A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical reaction networks

Alexander Slepoy,[1,a)] Aidan P. Thompson,[2] and Steven J. Plimpton[2,b)]
[1]*National Nuclear Security Administration, U.S. Department of Energy, Washington D.C. 20585, USA*
[2]*Sandia National Laboratories, Albuquerque, New Mexico 87185, USA*

(Received 21 December 2007; accepted 14 April 2008; published online 23 May 2008)

The time evolution of species concentrations in biochemical reaction networks is often modeled using the stochastic simulation algorithm (SSA) [Gillespie, J. Phys. Chem. **81**, 2340 (1977)]. The computational cost of the original SSA scaled linearly with the number of reactions in the network. Gibson and Bruck developed a logarithmic scaling version of the SSA which uses a priority queue or binary tree for more efficient reaction selection [Gibson and Bruck, J. Phys. Chem. A **104**, 1876 (2000)]. More generally, this problem is one of dynamic discrete random variate generation which finds many uses in kinetic Monte Carlo and discrete event simulation. We present here a constant-time algorithm, whose cost is independent of the number of reactions, enabled by a slightly more complex underlying data structure. While applicable to kinetic Monte Carlo simulations in general, we describe the algorithm in the context of biochemical simulations and demonstrate its competitive performance on small- and medium-size networks, as well as its superior constant-time performance on very large networks, which are becoming necessary to represent the increasing complexity of biochemical data for pathways that mediate cell function. © *2008 American Institute of Physics*. [DOI: 10.1063/1.2919546]

I. INTRODUCTION

The metabolic, regulatory, and signaling pathways in biological cells are often represented by biochemical networks involving reactions between proteins, genes, and other molecular species. The response of such networks to perturbation is a ubiquitous modeling problem in computational biology. Simulations of the response track the time-dependent concentrations of individual species. Continuum versions of the models can be formulated as sets of coupled ordinary differential equations and integrated by standard methods. In cases where the concentrations of some species are small, stochastic effects impact the behavior of the system.[1]

In 1976, Gillespie developed the stochastic simulation algorithm (SSA) to model these networks via Monte Carlo methods, in a way that correctly samples the dynamic probability distribution of possible reactions.[2,3] Derived from the chemical master equation, the algorithm evolves the system one reaction at a time, choosing the specific reaction to perform, advancing time by an appropriate interval, and updating the probability distribution of future reactions to reflect the outcome of the selected reaction. The method is widely used to model biochemical networks (the original two papers have been cited over 1500 times), and to analyze the effects of stochasticity within the small reaction volumes of cells.[4]

With rapid growth in experimental data characterizing biochemical interactions, researchers simulate ever-larger reaction networks,[1,5–13] where a network represents the interactions between biochemically reactive molecular species in and around the cell. The nodes of the network are biochemical species with concentrations; the edges are the relationships implied by the reactions, with edge weights corresponding to reaction rates. Currently, the largest protein interaction network[14] known to the authors contains 18 000 proteins and 44 000 interactions.

Note that each bound state of two or more molecules is typically counted as a separate "species" in formulations of these networks, though alternate methodologies have been proposed as dicussed below.[15,16] Thus, if the combinatorial richness of protein complex formation is included, network sizes can grow exponentially, since even a relatively small number of proteins which bind together can exhibit great variety in complexation and post-translational modification, with the enumeration of possible states for a single receptor complex reaching 10^6 to 10^8 states.[17] Estimates of the average connectivity in such networks are as high as 38 interactions per protein.[18] The kinetic rate constants associated with biochemical reactions span many orders of magnitude. For example, fast enzymes operate at $\sim 10^5/\text{s}$, while slow enzymes operate at $2/\text{s}$ or slower.[19,20] These characteristics define the size and other properties of the network which produces a probability distribution of reactions that must be sampled by the SSA in order to accurately model biochemical network dynamics.

The computational cost of the original SSA to perform a single reaction scaled as $O(N)$, i.e., linearly in N, the number of reactions in the network. Since the time increment per reaction also tends to shrink with increasing N, such scaling limited the size of networks that could be efficiently simulated. Gibson and Bruck[21] proposed an alternate implementation of the SSA which scales as $O(\log_2 N)$, enabling much

[a)]Electronic mail: Alexander.Slepoy@nnsa.doe.gov.
[b)]Electronic mail: sjplimp@sandia.gov.

larger networks to be modeled. Other optimizations have also been proposed, including preranking the list of reaction probabilities from large to small,[22] which allows a sequential search to outperform a logarithmic search for some probability distributions and small networks.

A variety of enhancements to the SSA have also been developed to enable its use for different problems. For example, much work has been done[23–28] to address the issue of stiffness of the dynamical systems embodied in biological networks, and extend the timescale over which the SSA can be used, while bounding the errors induced by multitimescale approximations. To address the issue of exponential growth in the number of possible reactions due to protein complexation, rule-based approaches have been developed,[15,16] which limit the number of reactions by generating new ones only as needed during a simulation as specific reactants are produced. Properties of the new reaction (e.g., its rate constant) are inferred from properties of its reactants, limiting the amount of information that must be stored. Depending on how reactions are selected, the computational cost in these approaches can depend on the number of molecules currently present in the simulation, rather than the number of possible reactions.[29,30] Computationally, this can be a win when the number of molecules is smaller than a very large list of possible reactions.

In this paper, we do not address these enhancements directly. Rather we note that they are all built on top of some version of the SSA at their core. Thus improving the scalability of the SSA itself could benefit any of these approaches.

All of the algorithms discussed thus far solve the generalized problem of random variate generation (RVG) from a dynamic discrete probability distribution. The generated variate determines what "event" takes place in the next time increment. "Dynamic" means the distribution changes each time an event occurs. For biochemical networks, the event is a reaction, and the system is dynamic because the occurrence of a reaction changes the concentration of various species (its reactants and products) and hence the probabilities for other reactions to occur at the next iteration. In this more general context, RVG is a well-studied problem. Devroye[31] provides a classification scheme for RVG and describes a rich compendium of algorithms, including all of the event-selection algorithms commonly used in SSA implementations. Efficient RVG is also a key kernel in discrete-event and kinetic Monte Carlo (KMC) simulators which model phenomena as diverse as factory scheduling (operations research) or grain growth and chemical vapor deposition (materials science). It is worth noting that the classic KMC algorithm for choosing events and the associated timestep, known as the n-fold way or BKL algorithm,[32] is, in fact equivalent to the SSA, though it was formulated independently.

In this paper, we adapt a particular RVG algorithm known as composition and rejection, which has been developed and enhanced in Refs. 33–35, and apply it to the SSA. It is well-suited to the simulation of large biochemical networks, because its scaling is $O(1)$, i.e., the computational cost to perform a reaction is constant, *independent* of N. This surprising result requires only two assumptions be met, both of which we argue in later sections, that are reasonable for biochemical networks. The first is that the ratio of maximum to minimum probability for any two reactions is bounded. The second is that the average number of other reactions directly coupled to each reaction (products of one are reactants of others) does not grow continuously as large numbers of new reactions are added to the network.

In the remainder of the paper, we briefly describe the original SSA, the widely used Gibson/Bruck enhanced algorithm, the new constant-time algorithm, and its implementation details. We compare the computational cost of the logarithmic and constant-time algorithms on a synthetic test suite, showing the new algorithm to be competitive even for small networks, and to perform significantly faster as the network grows to large numbers of reactions.

II. LINEAR TIME ALGORITHM

Consider a collection of molecules of different chemical species in a volume V. The initial count of molecules of each species i is n_i, so that molar concentrations are $c_i = n_i/(N_A V)$, where N_A is Avogadro's number. The species interact via a set of N chemical reactions. The *nth* reaction can be written in familiar form as follows:

$$\sum_{i=1}^{r_n} R_i^n \xrightarrow{k_n} \sum_{i=1}^{p_n} P_i^n,$$

where k_n is a reaction rate constant, R_i^n is a reactant molecule of a particular chemical species, and similarly for product molecules P_i^n. The number of reactants r_n on the left side of the equation can be limited to 0, 1, or 2 without loss of generality, while the number of products p_n can be 0, 1, or any number. The N reactions are "coupled" in the sense that products of each reaction can be reactants of others. The computational task is to evolve the species concentrations c_i over time, assuming the volume is a well-stirred reaction chamber, where each molecule is equally likely to encounter any other molecule.

The continuum formulation of this problem converts reactions to ordinary differential equations (ODEs) and integrate the set of coupled ODEs forward in time, where continuous concentrations are the variables of interest. As an alternative, Gillespie proposed the SSA, which treats individual molecules discretely, and showed it was rigorously equivalent to simulating the tome evolution of the chemical master equation formulated for the system of reacting molecules and, in the limit of large numbers of molecules, to the continuum formulation as well.[2,3]

In the SSA, the system evolves one reaction at a time, changing the counts of reactant and product molecules appropriately, and thus the associated species concentrations. Which reaction occurs next and the time at which it occurs are chosen using random numbers and probabilistic rules that ensure accurate sampling. In the SSA, a "propensity" is computed for each reaction which is proportional to its probability of occurrence relative to other reactions. The propensity is $k_n N_A V$ for a zeroth-order reaction, $n_1 k_n$ for a first-order, and $n_1 n_2 k_n/(N_A V)$ for a second-order reaction, where n_1 and n_2 are the molecular counts of the reacting species, and the

```
(1) Generate two random numbers $r_1$ and $r_2$
(2) $\Delta t = \frac{1}{p_s}\ln(\frac{1}{r_1})$
(3) Find the smallest $m$ such that $r_2 p_s < \sum_{i=1}^{m} p_i$
(4) Perform $mth$ reaction, incrementing reactant/product counts
(5) Compute propensity $p_i$ of each reaction
(6) $p_s = \sum_{i=1}^{N} p_i$
```

FIG. 1. A single iteration of the original SSA, the Gillespie stochastic simulation algorithm, with $O(N)$ scaling in the number of reactions N.

```
(1) Generate two random numbers $r_1$ and $r_2$
(2) $\Delta t = \frac{1}{p_s}\ln(\frac{1}{r_1})$
(3) Search binary tree for smallest $m$ such that $r_2 p_s < \sum_{i=1}^{m} p_i$
(4) Perform $mth$ reaction, incrementing reactant/product counts
(5) Compute propensity $p_i$ of affected reactions
(6) Percolate changed propensities up binary tree, yielding $p_s = \sum_{i=1}^{N} p_i$
```

FIG. 2. A single iteration of SSA-GB, the Gibson/Bruck stochastic simulation algorithm, with $O(\log_2 N)$ scaling in the number of reactions N.

input k_n values have units of M/s, 1/s, and $1/M$ s for zeroth-, first-, and second-order reactions, respectively.

With these definitions, the "direct" SSA is outlined in Fig. 1. A "first-reaction" version of the SSA was also discussed by Gillespie and shown to be exactly equivalent. Its scaling properties are the same as discussed here. Note that as presented in Fig. 1 the following values must be precomputed before the first iteration: p_i=the propensity for each reaction and p_s=the sum of p_i for all N reactions.

Two random numbers are used per iteration, each sampled from a uniform distribution bounded by 0 and 1. The first is used to compute a time increment in step (2). The second is used to pick a reaction in step (3). Conceptually, step (3) can be thought of in the following way. Each propensity represents a short line segment of length equal to p_i. If these segments are concatenated, the resulting long segment has length p_s. If a random point along this long segment is chosen, step (3) determines which short segment the point falls inside of. Step (4) updates molecular counts due to reaction m. Steps (5) and (6) compute new propensities resulting from changed molecular counts, in preparation for the next iteration.

As originally proposed by Gillespie, step (3) scales as $O(N)$ with the number of reactions N, using the following approach. Sum the N propensities in order from 1 to N, adding each in turn, continuing until the the p_m term causes the accumulating sum to exceed $r_2 p_s$. Step (4) scales as $O(1)$ since we assume each reaction has a small bounded number of products. As written, steps (5) and (6) also scale as $O(N)$.

In the nomenclature of random variate generation, step (3) "generates" a random variate from a dynamic discrete probability distribution (the set of propensities), and steps (5) and (6) "update" the distribution. Thus the scaling of the original Gillespie SSA is $O(N)$ (linear) in both its generation and update times. However, a simple enhancement improves the scaling of the update to $O(1)$ (constant). If we assume that each chemical species occurs as a reactant in a small bounded number of reactions (a plausible assumption for biochemical networks), then the number of propensities that need to be updated in step (5) is also small and bounded, i.e., $O(1)$. The sum of step (6) can be similarly updated in $O(1)$ time, yielding an overall update scaling of $O(1)$.

The idea of only updating propensities for affected reactions was formalized as a "dependency graph" by Gibson and Bruck,[21] though others may have implemented similarly efficient forms of steps (5) and (6) before this paper. Reactions are nodes of the graph representing the biochemical network and a directed edge from node i to j exists if a product of reaction i is a reactant of reaction j. Storing such a dependency graph enables a straightforward implementation of an $O(1)$ version of steps (5) and (6).

Another optimization of the direct method, proposed in Cao *et al.*,[22] is to preorder the set of propensities so that large values appear at the beginning of the list. Then the accumulating sum in step (3) is likely to exceed $r_2 p_s$ quickly, yielding a small m. Cao *et al.* argue that this preordering is feasible for some biochemical networks and can be recomputed periodically as concentrations change, leading to a faster algorithm.

We note that when N is small the number of required updates (although fixed) can be close to N. Thus an algorithm whose update scaling is constant is a good choice even if its generation cost scales linearly. Hence the original SSA performed satisfactorily in a computational sense until the size of simulated biochemical networks grew larger. This motivated the algorithm of the next section.

III. LOGARITHMIC TIME ALGORITHM

The key advance of the Gibson/Bruck version of the SSA was to convert it from an algorithm with linear generation time and constant update time to one that is logarithmic in both generation and update time, thus enabling large networks to be simulated more efficiently.[21] The paper mainly focused on enhancements to the first-reaction version of the SSA yielding a "next-reaction" method, but enhancements to the direct SSA were also proposed. The resulting scaling is the same for both algorithms; we discuss the enhanced direct SSA here, which we refer to as SSA-GB.

The SSA-GB algorithm is outlined in Fig. 2. It has the same sequence of steps as in Fig. 1. Steps (3) and (6) now use a binary tree so that random variate generation and the update of the distribution scale more efficiently. Note that step (5) is now the $O(1)$ update discussed in the previous section using a dependency graph, as suggested by Gibson and Bruck.

A binary tree is used to store the set of N reaction propensities, assuming N is a power of 2. Each propensity is a "leaf" in the tree. Pairs of propensities (siblings) are summed to a parent value, stored at a "branch" location in the tree. Pairs of parent values are summed iteratively at the next level (grandparents) until a single "root" value results which is p_s, the sum of all N propensities. The resulting tree has $\log_2 N$ levels. Since $N-1$ partial sums are stored, the entire tree can be stored in $2N$ memory locations. Generalization to a tree where N is not a power-of-two is straightforward, e.g., by padding the list of leaves with zeros.

Step (3) can now be performed in a logarithmic number of operations, yielding an algorithm whose generation time scales as $O(\log_2 N)$. Begin at the root of the tree with a search value $s = r_2 p_s$. If s is less than the left child value p_l, branch to the left keeping s as the search value, else branch to the right using a new search value $s = s - p_l$. This operation is applied successively at each of the $\log_2 N$ levels of the tree, until a specific leaf is arrived at. This is the mth leaf of step (3).

Similarly, in step (6), the effect of each changed propensity on the summed p_s can be computed in a logarithmic number of operations. First, the appropriate leaf value is changed, then its parent value (changed value+sibling value) is recomputed. Then the grandparent value is recomputed and so forth until the root value is recomputed. This operation is performed once for each changed propensity (the number of which is small and bounded); thus, the overall scaling of the update operation is also now logarithmic, i.e., $O(\log_2 N)$.

The overall logarithmic scaling of the Gibson/Bruck enhanced SSA algorithms (both next-reaction and SSA-GB) results in a large performance improvement over the linear time method for networks with even a few dozen reactions. Hence these algorithms are currently widely used in many biochemical network modeling codes.

IV. CONSTANT-TIME ALGORITHM

All the algorithms of the preceding sections are discussed (from a RVG perspective) in Devroye's compendium.[31] Methods linear in generation time and constant in update time (the original Gillespie SSA) are presented in Chapter 3.2.3 (inversion by sequential search). The optimization[22] of Cao *et al.* is described in the same chapter (inversion by sequential search with reorganization). Methods logarithmic in both generation and update time (SSA-GB) are discussed in Chapter 3.3.2 (inversion by binary search). For these methods, Huffman trees are proposed to further reduce the generation time, but this does not change the fundamental logarithmic scaling behavior.

For better performance on networks with a very large number of reactions N, we turn to a class of methods called composition and rejection (CR) algorithms (Chapter 2.4) that are constant in both generation and update time, i.e., their $O(1)$ scaling is independent of the number of reactions.

The rejection idea is illustrated in the left panel of Fig. 3. Consider a set of N reaction propensities, listed along the x-axis. The y-axis height of each bar represents the propensity for that reaction. If we draw a rectangle that bounds the N vertical bars, then a valid algorithm for randomly choosing a reaction is as follows. Let the height of the bounding rectangle be $p_{\max}$. Pick a uniform random integer i from 1 to N. Pick a second uniform random number r from 0 to $p_{\max}$. If $p_i < r$, then reaction i is selected. If not, the selection is "rejected" and the algorithm is repeated. Thus, in the figure, point A would be rejected, while point B would select reac-

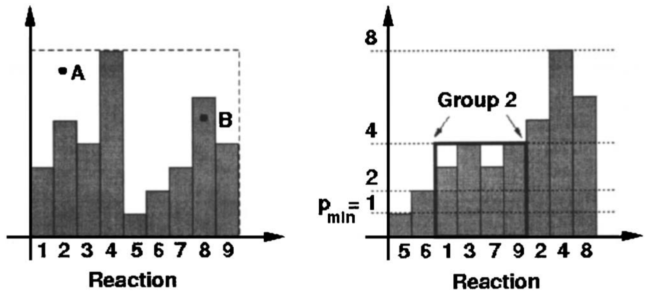

FIG. 3. Composition and rejection algorithm for random variate generation. A reaction is selected from a set of reaction propensities (left) by picking random points (A and B) from a bounding rectangle until a point inside a vertical bar (B) is found. Grouping the propensities by their magnitude (right) makes rejected points less likely.

tion 8. Effectively, this algorithm iterates until a point inside one of the bars is selected, using two random numbers at each iteration. Note that the cost of selecting a test point is independent of N. If the set of bars covers a high fraction of the bounding rectangle's area, the average rejection count per selection will also be small.

Now imagine the set of N propensities are first grouped by their propensity values, as illustrated in the right panel of Fig. 3. In this case, three groups from $p_{\min}$ to $p_{\max}$ are used. The first group (reactions 5 and 6) contains propensities ranging from $p_{\min}$ to $2p_{\min}$, the second group (1,3,7, and 9) from $2p_{\min}$ to $4p_{\min}$, and the third group (2,4, and 8) from $4p_{\min}$ to $p_{\max}$. The selection of a reaction can now be done efficiently via an algorithm composed of two stages (hence the "composition" aspect of the CR algorithm). The first stage selects a group. Let G be the number of groups. If the total propensity of all reactions in a group is p_g and the total propensity of all reactions is $p_s = \Sigma_{g=1}^{G} p_g$, then this requires one random number and a linear scan or binary search of the G values, as discussed in the preceding two sections. Once a group is selected, the reaction within the group is chosen via the rejection procedure, using a rectangle that bounds only the reactions in that group, as illustrated for the second group in the figure. The key point is that by choosing the group boundaries as cascading factors of 2 ($p_{\min}, 2*p_{\min}, 4*p_{\min}$, etc.), we have guaranteed that the area covered by the bars of each group's reactions is greater than half the area of the group's bounding rectangle. Thus, on the average, the selection of each reaction will require less than two iterations of the "rejection" portion of the CR algorithm.

The two stages together constitute the generation portion of the CR algorithm. Its scaling is $O(1)$, independent of N, if the number of groups G is also independent of N. We now argue why this is a valid assumption for biochemical networks. Clearly, for a set of reactions, there is a $p_{\min}$ which can be computed from the propensity formulas for the zeroth-, first-, and second-order reactions, assuming only one molecule of each reactant exists in the volume V. Note that reactions with propensity $p = 0$ need not be included as possible selections. Similarly, one can bound the maximum number of reactant molecules of any species by physical constraints or knowledge of the reaction network. For example, only so many molecules of a given species will be present in a cell. Thus the maximum propensity $p_{\max}$ for the

```
(1)  Generate four random numbers $r_1$ thru $r_4$
(2)  $\Delta t = \frac{1}{p_s}\ln(\frac{1}{r_1})$
(3a) Use $r_2$ to select a group of reactions (composition)
(3b) Use $r_3$ and $r_4$ to select reaction $m$ within the group (rejection)
(4)  Perform $mth$ reaction, incrementing reactant/product counts
(5)  Compute propensity $p_i$ of affected reactions
(6)  Assign affected reactions to new groups, yielding new $p_g$ and $p_s = \sum_{i=1}^{G} p_g$
```

FIG. 4. A single iteration of SSA-CR, the composition and rejection stochastic simulation algorithm, with $O(1)$ scaling, independent of the number of reactions N.

set of reactions is also computable from the reaction rate constants k_i and the maximum molecular counts. The largest number of groups possible is then $G_{\max}=\log_2(p_{\max}/p_{\min})$, though many fewer will likely be required when the SSA model executes. In a biochemical sense, adding new reactions to the model does not change G, assuming their rates are in the same range as those of previous reactions.

G can also be bounded by practical considerations. If $p_{\max}$ is computed, propensities below a chosen threshold value of $p_{\min}$ could be discarded, since statistically speaking, they will not occur frequently enough to impact the network dynamics. For example, if reactions with propensities 1×10^9 times smaller than $p_{\max}$ are discarded, then G is bounded at 30. Alternatively, $p_{\min}$ can be computed, and groups added on-the-fly as $p_{\max}$ grows during a simulation. If G grows too large a larger $p_{\min}$ can be used for future simulations of the same network. In practice, G remains small (around 10–20) for networks we have modeled. We also note that even if G grows slowly with increasing N, this only affects the generation time for the algorithm, which scales as $\log_2(G)$ using a binary search of the group propensities. As shown in the next section, the generation cost is considerably smaller than the update cost, which is independent of the number of groups.

Using the CR algorithm for reaction selection leads to a constant-time algorithm that is exactly equivalent to the original Gillespie SSA. We refer to this new algorithm as SSA-CR; it is outlined in Fig. 4. Before the first iteration, reactions are assigned to groups, the summed propensities for each group p_g are computed, as is the total summed propensity $p_s=\sum_{g=1}^{G}p_g$.

Steps (3a) and (3b) are the reaction selection procedure outlined above. Step (3a) is the same as step (3) in either Fig. 1 or 2, except that now the selection is from G groups instead of N reactions, thus it scales as $O(G)$ or $O(\log_2 G)$. Step (3b) may require additional random numbers if rejection occurs, but this will happen less than half the time (on average), regardless of the distribution of reaction propensities. For randomly distributed propensity values, it will occur only one quarter of the time (75% area coverage by the vertical bars within a group). Note that a constant-time implementation of step (3b) requires that the mth reaction in a group can be accessed in a one-step operation. This is easily done by having each group maintain a linear list of its reactions, which can simply be integer indices from 1 to N.

Once a reaction has been performed in steps (4) and (5), the update portion of the SSA-CR algorithm is performed in step (6). The new propensity of each dependent reaction is compared to its old value. If the reaction stays in the same group, only the group sum p_g and total sum p_s need updating. If the group assignment has changed, the reaction is deleted from the old group and added to the new group, then p_g and p_s values for both groups are updated. Adding/deleting a reaction to/from a group is a constant-time operation. For addition, a new index is added to the end of the group list and the group size is incremented. For deletion, the reaction at the end of the group list replaces the deleted reaction and the group size is decremented. Thus the update portion of SSA-CR also scales as $O(1)$.

There are two requirements implicit in this scaling result. The first is that a dependent reaction can be located within its group in a one-step operation. This is easily done by having each of the N reactions store two integers: its current group assignment and its location within that group. The second is the assumption that the average number of dependencies per reaction does not grow continually larger as the number of reactions grows. Since extremely large networks have not been formulated, this assumption is hard to test empirically, but we note that any implementation of the SSA will suffer in performance if this is not the case, since the update time is necessarily proportional to the average number of dependencies.

Overall, in addition to propensities, the memory cost of the SSA-CR is three integers per reaction. This is similar to the SSA-GB memory cost for its binary tree of one additional floating point value per reaction.

Nothing in the preceding discussion requires that propensity boundaries between groups be chosen such that the ratio of upper and lower bounds for a group is $r=p_{\text{upper}}/p_{\text{lower}}=2$. If $r<2$, the cost of the rejection portion of the algorithm would decrease (less rejections) while the cost of the composition portion would increase (more groups). The converse would be true for $r>2$. In either case, both portions would still be constant-time operations. A practical reason to use groups with $r=2$, as illustrated in Fig. 3, is that calculating which group a newly computed propensity value p falls into can be done in a single operation by calling a standard C math library function, namely, frexp$(p/p_{\max},\&\text{gneg})$, which returns the negative of the group ID as the variable gneg.

V. PERFORMANCE AND DISCUSSION

While the CR version of the Gillespie SSA has better theoretical performance than the Gibson/Bruck version, actual performance depends on prefactors of the scaling terms and other implementation details. For SSA-CR, the prefactors also depend on the number of groups G.

To test the algorithms for widely varying numbers of reactions N, we generated random reaction networks. We represented the network of reactions as an $1d$ array of N doubles representing reaction propensities and a $2d$ $N\times M$ array of connectivity for each of the propensities representing the dependency of the reactive species. For each of N reactions, initial propensities varying by a factor of a 1

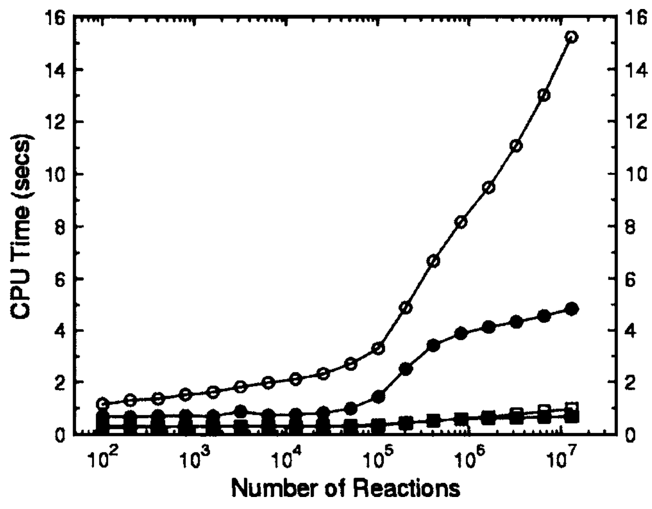

FIG. 5. CPU time in seconds for 1 000 000 iterations of the logarithmic-time Gibson/Bruck (open symbols) and constant-time composition and rejection (filled symbols) versions of the direct-method Gillespie stochastic simulation algorithm. Squares are generation times; circles are update times; the total time is the sum of generation and update. This is the high-memory version of the test program which stores a reaction dependency graph.

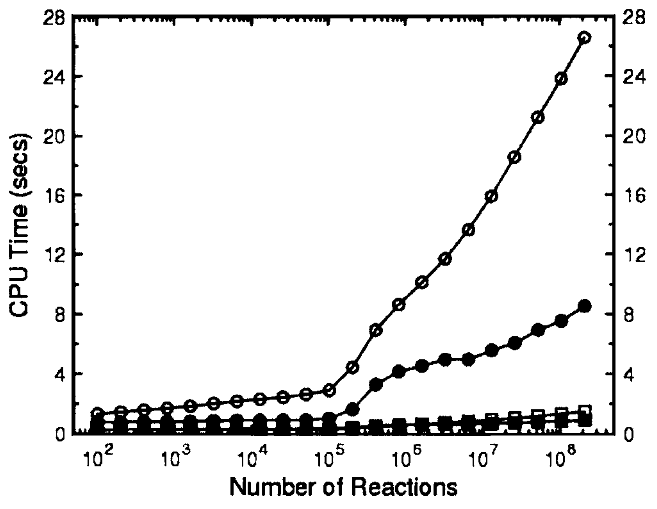

FIG. 6. CPU time in seconds for 1 000 000 iterations of the Gibson/Bruck and composition and rejection versions of the direct-method Gillespie stochastic simulation algorithm. The symbols have the same meaning as in Fig. 5. This is the low-memory version of the test program which does not store a reaction dependency graph. These runs were performed on a Xcon processor.

$\times 10^6$ $(1.0\times 10^6-1.0)$ were chosen randomly from an exponential distribution. For SSA-CR this created approximately 20 groups, with roughly equal numbers of reactions per group. Each reaction affected M other randomly chosen reactions where M is a uniformly distributed integer from 1 to 30. Each time one reaction is executed, the propensity of each of the M affected reactions was altered and the effect of the change on the overall probability distribution was accounted for, before the next reaction was selected. Specifically, the new propensity of each of the M reactions was set to a uniform random value between 95% and 105% of its current propensity.

We created two versions of our test program. The first, which we call a high-memory version, stores a precomputed random dependency graph, where the list of M affected reactions is generated in advance and stored for each of the N reactions. This requires 15 integers per reaction (on average), which limits the problem size that can be run for large N. So we also created a second, low-memory, version which generates M random dependencies on-the-fly, each time a reaction is selected. As before, M is a uniform random integer between 1 and 30. This second scheme could not be used for modeling an actual biochemical network, but allows the scaling of the SSA-GB and SSA-CR algorithms to be tested for much larger N.

Figure 5 shows timings for the high-memory version of the test program, which stores a dependency graph. Simulations of networks varying in size from $N=100$ to $N=100\times 2^{17}\approx 13.1\times 10^6$ reactions were run with the SSA-GB and SSA-CR algorithms outlined in Figs. 2 and 4. The CPU time is in seconds for 1 000 000 iterations of each algorithm, i.e., 10^6 reactions are executed. The generation and update times for both algorithms are shown; the total time is simply the sum of generation and update for either algorithm. These timing tests were run on a single processor (core) of a Dell 690 desktop machine with two 2.66 GHz quad-core Xeon chips and 16 GB of memory. The two algorithms were implemented in C++, though simple C-style data structures and coding syntax were used for the key operations.

The generation time for both algorithms is roughly equal and nearly constant for networks of any size. Both algorithms are dominated by the cost of updating, since there are many dependencies per reaction. For $N<100\,000$ reactions, the logarithmic and constant scaling of the update time for the two algorithms are evident; logarithmic dependence is a sloped line on a log/linear plot. Around $N=100\,000$, both algorithms begin to run slower due to cache effects when the data structure (tree, groups) for storing propensities no longer fits in second-level cache.

For SSA-GB this is manifested by a logarithmic dependence with a steeper slope. For SSA-CR, the new slope is not as flat as for smaller problems. As we discuss below, this is not due to the algorithm, which still has $O(1)$ or constant-time scaling, but to memory-access issues for very large problem sizes.

Figure 6 shows timings with the low-memory version of the test program, where the reaction dependencies are generated on-the-fly rather than stored. The update times are now somewhat slower than in Fig. 5 (note the difference in vertical scale) due to the cost of generating dependencies each time a reaction executes. But we can now run networks up to size $N=100\times 2^{21}\approx 210\times 10^6$ reactions. The difference between logarithmic- and constant-time scaling is now more evident for very large N.

To address the apparent nonconstant scaling of the update time for the SSA-CR algorithm for large N, we ran the same low-memory tests on a single processor (core) of a Cray XT3 with dual-core 2.4 GHz Opteron chips, each with 4 GB of memory. The operating system on the XT3 has the run-time option to configure itself with either small memory pages (4 kB) or large pages (2 MB). On most Linux machines, including the Dell desktop machine of Fig. 5, small memory pages are the default. Large-memory pages are an

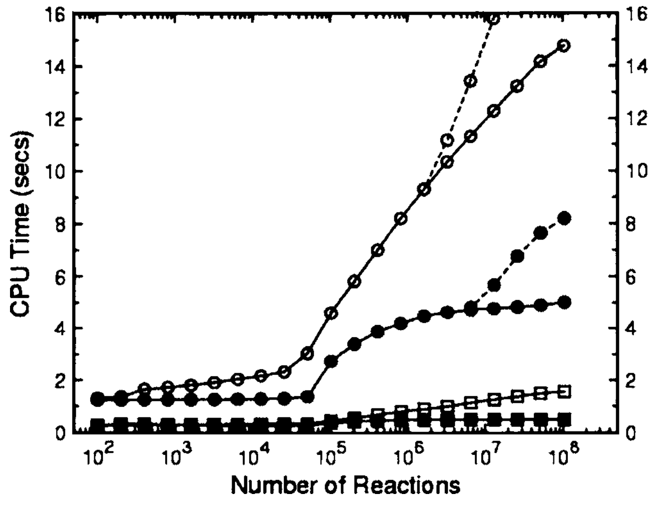

FIG. 7. Same timing runs as Fig. 6 but on an Opteron processor. The dashed-line data points are for Linux configured with small-size memory pages; the solid-line data are for large-size memory pages. The latter evidences the constant-time scaling of the composition and rejection algorithm.

available option, but typically requires changes in the setup procedure for the OS and a reboot of the machine.

The results for runs with small and large pages on the Opteron processor are shown in Fig. 7. The difference is only significant for the update times of the largest runs; elsewhere the dotted- and solid-line data overlay almost exactly. The update timings for small pages (dotted lines) are qualitatively the same as the Xeon timings in Fig. 6, with a nonconstant upturn for large N runs of the SSA-CR algorithm. Note that the vertical scales of Figs. 6 and 7 are different; the last dotted-line data point for a SSA-GB update (open circles) is off the plot at 22.3 s.

The large-memory page timings illustrate the constant-time scaling of the SSA-CR algorithm (solid lines and symbols), up to $N=100\times 2^{20}\approx 105\times 10^6$ reactions. The large-memory pages improve the speed of both the SSA-GB and SSA-CR algorithms. The reason is that memory access to huge data sets (several gigabytes for the problems with largest N) in Linux is through a translation lookaside buffer (TLB) which is a list of page addresses. For small pages the size of the TLB becomes large for a huge data set, so that access to the TLB itself causes additional cache misses. For large pages, the TLB still fits in cache and overall memory access is more efficient.

Two other features of Fig. 7 are interesting to note. First, the constant-time versus logarithmic scaling of even the less-costly generate time for the SSA-CR and SSA-GB algorithms is apparent. Second, the slow-down around $N=50\,000$ due to the size of the data set exceeding second-level cache happens for smaller problems than in Fig. 6, due to a smaller cache on the Opteron (1 Mb) versus the Xeon (4 Mb).

Our main conclusion is not the fine details of the performance plots, since these may depend on specific processor attributes or optimized implementations of the algorithms. Rather we focus on the fact that the SSA-CR algorithm is competitive or faster than the SSA-GB algorithm across a large range of biochemical network sizes and exhibits the desired constant-time scaling behavior. We again emphasize that the CR algorithm is applicable not only to simulations of biochemical networks, but can be used for efficient event selection in any large-scale kinetic Monte Carlo model whose event probabilities and interevent dependencies meet the same assumptions discussed in this paper.

While biochemical networks with a 1×10^6 or more reactions are not common today, due to limited experimental or bioinformatic data, this is likely to change in the future. As this occurs, the use of $O(1)$ reaction-selection algorithms, such as the composition and rejection scheme described here, will become increasingly advantageous.

ACKNOWLEDGMENTS

The authors thank Kevin Pedretti at Sandia for help in understanding the memory page-size issues discussed in the performance section. We thank one of the reviewers for calling our attention to the related recent work of Danos and Yang.[29,30] This work was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy, under Contract No. DE-AC04-94AL85000.

We note that since this paper was submitted, a paper by Schulze was published,[36] which discusses composition-rejection style algorithms in the kinetic Monte Carlo context, similar to the algorithm presented in Sec. IV.

The software discussed in this paper will be available for download from the WWW site www.sandia.gov/~sjplimp/download.html about the time the paper is published. It is part of the open-source SPPARKS kinetic Monte Carlo code.

[1] A. Arkin, J. Ross, and H. H. McAdams, Genetics **149**, 1633 (1998).
[2] D. T. Gillespie, J. Chem. Phys. **22**, 403 (1976).
[3] D. T. Gillespie, J. Phys. Chem. **81**, 2340 (1977).
[4] I. Bose, R. Karmakar, and S. Roy, Science **297**, 1183 (2002).
[5] C. V. Rao and A. P. Arkin, Annu. Rev. Biomed. Eng. **3**, 391 (2000).
[6] C. Versari and N. Busi, *Computational Methods in Systems Biology* (Edinburgh, Scotland, 2007), pp. 80–95.
[7] F. Ciocchetta, J. Hillston, M. Kos, and D. Tollervey, in *Computational Methods in Systems Biology* (Edinburgh, Scotland, 2007), pp. 32–47.
[8] H. H. McAdams and A. Arkin, Proc. Natl. Acad. Sci. U.S.A. **94**, 814 (1997).
[9] M. B. Elowitz, A. J. Levine, E. D. Siggia, and P. S. Swain, Science **297**, 1183 (2002).
[10] W. J. Blake, M. Kaern, C. R. Cantor, and J. J. Collins, Nature (London) **422**, 633 (2003).
[11] J. M. Raser and E. K. O'Shea, Science **304**, 1811 (2004).
[12] L. S. Weinberger, J. C. Burnett, J. E. Toettcher, A. P. Arkin, and D. V. Schaffer, Cell **122**, 169 (2005).
[13] M. Samoilov, S. Plyasunov, and A. P. Arkin, Proc. Natl. Acad. Sci. U.S.A. **102**, 2310 (2005).
[14] D. A. Bader and K. Madduri, "A Graph-Theoretic Analysis of the Human Protein-Interaction Network Using Multi-core Parallel Algorithms," *Parallel and Distributed Processing Symposium*, IEEE International, 2007, pp. 1–8.
[15] J. R. Faeder, M. L. Blinov, B. Goldstein, and W. S. Hlavacek, Complexity **10**, 22 (2005).
[16] L. Lok and R. Brent, Nat. Biotechnol. **23**, 131 (2005).
[17] W. S. Hlavacek, J. R. Faeder, M. L. Blinov, R. G. Posner, M. Hucka, and W. Fontana, Systems Biology, IEE Proc. **2**, 5 (2005).
[18] T. Wilhelm, H.-P. Nasheuer, and S. Huang, Mol. Cell Proteomics **2**, 292 (2003).
[19] S. Lindskog, Pharm. Ther. **74**, 1 (1997).

[20] S. K. Banerjee, I. Kregar, V. Turk, and J. A. Rupley, J. Biol. Chem. **248**, 4786 (1973).
[21] M. A. Gibson and J. Bruck, J. Chem. Phys. **104**, 1876 (2000).
[22] Y. Cao, H. Li, and L. Petzold, J. Chem. Phys. **121**, 4059 (2004).
[23] D. T. Gillespie, J. Chem. Phys. **115**, 1716 (2001).
[24] M. Rathinam, L. R. Petzold, Y. Cao, and D. T. Gillespie, J. Chem. Phys. **119**, 12784 (2003).
[25] Y. Cao, D. Gillespie, and L. Petzold, J. Comput. Phys. **206**, 395 (2005).
[26] W. E. , D. Lin, and E. Vander-Eijnden, J. Chem. Phys. **123**, 194107 (2005).
[27] D. T. Gillespie, Annu. Rev. Phys. Chem. **58**, 35 (2007).
[28] A. Samant and D. G. Vlachos, J. Chem. Phys. **123**, 144114 (2005).
[29] J. Yang, M. I. Monine, J. R. Faeder, and W. S. Hlavacek, Report number: LA-UR-8103, arXiv: 0712.3773v2 [q-bio.QM].
[30] V. Danos, J. Feret, W. Fontana, and J. Krivine, in *Proceedings of APLAS 2007*, edited by Z. Shao (Springer Berlin, Heidelberg, 2007), Vol. 4807, pp. 83–97.
[31] L. Devroye, *Non-uniform Random Variate Generation* (Springer-Verlag, New York, 1986).
[32] A. B. Bortz, M. H. Kalos, and J. L. Lebowitz, J. Comput. Phys. **17**, 10 (1975).
[33] B. L. Fox, ORSA J. Comput. **2**, 126 (1990).
[34] S. Rajasekaran and K. W. Ross, ACM Trans. Model. Comput. Simul. **3**, 1 (1993).
[35] T. Hagerup, K. Mehlhorn, and I. Munro, Lect. Notes Comput. Sci. **700**, 253 (1993).
[36] T. P. Schulze, J. Comput. Phys. **227**, 2455 (2008).