UC Santa Barbara
UC Santa Barbara Previously Published Works

Title
Efficient formulation of the stochastic simulation algorithm for chemically reacting systems

Permalink
https://escholarship.org/uc/item/27j1s73p

Journal
Journal of Chemical Physics, 121(9)

ISSN
0021-9606

Authors
Cao, Y
Li, H
Petzold, L

Publication Date
2004-09-01

Peer reviewed

Efficient Formulation of the Stochastic Simulation Algorithm for Chemically Reacting Systems *

Yang Cao [†]   Hong Li [‡]   Linda Petzold [§]

June 8, 2004

Abstract

In this paper we examine the different formulations of Gillespie's stochastic simulation algorithm (SSA) [J. Phys. Chem., 81:2340, (1977)] with respect to computational efficiency, and propose an optimization to improve the efficiency of the direct method. Based on careful timing studies and an analysis of the time-consuming operations, we conclude that for most practical problems the optimized direct method is the most efficient formulation of SSA. This is in contrast to the widely held belief that Gibson and Bruck's next reaction method [J. Phys. Chem. A, 104:1876, (2000)] is the best way to implement the SSA for large systems. Our analysis explains the source of the discrepancy.

[†]Department of Computer Science, University of California Santa Barbara. CA 93106. email: ycao@cs.ucsb.edu.

[‡]Department of Electrical and Computer Engineering, University of California Santa Barbara, Santa Barbara, CA 93106. hongli@engineering.ucsb.edu

[§]Department of Computer Science, University of California Santa Barbara, Santa Barbara, CA 93106. petzold@engineering.ucsb.edu

1   Introduction

The time evolution of a spatially homogeneous system of chemically reacting molecules is traditionally calculated by solving a set of coupled ordinary differential equations. This method is based on the *deterministic formulation* of chemical kinetics, in which the reaction constants are viewed as "rates" and the species concentrations are represented by continuous functions of time. Although the deterministic formulation is adequate in most cases, it does not reflect the stochastic nature of the system, which has been shown to be important in many biological systems. In particular, when there are some species with very few copy numbers in living cells, including DNA and important regulatory molecules, stochastic effects may account for cell to cell variation and play crucial roles in biological processes.[1-3] Numerical methods which can capture the correct stochasticity of the system are then needed. Gillespie's stochastic simulation algorithm (SSA)[4,5] is a numerical simulation procedure that is essentially exact for spatially homogeneous or well stirred chemical systems. The SSA is considered exact because it is rigorously based on the same microphysical principles that underly the chemical master equation (CME). There are also inexact stochastic simulation algorithms[6,7] that generate approximate trajectories. Our interest in this paper is only in exact methods.

The SSA is a Monte Carlo type method. With the SSA, one may approximate any variable of interest by generating many trajectories and observing the statistics of the values of the variable. Since many trajectories are needed to obtain a reasonable approximation, the efficiency of the SSA is of critical importance. Gillespie developed two different but equivalent formulations of the SSA: the Direct Method (DM) and the First Reaction Method (FRM). A third formulation of the SSA was given by the Next Reaction Method (NRM) of Gibson and Bruck.[8] The NRM can be viewed as an extension of the FRM, but it is much more efficient than the latter. Based on a count of arithmetic operations, it is widely believed to be more efficient than the Direct Method for large systems.[8]

In this paper we present a detailed analysis of the computational costs of the three formulations: DM, FRM and NRM, focusing on the differences between the DM and the NRM. In our experiments with different biochemical models, we have observed that even with the best data structure, the NRM is less efficient than the DM except for a very specialized class of problems. A similar observation has also been reported by Markus Schwehm:[9] *"A profiling of the algorithm (NRM) has revealed that the simulator engine spends most of its execution time for maintaining the priority queue of the tentative reaction times."* Our analysis explains these observations. Based on the analysis, and motivated by an idea used in NRM, we propose a strategy to improve the efficiency of the Direct Method. The new Optimized Direct Method (ODM) is, to the best of our knowledge, the most efficient formulation of the SSA.

This paper is organized as follows. In Section 2 we briefly review the SSA and its three different formulations, and outline a model of the heat shock response of E. Coli that will be used in our numerical experiments. Section 3 presents a detailed timing study for the DM and the NRM, based on the numerical results of three test problems. Section 4 gives a computational cost analysis and comparison for DM and NRM. In Section 5, we introduce the ODM and present some numerical results which illustrate its efficiency.

2 Background

In this section we briefly review the three formulations of SSA: Direct, First Reaction and Next Reaction, and describe the Heat Shock Response (HSR) model that will be used in some of the numerical experiments.

2.1 Stochastic Simulation Algorithm

Suppose the system involves N molecular species $\{S_1, \ldots, S_N\}$, represented by the state vector $X(t) = (X_1(t), \ldots, X_N(t))$, where $X_i(t)$ is the number of molecules of species S_i at time t. M reaction channels $\{R_1, \ldots, R_M\}$ are involved in the system. Assume the system is well-stirred and in thermal equilibrium. The dynamics of reaction channel R_j is characterized by the *propensity function* a_j and by the *state change vector* $\nu_j = (\nu_{1j}, \ldots, \nu_{Nj})$: $a_j(x)dt$ gives the probability that, given $X(t) = x$, one R_j reaction will occur in the next infinitesimal time interval $[t, t + dt)$, and ν_{ij} gives the change in the population of S_i induced by one R_j reaction.

The dynamics of the system obeys the *chemical master equation* (CME)[4,5]

$$\frac{\partial P(x, t|x_0, t_0)}{\partial t} = \sum_{j=1}^{M}[a_j(x - \nu_j)P(x - \nu_j, t|x_0, t_0) - a_j(x)P(x, t|x_0, t_0)], \tag{1}$$

where the function $P(x, t|x_0, t_0)$ denotes the probability that $X(t)$ will be x, given that $X(t_0) = x_0$. The CME is hard to solve both theoretically and numerically except for very simple systems. In practice, simulation methods are used. The SSA[4,5] is a well-known stochastic simulation method which is rigorously equivalent to the CME. Starting from the initial states, the SSA simulates the trajectory by repeatedly answering the following two questions and updating the states:

- When (time τ) will the next reaction fire?

- Which (reaction channel index μ) reaction will fire next?

The distributions of τ and μ are formulated to answer the two questions. Let

$$a_0(X) = \sum_{j=1}^{M} a_j(X). \tag{2}$$

The time τ, given $X(t) = x$, that the reaction will fire at $t+\tau$, is the exponentially distributed random variable with mean $\frac{1}{a_0(x)}$

$$P(\tau = s) = a_0(x) \exp(-a_0(x)s), \tag{3}$$

and the index μ of that firing reaction is the integer random variable with probability

$$P(\mu = j) = \frac{a_j(x)}{a_0(x)}. \tag{4}$$

In each step, the SSA generates random numbers and calculates τ and μ according to the probability distributions (3) and (4). Three different but stochastically equivalent formulations for SSA are proposed as follows.

2.2 Direct Method

On each step, the Direct Method generates two random numbers r_1 and r_2 in $U(0,1)$ (the set of uniformly distributed random numbers in the interval $(0,1)$). The time for the next reaction to occur is given by $t + \tau$, where τ is given by

$$\tau = \frac{1}{a_0(x)} \log\left(\frac{1}{r_1}\right). \tag{5}$$

The index μ of the occurring reaction is given by the smallest integer satisfying

$$\sum_{j'=1}^{\mu} a_{j'}(t) > r_2 a_0(t). \tag{6}$$

The system states are updated by $X(t+\tau) = X(t) + \nu_\mu$. Then the simulation proceeds to the next occurring time.

Algorithm Direct Method

1. *Initialization (Set the initial numbers of molecules. Set $t = 0$).*

2. *Calculate the propensity functions $a_i, (i = 1, ..., M)$ and a_0 from (2).*

3. *Generate two random numbers r_1 and r_2 in $U(0,1)$.*

4. *Calculate τ according to (5).*

5. *Search for μ as the smallest integer satisfying (6).*

6. *Update the states of the species to reflect execution of reaction μ. Set $t \leftarrow t + \tau$.*
7. *Go to step 2.*

2.3 First Reaction Method

The First Reaction method generates a τ_k for each reaction channel R_k according to

$$\tau_k = \frac{1}{a_k(x)} \log\left(\frac{1}{r_k}\right), \quad (k = 1, ..., M), \tag{7}$$

where $r_1, ..., r_M$ are M statistically independent samplings of $U(0,1)$. Then τ and μ are chosen as

$$\tau = \min\{\tau_1, ..., \tau_M\}, \tag{8}$$

and

$$\mu = \text{the index of } \min\{\tau_1, ..., \tau_M\}. \tag{9}$$

Algorithm First Reaction Method

1. *Initialization (Set the initial numbers of molecules. Set t = 0).*

2. *Calculate the propensity functions $a_i, (i = 1, ..., M)$.*

3. *Generate M independent random numbers from $U(0, 1)$.*

4. *Generate the time $\tau_i, (i = 1, ..., M)$ according to (7).*

5. *Find τ and μ according to (8) and (9).*

6. *Update the states of the species to reflect execution of reaction μ. Set $t \leftarrow t + \tau$.*
7. *Go to step 2.*

The Direct and the First Reaction methods are fully equivalent to each other[4, 5] although they look different. The random pairs (τ, μ) generated by both methods follow the same distribution. The First Reaction Method discards $M - 1$ unused reaction times in (8). Thus it is much less efficient than the Direct Method.

2.4 Next Reaction Method

Gibson and Bruck[8] cleverly transformed the First Reaction Method into an equivalent but more efficient new scheme. The Next Reaction method is significantly faster than the First Reaction method. It is widely believed[8] to be more efficient than the Direct method when the system involves many species and loosely coupled reaction channels.

The Next Reaction method can be viewed as an extension of the First Reaction method in which the $M - 1$ unused reaction times (8) are suitably modified for reuse. Clever data storage structures are employed to efficiently find τ and μ. The following algorithm describes the Next Reaction method (see reference[8] for details).

Algorithm Next Reaction Method

1. *Initialize:*

   (a) *set initial numbers of molecules, set $t \leftarrow 0$, generate a dependency graph G;*

   (b) *calculate the propensity functions 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 P .*

2. *Let μ be the reaction whose putative time τ_μ stored in the P, is least. Set $\tau \leftarrow \tau_\mu$.*

3. *Update the states of the species to reflect execution of reaction μ. Set $t \leftarrow \tau$.*

4. *For each edge (μ, α) in the dependency graph G*

    (a) update a_α;

    (b) if $\alpha \neq \mu$, set

$$\tau_\alpha \leftarrow (a_{\alpha,old}/a_{\alpha,new})(\tau_\alpha - t) + t. \tag{10}$$

    (c) if $\alpha = \mu$, generate a random number r and compute τ_α according to an equation

similar to (5)

$$\tau_\alpha = \frac{1}{a_\alpha(t)} \log\left(\frac{1}{r}\right) + t. \tag{11}$$

    (d) replace the old τ_α value in P with the new value.
5. *Go to step 2.*

Two data structures are used in this method:

- The *dependency graph* defined in reference[8] is a data structure that tells precisely which a_i should change when a given reaction is executed. Each reaction channel is denoted as a node in the graph. A directed edge connects R_i to R_j if and only if the execution of R_i affects the reactants in R_j. One can use the dependency graph to recalculate only the minimal number of propensity functions in step 4.

- The *indexed priority queue* (also known as a heap tree in computer science) consists of a tree structure of ordered pairs of the form (i, τ_i), where i is the reaction channel index and τ_i is the corresponding time when the next R_i reaction is expected to occur, and an index structure whose ith element points to the position in the tree which contains (i, τ_i). In the tree, each parent has a smaller τ than either of its children. Note that the minimum τ always stays in the top node and the order is only *vertical*. In each step, the update changes the value of the node and then bubbles it up or down according to its value to obtain the new priority queue. Theoretically, this procedure takes at most $\log(M)$ operations. In practice, usually there are a few reactions that occur much more frequently. Thus, the actual update takes less than $\log(M)$ operations.

We note that it takes some CPU time to maintain the two data structures. For a small system, this cost dominates the simulation. For a large system, the cost of maintaining the data structures may be relatively smaller compared with the savings.

The argument for the advantage of the NRM over the DM is based primarily on two observations: First, in each step, the NRM generates only one uniform random number while the Direct Method requires two. Second, the search for the index μ of the next reaction channel takes $O(M)$ time for the Direct Method, while the corresponding cost for the Next Reaction method is on the update of the indexed priority queue which is $O(\log(M))$.

2.5 Heat Shock Response Model

In this paper we use a model of the heat shock response (HSR) of E. Coli[10, 11] as an example problem to test the different SSA formulations. The HSR system describes the mechanism of how the bacteria E. Coli responds to a temperature increase. When exposed to temperatures high enough to induce the denaturing (unfolding) of its constituent proteins, the *E. Coli* bacterium derives some measure of protection from an elaborate heat shock response mechanism. One of several important components of this mechanism is the heat shock sigma factor, σ_{32}. Elevated temperatures in the bacterium cause σ_{32} to be produced through transcription and translation at a very rapid rate. A free σ_{32} molecule can bind to RNA polymerase (RNAP), and the resultant complex σ_{32}:RNAP initiates the transcription of genes that encode a variety of chaperone enzymes. These chaperons take care of denatured proteins, either by refolding them or else by degrading them so that they will not cause problems by aggregating. But a newly produced σ_{32} molecule is usually much more likely to be promptly sequestered by DNAK, one of those chaperone enzymes, an occurrence that precludes its binding to RNAP. The details of the deterministic model for the HSR system can be found in Ref.[10] and a stochastic version was discussed in Ref.[11] In our stochastic model, 28 species participate in 61 chemical reactions.

3 Timing Studies for Direct Method and Next Reaction Method

Since the FRM is obviously much less efficient than the other two formulations, we will concentrate on an analysis of DM and NRM. Both DM and NRM need to calculate the propensities, generate random numbers, find the next occurring time τ and channel index μ and update the system states. The major difference between them is the way they generate τ and μ. To generate τ, the Direct Method computes a random number r_1 and sums all the propensities to obtain a_0. Then τ is obtained from (5). The Next Reaction Method generates a set of τ_α's. For the just-fired reaction channel, it generates a random number to calculate the τ_μ from (11) (step 4(c) in the NRM algorithm). For the other reaction channels, it calculates τ_α according to (10) (step 4(b)). To obtain μ, the Direct Method generates another random number r_2 and searches for μ satisfying (6). The Next Reaction Method directly takes the index of the root node in the tree structure as μ. But it needs to update the nodes in the heap to maintain the order of the indexed priority queue. Table 1 shows the major differences between the two methods.

The cost for the Direct Method in Table 1 consists of five items: the cost C_p for calculating all the M propensities, the cost C_{a_0} for calculating the summation a_0 of all the propensities, the cost $2 * C_{rand}$ for generating two random numbers, the cost C_τ for calculating τ from (5) and the cost C_s for searching the index μ. We can express it as

$$C_{DM} = 2 * C_{rand} + C_{a_0} + C_\tau + C_s + C_p + C_{rest},\tag{12}$$

where C_{DM} denotes the average cost of the Direct Method and C_{rest} denotes all the other costs that both the DM and NRM methods have to spend.

The cost for the Next Reaction Method can be similarly formulated as

$$C_{NRM} = C_{rand} + C_\tau + C_{p'} + C_\alpha + C_{heap} + C_g + C_{rest},\qquad(13)$$

where C_{NRM} denotes the average cost of the Next Reaction Method, $C_{p'}$ denotes the cost for calculating the propensities changed by the last reaction, C_α denotes the cost to compute τ_α from (10), C_{heap} denotes the cost to update and maintain the heap tree, and C_g denotes the extra cost for the dependency graph in the calculation of the propensities. C_τ and C_{rest} have the same meaning as in (12). Note that although the formula (11) is different from (5), the computational costs of them are almost the same. Thus we can use C_τ in (13).

NRM has an advantage when the system is of large size and loosely-coupled, or in other words, if the firing of one reaction does not affect many other reactions. Gibson and Bruck[8] did not specify how loosely-coupled the system should be to show the advantage of NRM. For the practical problems we have tried, NRM is usually less efficient than DM. In the following we present the timing results when applying DM and NRM to three test problems. One is the heat shock model described in Section 2.5. The other two examples are designed to illustrate the potential advantage of NRM. The simulations were performed on a 1.4Ghz Pentium IV Linux workstation.

Example 3.1 The HSR model

In the simulation of the HSR model, the average simulation time for the DM is 86.8 seconds per SSA simulation. The average simulation time for the NRM is 170.4 seconds. We list the detailed time costs for one simulation in Table 2.

For the HSR model, the cost to maintain the data structure in NRM is large (60%). Thus this example does not show the advantage of NRM. Unfortunately for many practical problems, the data structure cost is very large when applying NRM. The following two test models were constructed to illustrate the situation where NRM is most advantageous.

Example 3.2 Linear Chain System

This model contains M chain reactions with $M + 1$ species as follows:

$$S_1 \longrightarrow S_2 \longrightarrow \ldots \longrightarrow S_n,\qquad(14)$$

where $n = M + 1$. The propensity functions are uniform: $a_i(X) = cX_i$, where $c = 1$. The initial state is $x_1(0) = 10,000$, $x_i(0) = 0$, $i = 2, \ldots, 601$. We take $M = 600$, and the final

time $T = 30$.

For this model, each reaction channel affects at most two reaction channels. Compared with the total number of reactions, this system is very loosely-coupled. Thus the NRM should have a great advantage over DM. In our simulations, the average simulation time for DM was 2.13 seconds, while the average simulation time for NRM was 1.07 seconds. The CPU time for NRM is about half of that of the DM. Table 3 shows the detailed CPU costs for one simulation. Because now the cost for updating the heap tree is reduced, the cost percentage due to the data structure is not as bad as in the HSR model.

Example 3.3 A Totally Independent System

This model contains 600 independent decaying processes as follows:

$$S_i \longrightarrow \emptyset, \quad i = 1, \ldots, n, \tag{15}$$

where $n = 600$. The propensity functions are uniform: $a_i(X) = cX_i$, where $c = 1$. The initial states are $x_i(0) = 1,000$ for $i = 1, \ldots, 600$. The final time $T = 30$.

For this model, each reaction channel affects itself only. It is the most loosely-coupled system possible. In our simulations, the average simulation time for DM was 5.39 seconds, while the average simulation time for NRM was 2.63 seconds. Again, NRM is faster than DM. Table 4 shows the detailed CPU costs for each part.

4   Computational Cost Analysis for the Direct Method and the Next Reaction Method

In this section we present a detailed analysis of the computational costs of DM and NRM which explains the results of our timing studies. We begin by introducing the following

definitions.

4.1 Definitions

Definition 4.1 Average Search Depth $\mathbb{S}$ *is the average number of operations it takes to*

obtain μ *according to equation* (6) *in the Direct Method.*

Obviously, $\mathbb{S}$ varies depending on the problem. For each step in the Direct Method, the search procedure is actually the summation of a_i (see (6)). To find μ, μ ADDs and COMPAREs are needed. The search depth in each step is exactly the index of the firing reaction. Thus in practice, we use the following formula to measure $\mathbb{S}$

$$\mathbb{S} = \frac{\sum_i^M i k_i}{\sum_i^M k_i},\tag{16}$$

where k_i is the number of firing times of the ith reaction during a simulation.

In theory, if the firing times k_i are of the same magnitude for all the reaction channels, the average search depth $\mathbb{S}$ of the DM is about $\frac{M}{2}$. But in practice the firing times for different reaction channels have a multiscale property. For many systems, a very few reactions fire much more frequently than others. $\mathbb{S}$ is different from $\frac{M}{2}$ for those systems.

Definition 4.2 Average Weighted Degree $\mathbb{D}$ *is the average degree for the dependency*

graph of a system. It is also the average number of reactions affected by one occurring

reaction.

The Next Reaction Method generates the dependency graph before it simulates a system. It is easy to calculate the degree of each node (reaction channel) in the graph. Thus in practice, the average weighted degree can be measured by

$$\mathbb{D} = \frac{\sum d_i k_i}{\sum k_i}\tag{17}$$

where d_i is the degree for the ith node, or in other words, the number of reactions affected by the ith reaction, and k_i is the number of times the ith reaction fires during a simulation.

Definition 4.3 Update Depth $\mathbb{U}$ *is the average number of operations needed to update*

the heap tree for the changed τ_α *'s in the NRM.*

There are two major steps in the update of the heap tree. First, we need to update the τ_α for the affected reactions. Second, we need to move the nodes in the heap to maintain the priority order. In each step, if d reactions are affected by the firing reaction channel, at least d operations need to be done to update the corresponding τ's. To maintain the priority order of the heap tree, at most $\log(M)$ moves are needed for each node. Thus we have the following inequality

$$\mathbb{D} \leq \mathbb{U} \leq \mathbb{D}\log(M). \tag{18}$$

In practice, because of the multiscale nature of the system, some reactions fire much more frequently than others. Those reaction channels stay around the root of the heap array. Thus very few moves are needed for each step. $\mathbb{U}$ is much less than $\mathbb{D}\log(M)$ and is close to $\mathbb{D}$.

For the test models in Section 3, in the simulation of the HSR model (Example 3.1), $M = 61$, $\mathbb{S} = 26.3$ (before the optimization described later), $\mathbb{D} = 8.79$ and $\mathbb{U} = 10.33$. In the simulation of the linear model (Example 3.2), $M = 600$, $\mathbb{S} = 16$, $\mathbb{D} = 2$ and $\mathbb{U} = 2.001$. For Example 3.3, $M = 600$, $\mathbb{S} = 300$, $\mathbb{D} = 1$ and $\mathbb{U} = 5.15$.

4.2   Cost Analysis and Comparison

It is widely believed that NRM should perform better than DM for a system of large size. The reasoning is based on an analysis[8] of the costs of the computational operations listed in Section 3. For the random number generator, DM always has one extra cost C_{rand}. For the computation of the propensities, DM needs to calculate M propensities, whereas NRM calculates only the changed ones (about $\mathbb{D}$ propensities). For the other terms, C_{a_0} consists of M additions and C_s consists of $O(M)$ additions and compares, while C_α and C_{heap} are of magnitude $O(\log M)$. Thus when the system has a large number of reactions, NRM might have less cost. But this analysis considers only the magnitude of the *computational* operations. It does not count the extra costs due to the data structure in C_{heap}, C_g. In practice, we have observed that DM is generally more efficient than NRM. Although one can argue that this is because the system size is not large enough to show the benefit of the NRM yet, only for the very loosely coupled problems Examples 3.2 and 3.3 did we begin to observe a relatively large savings. We believe that a fully optimized implementation of the DM can be more efficient than the NRM. There are two reasons for this. First, the original DM was not optimized for large systems. Second, the simple operation analysis in Gibson and Bruck[8] does not take into account all the components of the CPU cost. In this section we focus on the second point, that is, we extend the cost analysis to take into account costs such as maintenance of the data structure. Optimization of the DM for large systems will be discussed in the next section.

From (12) and (13), we can formulate the difference between the cost of DM and NRM as follows

$$C_{DM} - C_{NRM} = C_{rand} + C_1 + C_2 - C_{heap}, \tag{19}$$

where $C_1 = C_{a_0} + C_s - C_\alpha$ and $C_2 = C_p - C_{p'} - C_g$. In the following we discuss the details for each term.

For the random number generation, the NRM reduces the corresponding cost to half that of the DM. We note that the cost of generating a uniform random number is computationally small, and it does not increase with M. When M increases, C_{rand} is almost negligible. In our simulation of the HSR model with the Direct Method, C_{rand} accounts for 5% of the total CPU time. We expect an even lower percentage for C_{rand} when the system size becomes larger. Thus this saving is not critical.

The cost C_{a_0} consists of M ADD operations. We formulate it as

$$C_{a_0} \approx MC_{ADD}. \tag{20}$$

The cost C_s depends linearly on the average weighted search depth $\mathbb{S}$. The major operations are ADD and COMPARE. Thus it can be written as

$$C_s \approx \mathbb{S}(C_{ADD} + C_{COMP}). \tag{21}$$

The cost C_α depends linearly on the average weighted depth $\mathbb{D} - 1$ (C_α does not count the case when $\alpha = \mu$). Each update of τ_α consists of one MULTIPLY and one DIVIDE operation. We can express this as

$$C_\alpha \approx (\mathbb{D} - 1)(C_{MULT} + C_{DIV}). \tag{22}$$

We note that the costs for MULTIPLY and DIVIDE are usually much larger than the costs for ADD and COMPARE.

The cost C_p is based on the calculation of M propensities, while C_p' requires the calculation of only $\mathbb{D}$ propensities. The cost C_p is definitely larger than that of $C_{p'}$. But to distinguish which reactions are affected by the firing reaction, the directed graph is needed. Thus we have to also take the extra cost C_g into account. In each step, we need to visit the dependency graph $\mathbb{D}$ times to obtain all the affected reactions. Thus C_g is of order $O(\mathbb{D})$.

The cost C_{heap} for updating the heap data structure depends linearly on the average updating depth $\mathbb{U}$. The operations in the update include changing values in an affected node, swapping nodes to maintain the order, etc. We denote it by C_H. Then we have

$$C_{heap} = \mathbb{U}C_H. \tag{23}$$

Remark 4.1 *Although the costs C_α, C_g and C_{heap} are of $O(\mathbb{D})$ or at most $O(\mathbb{D}\log(M))$, the corresponding constants are different. For example, comparing C_s and C_α, the MULTIPLY and DIVIDE in C_α cost much more than the ADD and COMPARE in C_s. Thus although C_s is of order $O(M)$, practically C_s is less than C_α. In the maintenance of the heap tree, accessing the data costs much more than that in DM. Thus even for the same computational operation, due to the data structure cost, the CPU cost is significantly different. This is the reason why C_{heap} is so large in the HSR model.*

5　Optimized Direct Method (ODM)

According to the analysis in Section 4, for a large system the bottleneck for DM is the costs C_{a_0} and C_p, which are of magnitude $O(M)$, and the cost C_s, which is of magnitude $O(\mathbb{S})$. In this section we propose two optimizations to reduce these costs.

First, to reduce the cost C_s in the Direct Method, we note that in a large system, the reactions will undoubtedly be *multiscale*. Some reactions fire much more frequently than others. For example, in the HSR model, the six most frequent reactions fire about 95% of the total times, while the twelve most frequent reactions fire about 99% of the total times.[12] By utilizing this multiscale property, we can optimize the Direct Method implementation by making $\mathbb{S}$ very small. From (16), the order of the indexes of the reaction channels affects the search depth $\mathbb{S}$. By reindexing the reaction channels so that

$$k_i > k_j, \quad \text{for } i < j, \tag{24}$$

we can obtain an optimized value of $\mathbb{S}$ which we denote by $\mathbb{S}^*$. The optimized search depth $\mathbb{S}^*$ yields an optimized cost C_s. To obtain an optimized index, we sort the index of the reactions in decreasing order based on how often they fire. But before we really run a simulation, we do not have a quantitative knowledge of k_i's. Thus the ODM requires one or a few pre-simulations to determine the scale of the k_i's. We denote $\bar{k}_i$ as the number (or the mean value if there are multiple pre-simulations) of firing times for each reaction channel from the pre-simulations. We reindex the reaction channels such that

$$\bar{k}_i > \bar{k}_j, \quad \text{for } i < j. \tag{25}$$

Then we can use the new group of indexes to run the many simulations needed to generate an ensemble of the system. After this reindexing, usually the new search depth will be much smaller than a non-optimized one. For a system with multiscale reactions, $\mathbb{S}^*$ will be much smaller than $M/2$.

For the HSR model, this optimization method yields $\mathbb{S}^* = 3.37$. The average simulation time is reduced to 76.5 seconds, which is 11.87% more efficient than the DM with the original index ($\mathbb{S} = 26.2$, average simulation time is 86.8 seconds). In the worst case (with indexes in the opposite order), $\mathbb{S} = 58.6$ and the average simulation time is 102.2 seconds. Thus the optimized DM is 25.15% more efficient than the worst case. Table 5 shows the detailed CPU costs after we optimized the index.

Second, in the case of $D << M$, we can apply an idea from the development of NRM to DM. To reduce the costs C_{a_0} and C_p, we recalculate the propensities only for those reaction channels affected by the last reaction. Thus, only $\mathbb{D}$ propensities are needed for recalculating in C_p. But an extra cost must be paid for accessing the directed graph. With this change, $C_p \approx C_{p'} + C_g$. For the same reason, we can modify a_0 by subtracting old values and adding new values. Then C_{a_0} will consist of approximately $\mathbb{D}$ operations rather than M, resulting in $C_{a_0} + C_p \approx \mathbb{D} C_{ADD} + C_{p'} + C_g$. The costs C_{a_0} and C_p are now $O(\mathbb{D})$ instead of $O(M)$. In this strategy, we need the directed graph G introduced in NRM. But there is no need to maintain an expensive heap tree. Note that the extra data structure cost C_g is still large for many practical models. Thus the second optimization step should be applied only when D is much less than M.

For Example 3.2, $D << M$. After we make use of the second optimization step, the average time for one simulation reduces to 0.86 seconds, which is much less than the CPU time of the original DM, and also less than that of NRM. The detailed timings for the optimized DM are listed in Table 6.

For Example 3.3, $D << M$. After the second step, the average time for one simulation reduces to 3.72 seconds, which is less than that of the original DM, but more than that of

the NRM. The reason is that here $\mathbb{S} = M/2$. The cost C_s is still large in this case. The detailed timings for the optimized DM are listed in Table 7.

With the two optimization steps, the Optimized Direct Method (ODM) is more efficient than the original DM. In the two extreme cases Example 3.2 and 3.3, ODM is close to NRM. Table 8 lists the operation comparison for DM, NRM and ODM. The only situation that ODM is less efficient than NRM is when $\mathbb{D} << M$ and $S^* \approx M/2$. In a practical code, we can evaluate $\mathbb{D}$, $\mathbb{U}$ and $\mathbb{S}$ and compare it with M. We note that in practice, large systems usually always have the multiscale nature which makes $S^* << M/2$. Thus ODM is almost always preferred.

6   Further Discussion

6.1   The Cost Comparison with Different Scales of M

To study how the CPU cost varies with the scale of the problem, we modified M in Example 3.2 and compared the CPU time for DM, NRM and ODM. The results are shown in Figure 1. We can see that when M is small, the CPU time for DM and NRM are close. When M is larger, NRM performs better. For all M, the CPU times of ODM are smaller than those of NRM. But it seems that the CPU times for all methods increase linearly when M increases. This is because for this simple model, when M increases, the total number of reactions increases. The total CPU cost is the product of the average cost and the total number of reactions. The reason why SSA is slow is due to the total number of simulation steps. Thus we could expect a large savings if we could reduce the total number of simulation steps. We address this in a forthcoming paper.[12]
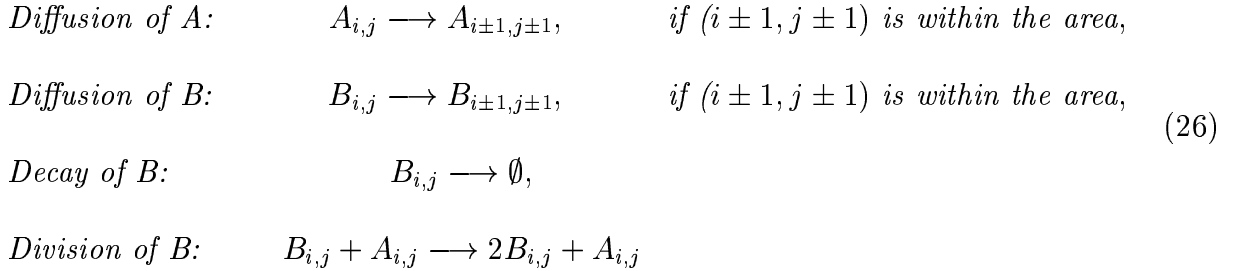
6.2   Spatially Inhomogeneous System

SSA is based on the assumption of a spatially homogeneous system. However, by discretizing the space into cells and introducing variables associated to the population of the species in each cell, SSA can also be applied to spatially inhomogeneous systems. With that configuration, for each cell reactions occur only in its own cell or with its neighbours. This feature makes the discretized spatially inhomogeneous system a good example of a loosely-coupled system. The Optimized Direct Method will have a great advantage for these systems. A theoretical discussion of the spatially inhomogeneous system will be a topic of future research. Here we only construct a simple example to show the efficiency of the optimization.

Example 6.1 Spatially Inhomogeneous Example

This example was discussed by Shnerb[13] to show the difference between the continuous

deterministic approach and the discrete stochastic approach. Two species A and B spread over a square area and move randomly with a certain diffusion coefficient. Two types of reactions are involved. The species B decay with a constant rate μ and divides with rate λ when it meets the catalyst A. To numerically simulate this system, the square area is discretized as a 10 by 10 grid. To each grid cell (labeled by (i,j)), we assign variables $A_{i,j}$ for species A and $B_{i,j}$ for species B. The reactions are listed as follows

Diffusion of A:	$A_{i,j} \longrightarrow A_{i\pm1,j\pm1}$,	*if $(i\pm1, j\pm1)$ is within the area,*
Diffusion of B:	$B_{i,j} \longrightarrow B_{i\pm1,j\pm1}$,	*if $(i\pm1, j\pm1)$ is within the area,*
Decay of B:	$B_{i,j} \longrightarrow \emptyset$,	
Division of B:	$B_{i,j} + A_{i,j} \longrightarrow 2B_{i,j} + A_{i,j}$	

$$(26)$$

The diffusion rate for both A and B is $\mu = 0.2$. The decay rate for B is 1. The division rate when B meets A is set to $\lambda = 2.8$. The initial states are set so that each cell contains one species B and there is one species A in the whole area.

$$B_{i,j} = 1, \qquad A_{i,j} = \begin{cases} 1, & i = 10, j = 10, \\ 0, & else. \end{cases} \tag{27}$$

The simulation time interval is $[0, 100]$. When the population of B blows up, the simulation will be very slow. Thus we terminate the simulation when the total population of B exceeds $10,000$. The corresponding state is called "survival of B". If all B species die, the simulation is also terminated and the corresponding state is called "extinction of B". With the above configuration, the species B has probability around 30% to survive, while a classical deterministic differential equation method will always predict extinction. We applied the three SSA formulations to this example. Since the species states have a big variation, the CPU cost for a single simulation has a large fluctuation, depending on whether the final state is survival

15

or extinction. But for a large number of simulations, the CPU time is stable. We measured the CPU time for 10^4 simulations. The corresponding CPU time for the original DM was 2045 seconds, the CPU time for NRM was 770 seconds and the CPU time for ODM was 406 seconds. We expect an even larger difference for a finer grid. This example shows that the ODM is much preferred for discretized spatially inhomogeneous systems. The detailed timings for the three formulations of SSA are listed in Table 9.

7 Conclusion

Gillespie's stochastic simulation algorithm (SSA) is in widespread use for the stochastic simulation of chemically reacting systems, consuming a great many CPU cycles. In this paper we studied the different formulations of SSA: Direct Method (DM), First Reaction Method (FRM) and Next Reaction Method (NRM). We found that for all but a very specialized class of problems, DM is most efficient. This is in contrast to the widely held belief that NRM is most efficient. Our timing studies reveal that the NRM is spending a substantial fraction of its time on maintaining the data structure. The original operation count for the NRM[8] considered computational operations only; hence it did not include these costs. We have presented a more comprehensive operation count which explains the observed results and suggests some further optimizations that can be applied to the DM. After applying these optimizations, the Optimized Direct Method (ODM) now appears to be the most efficient formulation for the vast majority of problems.

Acknowledgements

List of Captions
Figure 1: CPU time of DM (plot with 'o' and solid line), NRM (plot with '+' and dashed line) and ODM (plot with '*' and dotted line) vs. the number of reactions M for the linear chain Example 3.2.

References

[1] A. Arkin, J. Ross and H. McAdams. *Genetics*, 149, 1998.

[2] M. Elowitz, A. Levine, E. Siggia and P. Swain. *Science*, 297, 2002.

[3] H. McAdams, A. Arkin. *Proc. Natl. Acad. Sci. USA*, 94, 1997.

[4] D. Gillespie. *J. Comput. Phys.*, 22, 1976.

[5] D. Gillespie. *J. Phys. Chem.*, 81, 1977.

[6] C. Morton-Firth and D. Bray. *J. Theor. Biol.*, 192, 1998.

[7] D. Gillespie. *J. Chem. Phys.*, 115, 2001.

[8] M. Gibson and J. Bruck. *J. Phys. Chem. A*, 104, 2000.

[9] M. Schwehm. *Poster in German Conference on Bioinformatics*, 2001.

[10] H. Kurata, H. El-Samad, T. Yi, M. Khammash and J. Doyle. *Proceedings of the 40th IEEE Conference on Decision and Control*, 2001.

[11] H. Kurata, M. Khammash and J. Doyle. *3rd International Conference on Systems Biology*, 2002.

[12] Y. Cao, L. Petzold and D. Gillespie. Multiscale stochastic simulation algorithm with stochastic partial equilibrium assumption for chemically reacting systems. In preparation.

[13] N. Shnerb, Y. Louzoun, E. Bettelheim, S. Solomon. *Proc. Natl. Acad. Sci. USA*, 97, 2000.

Direct Method	Next Reaction Method
1. Generate random number r_1 and r_2 2. Calculate M propensities 3. Calculate the summation a_0 of all propensities 4. Calculate τ with (5) 5. Search for μ according to (6)	1. Generate random number r 2. Calculate propensities changed by last reaction 3. Calculate τ_μ with (11) 4. Calculate the other $\tau_\alpha s$ ($\alpha \neq \mu$) with (10) 5. Update the heap data structure

Table 1: Comparison of operations in DM and NRM.

DM	C_{a_0}	C_s	C_p	$2 * C_{rand}$	C_τ	$C_{updateX}$	C_{rest}
CPU cost (seconds)	15.48	14.98	11.71	4.09	6.45	7.36	9.30
Percentage	22.24%	21.52%	16.82%	5.88%	9.27%	10.57%	13.36%
NRM	C_{heap}	C_α	$C_{p'} + C_g$	C_{rand}	C_τ	$C_{updateX}$	C_{rest}
CPU cost (seconds)	99.16	23.34	19.34	1.74	6.36	7.89	6.42
Percentage	60.0%	14.12%	11.7%	1.05%	3.85%	4.77%	3.89%

Table 2: CPU costs of DM and NRM for the HSR model.

DM	C_{a_0}	C_s	C_p	$2 * C_{rand}$	C_τ	$C_{updateX}$	C_{rest}
CPU cost	0.66	0.05	0.56	0.03	0.03	0.65	0.05
Percentage	32.54%	2.46%	27.50%	1.45%	1.63%	31.86%	2.56%
NRM	C_{heap}	C_α	$C_{p'} + C_g$	C_{rand}	C_τ	$C_{updateX}$	C_{rest}
CPU cost	0.17	0.03	0.03	0.02	0.03	0.81	0.05
Percentage	14.8%	2.61%	2.61%	1.74%	2.61%	70.43%	4.35%

Table 3: CPU costs of DM and NRM for the linear model (14).

DM	C_{a_0}	C_s	C_p	$2 * C_{rand}$	C_τ	$C_{updateX}$	C_{rest}
CPU cost	1.26	1.27	1.07	0.03	0.03	1.67	0.06
Percentage	23.38%	23.56%	19.85%	0.56%	0.56%	30.98%	1.11%
NRM	C_{heap}	C_α	$C_{p'} + C_g$	C_{rand}	C_τ	$C_{updateX}$	C_{rest}
CPU cost	0.27	0	0.12	0.03	0.07	1.64	0.19
Percentage	10.7%	0%	4.74%	1.19%	2.77%	64.82%	7.51%

Table 4: The CPU costs of DM and NRM for the totally independent model (15).

ODM	C_{a_0}	C_s	C_p	$2 * C_{rand}$	C_τ	$C_{updateX}$	C_{rest}
CPU cost (seconds)	15.27	4.21	9.27	6.23	6.90	7.97	12.13
Percentage	24.64%	6.79%	14.96%	10.05%	11.13%	12.86%	19.57%

Table 5: CPU costs of the Optimized Direct Method for the HSR model

ODM	$C_{a_0} + C_p$	C_s	$2 * C_{rand}$	C_τ	$C_{updateX}$	C_{rest}
CPU cost	0.02	0.05	0.03	0.02	0.67	0.06
Percentage	2.35%	5.88%	3.53%	2.35%	78.82%	7.06%

Table 6: CPU costs of the Optimized Direct Method for the linear model (14).

ODM	$C_{a_0} + C_p$	C_s	$2 * C_{rand}$	C_τ	$C_{updateX}$	C_{rest}
CPU cost	0.20	1.42	0.08	0.06	1.88	0.08
Percentage	5.38%	38.17%	2.15%	1.61%	50.54%	2.15%

Table 7: CPU costs of the Optimized Direct Method for the totally independent model (15).

		C_{a_0}	C_p	C_s	extra data structure cost
DM		$O(M)$	$O(M)$	$O(\mathbb{S}) \approx O(M)$	no
ODM	when $D << M$	$O(\mathbb{D})$	$O(\mathbb{D})$	$O(\mathbb{S}^*)$	directed graph
	else	$O(M)$	$O(M)$	$O(\mathbb{S}^*)$	no
		C_α	$C_{p'}$	C_{heap}	extra data structure cost
NRM		$O(\mathbb{D})$	$O(\mathbb{D})$	$O(\mathbb{U}) \approx O(\mathbb{D})$	directed graph and heap tree

Table 8: Operation count comparison for DM, NRM and ODM.

DM	C_{a_0}	C_s	C_p	$2 * C_{rand}$	C_τ	$C_{updateX}$	C_{rest}
CPU cost	581.38	786.51	353.82	19.25	17.80	129.20	157.68
Percentage	28.42%	38.45%	17.30%	0.94%	0.87%	6.32%	7.71%
NRM	C_{heap}	C_α	$C_{p'} + C_g$	C_{rand}	C_τ	$C_{updateX}$	C_{rest}
CPU cost	331.54	86.37	58.68	10.21	17.28	119.08	145.91
Percentage	43.05%	11.22%	7.62%	1.33%	2.24%	15,45%	18.94%
ODM	$C_{a_0} + C_p$	C_s		$2 * C_{rand}$	C_τ	$C_{updateX}$	C_{rest}
CPU cost	64.83	43.75		17.13	18.84	115.28	144.45
Percentage	16.03%	10.82%		4.24%	4.66%	28.51%	35.72%

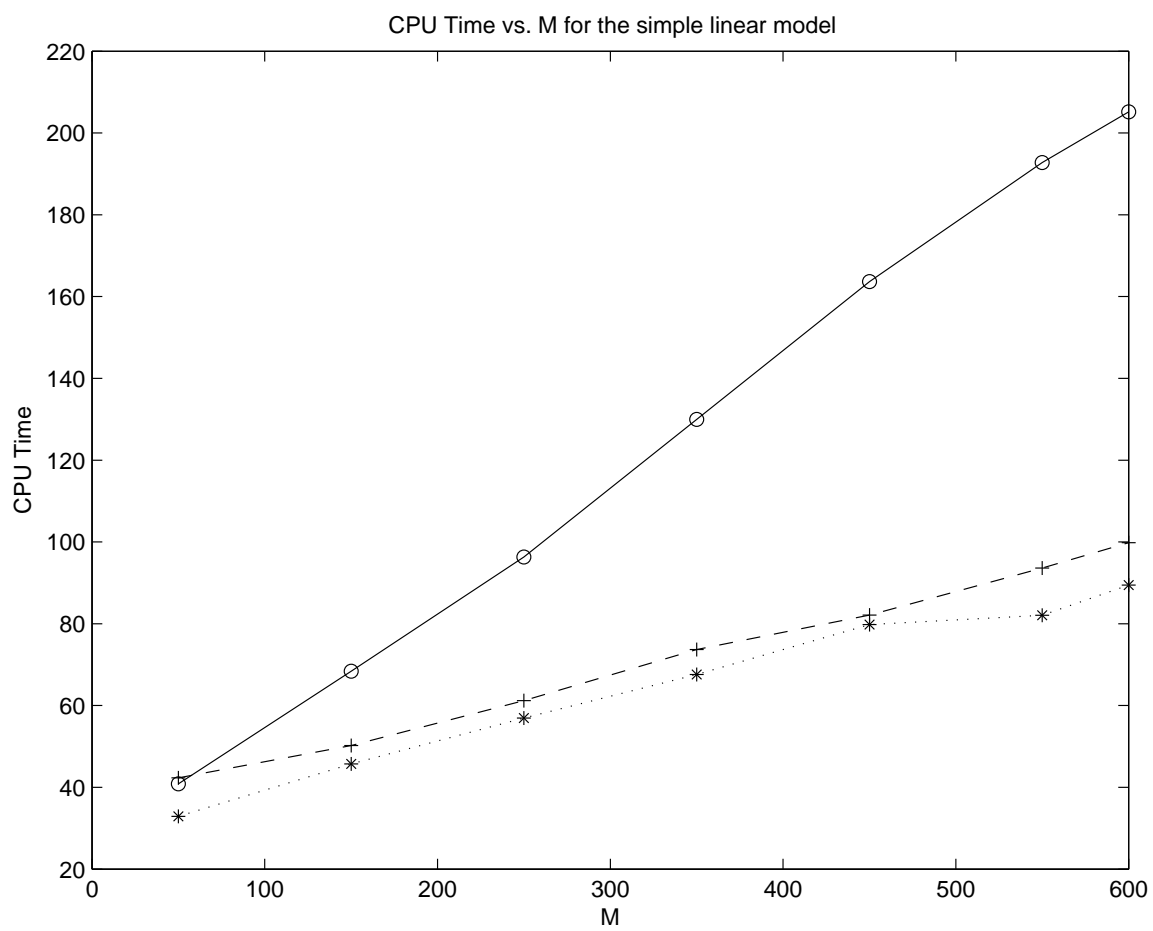Table 9: The detailed CPU costs of 10^4 similations for DM, NRM and ODM for the spatially inhomogeneous example.

19

Figure 1: