An Efficient Algorithm for Generating Trajectories of Stochastic Gene Regulation Reactions *

Michael Gibson Jehoshua Bruck

California Institute of Technology
Department of Computation and Neural Systems
Mail Code 136-93, Pasadena, CA 91125
Email: {gibson, bruck}@paradise.caltech.edu

Abstract

Systems of weakly coupled chemical equations occur in gene regulation and other biological systems. For small numbers of molecules (as in a small cell), the usual differential equations approach to chemical kinetics must be replaced with a stochastic approach. To deal with this kind of system, one generates trajectories through stochastic phase space. By generating a large enough number of trajectories, one can understand the statistics of the behavior of the complex, non-linear system.

The algorithms for dealing with sparsely connected stochastic processes are not as advanced as those for sparse deterministic processes. In particular, the existing algorithm of choice for generating trajectories, which is not optimized in any way for sparseness, is $\mathcal{O}(rE)$, where r is the number of reactions and E is the number of reaction events in the trajectory. We present two algorithms of $\mathcal{O}(r + E \log r)$, one of which is a simple extension of the existing algorithm, and the other of which is more subtle. The latter is more easily extended to include stochastic processes of different types.

We apply our faster algorithm to a model of bacteriophage lambda and are able to run the same calculations on a cluster of desktop workstations that previously required a supercomputer. This allows us to run more complicated calculations than could be done previously. As an example of this, we analyse the sensitivity of the lambda model to the values of several of its parameters. We find that the model is relatively insensitive to changes in the translation rate, protein dimerization rates and protein degradation rates; is somewhat sensitive to the transcription rate, and is extremely sensitive to the average number of proteins per mRNA transcript.

1 Introduction

Consider a chemical reaction, such as

$$S_1 + S_2 \xrightarrow{k} S_3 \tag{1.1}$$

which states that substances S_1 and S_2 react to give a new substance, S_3. Arbitrary biological process can be modeled as a system of chemical equations, with some physical constraints (such

*Supported in part by ONR grant N00014-97-1-0293, by a JPL-CISM grant, by NSF Young Investigator Award CCR-9457811 and by a Sloan Research Fellowship.

as diffusion, localization of molecules, cell growth, etc.) thrown in. A quantitative model of a system consists of a set of chemical equations, their corresponding parameter values (e.g., k in Equation 1.1) and physical constraints. Given such a model, one wants to know what predictions it makes. Going from an actual biological system to chemical equations and physical constraints is fundamentally a biological problem; going from these equations to a set of predictions is a mathematical or computer science problem. In this paper we will deal first with a computer science issue — how to get predictions from a model in significantly less time — then deal with a biological problem — how to use this computer science tool to refine a model.

1.1 The Problem: Stochastic Chemical Reactions

In the usual differential equations approach, the dynamics of the reaction Equation 1.1 can be expressed by the differential equation

$$\frac{d[S_3]}{dt} = -\frac{d[S_1]}{dt} = -\frac{d[S_2]}{dt} = k[S_1][S_2],$$

where $[S_1]$ means the concentration of substance S_1, etc. The concentration is, of course, the number of molecules per unit volume (usually expressed in units of moles per liter, where 1 mole $= 6.02 \times 10^{23}$ molecules). There are efficient numerical tools to solve a set of non-linear differential equations.

Notice that the differential equation above *implicitly* assumes that the concentrations are quantities that vary *continuously* and *deterministically* over time. In many biological systems, these assumptions are not valid. For example, a typical cell will have one molecule of DNA. To assume that the concentration of DNA varies continously is obviously inappropriate. Furthermore, in small cells, the number of molecules of mRNA and of proteins will also be sufficiently small that the continuous, deterministic approximation is invalid. In such systems, a different approach is required. In particular, one considers the *number* of molecules of each substance, rather than the *concentration* [9]. The number of molecules varies discretely, not continuously. In particular, let n_1, n_2 and n_3 be the number of molecules of S_1, S_2 and S_3, respectively. Since the reactions are now stochastic, we can transform Equation 1.1 into the probablistic equation[1]

$$P(n_1 - 1, n_2 - 1, n_3 + 1 | t + \Delta t) = P(n_1, n_2, n_3 | t) \times k \times n_1 \times n_2 \times \Delta t \tag{1.2}$$

and, of course,

$$P(n_1, n_2, n_3 | t + \Delta t) = P(n_1, n_2, n_3 | t) \times (1 - k \times n_1 \times n_2 \times \Delta t) \tag{1.3}$$

Consider a list of r reactions of the form in Equation 1.1. (For example, Arkin *et al.* [1] have written out such a set of equations. for gene regulation in the temperate bacteriophage lambda.) The function P is defined over the continous variable time and the discrete variables n_1, n_2, etc. Since the domain of P is the range of values of n_1 times the range of values of n_2, etc., it is not feasible to write out the transition probabilities for each state (i.e., discrete point of the domain) as in Equation 1.3 and deal with each state separately. Furthermore, it is typically not feasible to write an analytical form for P as a function of time and of n_1, n_2, etc.

The standard approach to solve this problem is to do a Monte Carlo simulation, i.e., generate a number of trajectories through state space which follow P. By doing this, it is only necessary to calculate P for the current state $(n_1, n_2, ..., t)$ rather than for *all* possible values of the state.

[1] We should be careful about the constant k. For a second order reaction such as Equation 1.1, the deterministic rate constant and the stochastic rate constant will differ by a factor of $A_g \times V$, where A_g is Avagadro's number, and V is the volume. This follows immediately from dimensional analysis.

1.2 Results and Organization of Paper

Suppose that in a given trajectory, E events occur, where an event is the execution of one reaction as in Equation 1.1. The current algorithm, that of Gillespie [6] is $\mathcal{O}(rE)$. In this paper, we present algorithms that are $\mathcal{O}(r + E \log r)$, subject to some weak assumptions on the nature of the reactions. These assumptions hold for the gene regulation example, and may for many other biological systems. Using our faster algorithm, we are able to do calculations that were previously intractable.

In Section 2, we explain in detail the Gillespie algorithm and how it can be applied to a gene regulation example. In Section 3, we explain our improvements of that algorithm, and give simple examples. In Section 4, we apply our faster algorithm to an extensive gene regulation example, and show some examples of how a faster algorithm, which is really a computer science construct, might be useful for biological research.

2 Background

In this section we describe the existing simulation algorithm and how it has been applied to gene regulation.

2.1 The Gillespie Algorithm

To review the problem from the introduction, suppose we have r reactions involving s different chemical substances. At any given time, the *state* of the system will be an s-tuple, containing the number of molecules of substance S_1, the number of molecules of substance S_2, etc. If we (arbitrarily) limit the maximum number of molecules of a given substance to MAX, the number of possible states is MAX^s, which gets large very quickly. In principle, one could write out a MAX^s by MAX^s matrix, consisting of the transition probabilities per unit time from each state to every other state. Solving the problem analytically for $P($ state, time $)$ is, formally, taking the matrix exponential of this matrix, which is out of the question except for systems where s is small. In some cases, it may be possible to parameterize P in terms of the state and solve it explicitly. However, there is no general parameterization of this form, so explicit solutions are only available for some special cases.

Gillespie [6] proposed an *exact stochastic simulation*, i.e. one simulates the system rather than solving explicitly for P. At each time step, the system is in exactly one state. A transition consists of executing a reaction, so there are at most r possible transitions from a given state. In fact, there are typically fewer possible transitions, say m, with $m < r$. (See Figure 1.) Equivalently, $\mu_i = 0$ for $i > m$.

For each possible transition i, there is some corresponding probability per unit time, μ_i. In the example in the introduction $m = 1$, and $\mu_1 = k \times n_1 \times n_2$, where n_1 and n_2 are part of the current state. In general, the μ_i's will be some constant k_i times some product of the number of different types of molecules. So μ_i might be $k_i \times n_1$, $k_i \times n_2 \times n_5$, $k_i \times n_1 \times n_3 \times n_5$, etc.[2]

The Gillespie algorithm now asks two questions:

- Which reaction occurs next?
- When does it occur?

[2] For reactions involving two copies of the same substance, μ_i will be of the form $k_i \times \binom{n_1}{2} = k_i \times n_1 \times (n_1 - 1)/2$.

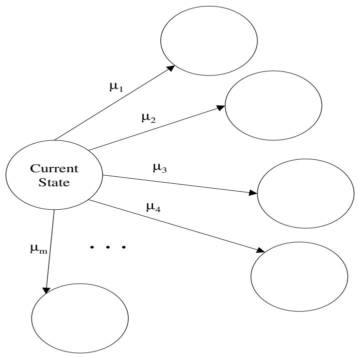


Figure 1: Transition from the current state to m possible next states.

Clearly, the answer to both of these questions must be probabilistic, so we must specify the probability $P(\rho, \tau)$ that the next reaction is ρ and it occurs at time τ. Let us answer the first question. The probability per unit time that reaction 1 occurs is μ_1/μ_2 times the probability per unit time that reaction 2 occurs, etc. The total probability that *some* reaction occurs is 1. Thus,

$$P(\rho = \text{reaction i}) = \mu_i / \sum_j \mu_j \tag{2.1}$$

For the second question, we write out an equation analagous to Equation 1.3, namely

$$P(\text{current state}|t + \Delta t) = P(\text{current state}|t) \times (1 - \sum_j \mu_j \times \Delta t)$$

Rearranging and taking the limit as Δt goes to 0 gives a differential equation whose solution shows that τ is distributed according to an exponential random variable with parameter $\sum_j \mu_j$.

The Gillespie algorithm [6] can thus be stated as:

Algorithm 1 (Exact Stochastic Simulation)

1. Initialize(i.e., set initial numbers of molecules, set $t = 0$.)

2. Calculate μ_i for all i.

3. Choose ρ according to the distribution in Equation 2.1.

4. Choose τ according to an exponential with parameter $\sum_j \mu_j$.

5. Change number of molecules to reflect execution of reaction ρ. Set $t = t + \tau$.

6. Go to Step 2.

Each step through the loop (Steps 2 to 6) is $\mathcal{O}(r)$, due to the calculations in Steps 2 and 3. If E events occur, we go through the loop E times, so the algorithm is $\mathcal{O}(rE)$.

2.2 The Arkin, Ross and McAdams Model

Arkin, Ross and McAdams [1] have written out the complete set of equations of gene regulation in the temperate bacteriophage lambda. The key reactions are summarized in Table 1 and will be discussed further in Section 4. The model considers five genes, so there will be five sets of equations similar to the ones in Table 1. Two of the five genes have protein products that dimerize, i.e., can exist as a protein$_2$ form, as in Table 1, Equations 9 and 10. The other three proteins exist only as monomers, so we can ignore Equations 9 and 10 for those proteins. In addition to the equations shown, Arkin *et al.* provide a more detailed model of degradation for two of the proteins and also a more detailed model of transcriptional termination. Also, these reactions take place in a growing *E. coli* host cell, so they model the volume increase over time as well.

$$
\begin{aligned}
\mathrm{RNAP}\bullet\mathrm{DNA}_{\mathrm{closed}} &\xrightarrow{k_1} \mathrm{RNAP}\bullet\mathrm{DNA}_{\mathrm{open,\ 0}} \\
\mathrm{RNAP}\bullet\mathrm{DNA}_{\mathrm{open,\ n}} &\xrightarrow{k_2} \mathrm{RNAP}\bullet\mathrm{DNA}_{\mathrm{open,\ n+1}} \\
\mathrm{RNAP}\bullet\mathrm{DNA}_{\mathrm{open,\ MAX}} &\xrightarrow{k_3} \mathrm{RNAP}_{\mathrm{free}} + \mathrm{DNA}_{\mathrm{free}} + \mathrm{mRNA}_{\mathrm{free}} \\
\mathrm{RNase} + \mathrm{mRNA}_{\mathrm{free}} &\xrightarrow{k_4} \mathrm{RNase} \\
\mathrm{Ribosome} + \mathrm{mRNA}_{\mathrm{free}} &\xrightarrow{k_5} \mathrm{Ribosome}\bullet\mathrm{mRNA}_0 \\
\mathrm{Ribosome}\bullet\mathrm{mRNA}_{\mathrm{n}} &\xrightarrow{k_6} \mathrm{Ribosome}\bullet\mathrm{mRNA}_{\mathrm{n+1}} \\
\mathrm{Ribosome}\bullet\mathrm{mRNA}_{\mathrm{MAX}} &\xrightarrow{k_7} \mathrm{Ribosome}_{\mathrm{free}} + \mathrm{mRNA}_{\mathrm{free}} + \mathrm{protein} \\
\mathrm{protein} &\xrightarrow{k_8} \mathrm{no\ protein} \\
\mathrm{protein} + \mathrm{protein} &\xrightarrow{k_9} \mathrm{protein}\bullet\mathrm{protein} \\
\mathrm{protein}\bullet\mathrm{protein} &\xrightarrow{k_{10}} \mathrm{protein} + \mathrm{protein}
\end{aligned}
$$

Table 1: Reactions involved in gene expression. In all cases, 'free' means 'in solution', i.e., not bound, and 'MAX' is the transcription length or translation length, depending on context. The notation $A \bullet B$ means "A is bound to B." In particular, "protein•protein" is the dimer form of a given protein, typically written as "protein$_2$." If we model p different proteins, we will have p sets of reactions of this form.

The key idea in gene regulation is that the expression of certain genes regulates the expression of other genes: the "constants" k_i in the equations may depend in some way on the number of proteins present. For the lambda model in particular, the k_1's can be calculated from the concentrations of the various proteins using a straight-forward equilbrium thermodynamics model. See [1] or [8] for a complete description of how to calculate the k_1 values.

Given their complete set of gene regulation reactions, Arkin *et al.* generate trajectories of the system using Gillespie's algorithm (Section 2.1). This gives them a series of trajectories through phase space that describe the expression of lambda genes as a function of time. By running many trajectories, they calculate probabilities that lambda adopts various fates, namely *lysis*, in which lambda replicates and destroys the host cell, or *lysogeny*, in which lambda inserts its DNA into the host DNA and does not destroy the host cell.

Because this sort of simulation involves so many reactions, one is interested in data structures that will save calculation time. For example, Equation 2 is really shorthand for MAX different equations, one for each n between 0 and MAX-1. At a given time, there will be very few RNA

Polymerase (RNAP) molecules bound to DNA, so rather than simulate many equations with μ_i's of 0, one can keep track of which n's will have non-zero μ_i's and only include those equations in calculation. There are numerous possible ad-hoc improvements, but we shall instead concentrate on a systematic treatment.

3 Our Algorithmic Improvements

3.1 Definitions

Consider the equations

$$S_1 + S_2 \xrightarrow{k_1} S_3 \tag{3.1}$$
$$S_2 + S_4 \xrightarrow{k_2} S_5 \tag{3.2}$$
$$S_3 + S_5 \xrightarrow{k_3} S_8 \tag{3.3}$$

In chemical equations, the substances on the *left* side of the arrow are called *reactants* and the substances on the *right* side called *products.*

Definition 3.1 Let *Reactants*(ρ) and *Products*(ρ) be the sets of reactants and products, respectively, of reaction ρ. So, for example, *Reactants*(Equation 3.1) = $\{S_1, S_2\}$ and *Products*(Equation 3.1) = $\{S_3\}$.

For each reaction, we can calculate the μ_i value by multiplying the rate constant and the numbers of molecules of each reactant.[3] Thus, $\mu_{3.1} = k_1 \times n_1 \times n_2$, $\mu_{3.2} = k_2 \times n_2 \times n_4$ and $\mu_{3.3} = k_3 \times n_3 \times n_5$. Thus, each μ_ρ depends on some constant k, which is a property of the reaction, and on the number of molecules of each substance in *Reactants*(ρ).

Definition 3.2 Let μ*DependsOn*(ρ) be the set of substances which affect the value μ_ρ. Then *Reactants*(ρ) $\subseteq$ μ*DependsOn*(ρ), with equality in those reactions where k_ρ is a true constant.

For example *Reactants*(Equation 3.1) = μ*DependsOn*(*Equation* 3.1), but in the example of Table 1, *Reactants*(1) $\subsetneqq$ μ*DependsOn*(1).

Now suppose we execute Equation 3.1. Then one molecule of S_1 and one molecule of S_2 will disappear, and a molecule of S_3 will be created. In general, the number of molecules of each of the reactants will decrease and the number of molecules of each of the products will increase.

Definition 3.3 Let *Affects*(ρ) be the set of substances whose number of molecules changes when reaction ρ is executed. Then *Affects*(ρ) = *Reactants*(ρ) $\cup$ *Products*(ρ)

3.2 The Dependency Graph

Suppose we execute Equation 3.1 above and ask the question: *Which μ_i's change?* The answer is $\mu_{3.1}$, $\mu_{3.2}$ and $\mu_{3.3}$. Now suppose we execute Equation 3.3 and ask the same question. The answer this time is $\mu_{3.3}$ but *not* $\mu_{3.1}$ or $\mu_{3.2}$. For an arbitrary set of equations, we can create a *dependency graph*, which captures the information in this question for all equations.

[3] We shall try to avoid "the product of the reactants," as "product" already has a meaning in chemical reactions.

Definition 3.4 (Dependency Graph) Let a set of reactions $\mathcal{R}$ be given. Let $\mathcal{G}(V, E)$ be a digraph with vertex set $V = \mathcal{R}$, and with a directed edge from v_i to v_j if and only if *Affects*(v_i) $\cap$ μ*DependsOn*(v_j) $\neq \emptyset$. Then $\mathcal{G}$ is called the *dependency graph* of the set of reactions $\mathcal{R}$.

For example, let $\mathcal{R}$ be the set of Equations 3.1 through 3.3. Then $V = \{e_{3.1}, e_{3.2}, e_{3.3}\}$ and $E = \{(e_{3.1}, e_{3.1}), (e_{3.1}, e_{3.2}), (e_{3.1}, e_{3.3}), (e_{3.2}, e_{3.1}), (e_{3.2}, e_{3.2}), (e_{3.2}, e_{3.3}), (e_{3.3}, e_{3.3})\}$, as illustrated in Figure 2a.

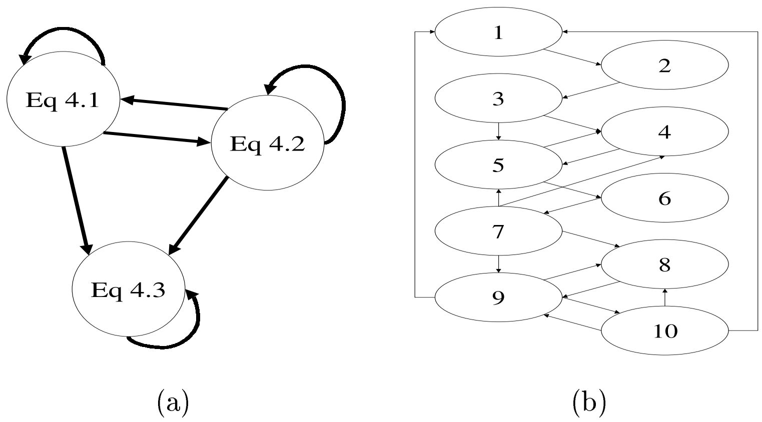


Figure 2: Dependency graphs for: (a) Equations 3.1 through 3.3. (b) Lambda model (Table 1). For simplicity, self-edges are not shown in (b).

3.3 One Algorithm

In subsequent sections, we shall present different algorithms which we feel are significantly better than the one in this section, but we start here as this algorithm follows immediately from Gillespie's algorithm and the idea of dependency graphs. The basic idea is to update the μ_i's only when needed. Basically, we store the μ_i's as the leaves of a complete tree, and give each node a value equal to the sum of its left child and right child. Thus, the root will have value $\sum_i \mu_i$. When we need to update, we update 1) those μ_i's that have changed and 2) their ancestors. So replace Step 2 with

Algorithm 2 1. If this is the initial iteration, calculate μ_i for all i.

2. Otherwise, let ρ be the reaction we just executed. For each edge (ρ, η) in the dependency graph $\mathcal{G}$, update μ_i and each ancestor (in the tree) of μ_i.

Each update is $\mathcal{O}(\log r)$. We could do each in $\mathcal{O}(1)$, but we need the partial sums to get an efficient Step 3. In particular, when we need to generate random numbers, the τ value will be easy, since the root of the tree contains its parameter. For the ρ value, we generate a random number x between 0 and $\sum_i \mu_i$ and then use the following algorithm, starting at the root:

Algorithm 3 (Efficient Uniform Random Number Generation)

1. If the current node is a leaf, return the index of the leaf for ρ.

2. Otherwise, if $0 \leq x \leq$ (left child value), then call this algorithm recursively on the left child with parameter x.

3. Otherwise, (left child value) $\leq x$, so call this algorithm recursively on the right child with parameter $x -$ (left child value).

Generating a random number is thus $\mathcal{O}(\log r)$, so, assuming that the dependency graph is sparse, the total complexity is $\mathcal{O}(r + E \log r)$. It is clear that this algorithm is equivalent to Algorithm 1.

3.4 Our Main Algorithm

We now take a different view of the problem than the Gillespie algorithm (Section 2.1). The Gillespie algorithm asks: *Of all the reactions, which is the next to occur, and when?* We ask: *For* each *reaction ρ, when is the next time ρ occurs?*

To implement our algorithm, we use a *priority queue* [2], a data structure for maintaining a set S of elements, each with a **key**. A priority queue supports the following operations:

- Insert(S, x) - inserts element x into the queue.
- Min(S) - returns the element with the lowest **key**.
- Build(S) - makes the set S into a priority queue.
- Remove(S, x) - removes element x from the queue.

The last is non-standard, but can be implemented efficiently. In addition, we will index our elements by the reaction index ρ, and use that indexing for the Remove operation. Priority queues can be implemented using heaps, in which case the operations Insert and Remove are $\mathcal{O}(\log n)$, Min is $\mathcal{O}(1)$, and Build is $\mathcal{O}(n)$.

Our algorithm is this:

Algorithm 4 (Efficient, Exact Stochastic Simulation)

1. Initialize. (I.e. set initial numbers of molecules, set $t = 0$.)
2. Create the dependency graph $\mathcal{G}$ of the reactions $\mathcal{R}$.
3. For each reaction $\rho \in \mathcal{R}$, calculate μ_ρ. Generate a random number τ_ρ according to an exponential distribution with parameter μ_ρ. Insert the pair (ρ, τ_ρ) into a priority queue P, with **key** value τ.
4. Pick the element (ρ, τ_ρ) of P with the minimum τ.
5. Change number of molecules to reflect execution of reaction ρ. Set $t = \tau_\rho$.
6. For each edge (ρ, η) in the dependency graph $\mathcal{G}$,
   - Remove the pair (η, τ_η) from P.
   - Recalculate μ_η and generate a new τ'_η.
   - Insert the pair $(\eta, \tau_\rho + \tau'_\eta)$ into P.
7. Go to Step 4.

3.5 Discussion of our Algorithm

Gillespie [5] refers to his algorithm (Algorithm 1) as a "direct" algorithm, which he contrasts to the "first reaction" approach (such as Algorithm 4). Without the concept of a dependency graph, the direct algorithm is preferable. However, we prefer the first reaction approach of Algorithm 4, for reasons that will be explained in Sections 3.6 and 3.7.

3.5.1 Correctness

Returning to the "first reaction" approach, we now prove that Algorithm 4 is equivalent to Algorithm 1.

Theorem 1 (Correctness) *Algorithm 4 is equivalent to Algorithm 1 in the following sense: for any trajectory $\mathcal{T}$, the probability $P_4(\mathcal{T})$ that Algorithm 4 produces $\mathcal{T}$ is equal to the probability $P_1(\mathcal{T})$ that Algorithm 1 produces $\mathcal{T}$.*

Proof: The invariant for our loop is that τ_α is drawn from the distribution

$$P(T_\alpha > u) = \begin{cases} \exp(-\mu_\alpha(u-t)) & \text{if } u > t \\ 0 & \text{otherwise} \end{cases}$$

where t is the variable t in our algorithm, not a dummy variable. First we assume the invariant and prove that the selection of ρ in Algorithm 4 is equivalent to the selection of ρ in Algorithm 1, then we prove that the selection of τ in Algorithm 1 is equivalent to our selection of τ_ρ. Finally, we prove the invariant.

In Algorithm 4, the ρ chosen is the one whose τ_ρ is least. Let R be the random variable describing the choice of reaction, and let T_ρ be the random variable describing the choice of τ_ρ. Then

$$\begin{aligned} P(R=\rho) &= P(\min_\alpha(T_\alpha) = T_\rho) \\ &= \int_{u=0}^{\infty} P(T_\rho = u) \prod_{\alpha\neq\rho} P(T_\alpha > u) du \\ &= \int_{u=t}^{\infty} \mu_\rho \exp(-\mu_\rho(u-t)) \prod_{\alpha\neq\rho} \exp(-\mu_\alpha(u-t)) du \\ &= \mu_\rho \int_{u=t}^{\infty} \exp(-\sum_\alpha(\mu_\alpha)(u-t)) du \\ &= \mu_\rho / \sum_\alpha(\mu_\alpha) \end{aligned}$$

This matches the distribution in Equation 2.1, which is used in Step 3 of Algorithm 1. The value of τ in Step 4 of Algorithm 1 matches as well, since for all $u > t$

$$\begin{aligned} P(\min_\alpha(T_\alpha) > u) &= \prod_\alpha P(T_\alpha > u) \\ &= \prod_\alpha \exp(-\mu_\alpha(u-t)) \\ &= \exp(\sum_\alpha(-\mu_\alpha)(u-t)) \end{aligned}$$

Algorithm	Type	Run time	Arbitrary Stoc. Proc.?	Reference
1	Direct	rE	No	Section 2.1, Refs. [5, 6]
Modified 1	Direct	$r + E \log r$	No	Section 3.3
4	First Reaction	$r + E \log r$	No	Section 3.4
5	First Reaction	$r + E \log r$	Yes	Section 3.6

Table 2: Comparison of algorithms for the gene regulation example in the text.

(Note that here u is an absolute time, but in Algorithm 1, the τ picked is the time *difference*, $u - t$, so the distributions do indeed match.)

Now we prove the invariant. Clearly, Steps 1 to 3 establish this invariant. Any η whose τ_η is recalculated in Step 6 will have $\tau_\eta = \tau'_\eta + \tau_\rho$, where τ'_η is drawn according to a standard exponential distribution, and $\tau_\rho = t$ is the shift factor. The only thing left to show is that it is legitimate to not recalculate values of τ_ρ whose μ_ρ values have not changed. This follows directly from the memoryless property of the exponential, namely $P(T_\alpha > u+t | T_\alpha > t) = P(T_\alpha > u)$. This completes the proof. $\square$

3.5.2 Complexity

Notice that the correctness of the algorithm did not depend on any assumptions about the nature of the dependency graph. The complexity, however, does. In particular, we assume that the dependency graph is *sparse* in the sense that the out-degree of any vertex is bounded by some constant, which is independent of the number r of reactions. In biological terms, this means that the substances we are modeling act specifically, i.e. of the r reactions, any given substance is only in a small subset, the size of which does not depend on r.

Given this assumption, Steps 1 2 and 3 are $\mathcal{O}(r)$, and Steps 4 and 6 are of order $\mathcal{O}(\log r)$ [2]. Since the loop (Steps 4 to 7) is executed E times for E events, the entire algorithm is $\mathcal{O}(r + E \log r)$.

Notice that in the gene regulation example, the dependency graph is indeed sparse. Other than the obvious (ρ, ρ) self-edges, the dependency graph consists of the edges illustrated in Figure 2b. This is a total of 19 edges, plus the 10 self-edges, which gives us a mere 29 edges, compared to a maximum possible of $10^2 = 100$. For p proteins, we would have $10p$ reactions, so $100p^2$ possible edges. In addition to p copies of the edges above, there would be at most $2p(p-1)$ additional cross-dependencies, namely the edges of the form (e_9^i, e_1^j) and (e_{10}^i, e_1^j), where i and j are two different genes. This gives us a maximum of $27p + p^2$ edges for $10p$ vertices. Assume that a typical protein does not affect the rate of production of *all* others, but only a small subset. Then the p^2 term drops out and we have that a typical vertex has out-degree of around 3. Compare the performance of the algorithms in Table 2.

3.6 Refinement I: Non-exponential Waiting Times

In the last section, we proved that Algorithm 4 is equivalent to Algorithm 1. Consider the following generalization of Algorithm 4:

Algorithm 5 (Efficient, Exact Stochastic Simulation II)
Modify Algorithm 4 as follows: rather than an exponential, generate τ_ρ according to an arbitrary distribution $P(T_\rho < u)$, possibly with parameter μ_ρ.

Of course, if $P(T_\rho < u) = 1 - exp(-\mu_\rho u)$ for all ρ, then this reduces to Algorithm 4. In general, however, this is not the case.

There are at least two obvious reasons to use Algorithm 5. First is the simplification of Equations 2 and 6 in Table 1 using a gamma distribution and second is volume effects.

3.6.1 Gamma Distribution

Equations 2 and 6 in Table 1 are really shorthand for a large number of equations. The combined equations can either be viewed as a combination of many elementary reactions (in this case, reactions with exponential waiting times), or as one non-elementary reaction whose waiting time is a sum of exponentials, i.e., a gamma variable. If we switch to a single gamma variable, we only have to generate one random number, and, in fact, we decrease the value of E, the total number of events required to get the same trajectory. (So, for example, if there are 1000 steps along DNA to transcribe one molecule of mRNA, our algorithm will treat this as one gamma step rather than 1000 exponential steps, so E will be significantly smaller.) Using Algorithm 5, it is easy to incorporate this computational savings.

3.6.2 Volume

In Equation 1.3, we showed the probability that the system stays in the same state over a small time step, namely

$$P(\text{state}|t + \Delta t) = P(\text{state}|t) \times (1 - k \times n_1 \times n_2 \times \Delta t)$$

In a system such as lambda, the volume is changing. For second order reactions, the k term depends on the volume, and should be replaced with k'/V, where V is the volume and k' is a "true" constant.[4] This leads to the differential equation

$$\frac{dP(T_\rho > t)}{dt} = -\frac{\mu P(T_\rho > t)}{V(t)}$$

Where μ is a constant depending on k' and on the number of molecules of substance S_1, substance S_2, etc. This differential equation is separable, so for simple $V(t)$, we can solve it analytically and get an exact expression for $P(T_\rho > t)$. So, in other words, we can take care of volume expansion by using Algorithm 5.

For example, in Arkin *et al.*, the volume is modeled as increasing linearly. Thus $V(t) = V_0 + c_V t$, which leads to the distribution

$$P(T_\rho > t) = \left(\frac{V_0 + c_V t}{V_0}\right)^{-\mu/c_V}$$

It is a straightforward operation to generate random numbers according to this distribution [7]. Note also that in the limit as c_V goes to zero, this reduces to an exponential distribution with parameter μ/V_0, as expected.

3.6.3 Modifying the Gillespie Algorithm

Since Algorithm 1 is equivalent to Algorithm 4, it should be possible to modify Algorithm 1 to be equivalent to Algorithm 5. In this section we show that this requires a non-trivial modification, so that Algorithm 5 is preferable.

In Algorithm 1, one uses knowledge of the distributions $P(T_\alpha > u)$ to find a closed form for the random variables $P(R_{\text{next}} = \rho)$ and $P(T_{\text{next}} = \tau)$. There are two possible problems here:

[4] Of course, constant is a relative term. In particular, k' will still depend on the temperature, concentrations of various enzymes, possibly ionic strength, etc. If we assume these influences are constant, then k' is also constant.

1. For arbitrary distributions, it may be hard to find a closed-form solution for these two random variables.

2. Non-exponential distributions do not have the memoryless property, so these random variables will (in general) depend on time t in the algorithm, not just on the μ_is.

Thus it is not straightforward to modify Algorithm 1 to deal with arbitrary, non-exponential waiting times. For this reason, Algorithm 5 is preferable.

3.7 Refinement II: Generating Fewer Random Numbers

Gillespie [6] points out that generating pseudo-random numbers is not the same as generating true random numbers, in particular, pseudo-random number algorithms will have finite cycle length, so we would like to improve Algorithm 4 so as to generate fewer random numbers.

Consider Step 6 of Algorithm 4. Assume reaction η follows a probability distribution with given $F(t) \equiv P(\tau_\eta \leq t)$. In particular, the random variable $Y = F(\tau_\eta)$ is uniform on $[0,1]$. Given that $\tau_\eta > \tau_\rho$ when we reach Step 6, it must be the case that $F(\tau_\eta) > F(\tau_\rho)$. Hence, the random variable $Y' \equiv Y|(\tau_\eta > \tau_\rho)$ is uniform on $[F(\tau_\rho), 1]$. It follows that the random variable $Y'' \equiv \frac{Y' - F(\tau_\rho)}{1 - F(\tau_\rho)}$ is uniform on $[0,1]$. So then we can generate τ'_η according to $F^{-1}(Y'')$.

Specifically, we should use the old value of μ_η in calculating all of the F terms, and the new (i.e., recalculated) value of μ_η in calculating all of the F^{-1} terms.

Example 3.1 Generating new probability distributions.
Consider the simple case where all the variables follow exponential distributions. Then $F(t) \equiv P(\tau_\eta \leq t) = 1 - exp(-\mu_{old} \times t)$. Now,

$$Y'' = \frac{exp(-\mu_{old} \times \tau_\rho) - exp(-\mu_{old} \times \tau_\eta)}{exp(-\mu_{old} \times \tau_\rho)} = 1 - exp(-\mu_{old} \times (\tau_\eta - \tau_\rho).$$

Also, $F^{-1}(U) = -\ln(1-U)/\mu_{new}$. Therefore, $\tau'_\eta \equiv \frac{\mu_{old}}{\mu_{new}}(\tau_\eta - \tau_\rho)$ follows the correct distribution for τ'_η.

For sufficiently simple distributions, e.g. the exponential, we may be able to calculate a closed form solution of this transformation. In such cases, we can replace Step 6 by:

Algorithm 6 (Replacement for Step 6)
For the edge (ρ, ρ), generate a new random number using the pseudo-random number generator. For each edge $(\rho, \eta), \rho \neq \eta$ in the dependency graph $\mathcal{G}$,

- Remove the pair (η, τ_η) from P.
- Recalculate μ_η and calculate a new τ'_η using the closed form version of

$$\tau'_\eta = F^{-1}\left(\frac{F(\tau_\eta) - F(\tau_\rho)}{1 - F(\tau_\rho)}\right),$$

  with the old value of μ_η in all of the F terms and the new value of μ_η in the F^{-1} term.
- Insert the pair $(\eta, \tau_\rho + \tau'_\eta)$ into P.

If we are able to find a closed form for all the random variables (e.g., in the all-exponential case), the total number of calls to the pseudo-random number generator is $r + E$, as opposed to $2E$ in Gillespie's algorithm.

3.8 Refinement III: Approximations

Certain calculations, for example, calculating k_1 in Table 1 as a function of the concentrations of all the proteins [1, 8], may be computationally expensive. We have experimented with the approximation that the k_1's are constant over small regions, rather than varying continuously [4]. In Algorithm 4, that means that we do not execute Step 6 for every change in the number of molecules present, but only when those changes exceed certain limits.

4 Results

In this section, we describe the full lambda model and provide results.

4.1 The Full Model

In Section 2.2, we hinted at the lambda model. Let us describe it fully here. This model is more or less that of Arkin *et al.*[1]. We model the five genes *cI*, *cII*, *cIII*, *cro*, and *N*, and the corresponding mRNA and proteins. Like Table 1, the full model consists of transcription, translation and degradation. Some additional reactions are shown in Table 4. We will talk about the equations in each table with numbers corresponding to the rate constant numbers.

$$\begin{array}{c} \text{OR, right} \rightarrow \ \textit{cro} \ \rightarrow \ \text{NUT-R} \ \rightarrow \text{TR}_1 \rightarrow \ \textit{cII} \\ \textit{cIII} \ \leftarrow \text{TL}_1 \leftarrow \ \textit{N} \ \leftarrow \ \text{NUT-L} \ \leftarrow \text{PL, left} \leftarrow \ \textit{cI} \ \leftarrow \text{OR, left} \leftarrow \text{PRE, left} \end{array}$$

Table 3: The linear order of genes (in italics), promoters(OR, PRE, PL), termination sites(TR_1, TL_1) and anti-termination sites(NUT-R, NUT-L).

$\text{N}\bullet\text{RNAP}\bullet\text{DNA}_{\text{open, n}} \xrightarrow{k_{11}} \text{N}\bullet\text{RNAP}\bullet\text{DNA}_{\text{open, n+1}}$	$\text{CIII} + \text{P1} \xrightarrow{k_{22}} \text{P1}\bullet\text{CIII}$
$\text{RNAP}\bullet\text{DNA}_{\text{NUT}} \xrightarrow{k_{12}} \text{RNAP}\bullet\text{DNA}_{\text{NUT+1}}$	$\text{P1}\bullet\text{CIII} \xrightarrow{k_{23}} \text{CIII} + \text{P1}$
$\text{N} + \text{RNAP}\bullet\text{DNA}_{\text{NUT}} \xrightarrow{k_{13}} \text{N}\bullet\text{RNAP}\bullet\text{DNA}_{\text{NUT}}$	$\text{P1}\bullet\text{CIII} \xrightarrow{k_{24}} \text{P1}$
$\text{N}\bullet\text{RNAP}\bullet\text{DNA}_{\text{NUT}} \xrightarrow{k_{14}} \text{N} + \text{RNAP}\bullet\text{DNA}_{\text{NUT}}$	$\text{CII} + \text{P2} \xrightarrow{k_{25}} \text{P2}\bullet\text{CII}$
$\text{N}\bullet\text{RNAP}\bullet\text{DNA}_{\text{NUT}} \xrightarrow{k_{15}} \text{N}\bullet\text{RNAP}\bullet\text{DNA}_{\text{NUT+1}}$	$\text{P2}\bullet\text{CII} \xrightarrow{k_{26}} \text{CII} + \text{P2}$
$\text{RNAP}\bullet\text{DNA}_{\text{T}} \xrightarrow{k_{16}} \text{RNAP}\bullet\text{DNA}_{\text{T+1}}$	$\text{P2}\bullet\text{CII} \xrightarrow{k_{27}} \text{P2}$
$\text{RNAP}\bullet\text{DNA}_{\text{T}} \xrightarrow{k_{17}} \text{RNAP}_{\text{free}} + \text{DNA}_{\text{free}}$	$\text{CIII} + \text{P2} \xrightarrow{k_{28}} \text{P2}\bullet\text{CIII}$
$\text{N}\bullet\text{RNAP}\bullet\text{DNA}_{\text{T}} \xrightarrow{k_{18}} \text{N}\bullet\text{RNAP}\bullet\text{DNA}_{\text{T+1}}$	$\text{P2}\bullet\text{CIII} \xrightarrow{k_{29}} \text{CIII} + \text{P2}$
$\text{CII} + \text{P1} \xrightarrow{k_{19}} \text{P1}\bullet\text{CII}$	$\text{P2}\bullet\text{CIII} \xrightarrow{k_{30}} \text{P2}$
$\text{P1}\bullet\text{CII} \xrightarrow{k_{20}} \text{CII} + \text{P1}$	$\text{no protein} \xrightarrow{k_{31}} \text{protein}$
$\text{P1}\bullet\text{CII} \xrightarrow{k_{21}} \text{P1}$	$\text{Volume} = \text{Initial Volume} + k_{32} \times \text{Time}$

Table 4: Reactions for the lambda model that were not covered in Table 1.

4.1.1 Transcription

Table 3 contains the linear structure of DNA that is relevant for the model. Transcription can initiate at any of the promoters, according to Equation 1. The parameter k_1 is determined by an equilibrium binding model — Shea *et al.* [8] for O_R, Arkin *et al.*[1] for P_L and P_RE.

Once transcription has initiated at a given promoter in a given direction, RNAP moves along the DNA following Equation 2. Whenever RNAP reaches the end of a gene, an mRNA molecule is produced — this is similar to Equation 3, but RNAP will stay bound to the DNA for genes *cro*, *cI*, and *N* and will continue transcribing. For genes *cII* and *cIII*, RNAP will come off the DNA exactly as in Equation 3.

RNAP slows down at the N-utilization sites, NUT-R and NUT-L. From either site, RNAP can either continue transcribing according to Equation 2, or it can bind N and then continue transcribing, according to Equation 11 in Table 4. Equations 12 through 15 show the full model of the N-utilization sites.

When RNAP reaches one of the termination sites, TR_1 or TL_1, it can continue transcribing or can fall off the DNA. These two options are shown in Equations 16 and 17. The relevant rate constants are set to get the correct pass-through ratios (50% for TR_1 and 20% for TL_1). If RNAP was antiterminated at one of the NUT sites, it passess through with probability 1 (Equation 18).

Parameter	Value	Parameter	Value
k_1	From binding model	$k_{17}(TL_1)$	25 sec^{-1}
k_2	30 nt sec^{-1}	$k_{16}(TR_1)$	15 sec^{-1}
k_3	30 nt sec^{-1}	$k_{17}(TR_1)$	15 sec^{-1}
RNase$\times k_4$	0.03 sec^{-1}	k_{18}	30 sec^{-1}
Ribosome$\times k_5$	0.3 sec^{-1}	k_{19}	0.01 mol^{-1} sec^{-1}
k_6	100 nt sec^{-1}	k_{20}	0.01 sec^{-1}
k_7	100 nt sec^{-1}	k_{21}	0.015 sec^{-1}
k_8(cI)	0.0007 sec^{-1}	k_{22}	0.05 mol^{-1} sec^{-1}
k_8(cro)	0.0025 sec^{-1}	k_{23}	0.001 sec^{-1}
k_8(N)	0.0023 sec^{-1}	k_{24}	0.0001 sec^{-1}
k_9(cI)	0.05 mol^{-1} sec^{-1}	k_{25}	0.0001 mol^{-1} sec^{-1}
k_9(cro)	0.05 mol^{-1} sec^{-1}	k_{26}	0.065 sec^{-1}
k_{10}(cI)	0.5 sec^{-1}	k_{27}	0.6 sec^{-1}
k_{10}(cro)	0.5 sec^{-1}	k_{28}	0.01 mol^{-1} sec^{-1}
k_{11}	30 nt sec^{-1}	k_{29}	0.01 sec^{-1}
k_{12}	5 nt sec^{-1}	k_{30}	0.001 sec^{-1}
k_{13}	0.145 mol^{-1} sec^{-1}	k_{31}(RNAP)	0.0146 mol^{-1} sec^{-1}
k_{14}	0.1 sec^{-1}	k_{31}(P1)	0.0116 mol^{-1} sec^{-1}
k_{15}	30 nt sec^{-1}	k_{31}(P2)	0.0465 mol^{-1} sec^{-1}
$k_{16}(TL_1)$	5 sec^{-1}	k_{32}	$(35\ \text{min})^{-1}$

Table 5: Parameter values used for lambda model. N.B. mol = molecules, not moles. All parameters are in microscopic units, which differ from the traditional macroscopic values by a fixed constant.

4.1.2 Translation, degradation

Translation and mRNA degradation follow Equation 4 through Equation 8 in Table 1 exactly. Proteins N, cI and cro are degraded according to Equation 8. Proteins cII and cIII follow a more complicated model, which follows in the next subsection. Proteins cI and cro can exist as dimers and hence follow Equations 9 and 10.

4.1.3 cII, cIII degradation

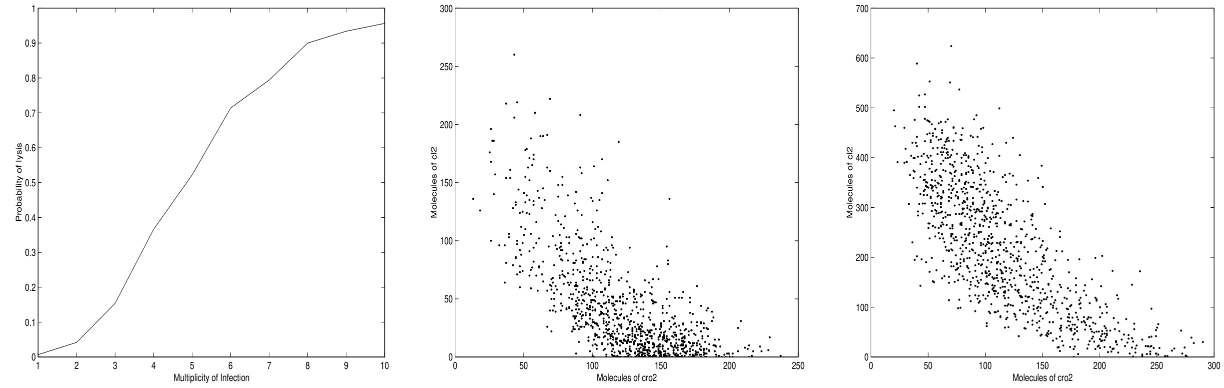
It is thought that saturation of the cII-degradation machinery in the cell is the main reason why hosts that are infected by multiple phages favor lysogeny. The simple protein degradation model in Equation 8, which works for the above proteins, does not saturate, so a more detailed model is necessary. That model is in Table 4, Equations 19 through 30.

4.1.4 Miscellaneous

There are a few more miscellaneous equations to complete the model. First, there are several proteins whose concentration is assumed to be constant, so they need to be produced at an appropriate rate, as in Equation (31). Second, there is the growth of the host cell, Equation 32.

4.2 Comparison to Previous Results

Using the values in Table 5 for the rate constants, and the values in Shea *et al.* [8] and Arkin *et al.*[1] the equilibrium constants of the promoters, we can do the same calculation as Arkin *et al.*[1]. Figure 3a is a plot of the probability of lysogeny as a function of Multiplicity of Infection(MOI), i.e., the number of phages per host cell. As we have proved our algorithm to be equivalent to Gillespie's algorithm, the calculation plotted here is equivalent to that in Arkin *et al.* [1]. The difference in the two algorithms is speed. Our calculation and theirs took comparable amounts of time, but theirs ran on a 200-node supercomputer, whereas ours ran on 10 desktop workstations. Figures 3b and c show the final number of cro_2 and cI_2 protein molecules present in a simulated population for MOI 3 and 6, respectively.



(a) (b) (c)

Figure 3: (a) Probability of lysogeny as a function of Multiplicity of Infection(MOI). (b) The number of molecules of cI_2 and of cro_2 at the time when the host divides, for MOI=3, and (c) for MOI=6.

4.3 Sensitivity Analysis

Using our fast algorithm, we can do some biologically interesting calculations that would take prohibitively long using slower algorithms. For example, we can look at the parameter values

Parameter	Change	Distance
Control		0.025
k_2, k_{11}	$\times 2$	0.313
k_2, k_{11}	$\times 1/2$	1.276
k_4	$\times 2$	10.300
k_4	$\times 1/2$	4.555
k_5	$\times 2$	4.551
k_5	$\times 1/2$	11.175
k_6	$\times 2$	0.045
k_6	$\times 1/2$	0.056
k_8(cI)	$\times 2$	0.040
k_8(cI)	$\times 1/2$	0.023
k_9(cI)	$\times 2$	0.036
k_9(cI)	$\times 1/2$	0.045
k_{10}(cI)	$\times 2$	0.067
k_{10}(cI)	$\times 1/2$	0.040

Table 6: Kullbach Liebler distance between the probability distribution plotted in Figure 3a and the corresponding distribution calculated with different parameter values.

in Table 5, and test systematically how much the final results depend on the exact parameter values. The experiment is as follows: we change one parameter value at a time, and run the same calculation as in Section 4.2. This gives us (among other things) the probability that a host cell will go to lysogeny as a function of MOI. We compare this probability distribution with the baseline distribution from Section 4.2 using the Kullbach Liebler distance [3] as a measure of the difference between the two distributions.[5]

Rather than do this analysis for every possible parameter, we consider only those parameters from Table 5 which affect gene *cI* and its mRNA and protein products. The results of this analysis are shown in Table 6. The first line is a control, i.e., the parameters were the same, but the random number generator was reseeded. In each subsequent run, one parameter was either doubled or halved.

Based on this analysis, the simulation is not particularly sensitive to some of the parameters. For example, the distance measure for changing $k_8(cI)$ by a factor of 2 is on the same order as the control. On the other hand, the simulation is *very* sensitive to the values of k_4 and k_5. Figure 4 shows the probability distributions corresponding to changed values of k_4.

In Figure 5, we have plotted Kullbach-Liebler distance as a function of parameter values. Note that decreasing k_8 by an arbitrary amount has little effect, and increasing it within an order of magnitude also has very little effect. On the other hand, changing k_5 by a factor of two in either direction has a huge effect on the probability of lysis and of lysogeny.

In particular, k_4 and k_5 give the expected number of proteins per mRNA transcript, namely $(k_5 \times \text{Ribosome})/(k_4 \times \text{RNase})$. Thus multiplying k_4 by 2 should have similar effects to dividing k_5 by 2, as can be seen from Table 6. The meta-parameter, expected number of proteins per mRNA transcript, is estimated to have a value of 10 in Arkin *et al.* [1]. Our analysis indicates that the simulation is very sensitive to that parameter. Other parameters, such as k_8, do not affect the results significantly, so the current estimates (assuming they are in the right order of magnitude) are sufficient.

[5] We have actually used a symmetric version of the Kullbach Liebler distance.

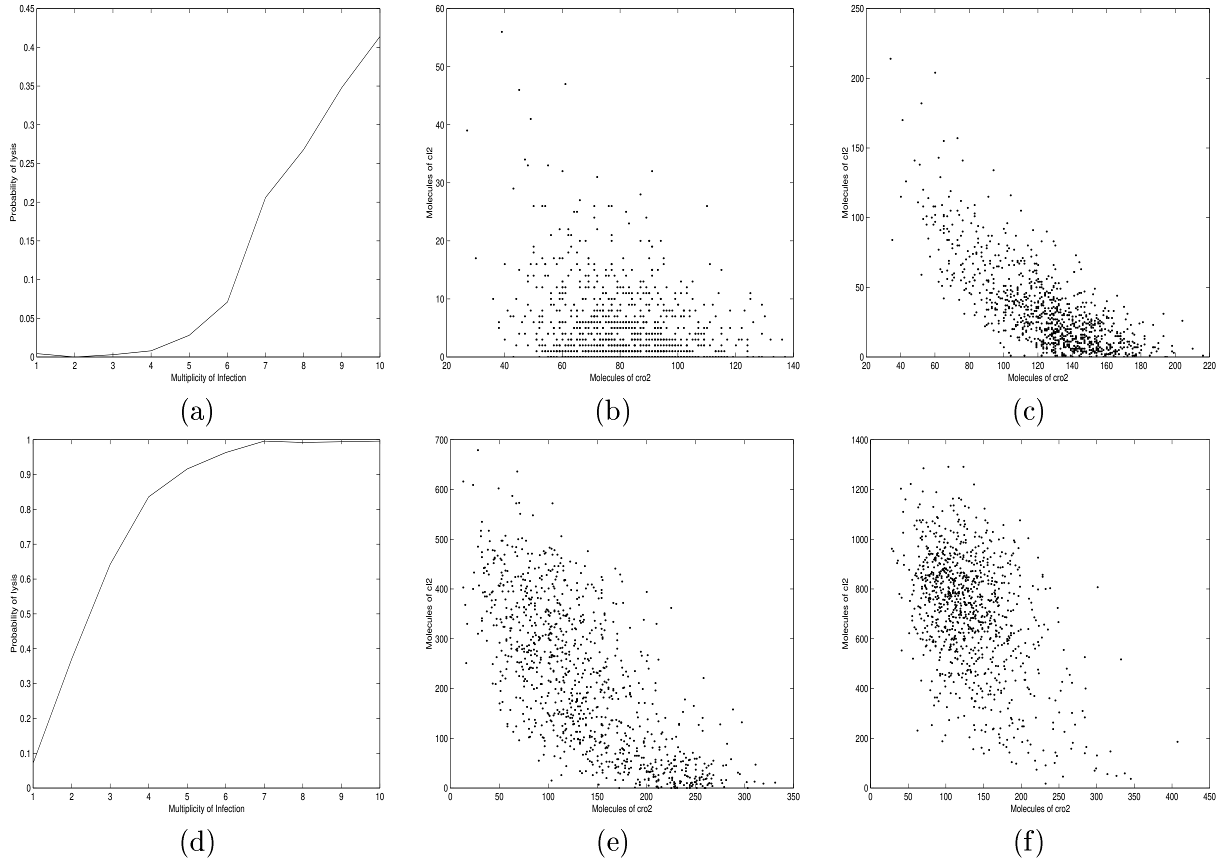


Figure 4: The plots that correspond to those in Figure 3. (a)-(c) $k_4 \times 2$ (d)-(f) $k_4/2$

There are three ways to think about parameter sensitivity:

1. Sensitivity analysis tells us which parameters to measure with high precision in order to improve the model. In this example, one should endeavor to measure the average number of proteins per mRNA transcript more carefully.

2. Sensitivity analysis makes predictions about which parameters should be varied to have maximum effect. For example, an experimental manipulation of [RNase] should affect the probability of lysogeny very strongly.

3. Sensitivity analysis makes suggestions as to possible effects. In the present treatment, we have noted that as the average number of proteins per transcript increases, so does the probability of lysogeny. Conceivably, as MOI increases, the mRNA degradation machinery could saturate faster than the mRNA translation machinery. This would cause the average number of transcripts to increase, which would in turn favor lysogeny. Thus, saturation of mRNA degradation *could be* an important effect that leads to lysogeny at higher MOI. Compare this to the analysis of the degradation rate of the cI protein. From sensitivitiy analysis, we can see that even if the cI degradation machinery saturates, it will have no significant effect on the probability of lysogeny.

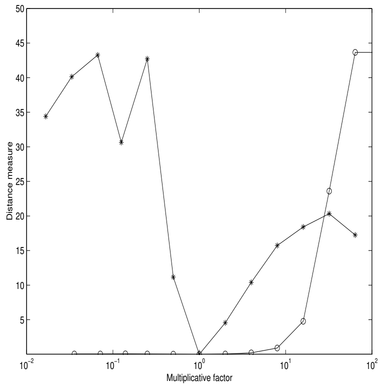


Figure 5: Kullbach-Liebler distance as a function of parameter values. Star = k_5, circle = k_8. The x-axis is the parameter value divided by the parameter value in Table 5.

5 Conclusions

We consider algorithms for creating a stochastic trajectory of a system of chemical equations. The current algorithm uses a "direct" approach, and is $\mathcal{O}(rE)$, where r is the number of reactions, and E is the number of events which are included in the trajectory. We present two basic approaches, one "direct" and one based on a "first reaction" methodology, both of which are $\mathcal{O}(r + E \log r)$ for loosely coupled reactions such as those found in gene regulation models. Additionally, our "first reaction" algorithm is more easily extended to include stochastic processes of different types, which may come up either as actual higher-order phenomina or may be be used strictly for their computational advantage.

We apply our fast algorithm to analysing a model of the bacteriophage lambda. Using only workstations, we can do as much calculation as previously could be done only on supercomputers. Further, we can do calculations that would have been prohibitively expensive before, such as analysing which parameters of the model affect the final outcome. This sort of analysis may be used to improve the model, to make experimental predictions and to develop a deeper understanding of which processes are key.

References

[1] A. Arkin, J. Ross and H. H. McAdams. Stochastic Kinetic Analysis of Developmental Pathway Bifurcation in Phage Lambda-infected *Escherichia coli* Cells *Genetics*, 149(4): 1633-1648, Aug 1998.

[2] T. H. Cormen, C. E. Leiserson and R. L. Rivest. Introduction to Algorithms *The MIT Press* and *McGraw-Hill Book Company*, 1990.

[3] T. M. Cover and J. A. Thomas Elements of Information Theory *John Wiley & Sons, Inc.*, 1991.

[4] M. A. Gibson and J. Bruck. The Stochastic Competition Model of Gene Regulation in Lambda Phage. In preparation.

[5] D. T. Gillespie. A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. *Journal of Computational Physics*, 22: 403-434, 1976.

[6] D. T. Gillespie. Exact Stochastic Simulation of Coupled Chemical Reactions. *J Phys Chem*, 81(25):2340-2361, 1977.

[7] A. Leon-Garcia. Probability and Random Processes for Electrical Engineering. *Addison-Wesley Publishing Company*, 1994.

[8] M. A. Shea and G. K. Ackers. The O_R Control System of Bacteriophage Lambda, A physical-Chemical Model for Gene Regulation. *J Mol Biol*, 181:211-230, 1985.

[9] N. G. van Kampen. Stochastic Processes in Physics and Chemistry. *North Holland*, 1981.