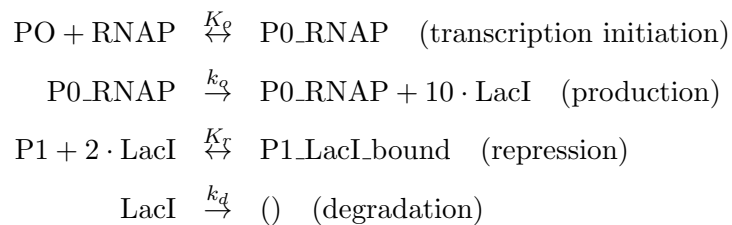


Figure 2.2: Repressilator modeled as chemical reactions.

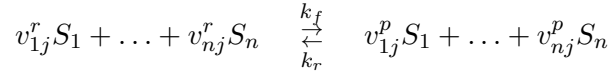
Some chemical reactions from the model are below:



Note that parameters such as k_o and k_d are known as *rate constants*, and they indicate the speed or likelihood of the reaction. Parameters such as K_r and K_o are known as *equilibrium constants*, and they are ratios of the forward and reverse rate constants (i.e., $K_r = k_{rf}/k_{rr}$).

2.3 ODE Simulation

A chemical reaction model can be converted into a set of *ordinary differential equations* (ODEs) using the *law of mass action*. This law states that the rate of a reaction is its rate constant times the concentration of the reactants raised to the power of their stoichiometry. More formally, consider a model with n species $\{S_1, \dots, S_n\}$ and m reactions $\{R_1, \dots, R_m\}$ where each reaction is of the form:



where v_{ij}^r is the reactant stoichiometry for species S_i in reaction R_j and v_{ij}^p is its product stoichiometry. Therefore, the law of mass action states that the *rate equation*, V_j , for reaction R_j is:

$$V_j = k_f \prod_{i=1}^n [S_i]^{v_{ij}^r} - k_r \prod_{i=1}^n [S_i]^{v_{ij}^p}$$

where $[S_i]$ is the concentration of species S_i . The rate equations for all reactions that produce or consume a species, S_i , can be combined to form an ODE describing the time evolution of the concentration of that species as follows:

$$\frac{d[S_i]}{dt} = \sum_{j=1}^m v_{ij} V_j, \quad 1 \leq i \leq n$$

where $v_{ij} = v_{ij}^p - v_{ij}^r$ (i.e., the net change in species S_i due to reaction R_j). As an example, the ODE for LacI is as follows:

$$\begin{aligned} \frac{d[\text{LacI}]}{dt} = & 10k_o[\text{P0_RNAP}] - k_d[\text{LacI}] \\ & -2(k_{rf}[\text{P1}][\text{LacI}]^2 - k_{rr}[\text{P1_LacI_bound}]) \end{aligned}$$

ODE simulation results for the repressilator are shown in Figure 2.3. It is clear from these results that ODE simulation of this model is not an accurate representation of the repressilator circuit, since the circuit stabilizes rather than oscillates. ODE simulation is deterministic, meaning that multiple simulations starting from the same initial condition always produce the same result. Moreover, ODE methods assume a large count of the entities being analyzed. In electrical engineering, ODE methods are reasonable for simulating electronic circuits, since the number of electrons flowing through the wires is very large. However, ODE methods are inaccurate for genetic circuits because the numbers of molecules of each species in a genetic circuit are typically quite small. Furthermore, the chemical reactions in a genetic circuits occur sporadically making them extremely noisy, a behavior not captured by ODE methods. Though there do exist ODE models that produce oscillations, this ODE model which is directly derived from the chemical reaction network for this genetic circuit does not reproduce the expected oscillatory behavior.

2.4 Stochastic Simulation

A better method for reasoning about genetic circuits is to utilize Gillespie's *stochastic simulation algorithm* (SSA) [8]. There are several variants of the SSA. This paper uses

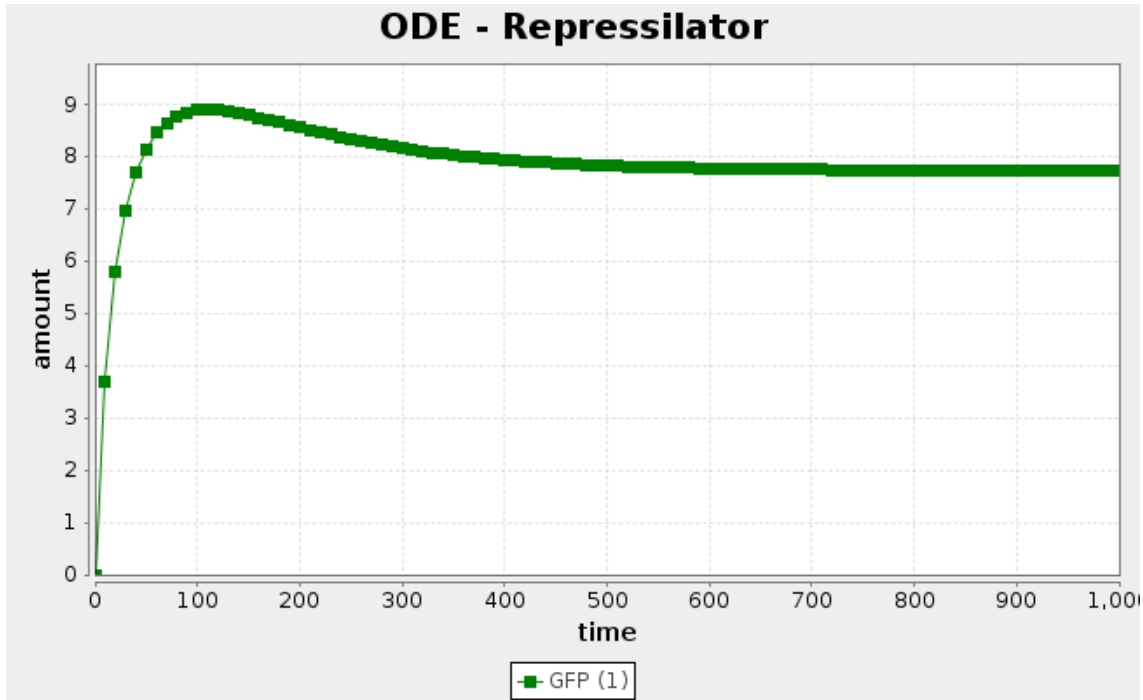


Figure 2.3: Repressilator ODE simulation results.

the *direct method* which is shown in Algorithm 2.1. The SSA takes a chemical reaction network model, M , and computes a *time series simulation*, α . The SSA is essentially a Monte Carlo algorithm which treats each reaction as a random event. The simulation begins by initializing α to an empty sequence, computes the initial time and state, $\langle t, \mathbf{x} \rangle$, from the model, M , and appends this *time point* to α . The state of the network is $\mathbf{x} = \langle x_1, \dots, x_n \rangle$ where x_i is the current amount of species S_i . The next step computes the *reaction propensities*, $\mathbf{a} = \langle a_1, \dots, a_m \rangle$, where a_j is the propensity for reaction, R_j , and can be approximated using the rate equation as follows:

$$a_j = k_j \prod_{i=0}^n (x_i)^{v_{ij}^r}$$

where k_j is the rate constant for reaction R_j and v_{ij}^r is the number of reactant molecules of species S_i consumed by the reaction. For example, the propensity for the forward reaction for transcription initiation on promoter P0 is approximately:

$$k_{of} \cdot P0 \cdot RNAP$$

The total propensity, a_0 , is the sum of all propensities. The total propensity is used to determine time until the next reaction using the following equation:

$$\tau = \frac{1}{a_0} \ln \frac{1}{r_1}.$$

where r_1 is a random number drawn from a uniform distribution from $[0, 1]$. Next, the propensities are used to compute the next reaction, μ , as follows:

$$\mu = \text{smallest integer s. t. } \sum_{j=1}^{\mu} a_j > r_2 a_0$$

where r_2 is a random number drawn from a uniform distribution from $[0, 1]$. Finally, the time and the state are updated as shown, where \mathbf{v}_μ is a vector representing the change in state due to reaction R_μ . This process repeats until the time, t , exceeds the simulation time limit. Using the SSA method, the repressilator model indeed oscillates as shown in Figure 2.4.

Algorithm 2.1: Gillespie's SSA

```

1 Input: Chemical reaction network model,  $M$ ;
2 Output: Time series simulation,  $\alpha$ ;
3  $\alpha := \langle \rangle$ ;
4  $\langle t, \mathbf{x} \rangle := \text{initialize}(M)$ ;
5 repeat
6    $\alpha := \alpha \cdot \langle t, \mathbf{x} \rangle$ ;
7    $\langle \mathbf{a}, a_0 \rangle := \text{computePropensities}(M, \mathbf{x})$ ;
8    $\tau := \text{computeNextReactionTime}(a_0)$ ;
9    $\mu := \text{selectNextReaction}(\mathbf{a}, a_0)$ ;
10   $\langle t, \mathbf{x} \rangle := \langle t + \tau, \mathbf{x} + \mathbf{v}_\mu \rangle$ ;
11 until  $t > \text{timeLimit}$  ;
```

2.5 Cellular Population Models

Genetic circuits have been constructed for many applications, such as genetic timers, oscillators, and logic gates, among others [12]. These applications can be developed in single-celled organisms. However, there are applications in which population modeling is a necessity, such as biomedical applications [19]. For example, genetic circuits can potentially be used for the treatment of infectious diseases and cancer, vaccine development, and gene therapy.

Stevens and Myers describe in [21] a method to model, analyze, and visualize dynamic populations of cells. Population-based models within `iBioSim` are represented in a two dimensional grid as shown in Figure 2.5. Each grid location is distinct and each location

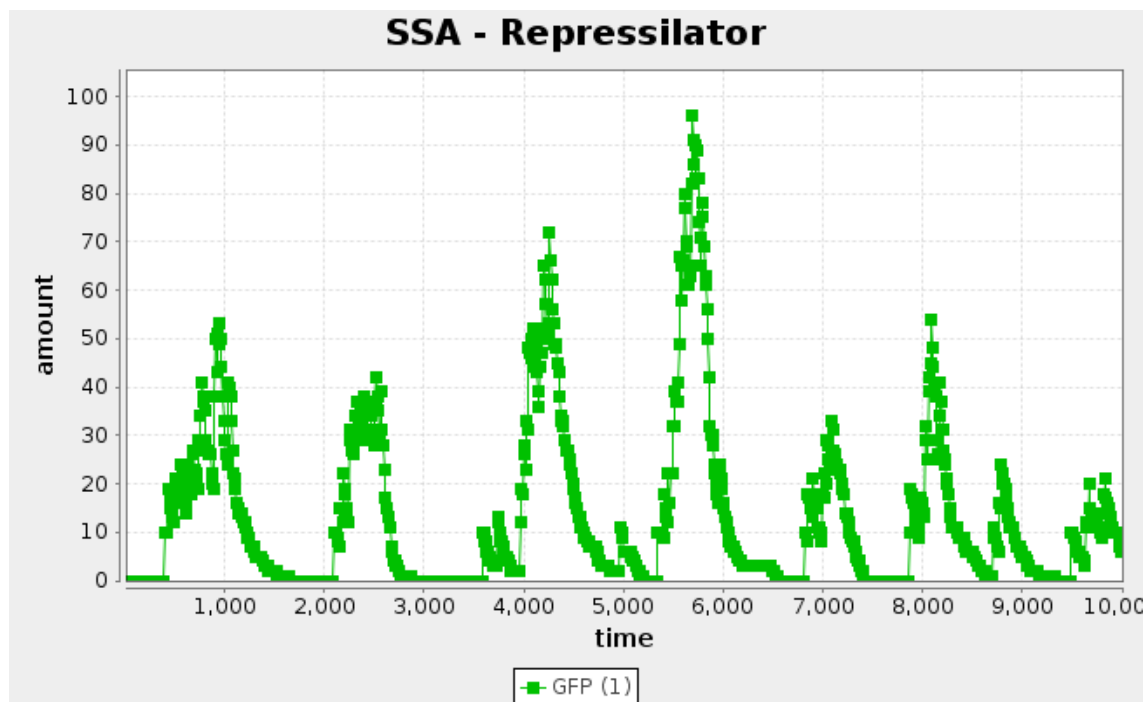


Figure 2.4: Repressilator SSA simulation results.

can include a single cell. As shown in Figure 2.7, diffusible species within a cell can move through membranes and between neighboring grid locations. The simulator described in [21] also supports dynamic events such as cell movement, division, and death. When a cell moves, the cell changes grid location. When a cell divides, a copy of the cell being divided is created. When a cell dies, the cell is removed from the model. Simulations can be visualized in *iBioSim* by coupling color gradients to species. The higher the concentration (or amount) of a species, the darker the color. When visualizing the repressilator circuit, it is possible to see every cell periodically turning on and off over time as shown in Figure 2.6. While two dimensional grids are not as realistic as three dimensional models in biology, two dimensional grids are a useful simplification.

The hierarchy in grid models is represented using SBML's hierarchical model composition package. This package allows the expression of hierarchy in SBML by allowing a top-level model to be constructed from a collection of sub-models. This package also enables the customization and connection of sub-models using *replacements* and *deletions*. A replacement can be used to state that an element in the top-level model replaces an element in a sub-model. A replacement can, for example, be used to state that a species

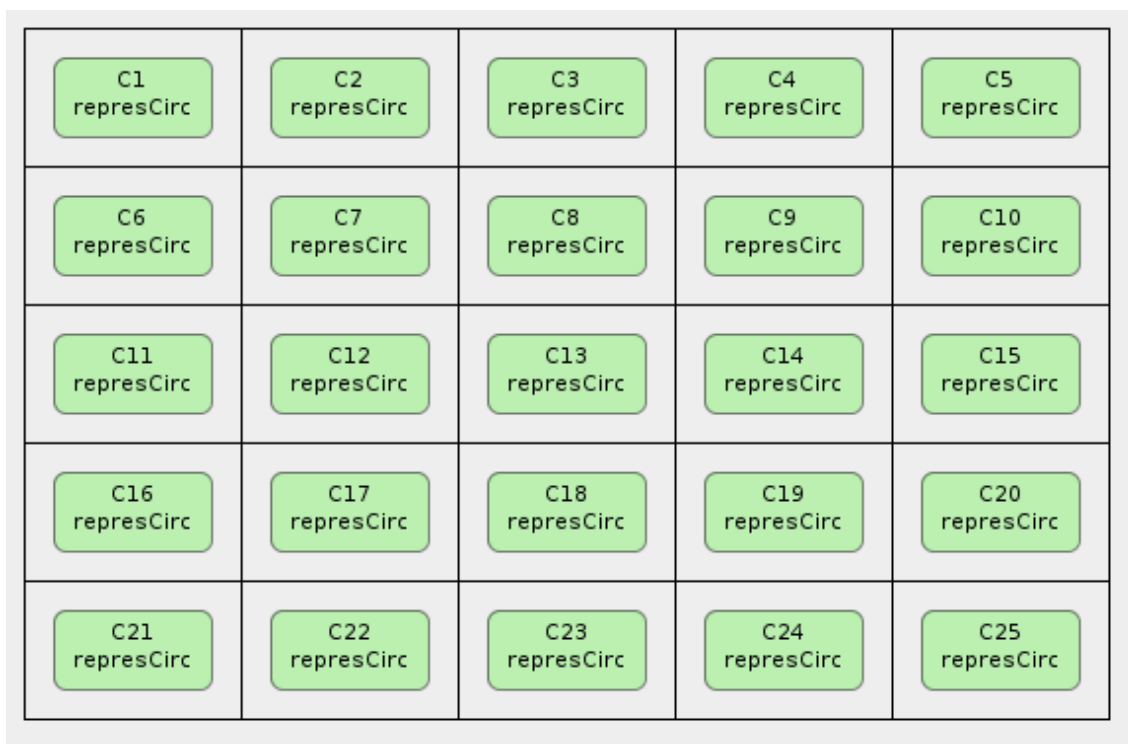


Figure 2.5: Cellular population modeling in iBioSim.

in the top-level model is to replace a species in two sub-models which effectively connects the two sub-models through this species. A deletion can be used to remove part of a sub-model that is not relevant to this use of the sub-model. A deletion, for example, can be used to remove a reaction that is not needed for this particular instantiation of a sub-model.



Figure 2.6: Cellular population visualization in iBioSim.

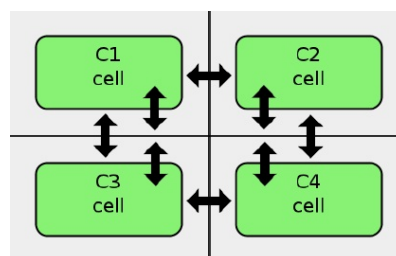


Figure 2.7: Diffusion in an iBioSim grid model [21].

CHAPTER 3

HIERARCHICAL SIMULATION

Dealing with the hierarchy inherent in cellular population models can be difficult because there are many dependencies that need to be handled. Therefore, it is a common practice for many modeling tools to flatten (inline) the hierarchy of a model before simulation. In other words, a typical simulator would instantiate copies of each sub-model and perform replacements and deletions during this flattening process resulting in a potentially much larger model that no longer includes any hierarchical modeling constructs. This approach has several disadvantages. First, the flattening routine causes the size of the model representation to grow quickly, consuming a lot of computational resources. Second, the flattening process itself can be very time consuming. Using the simulator described in [21], the time that `iBioSim` takes to flatten out a grid model composed of various numbers of repressilator components is shown in Table 3.1. According to this table, the time it takes to flatten out the hierarchy of a model can actually be larger than the time required to simulate the top-level model.

Table 3.1: Comparison of flattening to simulation runtime.

Num. Components	Flattening (sec)	Simulation (sec)
4	1.101	0.488
16	9.482	2.458
36	40.065	8.408
64	119.619	24.272
100	285.714	62.709

This fact motivated the development of the hierarchical simulation method explained in Section 3.1. Section 3.2 illustrates the hierarchical method through an example. Section 3.3 compares the performance of the hierarchical simulator and the method using the flattening routine.

3.1 hSSA Algorithm Overview

Our hierarchical simulator avoids the cost of flattening while preserving identical simulation results through several steps. First, in the preamble stage, the simulator locates the sub-models, $\{M_1, \dots, M_p\}$, used by the top-level model, M_0 . The simulator, however, only stores in memory one copy of each unique type of sub-model. The state of the simulator is now a vector of state vectors (i.e., $\mathbf{x} = \langle \mathbf{x}^0, \dots, \mathbf{x}^p \rangle$ where \mathbf{x}^i is the state corresponding to model M_i). Currently, the simulator only supports two-levels of hierarchy, so sub-models which have sub-models are still flattened.

The SSA is modified as shown in Algorithm 3.1 to support hierarchical simulation. Structurally, the algorithms are similar. The main difference is the introduction of ν to indicate the model for the reaction to be executed. Since there is only one copy of each unique sub-model stored in memory, the key challenge is that replacements and deletions must be performed on the fly during simulation making each step a bit more involved. In the description of the algorithm below, the notation $replaces(S_i^k, S_j^l)$ is used to indicate that species S_i^k in model M_k replaces species S_j^l in model M_l , and the notation $delete(R_j^k)$ indicates that reaction R_j^k is to be deleted from model M_k .

Algorithm 3.1: Hierarchical SSA

```

1 Input: Hierarchical reaction model,  $M = \langle M_0, \dots, M_p \rangle$ ;
2 Output: Time series simulation,  $\alpha$ ;
3  $\alpha := \langle \rangle$ ;
4  $\langle t, \mathbf{x} \rangle := initialize(M)$ ;
5 repeat
6    $\alpha := \alpha \cdot \langle t, \mathbf{x} \rangle$ ;
7    $\langle \mathbf{a}, a_0 \rangle := computePropensities(M, \mathbf{x})$ ;
8    $\tau := computeNextReactionTime(a_0)$ ;
9    $\langle \nu, \mu \rangle := selectNextReaction(\mathbf{a}, a_0)$ ;
10   $\langle t, \mathbf{x} \rangle := updateState(M, t, \tau, \mathbf{x}, \nu, \mu)$ ;
11 until  $t > timeLimit$  ;
```

In the hierarchical SSA, replacements must be considered when determining the initial state which is accomplished with Algorithm 3.2. First, the initial state vector is set to the initial value defined within each model. Next, each state in the top model, x_i^0 must be updated to take the value, x_j^k , of the initial state of a species S_j^k when that species is specified to replace the top-level species S_i^0 . Finally, the algorithm updates any species in

a sub-model which is replaced by a species at the top-level. These steps are necessary to ensure that the states of species involved in replacements coincide initially.

Algorithm 3.2: *initialize*(M)

```

1  $\mathbf{x} := \langle x_0^0, \dots, x_0^p \rangle;$ 
2 for  $k := 1$  to  $p$  do
3   for  $i := 1$  to  $n^0$  do
4     for  $j := 1$  to  $n^k$  do
5       if replaces( $S_j^k, S_i^0$ ) then
6          $x_i^0 := x_j^k;$ 
7 for  $k := 1$  to  $p$  do
8   for  $i := 1$  to  $n^0$  do
9     for  $j := 1$  to  $n^k$  do
10      if replaces( $S_i^0, S_j^k$ ) then
11         $x_j^k := x_i^0;$ 
12 return  $\langle t_0, \mathbf{x} \rangle;$ 

```

Deletions are considered when evaluating reaction propensities. Namely, in Algorithm 3.3, the propensity of a deleted reaction is set to zero, so it does not participate in the simulation. The total propensity calculated, a_0 , is the sum of the propensities for all the non-deleted reactions in all models.

Algorithm 3.3: *computePropensities*(M, \mathbf{x})

```

1  $a_0 := 0;$ 
2 for  $l := 0$  to  $p$  do
3   for  $j := 0$  to  $m^l$  do
4     if delete( $R_j^l$ ) then
5        $a_j^l := 0$ 
6     else
7        $a_j^l := k_j^l \prod_{i=0}^{n^l} (x_i^l)^{v_{ij}^{r^l}}$ 
8      $a_0 := a_0 + a_j^l$ 
9 return  $\langle \mathbf{a}, a_0 \rangle;$ 

```

Computing the next reaction time is the same as for the original SSA, but the computation of the next reaction is modified as shown in Algorithm 3.4. Namely, the sum must be over all reactions in all models, and return both the model, M_ν , and the reaction, R_μ^ν in that model to execute.

Algorithm 3.4: *selectNextReaction*(\mathbf{a}, a_0)

```

1  $a := 0, r_2 = \text{uniform}(0, 1)$ ;
2 for  $k := 0$  to  $p$  do
3   for  $j := 0$  to  $m^k$  do
4      $a := a + \alpha_j^k$ ;
5     if  $a > r_2 \cdot a_0$  then
6       return  $\langle k, j \rangle$ ;
7 return  $\langle p, m^p \rangle$ ;

```

Finally, Algorithm 3.5 updates the simulation time and the state of the model M_ν as a result of the reaction R_μ^ν . The state of species involved in replacements must also be updated in order to ensure that the values of these species continue to coincide throughout simulation. Namely, in Algorithm 3.5, the value, x_j^ν , of any species, S_j^ν , in the model, M_ν , in which the reaction occurred that replaces or is replaced by a species, S_i^0 , in the top-level model is copied into its state value, x_i^0 . Then, the second loop propagates values from the top-level, x_i^0 , into values in the sub-models, x_j^k , for each species S_j^k that replaces or is replaced by the top-level species, S_i^0 .

Algorithm 3.5: *updateState*($M, t, \tau, \mathbf{x}, \nu, \mu$)

```

1  $t := t + \tau$ ;
2  $\mathbf{x}^\nu := \mathbf{x}^\nu + \mathbf{v}_\mu^\nu$ ;
3 for  $i := 1$  to  $n^0$  do
4   for  $j := 1$  to  $n^\nu$  do
5     if  $\text{replaces}(S_i^0, S_j^\nu) \vee \text{replaces}(S_j^\nu, S_i^0)$  then
6        $x_i^0 := x_j^\nu$ ;
7 for  $k := 1$  to  $p$  do
8   for  $i := 1$  to  $n^0$  do
9     for  $j := 1$  to  $n^k$  do
10      if  $\text{replaces}(S_i^0, S_j^k) \vee \text{replaces}(S_j^k, S_i^0)$  then
11         $x_j^k := x_i^0$ ;
12 return  $\langle t, \mathbf{x} \rangle$ ;

```

While the algorithm presented in this paper is limited to SBML models composed of only species and reactions, the actual implementation of our hierarchical simulator supports nearly all SBML Level 3 Version 1 core constructs, such as assignment and rate rules, events, and constraints. The modifications necessary to support these are similar to those for reactions. Namely, deleted elements are dropped from sub-models,

math expressions are computed on local states, and care must be taken to ensure that top-level model and local sub-model states for variables involved in replacements must always coincide throughout simulation. As a further memory saving optimization, our implementation does not keep duplicate copies of the local and top-level variables that are connected through replacements.

3.2 Example

The algorithm is better illustrated using an example. Assume there is a chemical reaction network as shown in Figure 3.1, where a molecule of A and B are taken as the reactants of a certain reaction R1 to form a molecule of C. In addition, a molecule of C is used to form a molecule of D through reaction R2. In this model, species A and D are put on a port, where the former is on an input port and the latter is on an output port.

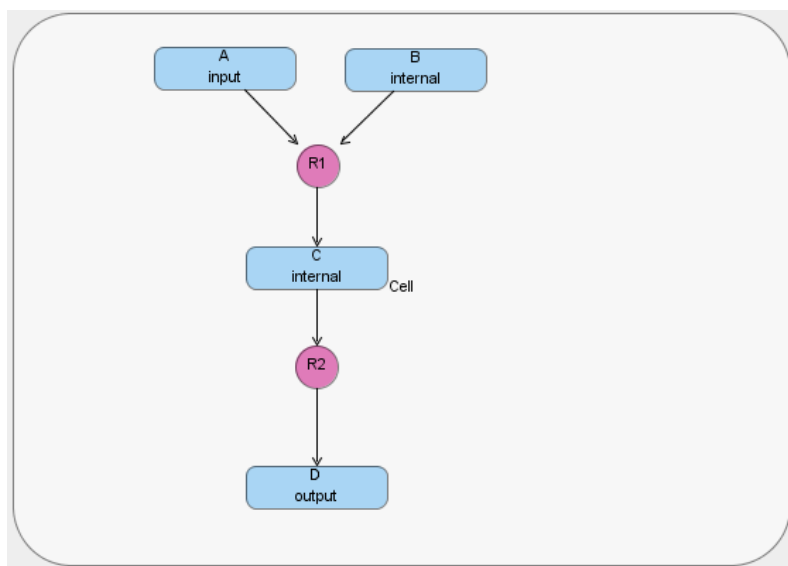


Figure 3.1: Simple chemical reaction network.

This chemical reaction network can be used to construct a hierarchical model as shown in Figure 3.2. In this model, the top-level model contains two instances, C1 and C2, of the chemical reaction network shown in Figure 3.1. In addition, the top-level model has three species X, Y, and Z, where species X replaces species A in instance C1, species Y is replaced by species D in C1 and replaces species A in C2, and species Z is replaced by species D in C2. Furthermore, reaction R2 in instance C2 is deleted.

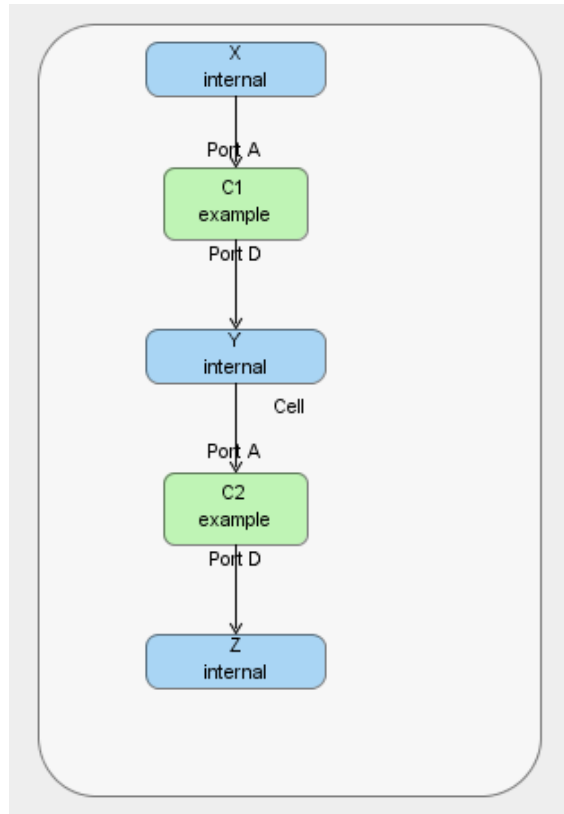


Figure 3.2: Hierarchical model example.

In hSSA, the input is a collection of models where M_0 represents the top-level model and there are p submodels. In this particular case, there are two submodels where M_1 represents instance C1 and M_2 represents instance C2. The output is a times series simulation α , which is set to be initially empty.

The first step in the initialization process is to set initial time to be equal to zero and the state vector \mathbf{x} to contain the initial amount of each species in each model as shown in Figure 3.3. Figure 3.3a shows the initial value of the state vector of the top-level model and Figures 3.3b and 3.3c show the initial value of the state vector of the model given in Figure 3.1. Note that the state vectors in Figure 3.3b and Figure 3.3c are equal since they refer to the same model definition. Once the state vectors are populated with the initial amount of each species, the simulator handles replacements. First, the simulator handles the case where a species in a submodel replaces a species in the top-level model. For example, in this particular model, species Y in the top-level model is replaced by species D in C1 and the value of Y is updated accordingly as shown in Figure 3.4. Similarly, species Z in the top-level model is replaced by species D in C2. Once this step is completed,

Top			
t	X	Y	Z
0	5	10	10

(a) Initial amount of each top-level species.

C1				
t	A	B	C	D
0	10	10	0	0

(b) Initial amount of each species in submodel C1.


C2				
t	A	B	C	D
0	10	10	0	0

(c) Initial amount of each species in submodel C2.

Figure 3.3: Initial amount for all species in the hierarchical model.

the simulator is going to handle the case where a top-level species replaces species in submodels. In the example, the value of species X percolates down to species A in instance C1. The same holds for the case where species Y replaces A in submodel C2. Figure 3.5 shows the final values once all replacements are handled.

Top			
t	X	Y	Z
0	5	10	10



Top			
t	X	Y	Z
0	5	0	10

C1				
t	A	B	C	D
0	10	10	0	0

C1				
t	A	B	C	D
0	10	10	0	0

Figure 3.4: Example of replacement.

After initialization is done, the simulator enters the loop. First, the simulator records the current state of the simulation. Then, the propensities for each reaction are calculated along with the total propensity, which is the sum of all reaction propensities as shown in Figure 3.6. The next reaction time is computed afterwards, which is a random variable drawn from an exponential distribution where the mean is the inverse of the total propensity. The next reaction to fire is selected based on the reaction propensities. That is, the next reaction is random with probability proportional to the contribution of this

Top				C1					C2				
t	X	Y	Z	t	A	B	C	D	t	A	B	C	D
0	5	0	0	0	5	10	0	0	0	0	10	0	0

Figure 3.5: Example of replacement.

reaction's propensity to the total propensity. From the table shown in Figure 3.6, it is possible to notice that the only possible reaction to be selected is reaction R1 in C1 given that it is the only reaction that has a propensity greater than zero. Figure 3.7 shows the computed time for the next reaction to occur, as well as, the next reaction to be fired. The last step is to update the state of the simulation. Time is advanced to the next time step and the state vector \mathbf{x} is updated based on the stoichiometry of the species involved in the selected reaction. In this particular case, a molecule of A and a molecule of B are consumed for the production of a molecule of C. Since species A in C1 is involved in a replacement, the new value of A needs to be percolated up to X in the top-level model. Now, the value of X is 4 as shown in Figure 3.8.

Propensities					
C1		C2		Total	
a_1	a_2	a_1	a_2	a_0	
5	0	0	0	5	

Figure 3.6: Propensity for each reaction.

τ	v	μ
0.1	C1	R1

Figure 3.7: Computed next reaction time and next reaction selected.

These steps are repeated until the current time exceeds the time limit. Assuming the current time is still lower than the time limit, another iteration is performed. First, the current state of the simulation is recorded. Then, the propensities are computed as shown in Figure 3.9. Note that reaction R2 in submodel C1 can be fired now since a molecule of C was produced in the last iteration. The next step is to compute the next reaction time, which, in this case, is 0.2. The next reaction selected is R2 in submodel C1. Once the next reaction is selected, the state of the simulation is updated. Time is advanced to the next

Top			
t	X	Y	Z
0	5	0	0
0.1	5	0	0
0.1	4	0	0

(a) Amount of each top-level species after first iteration.

C1				
t	A	B	C	D
0	5	10	0	0
0.1	4	9	1	0
0.1	4	9	1	0

(b) Amount of each specie in submodel C1 after first iteration.

C2				
t	A	B	C	D
0	0	10	0	0
0.1	0	10	0	0

(c) Amount of each species in submodel C2 after first iteration.

Figure 3.8: Amount for all species in the hierarchical model after first iteration.

time step and the state vector \mathbf{x} is updated after firing reaction R2 in C1. Now, the total amount of C is 0 and the amount of D is 1 in submodel C1. The state update for species D is effectively affecting the state of Y in the top-level model, and consequently, species A in C2 since Y replaces A. In the update function, the value of D in C1 is percolated up to species Y in the top-level model and the value of Y is percolated down to species A in C2. Figure 3.10 shows the state after handling replacements in the second iteration.

Propensities				
C1		C2		Total
a_1	a_2	a_1	a_2	a_0
3.6	1	0	0	4.6

Figure 3.9: Propensity for each reaction for the second iteration.

After recording the state of the simulation, the propensities are calculated as you can see in Figure 3.11. Up until this point, C2 is unable to fire any reaction. However, species A in C2 has a molecule now which enables reaction R1 to fire. The next reaction time

Top			
t	X	Y	Z
0	5	0	0
0.1	4	0	0
0.3	4	1	0

(a) Amount of each top-level species after second iteration.

C1				
t	A	B	C	D
0	5	10	0	0
0.1	4	9	1	0
0.3	4	9	0	1

(b) Amount of each specie in submodel C1 after second iteration.

C2				
t	A	B	C	D
0	0	10	0	0
0.1	0	10	0	0
0.3	1	10	0	0

(c) Amount of each species in submodel C2 after second iteration.

Figure 3.10: Amount for all species in the hierarchical model after second iteration.

that is drawn from an exponential distribution with mean $\frac{1}{a_0}$ is 0.2. The next reaction selected in this iteration is reaction R1 in C2. Once again, the state of the simulation is updated by advancing time to the next time step and the the reaction is fired. In this reaction, a molecule of both species A and B is consumed for the production of a molecule of C. Since the amount of A is changed, the value of species Y in the top-level model and species D in C1 must be updated accordingly. The new state is shown in Figure 3.12

Propensities				
C1		C2		Total
a_1	a_2	a_1	a_2	a_0
3.6	0	1	0	4.6

Figure 3.11: Propensity for each reaction for the third iteration.

Something interesting happens in the fourth iteration. After recording the state of the simulation, the propensities are calculated. Even though reaction R2 in C2 could, in

Top			
t	X	Y	Z
0	5	0	0
0.1	4	0	0
0.3	4	1	0
0.5	4	0	0

(a) Amount of each top-level species after third iteration.

C1				
t	A	B	C	D
0	5	10	0	0
0.1	4	9	1	0
0.3	4	9	0	1
0.5	4	9	0	0

(b) Amount of each specie in submodel C1 after third iteration.

C2				
t	A	B	C	D
0	0	10	0	0
0.1	0	10	0	0
0.3	1	10	0	0
0.5	0	9	1	0

(c) Amount of each species in submodel C2 after third iteration.

Figure 3.12: Amount for all species in the hierarchical model after third iteration.

theory, be fired since this reaction requires only a molecule of C, the propensity is zero as shown in Figure 3.13. This is because the reaction was deleted, causing the reaction propensity to be always zero. That is, this reaction can never be fired.

3.3 Results

The hierarchical simulation method provides substantial improvements in performance relative to a flat simulation method. The hierarchical simulator performance is compared against the SSA simulator in [21]. Tests are performed using an Intel(R) Core(TM) i5 CPU 2.80 GHz and 4GB RAM. The first test consists of a top-level grid model that is populated with repressilator sub-models without replacements or deletions. The test is performed using 4, 16, 36, 64, and 100 sub-models and the runtime results are shown in Figure 3.14. The increase in runtime for hierarchical simulation is clearly more scalable than for flat

Propensities				
C1		C2		Total
a_1	a_2	a_1	a_2	a_0
3.6	0	0	0	3.6

Figure 3.13: Propensity for each reaction for the fourth iteration.

simulation. Figure 3.15 shows the results in simulation time using both approaches. It is possible to observe that hSSA performed better since it deals with smaller data structures.

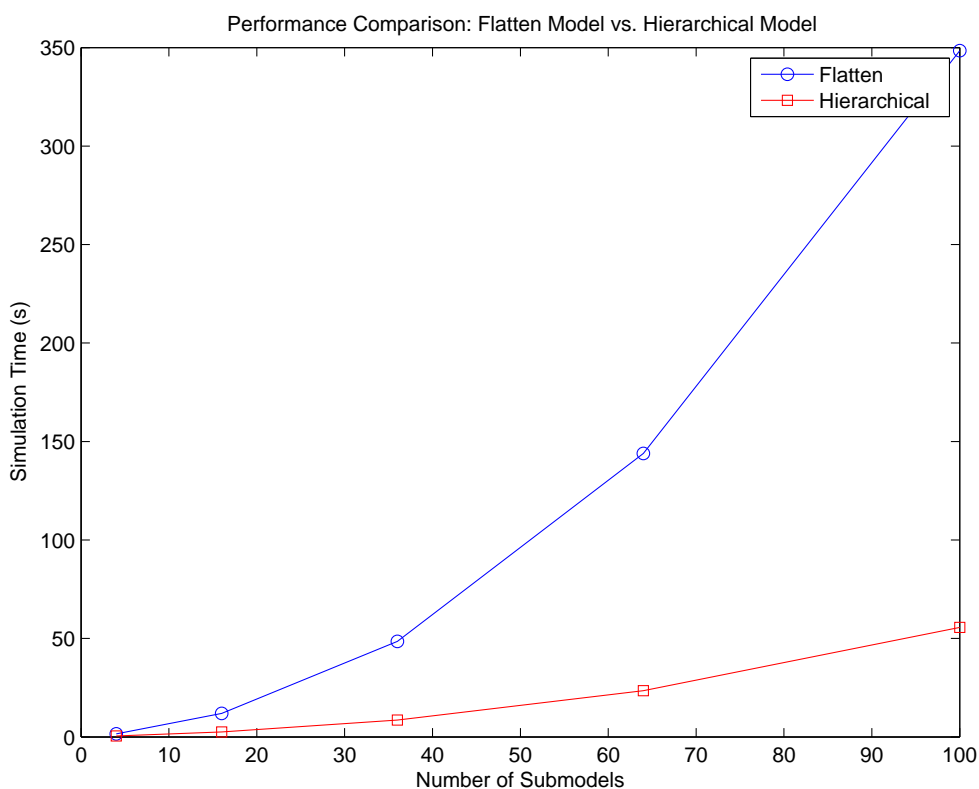


Figure 3.14: Comparison of performance of the SSA using flattening and our hierarchical approach.

A second test is performed using a top-level model populated with repressilator circuits in which the degradation reaction of the GFP reporter protein is deleted from all sub-models, and the GFP protein is replaced by a top-level GFP protein that tracks the total amount across all sub-models. A version with 25 sub-models is shown in Figure 3.16.

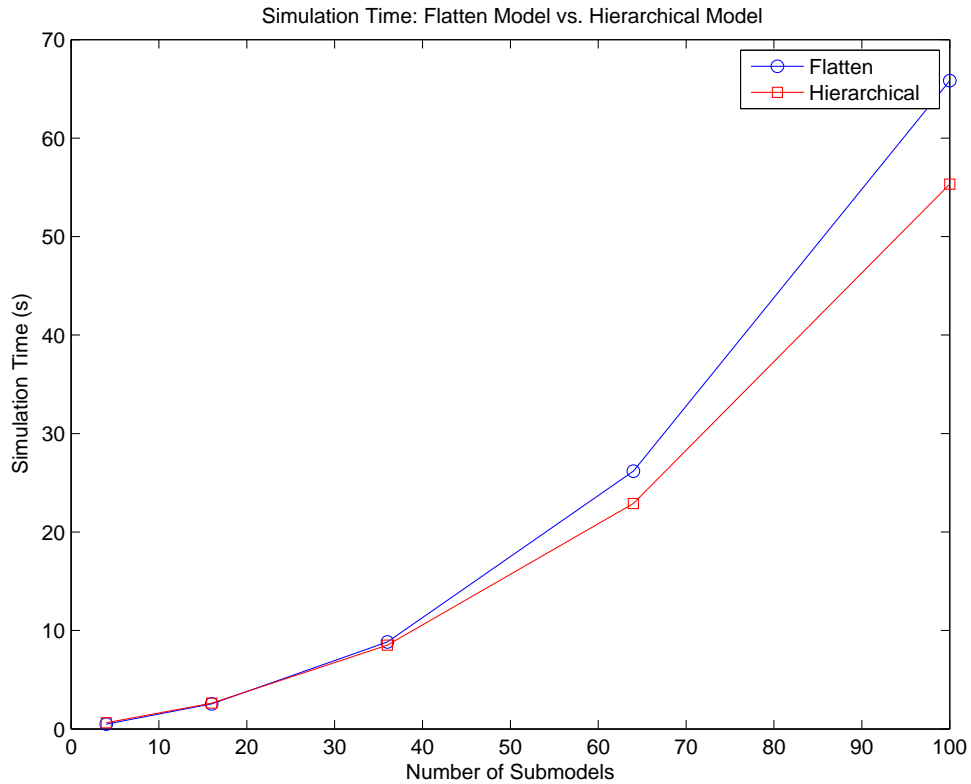


Figure 3.15: Comparison of simulation time of the SSA using flattening and our hierarchical approach.

Figure 3.17 shows the total amount of GFP in this circuit over time. Performance tests are performed using 1, 4, 9, 15, 25, and 50 sub-models, and the runtime results are shown in Figure 3.18. These results show that even with the added complexity of replacements and deletions that the performance of the hierarchical simulator still scales much better than an SSA simulator that uses flattening. Figure 3.19 shows the results in simulation time using both approaches for the test where the connection of submodels are customized using replacements and deletions. It is possible to observe that hSSA has some overhead, which implies that, for a sufficiently large time limit, the time required to simulate SSA with flattening and hSSA will intersect.

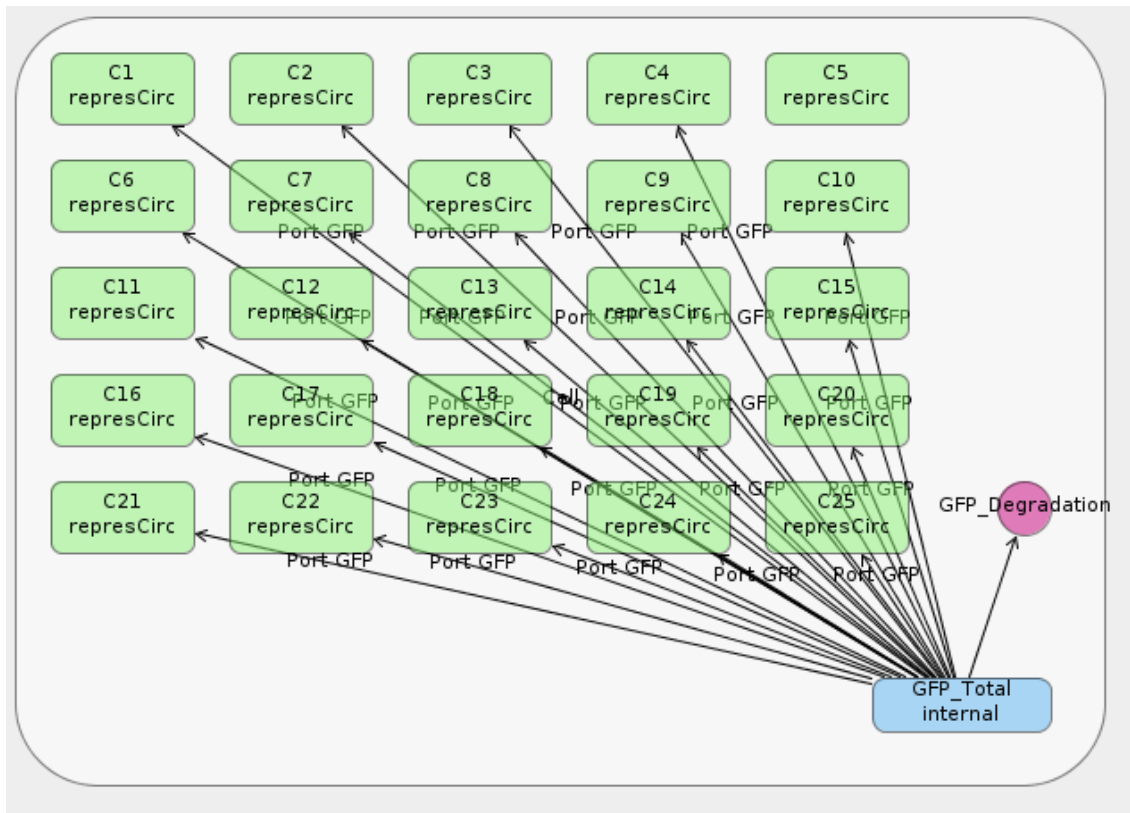


Figure 3.16: 25 Repressor circuits in which GFP degradation is deleted and has the local GFP species replaced with the top-level one.

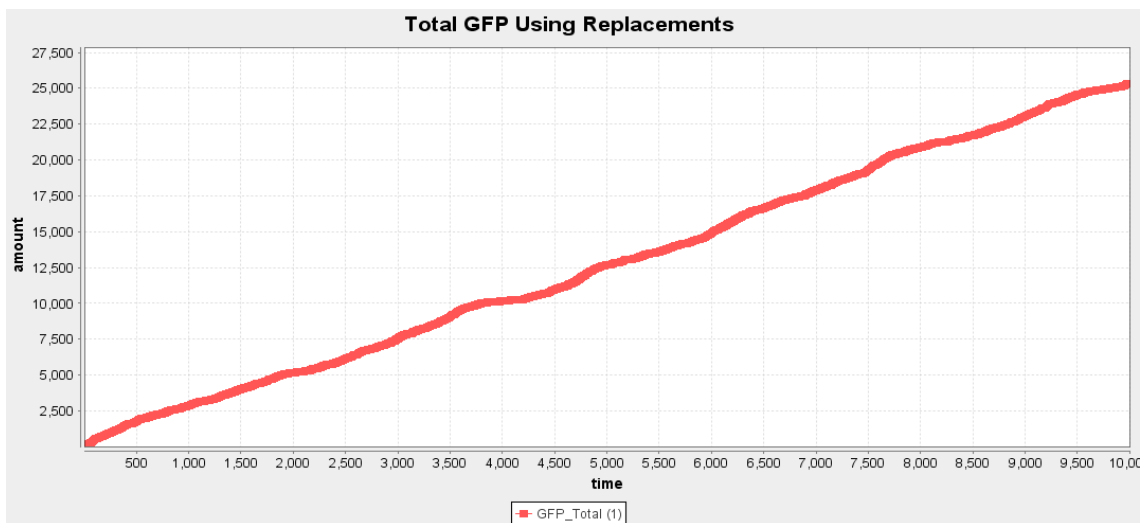


Figure 3.17: Total amount of GFP for 25 instances of the repressor circuit.

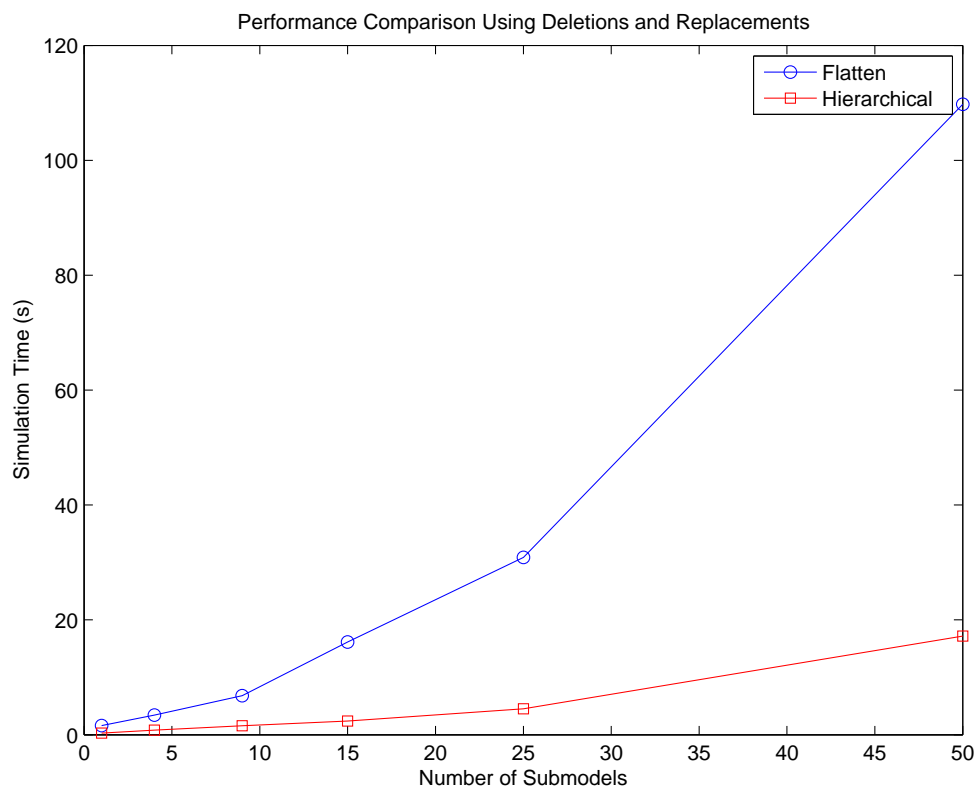


Figure 3.18: Comparison of performance of the SSA using flattening and our hierarchical approach for a model that includes replacements and deletions.

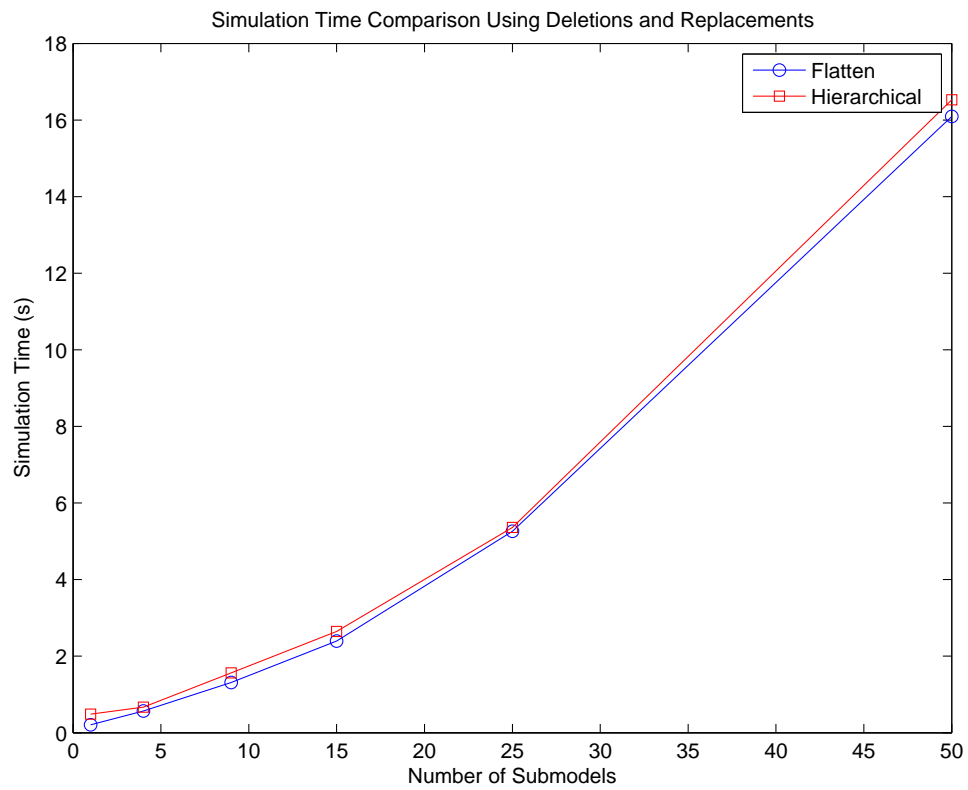


Figure 3.19: Comparison of simulation time of the SSA using flattening and our hierarchical approach for a model that includes replacements and deletions.

CHAPTER 4

CONCLUSIONS

Genetic circuits can potentially be used for many useful applications such as biomedical technologies. Modeling and analysis tools are going to have an important role in the synthetic biology field in the construction of complex circuits. However, efficient algorithms are necessary in the construction of complex models. This chapter concludes the thesis with a summary of the work in Section 4.1 and presents future directions of the research in Section 4.2.

4.1 Summary

This paper proposes a hierarchical simulation method for the analysis of genetic circuits. This method is intended for the analysis of dynamic systems. Results have shown that the extra time spent in the preamble stage of the simulation of a hierarchical model is substantially reduced by not flattening out the hierarchy while the simulation time is essentially equivalent. This fact might be counterintuitive since the hierarchical simulator has to perform replacements and deletions on the fly. However, the time spent on this task is more than compensated by the fact that the hierarchical method avoids flattening and deals with smaller data structures. The total simulation time for the hierarchical simulator grows at a nearly linear increase with respect to model size, whereas the simulation time for a flat simulation has an exponential increase.

4.2 Future Work

While the hierarchical simulator is promising, there are many extensions that can be implemented in the future. The next step is to support dynamic events to model cell division and death which add or remove models from the simulation dynamically. We feel that the hierarchical simulator is better suited to such changing model structures, and this requirement is actually a major motivation for the development of this simulator. Another future enhancement is dynamic model abstraction. In previous work,

significant improvements in analysis time is achieved by removing unimportant details using automated model abstraction before simulation which improves simulation time while still delivering accurate results [3, 11]. A dynamic hierarchical simulator has the potential to allow these abstractions to be performed on the fly to manage complexity as needed to balance computational cost with accuracy. Finally, given that dynamic hierarchical models are inherently concurrent, parallel processing can also be explored to further improve simulation time.

REFERENCES

- [1] ANDERSON, J. C., CLARKE, E. J., AND ARKIN., A. P. Environmentally controlled invasion of cancer cells by engineering bacteria. *J. Mol. Biol.* 355 (2006), 619–627.
- [2] BRAZIL, G. M., KENEFICK, L., CALLANAN, M., HARO, A., DE LORENZO, V., DOWLING, D. N., AND GARA, F. O. Construction of a rhizosphere pseudomonad with potential to degrade polychlorinated biphenyls and detection of *bph* gene expression in the rhizosphere. *Applied and Environmental Microbiology* 61, 5 (1995), 1946–1952.
- [3] C. MADSEN, C. MYERS, N. R. C. W., AND ZHANG, Z. Utilizing stochastic model checking to analyze genetic circuits. In *2012 Computational Intelligence in Bioinformatics and Computational Biology* (May 2012).
- [4] CASES, I., AND DE LORENZO, V. Genetically modified organisms for the environment: stories of success and failure and what we have learned from them. *International Microbiology* 8 (2005), 213–222.
- [5] CASTRO, R., KOFMAN, E., AND WAINER, G. A formal framework for stochastic DEVS modeling and simulation. In *Proceedings of the 2008 Spring Simulation Multiconference* (San Diego, CA, USA, 2008), SpringSim '08, Society for Computer Simulation International, pp. 421–428.
- [6] CONCEPCION, A. I., AND ZEIGLER, B. F. DEVS formalism: A framework for hierarchical model development. *IEEE Trans. Softw. Eng.* 14, 2 (Feb. 1988), 228–241.
- [7] ELOWITZ, M., AND LEIBLER, S. A synthetic oscillatory network of transcriptional regulator. *Nature* 403, 6767 (2000), 335–338.
- [8] GILLESPIE, D. T. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry* 81, 25 (1977), 2340–2361.
- [9] HUCKA, M., FINNEY, A., SAURO, H. M., BOLOURI, H., DOYLE, J. C., KITANO, H., AND ET AL. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* 19, 4 (Mar. 2003), 524–531.
- [10] KUWAHARA, H., MYERS, C., AND SAMOILOV, M. Temperature control of fibrillation circuit switch in uropathogenic escherichia coli: quantitative analysis via automated model abstraction. *PLoS Computational Biology* 6, 3 (Mar. 2010), e1000723.
- [11] KUWAHARA, H., MYERS, C., SAMOILOV, M., BARKER, N., AND ARKIN, A. Automated abstraction methodology for genetic regulatory networks. In *Transactions on Computational Systems Biology VI, LNBI 4220* (2006).

- [12] LUCKS, J., AND ARKIN, A. The hunt for the biological transistor. *IEEE Spectrum* 48 (2011), 38–43.
- [13] MADSEN, C., MYERS, C., PATTERSON, T., ROEHNER, N., STEVENS, J., AND WINSTEAD, C. Design and test of genetic circuits using iBioSim. *Design and Test of Computers, IEEE* 29, 3 (2012), 32–39.
- [14] MAUS, C., JOHN, M., RÖHL, M., AND UHRMACHER, A. Hierarchical modeling for computational biology. In *Formal Methods for Computational Systems Biology*, M. Bernardo, P. Degano, and G. Zavattaro, Eds., vol. 5016 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, pp. 81–124.
- [15] MYERS, C., BARKER, N., KUWAHARA, H., JONES, K., MADSEN, C., AND NGUYEN, N. Genetic design automation. In *2009 International Conference on Computer-Aided Design* (Nov. 2009).
- [16] MYERS, C. J. *Engineering Genetic Circuits*. Chapman and Hall/CRC, 2009.
- [17] NGUYEN, N., MYERS, C., KUWAHARA, H., WINSTEAD, C., AND KEENER, J. Design and analysis of a robust genetic muller c-element. *Journal of Theoretical Biology* 264, 2 (May 2010), 174–187.
- [18] RO, D. K., PARADISE, E. M., OUELLET, M., FISHER, K. J., NEWMAN, K. L., NDUNGU, J. M., HO, K. A., EACHUS, R. A., HAM, T. S., KIRBY, J., CHANG, M. C. Y., WITHERS, S. T., SHIBA, Y., SARPONG, R., AND KEASLING, J. D. Production of the antimalarial drug precursor artemisinic acid in engineered yeast. *Nature* 440, 7086 (2006), 940–943.
- [19] RUDER, W. C., LU, T., AND COLLINS, J. Synthetic biology moving into the clinic. *Science* 333 (2011), 1248–1252.
- [20] SAVAGE, D., WAY, J., AND SILVER, P. Defossilizing fuel: How synthetic biology can transform biofuel production. *ACS Chemical Biology* 3, 1 (2008), 13–16.
- [21] STEVENS, J., AND MYERS, C. Dynamic modelling of cellular populations within iBioSim. *ACS Synthetic Biology* 2, 5 (2012), 223–229.