Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels[1]

Michael A. 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
October 12, 1999

1 Abstract

There are two fundamental ways to view coupled systems of chemical equations: as continuous, represented by differential equations whose variables are concentrations, or as discrete, represented by stochastic processes whose variables are numbers of molecules. Although the former is by far more common, systems with very small numbers of molecules are important in some applications, e.g., in small biological cells or in surface processes. In both views, most complicated systems with multiple reaction channels and multiple chemical species cannot be solved analytically. There are *exact* numerical simulation methods to simulate trajectories of discrete, stochastic systems, methods that are rigorously equivalent to the Master Equation approach, but they do not scale well to systems with many reaction pathways.

This paper presents the Next Reaction Method, an exact algorithm to simulate coupled chemical reactions that is also *efficient*: it (a) uses only a single random number per simulation event, and (b) takes time proportional to the *logarithm* of the number of reactions, not to the number of reactions itself. The Next Reaction Method is extended to include time-dependent rate constants and non-Markov processes, and it is applied to a sample application in biology: the lysis/lysogeny decision circuit of lambda phage. When run on lambda, the Next Reaction Method requires approximately 1/15th as many operations as a standard implementation of the existing methods.

2 Introduction

The process of creating a mechanistic, predictive model of a system can be broken into two steps: (a) creating a complete description of the chemical, physical, and biological processes involved; (b) using mathematics to generate predictions. For chemical processes, the first step is accomplished by writing a system of chemical equations and perhaps a description of certain physical processes — temperature, volume, electric field, diffusion, stirring, etc. In principle, (a) can be accomplished without any thought about or care for (b); one may describe a system completely without reference to the subsequent process of calculation. In fact, such a calculation-independent model is more fundamental than calculation-based models: the same process, e.g., $A + B \longrightarrow C$, occurs whether there are 20 or 10^{20} molecules of A and B. The *mathematical* problem, (b), is very much different if there are 20 or 10^{20} molecules of A and B [4]. Additional mathematical or computational assumptions may be used to simplify calculations. These assumptions are just that: simplifying computational assumptions; they have nothing to do with the process, but rather with how one represents that process so as to allow efficient computation.

2.1 Mathematical descriptions of chemical processes

A coupled system of chemical reactions, of the form:

$$X_1 + X_2 \longrightarrow X_3 + ...$$

states that one molecules of substance X_1 reacts with one of substance X_2 to give one molecule of substance X_3, etc. A complicated chemical process can be decomposed into a set of many such reactions.

[1]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.

(Although it is possible to write arbitrarily high order reactions, virtually all real systems can be broken up into *elementary* reactions that have at most two reactants and rarely more than three products.)

One may make certain computational assumptions and proceed through to predictions. For example, one may *assume* that there are sufficiently many molecules that the number of molecules can be *approximated* as a continously varying quantity that varies deterministically over time. In this approach, one writes a coupled system of differential equations for the concentration of each substance in terms of the concentrations of all others:

$$\begin{aligned} \frac{d[X_1]}{dt} &= f_1([X_1],[X_2],[X_3]...) \\ \frac{d[X_2]}{dt} &= f_2([X_1],[X_2],[X_3]...) \end{aligned} \tag{1}$$

and so on. Solving these differential equations results in the concentration of each substance as a function of time.

Sometimes, one *assumes* that the process in question is fast compared to the time scale of interest, and can be considered to have reached equilibrium. In that case, one replaces the differential equation Eq. (1) with the algebraic equations

$$\begin{aligned} 0 &= f_1([X_1],[X_2],[X_3]...) \\ 0 &= f_2([X_1],[X_2],[X_3]...) \end{aligned}$$

and so on, whose solutions give only equilibrium concentrations, not dynamics. The assertion 'the system is in equilibrium' is recognized as an *assumption*; it is not true in general. In fact, the very statement of Eq. (1) was based on another assumption, namely that the number of molecules can be approximated as a *continously* varying quantity that varies *deterministically* over time. This assumption is so fundamental to the majority of chemical kinetics that it is frequently not even viewed as an assumption, but as a rigorous consequence of chemical theory. Although this assumption holds for most systems one deals with, this does not hold in very small systems (which are very important in biology).

Where does that leave us?

One may assume only that all molecules involved obey the laws of quantum mechanics; for systems consisting of complex molecules interacting in complex ways over long times, this is completely intractable.

One may assume that quantum effects are small, and that molecules obey Newton's laws of motion. For very simple systems, this molecular dynamics approach has some merit; for systems with complex macromolecules (e.g., biological proteins), long time scales, and interactions of several different molecule types, this too becomes intractable.

Finally, one may assume that the solution is well mixed — that non-reactive collisions occur far more often than reactive collisions — and hence that the fast dynamics of motion can be neglected, and one may represent the system simply by the number of each kind of molecule. This approach leads rigorously to the following statement: the probability that a certain reaction μ will take place in the next instant of time dt is given by $a_\mu dt + o(dt)$, where a_μ is independent of dt, and $o(dt)$ denotes terms that are negligible for small dt [7]. However, a_μ may depend on (a) μ, (b) the current number of molecules of each kind, and (c) the current time. (In particular, a_μ depends on temperature and volume, which may change with time.) The remainder of this paper will assume the stochastic framework.

2.2 The stochastic framework[7, 15]

Consider, for example, the set of reactions:

$$
\begin{aligned}
A + B &\xrightarrow{k_1} C \\
B + C &\xrightarrow{k_2} D \\
D + E &\xrightarrow{k_3} E + F \\
F &\xrightarrow{k_4} D + G \\
E + G &\xrightarrow{k_5} A
\end{aligned} \tag{2}
$$

The propensities of the reactions are given by k_1, k_2, ... k_5. For example, the probability that a given molecule of A reacts with a given molecule of B in a small time dt is $k_1 dt + o(dt)$. The 'constants' k_1 may be a function of volume, temperature, electrolyte concentration, etc.

One way to proceed is to label each molecule of A: A_1, A_2, ... $A_{\#A}$, each molecule of B: B_1, B_2, ... $B_{\#B}$, etc. Now there are $(\#A) \times (\#B)$ distinct copies of Reaction 1 that can occur, $(\#B) \times (\#C)$ distinct copies of Reaction 2, etc. (The algorithm presented in [13] uses this approach, picks random molecules of A and B, and sees whether they react.) Previous work on efficient simulation [11] focussed on surface processes, where reactions may take place on a large matrix of (x,y) positions. The current work focusses on reactions in solution, for which position is not important, and one can group *by molecule type*. The $(\#A) \times (\#B)$ copies of Reaction 1 are thus grouped into a single reaction, whose propensity is $k_1 \times (\#A) \times (\#B) dt + o(dt)$.

The state of the system in the stochastic framework is defined by the number of molecules of each specie and changes discretely whenever one of the reactions is executed. The probability that a certain reaction μ will take place in the next instant of time dt is given by $a_\mu dt + o(dt)$. For example, the state $S = (\#A, \#B, \#C, \#D, \#E, \#F, \#G)$ will change to $S' = (\#A - 1, \#B - 1, \#C + 1, \#D, \#E, \#F, \#G)$ if Reaction 1 is executed. The probability of this occurrance is given by:

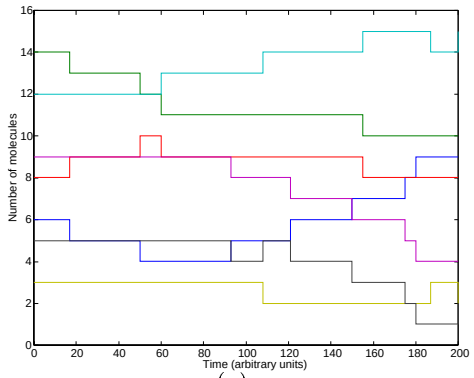
$$
P(S', t + dt | S, t) = a_1 dt + o(dt) \tag{3}
$$

(Note that since the transition probability depends only on the current state and not on previous states, the underlying process is Markov.) Grouping by molecule type allows immense improvements. Specifically, this paper will demonstrate speedups above and beyond those in [11]. However, there are also cases in which other kinds of groupings are even better, e.g., the case study in Section 5.2.1.

One standard way to deal with the stochastic framework is to create one probability variable for each possible state$(\#A, \#B, \#C, \#D, \#E, \#F, \#G)$. Then, using Eq. (3) and the definition of the a_i's as a function of state, one writes out a system of coupled differential equations that defines the system. This coupled set of differential equations has *probabilities as variables* and is called a Master Equation. For a system with very few states, such as an ion channel [2], one may write out this entire system of equations and solve it. For larger systems, however, this approach quickly becomes unreasonable. In the example above, suppose $0 \leq \#A \leq 9$, $0 \leq \#B \leq 9$, etc. Then there are 10^7 states in this simple 5-equation system. In the Arkin *et al.* [1] model of lambda phage, reasonable limits on the number of each kind of molecule lead to a number of states on the order of 10^{70}. Even if this estimate is off, or a clever reduction of the number of states is possible, say by a factor of 10^{50}, that still leaves 10^{20} states and is still untractable. Exact stochastic simulation provides a more feasible approach.

2.3 Exact stochastic simulation

As a different way of dealing with the stochastic framework, consider the problem of generating a single sample trajectory of a chemical process in the stochastic framework, as in Figure 1(a). The (intractable) Master Equation approach tries to write a system of equations and solve simultaneously for the probability of *all* possible trajectories. Generating *a single* trajectory is signficantly easier; as in Figure 1(b), one needs to generate a sequence of state transitions and the times at which they occur. A naive way to generate legal trajectories is to start with the initial state and repeatedly pick reactions arbitrarily and execute them, thus generating a legal trajectory. A better way to generate trajectories is to pick reactions and times *according the the correct probability distributions* so that the probability of generating a given trajectory with the simulation algorithm is exactly the probability that would come out of the solution of the Master Equation.



(a)

Time	0	17	50	60	93	108	121	150	155	175
#A	6	5	4	4	5	5	6	7	7	8
#B	14	13	12	11	11	11	11	11	10	10
#C	8	9	10	9	9	9	9	9	8	8
#D	12	12	12	13	13	14	14	14	15	15
#E	9	9	9	9	8	8	7	6	6	5
#F	3	3	3	3	3	2	2	2	2	2
#G	5	5	5	5	4	5	4	3	3	2
Reaction	—	1	1	2	5	4	5	5	2	5

(b)

Figure 1: Example trajectory. (a) Graphical representation. Legend: A-blue, B-green, C-red, D-cyan, E-purple, F-yellow, G-grey. (b) State representation. The 'Reaction' row merely indicates which reaction occured; it is not part of the state.

Amazingly, it is possible to create an algorithm that has this property, *even if* it is not possible to write out the entire Master Equation explicitly, let alone solve it. (There are also *in*exact stochastic simulation algorithms, that generate trajectories according to approximately the correct distribution. Our interest is only in exact methods, however.)

Given the ability to generate a single trajectory with the correct probability, one may estimate any parameter of interest by generating many trajectories, calculating the value of the parameter for each trajectory, and observing the statistics of those calculated values. For example, to find the average number of molecules of B present at time t, one can run many (say hundreds or thousands) of trajectories, and plot a histogram of the values of the number of molecules of B at time t.

Gillespie [5, 6] developed two *exact stochastic simulation* algorithms, discussed in the next section. The tricky mathematical part of such an algorithm is specifying how to generate random numbers so that they will have the correct distributions. The tremendous success of these exact stochastic simulation algorithms has led to them being applied to much larger systems than was originally designed for. For example Arkin *et al.* [1] used exact stochastic simulation to simulate a model of a simple virus, lambda phage, containing 75 equations in 57 chemical species. Because the original algorithms do not scale readily to large systems, we have developed new versions that do scale well with number of reactions. (The tricky computer science part is to develop *efficient* algorithms that do the right thing.)

2.4 Algorithms

Consider a system of r reactions as in Eq. (2). For now, assume that all rate constants (e.g., $k_1 \ldots k_5$ in Eq. (2)) are true constants; time-varying rate constants will be covered in Section 5. Gillespie [6] proposed two *exact stochastic simulation* algorithms. 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. The key is to choose random numbers using a computer's random number generator, and use those to pick transitions. One must be careful to choose from the correct distribution at each point in the algorithm.

Gillespie proposed two methods for accomplishing the simulation. The first method, which he calls the *Direct Method*, calculates explicitly *which* reaction occurs next and *when* it occurs. The second method, which he calls the *First Reaction Method*, generates, for each reaction μ, a putative time τ_μ at which reaction μ occurs, then choses the reaction μ^* with the smallest τ_μ^* (the *first* reaction) and executes reaction μ^* at time τ_μ^*. This section describes both of these methods.

2.5 Gillespie's Direct Method

For a system in a given state, Gillespie's direct algorithm asks two questions:

- Which reaction occurs next?

- When does it occur?

Clearly, both of these questions must be answered probabilistically, by specifying the probability density $P(\mu,\tau)$ that the next reaction is μ and it occurs at time τ. It can be shown [6], that

$$P(\mu,\tau)d\tau = a_\mu \exp\left(-\tau \sum_j a_j\right) d\tau \tag{4}$$

This leads directly to the answers of the two questions above. First, what is the probability distribution for reactions? Integrating $P(\mu,\tau)$ over all τ from 0 to ∞

$$\Pr(\text{ Reaction } = \mu\ ) = a_\mu / \sum_j a_j \tag{5}$$

Second, what is the probability distribution for times? Summing $P(\mu,\tau)$ over all μ,

$$P(\tau)d\tau = \left(\sum_j a_j\right) \exp\left(-\tau \sum_j a_j\right) d\tau \tag{6}$$

These two distributions lead to Gillespie's direct algorithm [6]:

Algorithm 1 *Exact Stochastic Simulation — Direct Method (Gillespie)*

1. *Initialize(i.e., set initial numbers of molecules, set* $t \leftarrow 0$.*)*
2. *Calculate the propensity function,* a_i*, for all* i.
3. *Choose* μ *according to the distribution in Eq. (5).*
4. *Choose* τ *according to an exponential with parameter* $\sum_j a_j$ *(as in Eq. (6)).*
5. *Change the number of molecules to reflect execution of reaction* μ*. Set* $t \leftarrow t + \tau$.
6. *Go to Step 2.*

As written, this algorithm uses two random numbers per iteration, takes time proportional to the number of reactions to update the a_i's, and takes time proportional to the number of reactions to calculate $\sum_j a_j$ and to generate a random number according to the distribution in Eq. (5). The ideas in the rest of this paper can be used to make the algorithm more efficient, so that the time it takes is proportional to the logarithm of the number of reactions; that discussion is in the appendix. The rest of the paper will focus ways of improving the First Reaction Method.

2.6 Gillespie's First Reaction Method

The algorithm of the previous subsection is direct in the sense that it generates μ and τ directly. Gillespie also developed the First Reaction Method [5], which generates a putative time τ_i for each reaction to occur — a time the reaction would occur if no other reaction occurred first — then lets μ be the reaction whose putative time is first, and lets τ be the putative time τ_μ. Formally:

Algorithm 2 *(Exact Stochastic Simulation — First Reaction Method)*

1. *Initialize(i.e., set initial numbers of molecules, set* $t \leftarrow 0$.*)*
2. *Calculate the propensity function,* a_i*, for all* i.

3. *For each i, generate a putative time, τ_i, according to an exponential distribution with parameter a_i.*

4. *Let μ be the reaction whose putative time, τ_μ, is least.*

5. *Let τ be τ_μ.*

6. *Change the number of molecules to reflect execution of reaction μ. Set $t \leftarrow t + \tau$.*

7. *Go to Step 2.*

At first glance, these two algorithms may seem very different, but they are provably equivalent [5], i.e., the probability distributions used to choose μ and τ are the same. We shall not repeat the proof here.

As written, this algorithm uses r random numbers per iteration (where r is the number of reactions), takes time proportional to r to update the a_i's, and takes takes time proportional to r to identify the smallest τ_μ.

2.7 Organization

The remainder of this paper presents a modified version of the First Reaction Method, which we call the Next Reaction Method, that is more efficient both in terms of number of operations and number of random numbers used. Section 5 shows how to extend the Next Reaction Method to time-dependent stochastic processes. The extended algorithm is applied to the Arkin *et al.* [1] model of lambda phage in Section 6.

3 The Next Reaction Method

Gillespie's First Reaction Method has three activities that occur every iteration and take time proportional to the number of reactions, r:

1. updating all r of the a_i's,
2. generating a putative time, τ_i, for each i, and
3. identifying the smallest putative time, τ_μ.

The *Next Reaction Method* will do away with each of these in turn. The main ideas used are:

- *Store τ_i, not just a_i.*
- *Be extremely sensitive in recalculating a_i (and τ_i); recalculate a_i only if it changes.* The preceeding statement may seem circular: how can one know that a_i has changed or not changed without calculating it and comparing to its old value? In fact, one can analyze the set of reactions beforehand and determine which reactions change which a_i's. Section 4.1 will introduce a data structure, called a *dependency graph*, which allows one to update the minimum number of a_i's.
- *Reuse τ_i's where appropriate.* Theorem 1 in Section 4.3 plus two simple transformations make it possible to reuse all τ_i's except for τ_μ, the time of the reaction that was just executed.
- *Switch from relative time (time between reactions) to absolute time.* This will obviate the need for one of the two transformations above; for reactions whose underlying a_i has not changed, the putative time τ_i will not have to change, either.
- *Use appropriate data structures to store a_i's (and τ_i's), so that updating those that change will be a very efficient operation.* Section 4.2 shows a data structure, called an *indexed priority queue*, that achieves this goal.

The formal statement of the algorithm is:

Algorithm 3 *(Exact Stochastic Simulation — Next Reaction Method)*

1. *Initialize:*

   (a) *Set initial numbers of molecules, set* $t \leftarrow 0$*, generate a dependency graph* $\mathcal{G}$*.*

   (b) *Calculate the propensity function,* a_i*, for all* i*.*

   (c) *For each* i*, generate a putative time,* τ_i*, according to an exponential distribution with parameter* a_i*.*

   (d) *Store the* τ_i *values in an indexed priority queue* $\mathcal{P}$*.*

2. *Let* μ *be the reaction whose putative time,* τ_μ*, stored in* $\mathcal{P}$*, is least.*

3. *Let* τ *be* τ_μ*.*

4. *Change the number of molecules to reflect execution of reaction* μ*. Set* $t \leftarrow \tau$*.*

5. *For each edge* (μ, α) *in the dependency graph* $\mathcal{G}$*,*

   (a) *Update* a_α*.*

   (b) *If* $\alpha \neq \mu$*, set* $\tau_\alpha \leftarrow (a_{\alpha,old}/a_{\alpha,new})(\tau_\alpha - t) + t$

   (c) *If* $\alpha = \mu$*, generate a random number,* ρ*, according to an exponential distribution with parameter* a_μ*, and set* $\tau_\alpha \leftarrow \rho + t$

   (d) *Replace the old* τ_α *value in* $\mathcal{P}$ *with the new value.*

6. *Go to Step 2.*

Consider the time used by the algorithm. Step 1 of the Next Reaction Method is only executed once; Steps 2-6 are executed once for each simulation event. Steps 3, 4, and 6 do not depend on the number of reactions, r. Step 2 does not either, because of the properties of indexed priority queues. Step 5 is executed once for every edge (μ, α) in $\mathcal{G}$; suppose there are k such edges, where k is typically much less than r. Step 5a, executed k times, depends on the number of reactants for each (elementary) reaction, so it should take no more than 3 multiplications (as was explained in the introduction). Step 5b, executed $k-1$ times, requires an addition, a subtraction, a multiplication and a division. Step 5c, executed 1 time, requires a call to the random number generator, which can be very slow compared to the other operations discussed (a simple test on our system indicates that a single call to the random number generator takes 10 times as long as a division). Step 5d, executed k times, requires *at most* $2\log(r)$ operations, although it may effectively take far fewer (see the discussion in Section 4.2). (Throughout this paper, log means logarithm base 2, as per the typical computer science usage.)

The total number of operations per iteration is at most $c_{2,3,4,5a,6} + c_{5b}(k-1) + c_{5c} + c_{5d}(k)(2\log(r))$, where each c is a machine specific constant. From a computer science perspective, this is $\mathcal{O}(\log(r))$, i.e., for very large r, only the last term will matter. From a more practical perspective, for r of 50 or 100, the other terms, particularly c_{5c}, may not be negligible. Let us be very clear on this point: the Next Reaction Method words even if k is large, but will achieve more of a speedup if k is small, relative to the number of reactions. (An equivalent way of saying the same thing is 'if the dependency graph is *sparse*.')

Lukkien *et al.* [11] discuss ways to improve the Direct Method and the First Reaction Method. Their improved First Reaction Method, which we shall call the Absolute Time First Reaction Method, consists of switching from relative to absolute times and using a standard priority queue. .They conclude that for time-invariant processes, the Direct Method is preferable to the Absolute Time First Reaction Method for two reasons, which do not apply to the Next Reaction Method. First, in their domain it is difficult to do the indexing necessary to implement the efficient update algorithm (Algorithm 4 of Section 4.2). Second, the Absolute Time First Reaction Method generates too many random numbers. Since typical computer pseudo-random number generators cycle with some regularity, using too many random numbers will quickly exhaust the abilities of the generator, and should be avoided with extreme prejudice. (Also, from a purely practical standpoint, generating random numbers is relatively slow.)

Amazingly, the Next Reaction Method uses just a single random number per iteration. Clearly, the optimum would be *exactly* one random number per iteration. Our algorithm is slightly sub-optimal, in that

Reaction	a_μ	*DependsOn*(a_μ)	*Affects*(μ)
$A + B \xrightarrow{k_1} C$	$k_1 \times (\#A) \times (\#B)$	A, B	A, B, C
$B + C \xrightarrow{k_2} D$	$k_2 \times (\#B) \times (\#C)$	B, C	B, C, D
$D + E \xrightarrow{k_3} E + F$	$k_3 \times (\#D) \times (\#E)$	D, E	D, F
$F \xrightarrow{k_4} D + G$	$k_4 \times (\#F)$	F	D, F, G
$E + G \xrightarrow{k_5} A$	$k_5 \times (\#E) \times (\#G)$	E, G	A, E, G

Table 1: Example reactions used in the text.

the initialization step will generate an extra r random numbers, and at the end of the algorithm, r random numbers will be left over. As the number of iterations increases, this initialization effect becomes negligible in comparison. The only new random number generated, τ_μ, corresponds to the reaction that was just executed and is generated in Step 5c. It is clear that reaction μ requires a new random number, since the value of the old random number has been used explicitly, thus reducing it to a sure variable. Section 4.3 will show that it is correct to do the other manipulations in 5, so as not to regenerate any other random numbers.

For this reason we assert that the Next Reaction Method is superior to the Direct Method.

4 Details of the Next Reaction Method

This section presents the details of the Next Reaction Method. First, how to implement the Next Reaction Method: Section 4.1 explains the *dependency graph* data structure, which tells exactly which propensities need to be updated in Step 5 of the Next Reaction Method, and Section 4.2 explains the *indexed priority queue* data structure used in Steps 2 and 5. Second, why the algorithm is correct: Section 4.3 shows it is correct to re-use random numbers in Step 5.

4.1 Dependency Graphs

Consider, once again, the reactions in Eq. (2).

Definition 1 *Let* Reactants(ρ) *and* Products(ρ) *be the sets of reactants and products, respectively, of reaction* ρ*. So, for example,* Reactants*(Reaction 1)* $= \{A, B\}$ *and* Products*(Reaction 1)* $= \{C\}$*.*

Definition 2 *Let* DependsOn*(* a_μ *) be the set of substances which affect the value* a_μ*.*

Evidently, *Reactants*(μ) = *DependsOn*(a_μ). It is sometimes useful to add additional dependencies (e.g., in the lambda model of Section 6.1), so we make this distinction.

Definition 3 *Let* Affects(μ) *be the set of substances that change quantity when reaction* μ *is executed.*

Typically, *Affects*(μ) = *Reactants*(μ) $\cup$ *Products*(μ), but again, there may be exceptions (e.g. catalytic reactions, such as Reaction 3).

Table 1 illustrates each of these concepts.

Definition 4 (Dependency Graph) Let a set of reactions $\mathcal{R}$ be given. Let $\mathcal{G}(V, E)$ be a directed graph with vertex set $V = \mathcal{R}$, and with a directed edge from v_i to v_j if and only if Affects(v_i) $\cap$ DependsOn*(* a_{v_j} *)*$\neq \emptyset$. (If for some strange reason, the self edges from v_i to v_i are not included in this definition, include them as well.) Then $\mathcal{G}$ is called the dependency graph of the set of reactions $\mathcal{R}$.

The dependency graph of the sample reactions is illustrated in Figure 2a.

In other words, a dependency graph is a data structure which tells *precisely* which a_i's to change when a given reaction is executed. Using the dependency graph allows one to recalculate only the minimum number of a_i's in Step 5 of the Next Reaction Method.

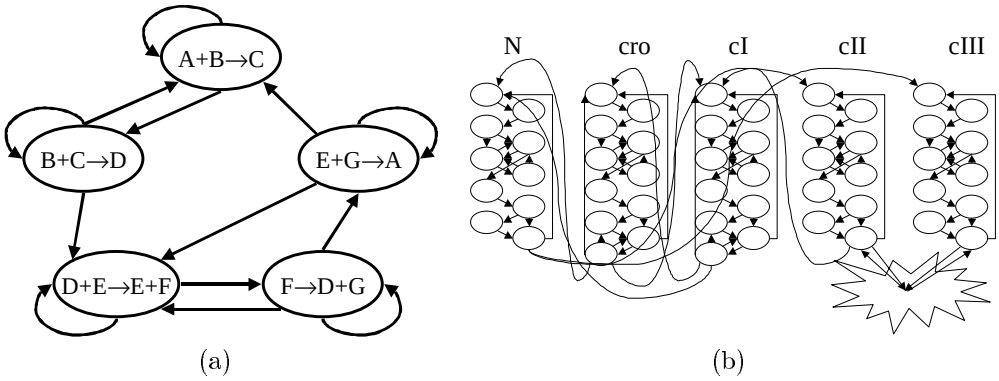


Figure 2: Dependency graphs for: (a) Example equations from Table 1, (b) lambda model (Table 2). For simplicity, self-edges are not shown in (b). The star at the bottom right of (b) indicates a set of reactions that are present in the complete model, but are not described in this paper.

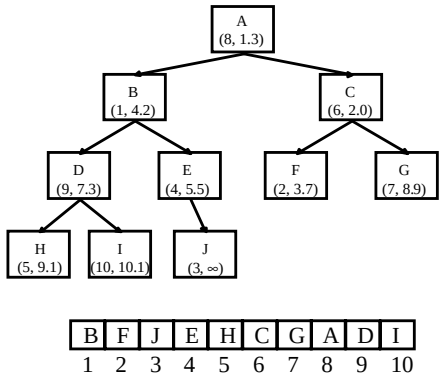


Figure 3: Example indexed priority queue. Top: tree structure. The positions in the tree structure are labeled with letters A–J for pedigogical purposes. Bottom: Index structure. Each number has a pointer to the corresponding position in the tree structure; these pointers are illustrated as letters A–J.

4.2 Indexed Priority Queues

Typically, the dependency graph is *sparse*, i.e., that the number of edges from a given vertex is small. It is important to have data structures that are very efficient at handling *a small number* of updates.

The Next Reaction Method deals with two kinds of variables, τ_i's and a_i's. The latter are easy to handle: the operations required are READ and UPDATE; they can be stored in a simple array. (A purist might not even store them, but rather recalculate them as needed.) The τ_i's require the operations: FIND_MINIMUM (in Step 2) and UPDATE (in Step 5d). The former is one of the operations of a priority queue (which is often implemented as a heap), and with a little thought, the other can be implemented in terms of the standard priority queue algorithms ADD_ELEMENT and DELETE_ELEMENT [3]. However, the standard algorithms, although used in some contexts for this speedup [11], are not really what is called for in this context. A better UPDATE, which takes into account the structure of the data, requires an indexing scheme and a separate UPDATE algorithm.

Definition 5 *An indexed* priority queue *consists of (a) a tree structure of ordered pairs of the form (i, τ_i), where i is the number of a reaction, and τ_i is the putative time when reaction i occurs, and (b) an index structure whose i-th element is a pointer to the position in the tree that contains (i, τ_i). The tree structure in (a) has the property that each parent has a lower τ_i than either of its children.*

Figure 3 shows an example priority queue. Note the following: a) finding the minimum element takes constant time — it is always in the top node, b) the ordering is *only* vertical, not horizontal, c) the number

of nodes is precisely the number of reactions r, not twice the number of reactions as in the efficient version of the Direct Method in the appendix, d) because of the indexing scheme, it is possible to find any arbitrary reaction in constant time, and e) $\tau_3 = \infty$, which corresponds to reaction 3 never occuring, i.e., $a_3 = 0$. In fact, ∞ is a perfectly legitimate floating point number, so it is possible to implement this feature (in C, for example) without any major headaches.

There are several algorithms that need to be defined in order to use the priority queue. Most of them are analogous to algorithms for priority queues. In particular, one needs:

- SWAP(i, j), which swaps the tree nodes i and j *and updates the index structure appropriately*,
- BUILD, which takes a tree and an index structure and moves entries until the tree has the property that each parent is less than its children,
- UPDATE(r), which updates a given reaction number.

SWAP is easy to implement. BUILD is completely analogous to the standard heap/priority queue BUILD operation, but uses SWAP so as to keep the index structure correct. UPDATE is non-standard and deserves comment.

Algorithm 4 *UPDATE(node* ***n****, value* ***new_value****)*
Change value of ***n*** *to* ***new_value***
UPDATE_AUX(***n*** *)*

Algorithm 5 *UPDATE_AUX(node* ***n****)*
*If value(****n****) < value(parent(* ***n*** *))*
SWAP ***n*** *and parent(* ***n*** *)*
Update_aux(parent(***n*** *))*
Else If value(***n*** *) > minimum value(children(* ***n*** *))*
SWAP ***n*** *and minimum child(* ***n*** *)*
Update_aux(minimum child(***n*** *))*
Else
Return

An example is called for.

Example 1 *Suppose the value of τ_1 changes from 4.2 to 16. Looking in the index array, τ_1 is stored in node B. In the tree structure, one changes node B's τ value to 16. Calling UPDATE_AUX on node B, one executes the 'Else If' statement and swaps the ordered pairs in nodes B and E and the corresponding indices (1 and 4) in the array. Calling UPDATE_AUX recursively on node E, one notes that the new value of 16 is in the correct position ($5.5 < 16 < \infty$), so the final 'Else' clause is executed, and the algorithm stops with: ordered pair (4, 5.5) in node B, (1, 16) in node E, index 'E' in position 1 of the array, and index 'B' at position 4. The rest of the structure remains unchanged.*

The converse case, where the new value is less than the old value, is completely analogous.

One way to implement UPDATE is simply to delete the offending node, and insert a new node with the same reaction number but a different time value. This takes something like $2 \log r$ operations. Our approach, on the other hand, changes the node in place, then bubbles it up or down the tree structure until the priority property is re-established. This evidently takes $\log r$, but has the advantage that if there are a small number of reactions that have fast rate constants compared to the others, say there are r' such reactions, most of the updates will involve those, and take $\log r'$ time, because once the algorithm reaches a node that is already in the right spot, it does not continue further. So, for example, if some of the reactions are "disabled" or "not possible" in the given state, and have $a = 0$ and $\tau = \infty$, they will not slow down the computation. This effect can be significant, for example, the chemotaxis system of [13] contains a large number of reactions which will not be "active" at any given time. Because of these inactive reactions, [13] avoided the standard Gillespie algorithm (i.e., the Direct Method), which grows with the number of reactions, and instead developed one that is not exact, but scales with number of molecules, since the number of molecules is much less than the number of possible reactions. Note that our algorithm is not only exact, but also scales with the *logarithm* of the number of "*active*" reactions. The Application section (Section 6) gives some performance numbers.

4.3 Reusing τ_i's

This section will demonstrate that the Next Reaction Method, with its switch from relative to absolute times and all of the strangeness in Step 5, is equivalent to the First Reaction Method. This demonstration, necessary to show that the algorithm works and why it works, is somewhat more mathematical than the rest of the paper. The reader whose primary interest is implementing the Next Reaction Method may skip ahead with impunity.

In what follows, T_i will denote the *random variable* corresponding to the i-th reaction, and τ_i, a number, will denote a *sample* from that random variable.

One of the differences between the First Reaction Method and the Next Reaction Method is that the former uses relative times, while the latter uses absolute times. This should not be a stumbling block or a source of confusion. Suppose that during the n-th iteration of the First Reaction Method, the random variables are denoted R_α, for $1 \leq \alpha \leq$ (number of reactions). Then $R_\alpha =$Exp(a_α), and the density of R_α is given by $P_{R_\alpha}(\tau) = \theta(\tau) a_\alpha \exp(-a_\alpha \tau)$. ($\theta(\tau)$, the Heaviside function, is 0 for $\tau' < 0$ and 1 for $\tau' \geq 0$.) The corresponding absolute time is given by the random variable $T_\alpha = R_\alpha + t_n$, the sum of the relative time and the variable t during the n-th iteration. (The n-th iteration ends, and the $n+1$-st begins, when t is updated in Step 5.) What is the density of T_α? By the random variable transformation (RVT) theorem [8],

$$P_{T_\alpha}(\tau) = \int_{-\infty}^{\infty} P_{R_\alpha}(\tau')\delta(\tau - [\tau' + t_n])d\tau' = P_{R_\alpha}(\tau - t_n) = \theta(\tau - t_n) a_\alpha \exp(-a_\alpha(\tau - t_n)),$$

or, equivalently,

$$\Pr(T_{\alpha,n} > u) = \begin{cases} \exp(-a_{\alpha,n}(u - t_n)) & \text{if } u > t_n \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

Clearly, an absolute-time version of the First Reaction Method with no other changes would be entirely equivalent to the original relative-time version.

Now we turn our attention to the Next Reaction Method. After Step 1, the random variables follow the distribution in Eq. (7); t_0 was set to 0 in Step 1a. The real core of the Next Reaction Method is that each subsequent iteration maintains Eq. (7).

At the risk of being overly mathematical, we state the key property that allows the Next Reaction Method as a theorem:

Theorem 1 *Assume that Eq 7 holds at the beginning of Step 2. Then, before Step 5 of the n-th iteration, for all $i \neq \mu$, τ_i is distributed according to*

$$\Pr(T_i > u) = \begin{cases} \exp(-a_{i,n}(u - t_{n+1})) & \textit{if } u > t_{n+1} \\ 1 & \textit{otherwise} \end{cases} \tag{8}$$

Proof. By assumption, before Step 2 of the n-th iteration, τ_i is distributed according to Eq. (7). Steps 2 and 3 identify the least τ, namely τ_μ. The act of identification reduces uncertainty. In particular, T_μ becomes the sure variable τ_μ, and all of the other τ_i's must be larger than τ_μ. Hence each of the other T_i's is distributed according to $\Pr(T_i > u | T_i > \tau_\mu)$. By definition, this is $\Pr((T_i > u) \text{ AND } (T_i > \tau_\mu))/\Pr(T_i > \tau_\mu)$. There are two cases. Case 1) for $u > \tau_\mu$, the numerator simplifies to $\Pr(T_i > u)$, and the resulting division is $\exp(-a_{i,n}(u - t_n))/\exp(-a_{i,n}(\tau_\mu - t_n)) = \exp(-a_{i,n}(u - \tau_\mu))$. Case 2) $u \leq \tau_\mu$, and the numerator simplifies to $\Pr(T_i > \tau_\mu)$. In this case, the numerator cancels the denominator, leaving 1. In Step 4, t_{n+1} is set to τ (which was set to τ_μ in Step 3), so the theorem holds. ■

Showing that Eq. (7) is maintained is just a matter of collecting the details:

- For those $i \neq \mu$ whose a_i remains constant from the n-th to $n+1$-st iteration, $a_{i,n+1} = a_{i,n}$, so Eq. (8) is equivalent to Eq. (7). There is no need to change these τ_i's in Step 5. In fact, reactions whose a_i does not change are not in the dependency graph, so the τ_i's are not changed.

- For those $i \neq \mu$ whose a_i *does* change, τ_i is now distributed according to Eq. (8). Simply plugging in to the RVT theorem shows that the random variable T_i', constructed by $\tau_i' = (a_{i,n}/a_{i,n+1})(\tau_i - t_{n+1}) + t_{n+1}$, is distributed according to Eq. (7).

  What is the intuition behind this transformation? By the theorem, τ_i is distributed according to Eq. (8). Going back to relative times, $\tau_i - t_{n+1}$ is distributed according to $\text{Exp}(a_{i,n})$. It can be shown (by the RVT, for example) that $(a_{i,n}/a_{i,n+1})\text{Exp}(a_{i,n}) = \text{Exp}(a_{i,n+1})$. Returning to absolute times gives the transformation.

  This transformation is applied to all the appropriate i's in Step 5b.

- Finally, for $i = \mu$, it is necessary to generate a new random number. Note that the theorem only holds for $i \neq \mu$. For $i = \mu$ the variable T_μ was reduced to a sure variable in Step 3, so a new random variable is needed. That new value is supplied in Step 5c.

One key point has been overlooked thus far: the First Reaction Method requires *statistically independent* random numbers. To complete the correctness argument amounts to showing that the manipulations done by the Next Reaction Method do not introduce any statistical dependencies.

At each step in the algorithm $T_i = f_i(R_i)$; each random variable in the Next Reaction Method is a transformed version of the corresponding random variable in the First Reaction Method, and there are no cross dependencies. By the RVT theorem,

$$\begin{aligned} P_{T_1 \cdots T_N}(\tau_1, \ldots \tau_N) &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \left\{ \prod_{i=1}^{N} \theta(r_i) \exp(-a_i r_i) \right\} \prod_{j=1}^{N} \delta\left(\tau_j - f_j(r_j)\right) dr_1 \cdots dr_N \\ &= \prod_{i=1}^{N} \left\{ \int_{-\infty}^{\infty} \theta(r_i) \exp(-a_i r_i) \delta\left(\tau_i - f_i(r_i)\right) dr_i \right\} \end{aligned}$$

The "product form" of this joint distribution function tells us that since the *original* variables R_i were statistically independent, then the *transformed* variables T_i are as well.

In summary, Step 1 sets up the T_i according to Eq. (7). Each subsequent iteration maintains that distribution, without introducing any statistical dependencies between the random variables. Thus the Next Reaction Method is equivalent to the First Reaction Method, and in turn to the Direct Method and the Master Equation approach.

5 Extension: Time Dependent and non-Markov Processes

The two approaches for efficient calculation of trajectories of chemical reactions in the stochastic framework presented thus far assume that the probability of a reaction μ occurring in a little bit of time dt (a) is given by $a_\mu \times dt$, where a_μ is a constant, and (b) depends only on the current state, not on the previous state or states of the system. It can be shown [7] that many reasonable chemical systems have these properties.

This section will show how to deal with systems in which (a) and (b) do not hold. In particular, it first relaxes assumption (a) by letting a_μ be a function time (as is necessary to model systems whose rate "constants" change, due to changing temperature, volume, etc.), and second it relaxes assumption (b), showing how to deal with non-Markov processes. Even though *elementary* reactions in the stochastic framework are Markov (i.e., have property (b)), it is sometimes useful to group consecutive steps to form a composite process. The full model of that process is, of course, still Markov, but if one is only interested in a subset of the variables, the resulting mathematical process is not guaranteed to be Markov.

5.1 Markov processes

Consider a system in which the probability of a reaction μ occurring in a little bit of time dt is given by $a_\mu \times dt$, *but* a_μ *is a function of time.* For now, assume that the transition probabilities do not depend on history, but only on the current state. For example, in Table 1, the first reaction has $a_1 = k_1 \times (\#A) \times (\#B)$. The rate constant k_1 is a function of temperature and of volume. In an engineering system, one typically

affects the rate constants by heating or cooling the reaction. In a biological system, cell growth changes the volume. Either of these mechanisms, or others, might change rate constants as a function of time, which requires a modification to the algorithms presented.

In place of the simple exponential distribution, the putative times are distributed [8] according to

$$P_\mu(\tau|S,t_n) = a_\mu(S,\tau)\exp\left(-\int_{t_n}^{\tau} a_\mu(S,t)dt\right) \tag{9}$$

Notice two things: first, it is not easy in general to find a closed form solution of Eq. (9) for arbitrary functions of time a_μ, and second, for non-constant a_μ, the resulting answer will not be a simple exponential distribution, and hence will not have the temporal homogeneity property that $\Pr(T_i > u|T_i > \tau_\mu) = \Pr(T_i > u - \tau_\mu)$. As a consequence of the latter, it will not, in general, be possible to let $t_0 = 0$, so absolute times should be used.

5.1.1 How to do it: Next Reaction Method, Markov processes

To extend the Next Reaction Method to arbitrary Markov processes, one simply changes Step 3 to generate τ_i according to the new process. This has two advantages over the time variant version of the Direct Method in the appendix:

- Since one considers each reaction separately, the computation is easier, and may be analytically solvable for some processes (e.g. in the example of the next section). The Direct Method, which considers all reactions at once, involves a sum within the integral in Eq. (9); the exact form is given in the appendix.
- Conditioning. Since the Next Reaction Method stores τ_i's and not just a_i's, it does not have to recondition on each iteration. Specifically, after executing reaction μ it does not need to re-generate τ_i according to $\Pr(T_i > u|T_i > \tau_\mu)$: the fact that the algorithm chose to execute reaction μ implies $T_i > \tau_\mu$. Therefore T_i is already distributed according to the correct distribution, for all $i \neq \mu$ whose a_i has not changed.

We now show an example of how to generalize the Next Reaction Method for time varying Markov processes, then consider the problem of re-using random numbers with the generalization.

5.1.2 Example: changing volume

Reaction 1 in Table 2 has propensity $k \times (\#A) \times (\#B)$. For second order reactions, such as this one, the k term depends on the volume, and should be replaced with $k'/V(t)$, where $V(t)$ is the volume and k' is independent of volume. This leads to the propensity $a'/V(t)$, where $a' = k' \times (\#A) \times (\#B)$ is a constant independent of volume (and hence time). For simple $V(t)$, Eq. (9) can be solved analytically, for example, in Arkin *et al.*, the volume is modeled as increasing linearly. Thus $V(t) = V_0 + ct$, which leads (by a simple integration) to the distribution

$$P(t|t_0) = \frac{a'\left(V(t_0)+ct\right)^{-a'/c-1}}{V(t_0)^{-a'/c}} \tag{10}$$

Note also that in the limit as c goes to zero, this distribution reduces to an exponential with parameter a'/V_0, as expected.

It is a straightforward operation to generate random numbers according to this distribution, using the inversion generating method [8, 10]: one calculates the cumulative distribution function F (a simple integral of P), takes a sample U from a uniform random number generator, and the variable $F^{-1}(U)$ has the correct distribution. For the preceeding example, the variable $R = V(t_0)[U^{-c/a'} - 1]/c$ is distributed according to Eq. (10).

5.1.3 Generating fewer random numbers

It was remarkably simple in the time-independent, exponential case to reuse the same random numbers. The extension is somewhat more difficult. The method presented here works for reusing random variables generated by the inversion generating method. Of course, not every random variable is generated that way, and in practice it may be hard to reuse other random numbers.

Theorem 2 *Let τ be a random number generated according to an arbitrary distribution with parameter a_n and distribution $F_{a,n}$. Suppose the current simulation time is t_n, and the new parameter (after the update in Step 2) is a_{n+1}. Then the transformation $\tau' = F^{-1}_{a,n+1}([F_{a,n}(\tau) - F_{a,n}(t_n)]/[1 - F_{a,n}(t_n)])$ generates a new random variable from the correct (new) distribution.*

The proof of this theorem is in Appendix 8.1. Here are some examples:

Example 2 *For exponentials,*

$$F_{a,n}(u) = \Pr(T_n \le u) = \begin{cases} 1 - \exp(-a_n(u - t_{n-1})) & \text{if } u > t_{n-1} \\ 0 & \text{otherwise} \end{cases}$$

and

$$F^{-1}_{a,n}(U) = \begin{cases} -\ln(1-U)/a_n + t_{n-1} & \text{if } 0 \le U \le 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

So, by the theorem, $\tau' =$

$$\begin{aligned} & F^{-1}_{a,n+1}([F_{a,n}(\tau) - F_{a,n}(t_n)]/[1 - F_{a,n}(t_n)]) \\ = \; & \frac{-1}{a_{n+1}} \ln\left(1 - \frac{[1 - \exp(-a_n(\tau - t_{n-1}))] - [1 - \exp(-a_n(t_n - t_{n-1}))]}{1 - [1 - \exp(-a_n(t_n - t_{n-1}))]}\right) + t_n \\ = \; & (a_n/a_{n+1})(\tau - t_n) + t_n \end{aligned}$$

This is the transformation used by the Next Reaction Method.

Example 3 *The previous section considered a process with $V(t) = V(t_{n-1}) + c(t - t_{n-1})$,*

$$F_{a,n}(u) = 1 - \left(\frac{V(t)}{V(t_{n-1})}\right)^{-a_n/c}$$

and

$$F^{-1}_{a,n}(U) = V(t_{n-1})[(1-U)^{-c/a_n} - 1]/c + t_{n-1}$$

By the theorem, one can reuse random numbers by the transformation

$$\begin{aligned} \tau' \quad = \quad & F^{-1}_{a,n+1}([F_{a,n}(\tau) - F_{a,n}(t_n)]/[1 - F_{a,n}(t_n)]) \\ = \frac{V(t_n)}{c} \quad & \left[\left(1 - \frac{\left(1 - \left(\frac{V(\tau)}{V(t_{n-1})}\right)^{-a_n/c}\right) - \left(1 - \left(\frac{V(t_n)}{V(t_{n-1})}\right)^{-a_n/c}\right)}{\left(\frac{V(t_n)}{V(t_{n-1})}\right)^{-a_n/c}}\right)^{-c/a_{N+1}} - 1\right] + t_n \\ = \quad & \frac{V(t_n)}{c}\left[\left(\frac{V(\tau)}{V(t_n)}\right)^{a_n/a_{N+1}} - 1\right] + t_n \end{aligned}$$

In some cases, as in the examples above, it is possible to calculate a closed form solution of the equations that is relatively simple, so this method is practical. In general, it may not be at all practical, and it may be easier to generate fresh random numbers.

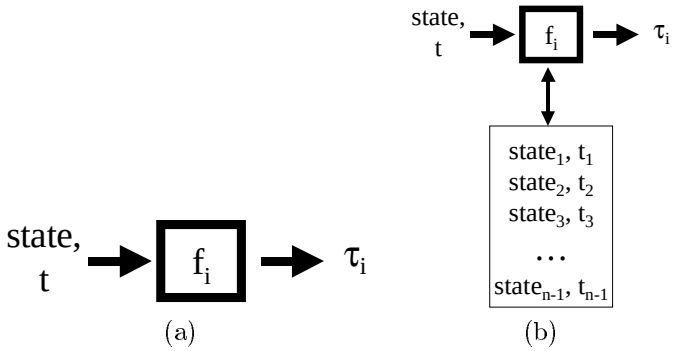


Figure 4: Procedures, f, for generating random numbers. (a) A Markov process, in which no history is stored. (b) A non-Markov process, which requires storing history.

5.2 Non-Markov processes

Even though elementary reactions are Markov (i.e., do not depend on history) in the stochastic framework, it is sometimes useful to deal with non-Markov.processes. For example, one may model a system using a certain set of variables for which the system is Markov, or one may use a smaller set for which the system is not. Provided the simulation algorithm still works for non-Markov processes, a smaller number of variables may be significantly faster to simulate.

For Markov processes, one generates τ_i directly from the state and the time, as in Figure 4a. For example, in the time independent case, the value a_i is calculated from the state, and τ_i is the sum of t and a random variable with exponential distribution and parameter a_i. Notice that 1) f_i is a *random* function, i.e., calling it multiple times with the same parameters will give multiple answers, and 2) f_i is a *function* in the mathematical usage (or in the computer science sense of functional programming), i.e., it does not contain any internal state. For non-Markov processes, as in Figure 4b, the distribution of τ_i depends on the history of (possibly all) states of the system from the initial time to the present. Hence one must use a *procedure* (in the computer science sense of procedural programming), which can store previous values of the system state and time.

In general, non-Markov processes are very difficult to deal with [8]. The distribution of next states, or of transition times to the next state, may depend on the entire history of the system. Fortunately, the sort of non-Markov processes that occur in chemical reaction simulations have some nice properties that make them easier to deal with. First, the complete history of the system is uniquely determined by the series of *discrete* transitions and transition times. Given the transitions and transition times, the state at any time t is the same as the state at the last transition before t. This is an enormous simplification: since continuous transitions are not possible, one can hope to store the entire state history. Second, one may not even need the entire history. For any given reaction μ, one only needs to store that fraction of the history that affects (in the dependency graph sense) the reaction μ. For systems with many reactions, that leads to another significant reduction in the amount of storage.

For *arbitrary* non-Markov chemical reaction models, even this reduction in storage may not be enough. It may be very difficult to generate τ_i given the appropriate subset of history. In that case, it may be preferable to include the full gamut of variables so as to make the system Markov. In those special cases where it *is* possible to generate τ_i, one may achieve a substantial performance improvement. An example follows.

5.2.1 Example: Gamma Distribution

Consider the set of equations:

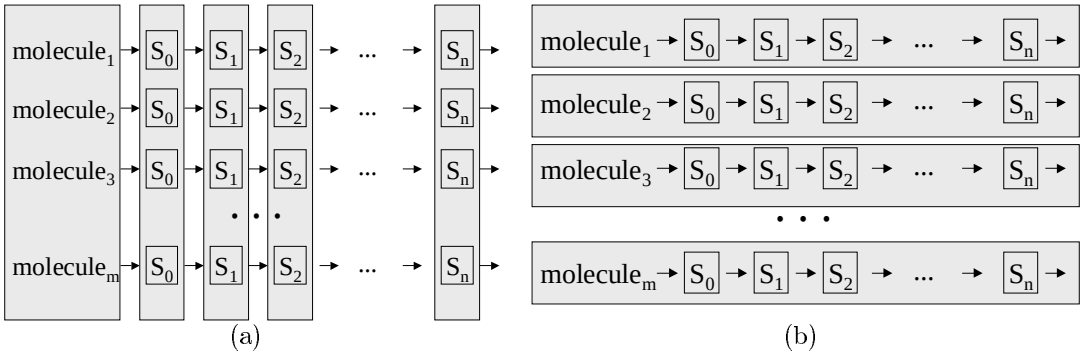


Figure 5: Ways of splitting reactions. (a) By state, (b) by molecule.

$$
\begin{aligned}
A + B &\xrightarrow{k_1} S_0 \\
S_0 &\xrightarrow{k} S_1 \\
S_1 &\xrightarrow{k} S_2 \\
&\cdots \\
S_{n-1} &\xrightarrow{k} S_n \\
S_n &\xrightarrow{k_2} C + D
\end{aligned}
$$

Systems of equations very much like this comes up in the Arkin *et al.* [1] model of lambda phage, both for transcription and for translation. Physically, this means that at some time a molecule of type S_0 is produced, it undergoes an n-step process, and then the resulting molecule, S_n, affects the rest of the system.

The first and last equations are different, but all the n intervening equations are identical. Assuming time-independence (as is the case in the model; first order reactions are not affected by change in volume) one may solve these n equations analytically. Rather than n exponentials, the combined waiting time is a gamma distribution. Specifically, consider a single molecule of S_0, produced at t_0, with no other molecules of S_0 produced. Then:

$$
\begin{aligned}
& \Pr(\text{one molecule of } S_n \text{ is produced at } t \mid \text{one molecule of } S_0, t_0) \\
= \quad & \frac{k[k(t-t_0)]^{n-1}}{(n-1)!} \exp[-k(t-t_0)] \qquad (11)
\end{aligned}
$$

This is simply a gamma distribution, and there are efficient ways to generate random numbers according to this distribution.

Now consider several molecules undergoing this process. This composite system can be described by ordered pairs of the form (molecule identity, state). There are to ways to simplify this: grouping by state and grouping by molecule identity. Thus far, the grouping has always by state, i.e., the number of molecules in state S_0, the number in state molecules of S_1, etc.

For the context in which this problem occurs, with many more states than molecules, one achieves a smaller system by grouping by molecule identity; by Eq. (11), one can simplify the n-step exponential process into a 1-step gamma process, thus the number of processes to consider is equal to the number of distinct molecules, much less than the number of states. (Note that this is possible because the reactions involved are first order.) These two possibilities are schematized in Figure 5.

The procedure f_i is as follows: rather than store state and time directly, f_i will keep a list L of processed values τ'. Every time a new molecule of S_0 is produced, f_i generates a τ' value for it according to Eq. (11) and adds that value to L. The value of τ_i' that f_i returns is simply the minimum of the τ values in L. (The astute reader will note that the operations required on L are insert, delete and minimum, so one could

$\mathrm{RNAP_{free}+DNA_{free}}$	$\xrightarrow{k_1}\mathrm{RNAP\bullet DNA_{open,\ 0}}$	
$\mathrm{RNAP\bullet DNA_{open,\ n}}$	$\xrightarrow{k_2}\mathrm{RNAP\bullet DNA_{open,\ n+1}}$	for $\mathrm{n} = 0 \ldots DMAX - 1$
$\mathrm{RNAP\bullet DNA_{open,\ DMAX}}$	$\xrightarrow{k_3}\mathrm{RNAP_{free}+DNA_{free}+RNA_{free}}$	
$\mathrm{RNase+RNA_{free}}$	$\xrightarrow{k_4}\mathrm{RNase}$	
$\mathrm{Ribosome+RNA_{free}}$	$\xrightarrow{k_5}\mathrm{Ribosome\bullet RNA_0}$	
$\mathrm{Ribosome\bullet RNA_n}$	$\xrightarrow{k_6}\mathrm{Ribosome\bullet RNA_{n+1}}$	for $\mathrm{n} = 0 \ldots RMAX - 1$
$\mathrm{Ribosome\bullet RNA_{RMAX}}$	$\xrightarrow{k_7}\mathrm{Ribosome_{free}+RNA_{free}+protein}$	
protein	$\xrightarrow{k_8}$no protein	
protein+protein	$\xrightarrow{k_9}$protein$\bullet$protein	
protein$\bullet$protein	$\xrightarrow{k_{10}}$protein+protein	

Table 2: 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 $\bullet$ protein" is the dimer form of a given protein, typically written as "$\mathrm{protein}_2$."

implement L as a priority queue. This implementation of L as a priority queue should not be confused with the indexed priority queue in Section 4.2.)

In the the current context, the transcription or translation lengths, and hence the n values, may be in the hundreds or even thousands, so this enhancement achieves quite a speedup (see Section 6).

6 Application

The Arkin *et al.* [1] model of the bacteriophage lambda provides a good, large, test case for the scalability of stochastic simulation algorithms. The details of the model can be found in the original paper; this paper shall just sketch out the model and focus on how the algorithmic improvements above apply to it.

6.1 The Model

The model consists of the complete set of equations governing the regulation of five genes in the temperate bacteriophage lambda. The key reactions are summarized in Table 2. The model considers five genes, *N*, *cro*, *cI*, *cII*, and *cIII*, and their protein products. There are five sets of equations similar to the ones in Table 2. Two of the five genes have protein products that dimerize, i.e., can exist as a $\mathrm{protein}_2$ form, as in Table 2, Eqs. (9) and (10). The other three proteins exist only as monomers, so Eqs. (9) and (10) do not apply to those proteins.

The key phenominon 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. This is the sort of extra dependency that was alluded to in the definition of $DependsOn(\ a_\mu\ )$. 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 [14] for a complete description of how to calculate the k_1 values.

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 (see the discussion in Section 5.1.2).

The dependency graph of these reactions is shown in Figure 2b.

6.2 Results

Table 3a shows performance data for the Next Reaction Method, with all the enhancements of Section 5, on the (full) Arkin *et al.* model. Note that the average out-degree — the average number of edges from a given

Reactions	78
Average Out-Degree	4.2
Trajectories Generated	500
Simulation Events	35,000,000
Updates	206,000,000
Operations	760,000,000
Operations/Event	22
Operations/Update	3.7

(a)

Gamma Reactions	21
Equivalent Elementary Reactions	10,000
Gamma Simulation Events	380,000
Equivalent Elementary Events	99,000,000

(b)

Table 3: Performance data for simulation. a) General information:. Average out-degree = (number of out-edges in dependency graph)/(number of vertices in dependency graph), Updates = number of executions of Step 5. b) Information about the gamma distribution enhancement.

vertex in the dependency graph — is significantly lower than the number of reactions, so the dependency graph is sparse, as mentioned in 4.1. Five hundred trajectories were generated for the condition '1 phage per cell;' each required about 70,000 simulation events.

Based on the number of simulation events and the average out degree, one would expect 150 million updates — one update per out edge per simulation event. The actual number is substantially higher at 206 million. This comes about because certain reactions are executed much more often than others. Under the conditions used (one phage per host cell), the two most frequent reactions constitute nearly 90% of all simulation events. Both of these most-frequent reactions have out-degree 6, which explains why the number of updates per simulation event is approximately 6, not 4.2 as would be expected.

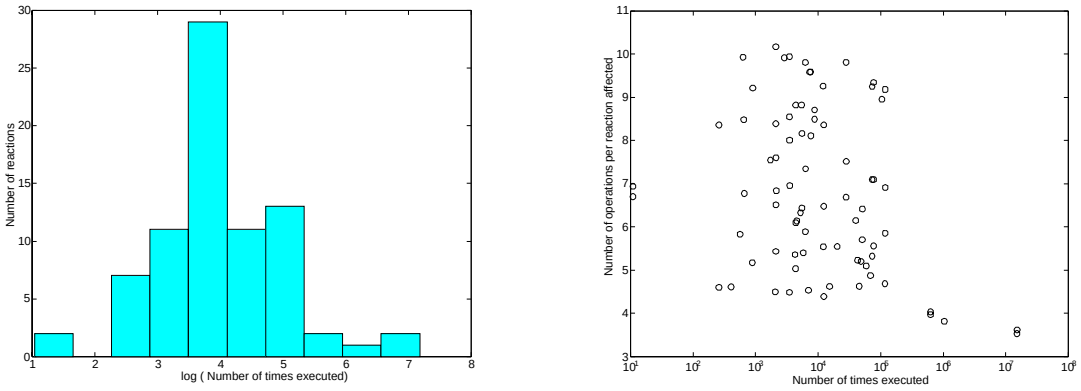
Which reactions are executed most often? By definition, those reactions with high propensities are more likely to be executed. Unfortunately, since propensity is a function of rate constant *and of state*, it is very difficult to calculate which reactions will occur most frequently without just running the whole simulation. (For example, by changing the conditions to 'six phages per host cell,' with all the same rate constants, two additional reactions occur with approximately the same frequency as the original two. Under these conditions, these four most likely reactions are responsible 95% of the the total of 180 million simulation events.) It is not obvious whether there is a way to identify such reactions and deal with them separately (e.g., by separation of time scales) *without affecting the results.*

Fortunately, as Figure 6b shows, the optimizations mentioned have the effect that the most frequently executed reactions are also the most efficient. Note that each time a reaction is executed and a new τ_i has to bubble down, at least two comparison operations are required - namely comparing with the left and right children. (Bubbling up only requires a single comparison.) The fact that the most frequently executed reactions have efficiencies around 3 and the overall total is 3.7 means that the updated τ_i's do not typically move very far from their original positions in the indexed priority queue, as mentioned in Section 4.2.

Also notice that by re-using random numbers, the number of calls to the pseudo random number generator is approximately equal to the number of simulation events, not to the number of updates. This decreases the number of random number generator calls 6 fold from the Absolute Time First Reaction Method in [11]. Equivalently, one can simulate 6 times as much without having to worry about numerical problems.

Another big time savings comes from using the gamma distribution, as detailed in Table 3b for the condition 'one phage per host cell.' One could, in principle, write the composite reactions as their elementary steps, however there would be nearly 500 times more reactions under that scheme. Furthermore, that would involve 260 times the number of simulation events — nearly three times the *total* number of simulation events — just for the gamma reactions. The total simulation would take four times as long without this particular refinement. Thus, using the gamma distribution provides enormous savings: all 21 gamma reactions together were only 1% of the simulation events.

Table 4 compares a simple implementation of Gillespie's Direct Method with our Next Reaction Method (including the gamma optimization) for the lambda model. All numbers are rounded. The simple implementation bogs down in Step 3: it calculates the cumulative sums a_1, a_1+a_2, $a_1+a_2+a_3$, $a_1+a_2+a_3+\ldots+a_{78}$, generates a random number between 0 and the sum $a_1+a_2+a_3+\ldots+a_{78}$, and then uses a binary search to choose the appropriate μ. This takes $78+\log_2(78)$ operations per simulation event. Also, the Direct



(a) (b)

Figure 6: Performance data for lambda model. (a) Number of times each reaction is executed. (b) Number of update operations per executed reaction.

	Direct Method	Next Reaction Method
Operations	11,300,000,000	760,000,000
Random Numbers	268,000,000	35,000,000

Table 4: Comparison of Gillespie's Direct Method with the fully optimized Next Reaction Method. "Operations" encompasses additions, subtractions, and comparisons.

Method only deals with elementary reactions, so the composite reactions discussed above must be broken into their components, increasing the number of simulation events. (We have assumed that one can keep track of which of those reactions are allowed at any given time, so the total number of reactions remains 78, not 10,078.) Finally, recall that the Direct Method uses two random numbers per simulation event, whereas the Next Reaction Method uses one. The Direct Method requires roughly 15 times as many operations and 7 times as many random numbers as the Next Reaction Method.

7 Conclusions

This paper presents an exact, efficient way to do calculations for large systems of loosely coupled reactions in the stochastic framework.

The algorithms presented are *exact*, i.e., provably equivalent to the chemical Master Equation approach, and they are *efficient*, both in running time and in number of random numbers generated. For both the Direct and the Next Reaction Method, the amount of time required per iteration is proportional to the logarithm of the number of reactions, not the number itself. The number of random numbers generated by the Next Reaction Method is $(reactions)+(simulation\ events)$, only slightly higher than the optimal number, $simulation\ events$. (For comparison, the Direct Method takes $2\times simulation\ events$.)

The paper extends the Next Reaction Method to time varying rate constants while maintaining both types of efficiency. As an aside, the appendix shows how to extend the Direct Method to time varying rate constants, but it is not clear whether this is useful.

The paper provides one specific example of extending of the Next Reaction Method to non-Markov processes, and sketches a framework in which other examples might be formulated.

For loosely coupled chemical reaction systems in solution, the Next Reaction Method is preferable to the Direct Method for a number of reasons:

- although the efficiency of updates in each is $O(\log r)$, if some reactions are much faster than others, the Next Reaction Method may be effectively $O(\log r')$, where $r' << r$

- the Next Reaction Method can easily be enhanced to reuse random numbers; this reduces the number of random numbers to 1/2 as many as the Direct Method uses
- the Next Reaction Method is easily extended to time-varying processes, both Markov and non-Markov

Finally, the paper presents an example of the performance of these algorithmic improvements on a test case from the biology literature.

8 Appendix

8.1 Proofs

Proof. (Theorem 2) The random number τ is orginally distributed according to distribution $F_{a,n}$, with density $P_{a,n}$. After Step 6, it is distributed according $P'_{a,n} = \Pr_{a,n}(T = u | T > t_n)$, which is equal to

$$P'_{a,n}(u) = \begin{cases} P_{a,n}(u)/[1 - F_{a,n}(t_n)] & \text{if } u > t_n \\ 0 & \text{otherwise} \end{cases}$$

By the Random Variable Transform Theorem [8], the random variable $Y = [F_{a,n}(\tau) - F_{a,n}(t_n)]/[1 - F_{a,n}(t_n)]$ has the density

$$\begin{aligned} Q(y) &= \int_{-\infty}^{\infty} P'_{a,n}(u)\delta\left(y - \frac{F_{a,n}(u) - F_{a,n}(t_n)}{1 - F_{a,n}(t_n)}\right) du \\ &= \frac{1}{1 - F_{a,n}(t_n)} \int_{t_n}^{\infty} P_{a,n}(u)\delta\left(y - \frac{F_{a,n}(u) - F_{a,n}(t_n)}{1 - F_{a,n}(t_n)}\right) du \\ &= \frac{1}{1 - F_{a,n}(t_n)} \int_{F_{a,n}(t_n)}^{F_{a,n}(\infty)=1} \delta\left(y - \frac{v - F_{a,n}(t_n)}{1 - F_{a,n}(t_n)}\right) dv \\ &= \int_0^1 \delta(y - w)\, dw \\ &= \begin{cases} 1 & 0 < y \leq 1 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

The second line is just the definition of $P'_{a,n}(u)$. The third line comes from the transformation $v = F_{a,n}(u)$, $dv = \frac{dF_{a,n}(u)}{du} du = P_{a,n}(u)du$. The fourth line comes from the transformation $w = \frac{v - F_{a,n}(t_n)}{1 - F_{a,n}(t_n)}$, $dw = \frac{1}{1 - F_{a,n}(t_n)} dv$. The final line comes from the definition of the delta function.

Hence, the random variable Y is distributed uniformly on $(0, 1]$. Finally, the inverse generation method works by transforming a uniform random number U and to $F^{-1}(U)$. Here Y is such a uniform random number, which proves the theorem. ■

8.2 Enhancing the Direct Method

The underlying ideas of the Next Reaction Method — use a dependency graph to update the minimal number of variables, and use an efficient data structure — can be applied to the Direct Method as well.

Algorithm 6 *(Efficient Exact Stochastic Simulation — Direct Method)*
Replace Steps 1 and 2 of the Direct Method with:

1. *Initialize(i.e., set initial numbers of molecules, set $t = 0$, generate a dependency graph $\mathcal{G}$)*
2. *Calculate the propensity function, a_i, for the following i:*

   If this is the initial iteration, calculate a_i for all i.

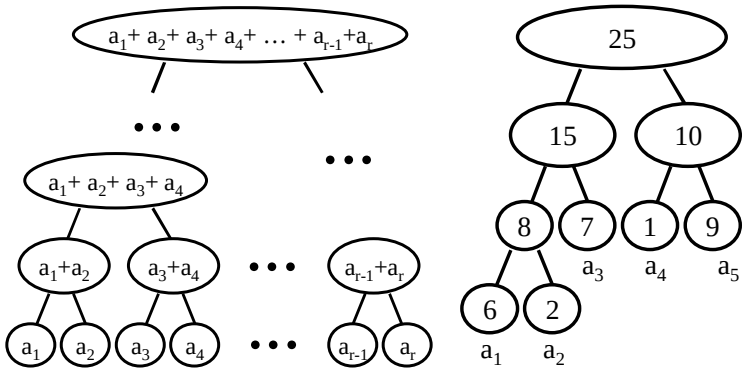


Figure 7: Data structure used for a_i's for an efficient version of the direct method. Each leaf contains an a_i value. Each other node contains the sum of its left and right child. (a) generic construction, (b) numerical example used in text.

Otherwise, let μ be the reaction that was just executed. For each edge (μ, α) in the dependency graph $\mathcal{G}$, update a_α

It is clear from the definition of the dependency graph that this will update only the a_i's that need to be updated. The only thing remaining is to use the right data structure to speed up the updates.

To complete the speed up of the Direct Method, one must do Steps 2, 3 and 4 efficiently. One might consider using a simple array to store each of the a_i's. In this scheme, updates would be very fast. However, Step 3 of the Direct Method would then take time proportional to the number of reactions. A better data structure, which takes time proportional to the logarithm of the number of reactions, follows.

Store the a_i's as the leaves of a complete tree, and store in each non-leaf node the sum of its left child and right child (see Figure 7a). Thus, the root will have value $\sum_i a_i$. When a_i's change, update 1) those a_i's that have changed and 2) their ancestors. Notice that the tree contains r leaves and $r/2 + r/4 + r/8 + ... + 1 \cong r$ non-leaves. The height of the tree is simply $\log 2r = 1 + \log r$. Each update affects one node at each level, hence is $\mathcal{O}(\log r)$.

Generating the random numbers τ will be easy, since the root of the tree contains the appropriate parameter. For the μ value, generate a random number x between 0 and $\sum_i a_i$ (which can be found at the root) and then use the following algorithm, starting at the root:

Algorithm 7 (Efficient Uniform Random Number Generation)

1. If the current node is a leaf, let μ be its index.
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).

The discussion thus far has been somewhat abstract and calls for an example.

Example 4 *Consider the numerical values in Figure 7b, the tree structure for $a_1 = 6$, $a_2 = 2$, $a_3 = 7$, $a_4 = 1$ and $a_5 = 9$. To calculate τ, generate an exponential random variable with parameter 25. For the μ value, generate a random number x between 0 and 25. Suppose $x = 15.5$. Because $x > 15$ (the left child), go to Step 3 of the algorithm, and descend the right subtree (i.e., the one whose root is "10") with parameter $x' = x - 15 = 0.5$. Now, $0 \leq x' \leq 1$, so use Step 2 of the algorithm, and go to the left node, labeled "1". Finally, this is the leaf corresponding to a_4, so stop and let μ be "4."*

One standard method of generating a random number of this distribution is to generate a random number R between 0 and 1, then find the index μ such that $\sum_{i=1}^{\mu-1} a_i \leq R(\sum_i a_i) < \sum_{i=1}^{\mu} a_i$. In fact, Algorithm 7 does precisely that, in an efficient way, and takes time proportional to the height of the tree, not the total number of nodes in the tree.

With this algorithm, an update takes $1+\log r$ operations. By the sparseness assumption, each simulation event (time through the loop) takes at most $k(1+\log r)$ operations, where k is a constant independent of r. For E simulation events, the algorithm takes $\mathcal{O}(E \log r)$ operations, not counting the initialization in Step 1.

As a side note, there are other efficient ways to generate random variates of a discrete distribution [12], which are somewhat esoteric but have better expected times. One could do a thorough analysis of the trade-off between programming complexity, run time, numerical stability, and number of uniform random numbers required. However, since the Next Reaction Method is more easily enhanced to use fewer uniform random numbers and handle time-varying processes, both Markov and non-Markov, we shall favor it.

8.3 Time varying Direct Method, Markov processes

It is well known how to generalize the Direct Method to arbitrary functions of time $a_i(t)$ [8, 9]. One writes an equation that is analogous to Eq. (4). If S is the state at time t_0, then the equation is:

$$P(\mu, \tau | S, t_0) = a_\mu(S, \tau) \exp\left(- \int_{t_0}^{\tau} \sum_j a_j(S, t) dt \right)$$

At each subsequent step of the algorithm, one must recondition, i.e., change the density $P(\mu, \tau | S, t_i)$ to the density $P(\mu, \tau | S, t_{i+1})$. Doing so works out to changing the lower limit of integration.

It may be hard to generate random numbers according to this distribution for arbitrary functions of time a_i. (Note, in particular, that the lower limit of integration changes each iteration, so methods which involve numerical storage of partial values of the integral will have to do significant recalculation each iteration.) If all the a_i's change *in the same way*, then one can use the enhancements of the previous section; if not, it is not immediately clear how to run this algorithm efficiently for many reaction channels. Once again, the Next Reaction Method is preferable.

Acknowledgement *The authors wish to thank Daniel Gillespie and Tau-Mu Yi for their careful reading of an earlier version of this paper.*

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] D. Colquhoun and A. G. Hawkes. On the Stochastic Properties of Bursts of Single Ion Channel Openings and Clusters of Bursts. *Phil. Trans. R. Soc. Lond. B*, 300:1-59, 1982.

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

[4] M. A. Gibson & E. Mjolsness. Modelling the Activity of Single Genes. Computational Methods in Molecular & Cellular Biology. *MIT Press*, expected publication late '99.

[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] D. T. Gillespie. A Rigorous Derivation of the Chemical Master Equation. *Physica A*, 188:404-425, 1992.

[8] D. T. Gillespie. Markov Processes: An Introduction for Physical Scientists. *Academic Press*, 1992.

[9] A. P. J. Jansen. Monte Carlo Simulations of chemical Reactions on a Surface with Time-dependent Reaction-rate Constants. *Computer Physics Communications*, 86: 1-12, 1995.

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

[11] J. J. Lukkien, J. P. L. Segers, P. A. J. Hilbers, R. J. Gelten and A. P. J. Jansen. Efficient Monte Carlo Methods for the Simulation of Catalytic Surface Reactions. *Physical Review E*, 58(2):2598-2610, Aug. 1998.

[12] Y. Matias, J. S. Vitter and W. C. Ni. Dynamic Generation of Discrete Random Variates. *Bell Labs Technical Report*, 1997.

[13] C. J. Morton-Firth. Stochastic Simulation of Cell Signalling Pathways. Ph. D thesis, University of Cambridge, 1998.

[14] 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.

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