Efficient Formulation of the Stochastic Simulation Algorithm for Chemically Reacting Systems

Yang Cao, Hong Li and Linda Petzold
Department of Computer Science
University of California, Santa Barbara
www.engineering.ucsb.edu/~cse/

Abstract

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 most efficient formulation of the SSA for large systems. Our analysis explains the source of the discrepancy.

Background – Foundations of SSA

Suppose the system involves N molecular species $\{S_1,\ldots,S_N\}$, represented by the state vector $X(t)=\{X_1,\ldots,X_N\}$, where X_i 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* $a_j(x)$ and by the *state change vector* $\nu_j=\{\nu_{1j},\ldots,\nu_{Nj}\}$: $a_j(x)$ 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. 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 $a_0(x)=\sum_{j=1}^{M} a_j(x)$.

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

$$p(\tau=s)=a_0(x)\exp(-a_0(x)s), \qquad (1)$$

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

$$P(\mu=j)=\frac{a_j(x)}{a_0(x)}. \qquad (2)$$

In each step, the SSA generates random numbers and calculates τ and μ according to the probability distributions (1) and (2). Three different but stochastically equivalent formulations for SSA have been proposed: Direct Method (DM), First Reaction Method (FRM) and Next Reaction Method (NRM).

Direct Method (DM) [Gillespie 1976]

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

$$\tau=\frac{1}{a_0(x)}\log\frac{1}{r_1}. \qquad (3)$$

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

$$\sum_{j=1}^{\mu} a_j(x) > r_2 a_0(x). \qquad (4)$$

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

First Reaction Method (FRM) [Gillespie 1977]

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

$$\tau_k=\frac{1}{a_k(x)}\log\frac{1}{r_k} \quad (k=1,\ldots,M),$$

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

$$\tau=\min\{\tau_1,\ldots,\tau_M\},$$
$$\mu=\text{the index of } \min\{\tau_1,\ldots,\tau_M\},$$

Next Reaction Method (NRM) [Gibson 2000]

Gibson and Bruck cleverly transformed FRM into an equivalent but more efficient new scheme. The Next Reaction Method (NRM) is significantly faster than FRM. It is widely believed to be more efficient than DM when the system involves many species and loosely coupled reaction channels. The NRM can be viewed as an extension of FRM in which the $M-1$ unused reaction times are suitably modified for reuse. Clever data storage structures are employed to efficiently find τ and μ. The following algorithm describes the NRM:

Algorithm Next Reaction Method:

1. Initialize
   (a) set initial numbers of molecules, set $t=0$, generate a dependency graph G
   (b) calculate the propensity functions $a_j(x)$, for all j
   (c) for each j, generate a putative time τ_j according to an exponential distribution with parameter $a_j(x)$
   (d) store the τ_j values in an indexed priority queue P
2. Let μ be the reaction whose putative time τ_μ stored in P is least. Set $\tau=\tau_\mu$
3. Update the states of the species to reflect execution of reaction μ. Set $t=t+\tau$
4. For each edge (μ,α) in the dependency graph G
   (a) update a_α
   (b) if $\tau\neq\mu$, set
   $$\tau_\alpha=a_{\alpha,old}/a_{\alpha,new}(\tau_\alpha-t)+t \qquad (5)$$
   (c) if $\mu=\mu$, generate a random number r and compute τ_α according to an equation similar to (3):
   $$\tau_\alpha=\frac{1}{a_\alpha(x)}\log\frac{1}{r}+t \qquad (6)$$
   (d) replace the old a_α value in P with the new value

Go to step 2.

Two data structures are used in this method:

- The *dependency graph* is a data structure that tells which a_j 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 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.

Which Algorithm is Most Efficient?

It has been widely believed that NRM is more efficient for large scale problems. The argument for the advantage of NRM over DM is based primarily on two observations: First, in each step, NRM generates only one uniform random number while DM requires two. Second, the search for the index μ of the next reaction channel takes $O(M)$ time for DM, while the corresponding cost for NRM is on the update of the indexed priority queue, which is $\log M$.

Our timing tests show a different result. The table below lists the major costs involved in DM and NRM:

DM	NRM
Generate two random numbers : $2C_{rand}$	Generate one random number : C_{rand}
Calculate M propensities : C_p	Calculate propensities affected : $C_{p'}+C_g$
Calculate a_0 : C_{a_0}	Calculate τ_μ : C_τ
Calculate τ,μ : C_τ	Calculate $\tau_\alpha,\alpha\neq\mu$: C_τ
Search for μ : C_s	Update heap data structure : C_{heap}

To access these costs and their relative impacts on the total simulation time, for both formulations of SSA, we performed careful timing tests of DM and NRM for several different test problems. The simulations were done on a 1.4 GHz Pentium IV Linux workstation.

The first problem is a stochastic model of the heat shock response of E. Coli [Kurata 2001], which contains 28 variables and 61 reactions. The average simulation time by DM was 86.8 seconds per SSA simulation, while the average simulation time by NRM was 170.4 seconds. The following tables give the detailed timing results for one simulation. We can see that the cost to maintain the data structure for NRM is large (60%). Unfortunately this is the case for many practical problems

DM	C_{a_0}	C_s	C_p	$2C_{rand}$	C_τ	$C_{updateX}$	C_{rest}	C_{total}
CPU (seconds)	15.48	14.98	11.71	4.09	7.45	7.36	9.30	86.8
Percentage (%)	22.24	21.52	16.83	5.88	9.27	10.57	13.36	100.0

NRM	C_{heap}	C_τ	$C_{p'}+C_g$	C_{rand}	C_τ	$C_{updateX}$	C_{rest}	C_{total}
CPU (seconds)	99.16	23.34	19.34	1.74	6.36	7.89	6.42	170.4
Percentage (%)	60.0	14.12	11.7	1.05	3.85	4.77	3.89	100.0

We constructed two test models to illustrate the situation when NRM is most advantageous. One is the linear chain reaction system, which contains M chain reactions with $M+1$ species as follows:

$$S_1 \xrightarrow{c} S_2 \xrightarrow{c} \cdots \xrightarrow{c} S_n \qquad (7)$$

where $n=M+1$. The propensity functions are uniform: $a_j(x)=cx_j$, where $c=1$. The initial state is $x_1(0)=10{,}000, x_i(0)=0, i=2,\ldots,M+1$. We let $M=600$, 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 NRM should have a substantial 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 following table 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 less than in the HSR model.

DM	C_{a_0}	C_s	C_p	$2C_{rand}$	C_τ	$C_{updateX}$	C_{rest}	C_{total}
CPU (seconds)	0.66	0.05	0.56	0.03	0.03	0.65	0.05	2.13
Percentage (%)	32.54	2.46	27.50	1.45	1.63	31.86	2.56	100.0

NRM	C_{heap}	C_τ	$C_{p'}+C_g$	C_{rand}	C_τ	$C_{updateX}$	C_{rest}	C_{total}
CPU (seconds)	0.17	0.03	0.03	0.02	0.03	0.81	0.05	1.07
Percentage (%)	14.8	2.61	2.61	1.74	2.61	70.43	4.35	100.0

The third model is a totally independent reaction system. This model contains 600 independent decaying processes as follows:

$$S_i \xrightarrow{c} \varnothing, \qquad i=1,\ldots,600. \qquad (8)$$

The propensity functions are uniform: $a_j(x)=cx_j$, where $c=1$. The initial states are $x_i(0)=1{,}000, \quad i=1,\ldots,600$. 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. The following table shows the detailed CPU costs for each part.

DM	C_{a_0}	C_s	C_p	$2C_{rand}$	C_τ	$C_{updateX}$	C_{rest}	C_{total}
CPU (seconds)	1.26	1.27	1.07	0.03	0.03	1.67	0.06	5.39
Percentage (%)	23.38	23.56	19.85	0.56	0.56	30.98	1.11	100.0

NRM	C_{heap}	C_τ	$C_{p'}+C_g$	C_{rand}	C_τ	$C_{updateX}$	C_{rest}	C_{total}
CPU (seconds)	0.27	0	0.12	0.03	0.07	1.64	0.19	2.63
Percentage (%)	10.7	0	4.74	1.19	2.77	64.82	7.51	100.0

From the timing tests above, we can draw the following conclusions:

- The cost to maintain the data structure in NRM is significant.
- For very loosely-coupled systems, NRM performs better than DM.
- For loosely-coupled systems, the major CPU costs are C_{a_0}, C_p and C_s.

We have done a careful cost analysis, taking into account the cost of maintaining the data structure, and found that by optimizing the direct method we can reduce the costs C_{a_0}, C_p and C_s. We call this new method the Optimized Directed Method (ODM). To reduce C_s, we note that in a large system, reactions usually are 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. Thus we re-index the reaction channels so that $k_i>k_j$ for $i<j$. This optimized index of reaction channels leads to an optimized search depth, which 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 the k_i's. Thus the ODM requires one or a few pre-simulations to determine the scale of the k_i's. 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. C_s can be dramatically reduced. For the HSR model, the optimized search depth is 3.37. By reindexing, the average simulation time is reduced to 76.5 seconds, which is 11.87% more efficient than the DM with the original index (search depth is 26.2). The following table shows the detailed time costs of ODM for the HSR model:

ODM	C_{a_0}	C_s	C_p	$2C_{rand}$	C_τ	$C_{updateX}$	C_{rest}	C_{total}
CPU (seconds)	15.27	4.21	9.27	6.23	6.90	7.97	12.13	76.5
Percentage (%)	24.64	6.79	14.96	10.05	11.13	12.86	19.57	100.0

In a second step of the cost optimization, to reduce C_{a_0} and C_p, we can use an idea from the development of NRM. ODM recalculates the propensities only for those reaction channels affected by the last reaction. Note that an extra cost must be paid for accessing the dependency graph. Thus this strategy applies only to loosely-coupled systems. (We can decide if a system is loosely coupled in the pre-simulation). For the linear chain reaction system, the average time for one simulation by ODM reduces to 0.86 second, which is much less than the CPU time of the original DM, and also less than that of the NRM. Detailed timing results for ODM are listed in the following table:

ODM	$C_{a_0}+C_p$	C_s	$2C_{rand}$	C_τ	$C_{updateX}$	C_{rest}	C_{total}
CPU (seconds)	0.02	0.05	0.03	0.02	0.67	0.06	0.86
Percentage (%)	2.35	5.88	3.53	2.35	78.82	7.06	100.0

For the totally independent system, the average time for one simulation by ODM 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 the search depth is about $\frac{M}{2}$. The cost C_s is still large in this case. Detailed timing results for the ODM are listed in the following table.

ODM	$C_{a_0}+C_p$	C_s	$2C_{rand}$	C_τ	$C_{updateX}$	C_{rest}	C_{total}
CPU (seconds)	0.20	1.42	0.08	0.06	1.88	0.08	3.72
Percentage (%)	5.38	38.17	2.15	1.61	50.54	2.15	100.0

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. 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 NRM is spending a substantial fraction of its time on maintaining the data structure. The original operation count for NRM 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 DM. After applying these optimizations, the Optimized Direct Method (ODM) now appears to be the most efficient formulation for the vast majority of problems.

References:

- [Gillespie1976] D. Gillespie, J. Comput. Phys. 22, 403 (1976).
- [Gillespie1977] D. Gillespie, J. Phys. Chem. 81, 2240 (1977).
- [Gibson2000] M. Gibson and J. Bruck, J. Phys. Chem. A 104, 1876 (2000).
- [Kurata2001] H. Kurata, H. El-Samad, T. Yi, M. Khammash and J. Doyle, Proceedings of the 40th IEEE conference on Decision and Control, 2001.

Acknowledgements:

This work was supported by the California Institute of Technology under DARPA award No. F30602-01-2-0558, by the U.S. Department of Energy under DOE award No. DE-FG03-00ER25430, by the National Science Foundation under NSF award CTS-0205584, and by the Institute for Collaborative Biotechnologies through grant DAAD19-03-D-0004 from the U.S. Army Research Office.

WebLink:

A detailed discussion has been published in J. Chem. Phys. 121(9), pp. 4059-4067, 2004.