On the Rejection-based Algorithm for Simulation and Analysis of Large-Scale Reaction Networks

Vo Hong Thanh,[1] Roberto Zunino,[2] and Corrado Priami[3]
[1)]*The Microsoft Research - University of Trento Centre for Computational and Systems Biology, Piazza Manifattura 1, Rovereto 38068, Italy.*[a)]
[2)]*Department of Mathematics, University of Trento, Italy.*[b)]
[3)]*Department of Mathematics, University of Trento, Italy and*
The Microsoft Research - University of Trento Centre for Computational and Systems Biology, Piazza Manifattura 1, Rovereto 38068, Italy.[c)]

(Dated: 6 June 2015)

Stochastic simulation for *in-silico* studies of large biochemical networks requires a great amount of computational time. We recently proposed a new exact simulation algorithm, called the rejection-based stochastic simulation algorithm (RSSA) [J. Chem. Phys. 141(13):134116 (2014)], to improve simulation performance by postponing and collapsing as much as possible the propensity updates. In this paper, we analyze the performance of this algorithm in detail, and improve it for simulating large-scale biochemical reaction networks. We also present a new algorithm, called simultaneous RSSA (SRSSA), which generates many independent trajectories simultaneously for the analysis of the biochemical behavior. SRSSA improves simulation performance by utilizing a single data structure across simulations to select reaction firings and forming trajectories. The memory requirement for building and storing the data structure is thus independent of the number of trajectories. The updating of the data structure when needed is performed collectively in a single operation across the simulations. The trajectories generated by SRSSA are exact and independent of each other by exploiting the rejection-based mechanism. We test our new improvement on real biological systems with a wide range of reaction networks to demonstrate its applicability and efficiency.

Keywords: Computational biology, Stochastic simulation, Rejection-based stochastic simulation algorithm.

I. INTRODUCTION

Stochastic modelling of biochemical reaction models the system state as a vector of species populations. A reaction between species is a random event with probability proportional to a *propensity* dependent on the reaction kinetics. The temporal evolution of biochemical networks can be realized by an exact simulation procedure called the stochastic simulation algorithm (SSA)[1,2], also known as the Direct Method (DM). Each SSA simulation step selects a reaction to fire with a probability proportional to its propensity, according to which the system jumps to a new state. Then, reaction propensities are updated to reflect the changes in the system state.

There are two main factors affecting performance of SSA: 1) searching for next reaction events, and 2) updating propensities of reactions. Furthermore, due to the inherent randomness in the simulation, many simulation runs should be done to have a reasonable statistical estimation that further increases the total simulation time. Many formulations have been introduced to accelerate the stochastic simulation. The Next Reaction Method (NRM)[3] exploits a special indexed structure, i.e., a binary heap, to store and extract smallest (absolute) putative times. Extracting the smallest time from the heap requires constant time, while updating requires logarithmic time. Furthermore, NRM uses a dependency graph, which points out which propensities need to be updated after a reaction firing, to reduce the number of propensity updates. The Optimized Direct Method (ODM)[4] and the Sorting Direct Method (SDM)[5] accelerate the search for next reaction of SSA by sorting reactions in descending order of propensities. The search for next reaction can further be improved by dividing reactions into groups[6,7] and performing two steps: 1) selecting the group, then 2) locating the reaction within that group. If groups are split into subgroups recursively until a group contains only two reactions, we obtain a tree structure with reactions on the leaves. Then, the search for the next reaction is done by traversing the tree[8–11]. The SSA with Composition Rejection algorithm (SSA-CR)[7,12] exploits the reaction grouping strategy and rejection-based mechanism to improve the search for next reaction. SSA-CR groups reactions with propensities between b^{i-1} and b^i (where b is a chosen base e.g., $b = 2$) into a group i. The search for the next reaction in group i is done through a rejection test with the hat function b^i. The search for next reaction firings by SSA-CR is thus propositional to the number of groups, depending on the ratio between the highest and lowest propensities. Thus, if the number of groups is bound by a small constant, the asymptotic time complexity of the search for the next reaction in SSA-CR is constant time. After a reaction firing, the propensities of affected reactions are updated and moved to appropriate groups. If the number of affected reactions is large and their propensities vary significantly, SSA-CR has to frequently update propensities and its underlying data structure, hence decreasing its overall performance[6]. Typically, the cost of propensity updates done by SSA contributes around 65% to 85%, and in some peculiar cases even up to 99%, of the entire simulation cost. Reducing propensity updates then provides an improvement for the simulation performance. An attempt to re-

[a)]Electronic mail: vo@cosbi.eu
[b)]Electronic mail: roberto.zunino@unitn.it
[c)]Electronic mail: priami@cosbi.eu

duces propensity update cost is the Partial-propensity Direct Method (PDM)[13–15] where propensities are factoring out by common reactants. Propensities of reactions with the shared reactant will be updated in one operation. However, due to a special form of the partial propensities, the reactant grouping approach is limited to class of reactions involving at most two reactants and their propensities must be in a form which can be factorized (e.g., mass-action)[14]. Although non-elementary reactions could be decomposed into elementary reactions, wet-lab experiments can more easily measure the propensity of the non-elementary reaction than the propensities of the intermediate elementary reactions that are involved in. When only the non-elementary reactions are measured, then simulation can only be performed using a complex propensity function. The Michaelis-Menten kinetics[16], for example, is widely used to model enzymatic reactions in biological systems.

We have recently proposed a new exact stochastic algorithm called Rejection-based Stochastic Simulation Algorithm (RSSA)[17,18] to accelerate the stochastic simulation. Our approach aims to reduce the propensity updates but is not relying on any specific form of reactions in the system. RSSA is thus able to simulate any types of reaction (e.g., non-elementary reactions). RSSA is specifically tailored for reaction networks in which propensity computations are time-consuming (e.g., complex propensity such as Michaelis-Menten rate function). During the simulation of RSSA, many propensity updates are completely avoided. Specifically, RSSA abstracts the propensity of a reaction with an interval including all possible concrete propensity values. The propensity bounds of reactions are derived by specifying an arbitrary bound on the population of each species (the choice of which affects the performance, but not the exactness of the results). RSSA uses these propensity bounds to select the next reaction firing in two steps. First, a candidate reaction is randomly chosen proportionally to its propensity upper bound. The selected candidate is then inspected through a rejection test to ensure that it fires with the same probability determined by SSA. The validation step postpones the evaluation of the exact propensity of the candidate reaction by exploiting its propensity lower bound. The exact propensity will be evaluated only if needed. The candidate reaction is either fired or (with low probability) rejected. If it is accepted to fire, only the state is updated, without recomputing the propensity except in uncommon cases. New propensity bounds are recomputed only when the population of a species exits the chosen bound. In case of rejection, a new candidate reaction is selected.

In this paper, we analyze the computational cost of RSSA and introduce efficient formulations to improve its performance for simulating large-scale biochemical reaction networks. We study how the search procedure applied to select candidates affects the simulation performance. We also focus on controlling the bounds of population of species which indirectly affect the propensity bounds and the acceptance probability of a candidate reaction to adaptively optimize itself.

The second contribution of this paper is a new algorithm, called simultaneous rejection-based stochastic simulation algorithm (SRSSA), that exploits the rejection-based principle to efficiently generate multiple independent trajectories simultaneously in one simulation run. SRSSA uses the same propensity bounds across simulations to select the next reaction firings instead of each one for separated simulation runs. The simulation performance is improved by reducing and lumping up together the computing of propensity bounds into one operation. The generated trajectories by SRSSA are still exact and independent of each other.

This paper is organized as follows. The next section reviews the standard approaches for stochastic simulation of biochemical reaction systems, and recalls the basic ideas behind the RSSA algorithm. Section III analyzes the performance of RSSA and its proposed improvements. Section IV presents our new SRSSA algorithm. Section V presents the experimental results of our improvements on concrete models in a range of problem sizes and complexities to demonstrate its applicability and efficiency. The concluding remarks are in section VI.

II. STOCHASTIC SIMULATION BACKGROUND

We consider a well-mixed volume containing n species denoted as $S_1 \dots S_n$. The state of the system is represented by a population vector $X(t) = (X_1(t), ..., X_n(t))$ where $X_i(t)$ denotes the population of species S_i at a time t. Species can interact through m reactions $R_1 \dots R_m$. The probability that a reaction R_j fires in the next infinitesimal time $t + dt$ is $a_j(X(t))dt$ where $a_j(X(t))$ is called the reaction propensity[1]. The propensity $a_j(X(t))$ is roughly proportional to the number of possible combinations of reactants involved in R_j and its kinetics information. Hereafter, we use a_j instead of $a_j(X(t))$ for a shorthand.

If a particular reaction R_μ is selected to fire, the state changes according to the state change vector v_μ, which expresses the changes in population of species involved in R_μ. The state transition of the system is therefore modeled as a (continuous-time) jump Markov process. The probability distribution of the system is completely described by the chemical master equation (CME)[19]; however, an analytic solution of CME is hard to find, unless the system is rather simple. Simulation is often the choice to construct possible realizations of CME. The stochastic simulation algorithm (SSA), in particular, is an exact method to sample temporal behavior encoded in CME.

SSA[1,2] realizes the next state by simulating the joint probability density function (pdf) $p(\tau, \mu)$ with $p(\tau, \mu)d\tau$ being the probability that a reaction R_μ fires in the next infinitesimal time $t + \tau + d\tau$, given the state $X(t)$ at time t. Eq. 1 gives a closed form of $p(\tau, \mu)$:

$$p(\tau, \mu) = a_\mu exp(-a_0\tau) \tag{1}$$

where $a_0 = \sum_{j=1}^{m} a_j$. Note that the reaction R_μ fires with corresponding discrete probability a_μ/a_0 and the firing time τ is exponentially distributed with parameter a_0.

SSA samples the pdf $p(\tau, \mu)$ and constructs a simulation trajectory as follows. It computes m propensities a_j for $j = 1 \dots m$ at beginning. Then, for each simulation step the next

reaction firing R_μ and its firing time τ are realized from Eq. 1 by:

$$\tau = \frac{1}{a_0} ln\left(\frac{1}{r_1}\right) \quad (2)$$

$$\mu = \text{smallest reaction index such that: } \sum_{j=1}^{\mu} a_j > r_2 a_0 \quad (3)$$

where r_1 and r_2 are two random numbers generated from a uniform distribution $U(0,1)$. The state is updated according to the selected reaction R_μ and moves to a new state $X(t+\tau) = X(t) + v_\mu$. The propensities are updated to reflect the changes in the system state. A reaction dependency graph is often used to decide which propensities need to be updated after a reaction firing.

A. Rejection-based Stochastic Simulation Algorithm

RSSA is an exact simulation algorithm which generates trajectories with the same statistical distribution as SSA. In fact, RSSA exactly samples the pdf $p(\tau, \mu)$ in Eq. 1, a reaction R_μ is selected with probability a_μ/a_0 and its firing time is exponentially distributed with parameter a_0. A complete proof for the exactness of RSSA is in Thanh *et al.*[17]. RSSA accelerates the simulation by reducing the number of propensity updates. In most of the simulation steps, RSSA does not require to update propensities, hence reducing the average number of propensity updates. RSSA is summarized in Algorithm. 1.

RSSA computes for each reaction a propensity lower bound $\underline{a}_j$ and an upper bound $\overline{a}_j$ for $j = 1 \dots m$ and uses these propensity bounds to select the next reaction firing. These propensity bounds are derived by imposing a bound on the population of each species in the state. For species S_i, a lower bound $\underline{X}_i$ and an upper bound $\overline{X}_i$ is defined around its current population $X_i(t)$. The population bounds for each species S_i could be chosen arbitrarily around its population without affecting the correctness of the algorithm. The state therefore satisfies $\underline{X} \leq X(t) \leq \overline{X}$ for each species. The population interval $[\underline{X}, \overline{X}]$ is called the *fluctuation interval* or *abstract state*. The invariant $\underline{a}_j \leq a_j \leq \overline{a}_j$ holds for all reaction R_j with $j = 1 \dots m$ when $X(t) \in [\underline{X}, \overline{X}]$. The propensity lower/upper bounds are chosen to be the minimum/maximum of the propensity function a_j over the fluctuation interval $[\underline{X}, \overline{X}]$. If the propensity a_j increases whenever the species population increases, its minimum and maximum values correspond to the evaluation of a_j at the lower extreme and upper extreme of the species population interval, respectively. For example, if a_j follows the mass action kinetics or the Michaelis-Menten kinetics where the monotonicity holds, we simply let $\underline{a}_j = a_j(\underline{X})$ and $\overline{a}_j = a_j(\overline{X})$. If a_j is a complex function, one can apply numerical techniques, e.g., interval analysis[20], to compute the bounds for propensity. The exact minimum and maximum, however, are not really needed. A reasonable tight bound for the propensity over the fluctuation interval is enough for the simulation, but the next reaction firing is always selected with the right probability.

Algorithm 1: Rejection-based SSA (RSSA)

procedure: rssa
output: a trajectory of the reaction network
initialize time $t = 0$ and state $X = x_0$
while $(t < T_{max})$ **do**
  define fluctuation interval $[\underline{X}, \overline{X}]$ of state X
  compute propensity upper bound $\overline{a_j}$ and lower bound $\underline{a_j}$ for each reaction R_j for $j = 1 \dots m$
  compute the total propensity upper bound $\overline{a_0} = \sum_{j=1}^{m} \overline{a_j}$
  repeat
    set $u = 1$
    set $accepted =$ **false**
    repeat
      generate three random numbers $r_1, r_2, r_3 \sim U(0,1)$
      select minimum index μ satisfied $\sum_{j=1}^{\mu} \overline{a_j} > r_1 \overline{a_0}$
      if $r_2 \leq (a_\mu / \overline{a_\mu})$ **then**
        $accepted =$ **true**
      else
        evaluate a_μ with state X
        if $r_2 \leq (a_\mu / \overline{a_\mu})$ **then**
          set $accepted =$ **true**
        end if
      end if
      set $u = u \cdot r_3$
    until $accepted$
    compute firing time $\tau = (-1/\overline{a_0}) \ln(u)$
    update time $t = t + \tau$ and state $X = X + v_\mu$
  until $(X \notin [\underline{X}, \overline{X}])$
end while

Having propensity bounds, a candidate reaction R_μ is selected with probability $\overline{a_\mu}/\overline{a_0}$ where $\overline{a_0} = \sum_{j=1}^{m} \overline{a_j}$. RSSA realizes the candidate reaction by linearly accumulating propensity upper bounds until it finds the smallest reaction index μ satisfying the inequality: $\sum_{j=1}^{\mu} \overline{a_j} > r_1 \cdot \overline{a_0}$ where r_1 is a random number in $U(0,1)$.

The candidate reaction R_μ then enters a rejection test for validation with success probability $a_\mu/\overline{a}_\mu$. In other words, we toss a (biased) coin with success probability $a_\mu/\overline{a}_\mu$. If the toss succeeds, we accept the candidate R_μ to fire, otherwise we reject it. The efficient simulation of this coin toss, however, is tricky since we do not know the exact value of the propensity a_μ in advance, and we want to avoid computing it as much as possible. To achieve that, we draw a random number $r_2 \sim U(0,1)$. We then check whether $r_2 \leq \underline{a}_\mu/\overline{a}_\mu$, which does not require us to compute a_μ. If the check succeeds, then we know that $r_2 \leq \underline{a}_\mu/\overline{a}_\mu \leq a_\mu/\overline{a}_\mu$, hence we can accept R_μ. Only when this test fails we indeed compute a_μ, and then test r_2 against $a_\mu/\overline{a}_\mu$. The computation of a_μ is infrequently performed when $\underline{a}_\mu/\overline{a}_\mu$ is close to 1, which is often the case in practice. If R_μ is accepted, its firing time is then computed. Otherwise, a new candidate reaction is selected.

The firing time τ of the accepted reaction R_μ is generated following an $Erlang$ distribution. This distribution is chosen to be faithful with SSA. The key idea is that each candidate selection step corresponds to a stochastic transition with total rate $\overline{a}_0$. Such transition can cause the state $X(t)$ to ei-

ther move to the new state $X(t+\tau) = X(t) + v_\mu$ if candidate R_μ is accepted or perform a self-loop in the current state $X(t+\tau) = X(t)$ if candidate R_μ is rejected. Hence, if we perform k trials before we accept (i.e., we rejected $k-1$ candidates), we need to advance the time according to the sum of k independent stochastic transitions of rate $\overline{a}_0$, which is an $Erlang$ distribution. The $Erlang$-distributed firing time τ is sampled as:

$$\tau = \sum_{i=1}^{k} (-1/\overline{a_0}) \ln(u_i) = (-1/\overline{a_0}) \ln(\prod_{i=1}^{k} u_i) \qquad (4)$$

where u_i is a random number from $U(0,1)$. RSSA implements this sampling technique by multiplying the variable u in every validation steps by a uniform random quantity r_3 until a trial succeeds.

Knowing the reaction and its firing time, the state is updated accordingly. RSSA postpones recomputing propensity bounds if the state is confined in its fluctuation interval. Thus, it checks whether the condition $X(t) \in [\underline{X}, \overline{X}]$ holds after the state is updated. If the condition is true, which is often the case, the next simulation step is performed. In the uncommon case in which the state is outside the current fluctuation interval, i.e. $X(t) \notin [\underline{X}, \overline{X}]$, a new fluctuation interval is defined. At that time, new propensity bounds for reactions are derived as well. We can reduce the number of reactions having to recompute their propensity bounds by applying a Species-Reaction (SR) dependency graph[17]. The SR dependency graph shows which reactions should recompute their propensity bounds when a species exits its fluctuation interval. Thus, only a subset of reactions requires to recompute propensity bounds.

III. PERFORMANCE ANALYSIS AND IMPROVEMENTS FOR RSSA

This section analyzes the performance of RSSA in generating a simulation trajectory. We discuss the factors affecting the simulation performance and study formulations to improve its efficiency. We measure the computation cost in terms of the average CPU time. In our discussion we use the O-notation to express the time complexity.

A. Computational cost of RSSA

Let T_{RSSA}^{search} be the average search time for a candidate reaction and T_{RSSA}^{update} be the average update time after a reaction fires. The average simulation step time T_{RSSA} is expressed as:

$$T_{RSSA} = \alpha T_{RSSA}^{search} + T_{RSSA}^{update}/\beta + O(1) \qquad (5)$$

where α is the average number of times the search conducted until the candidate reaction is accepted and, respectively, β is the average number of skipped updates during the simulation. α is equal to the reciprocal average acceptance probability of a candidate reaction i.e., $\alpha = \overline{a_0}/a_0$. β is the average frequency of $X(t) \in [\underline{X}, \overline{X}]$. The additional constant cost $O(1)$ in Eq. 5 denotes the CPU time for after-simulation data handling. Although this processing time may contribute a large portion to the simulation time, especially for simulating small models, it depends on the operating system and is the same for all algorithms so that we can still assume it to be a constant.

We can improve the overall performance of RSSA by reducing the search time T_{RSSA}^{search} that depends on the search procedures applied to realize a candidate reaction. The search strategy used in the basic RSSA is equivalent to a linear search. Its main advantage is that it does not require to build any complex data structure in advance. Indeed, in an implementation we only need an array of size m to store propensity upper bounds $\overline{a_j}$ for $j = 1 \ldots m$. However, the time complexity of the search is linear w.r.t. the number of reactions, i.e., $T_{RSSA}^{search} = O(m)$. The search performance can be improved by sorting the propensity upper bounds in decreasing order; however, the worst case complexity still remains linear. We discuss options for implementing fast search procedures that are different in speed and code simplicity in the following section. We also study the impact of their running times to be able to tune the performance for each specific problem.

An update step of RSSA is composed of defining a new fluctuation interval and recomputing propensity bounds of reactions. Defining new fluctuation intervals for each species whose population moves out of its current population bounds after a reaction firing is a constant because only a small number of species are involved in that reaction. Reactions that need recomputing propensity bounds when an involved species moves out of their population bounds are retrieved from the SR dependency graph. Let k be the average number of reactions in the SR dependency graph that needs recomputing propensity bounds, the complexity of update of propensity bounds in RSSA is therefore $T_{RSSA}^{update} = O(k)$. We remark that the propensity updates in RSSA are performed infrequently and controlled by tuning the fluctuation interval $[\underline{X}, \overline{X}]$. Generally, the narrower the interval $[\underline{X}, \overline{X}]$ we use, the more frequently the propensity updates perform resulting in increasing the updating time and the acceptance probability. If the fluctuation interval degenerates into the state $\underline{X} = \overline{X} = X(t)$, then $\alpha = \beta = 1$ which means a candidate reaction is always accepted and reactions have to update their propensities after every reaction firing as in SSA. On the other hand, if we increase the fluctuation interval, we reduce the number of updates for propensity bounds. We are even able to define a fluctuation interval so that no update occurs in the whole simulation ($\beta = \infty$). The update cost is thus zero, $T_{RSSA}^{update}/\beta = 0$, and has no effect on the simulation. T_{RSSA} will depend only the search cost; however, in this situation because $\underline{a}_j$ and $\overline{a}_j$ are very loose approximations of the exact propensities a_j, the acceptance probability decreases significantly. Consequently, it increases α since the candidate reaction is rejected frequently. This indirectly affects the search for the next reaction firing, which in turn negatively impacts the simulation performance. We discuss mechanisms to control the fluctuation interval in the next section.

B. Search for a candidate reaction

An efficient search algorithm can be applied to reduce the time complexity of RSSA in simulating large models. Typically, a fast search algorithm with fast asymptotic speed requires to build complex underlying data structures (trees, hash tables) before the actual search can be conducted. The choice for the search algorithm thus depends on the problem size and on the complexity of the data structures it needs to build.

Tree-based search. By applying the tree-based search, we reduce complexity of the search from linear to logarithmic time. This search method is based on a (binary) tree structure in which the leaf nodes will store the propensity upper bounds $\overline{a_j}$ and the inner nodes store the sums of values of their child nodes. The tree root therefore holds the sum of all values stored in the leaves i.e., $\overline{a_0}$. In a implementation, an array is used to represent the tree; however, the array requires $O(m)$ more elements than linear search array where m is the number of reactions. This is because we have to store also partial sums of propensity upper bounds in internal nodes, as well as the pointers to parent/children pairs. The search will traverse the tree to find a candidate reaction R_μ given the search value $r_1 \cdot \overline{a_0}$. Starting at the tree root, that we mark as the current node, the search recursively selects the next branch by comparing on the search value with the value stored in the left child node. The left branch is selected if the search value is less than the value stored in left child of the current node. The right branch is chosen otherwise. If the right branch is selected, the search value is subtracted by the value stored in current node. The search stops when it reaches a leaf. The reaction in this leaf is chosen as the candidate for the rejection test. Since the search complexity is linked to the depth of the tree, we use a Huffman tree[10,11,21] to optimize the average search length. The key idea of the Huffman tree is to have the leaves storing large values (hence more likely) close to the root than leaves with small values. The time complexity for the (complete) tree-based search is $T^{search}_{RSSA} = O(\log m)$ and the update time is also $T^{update}_{RSSA} = O(k \log m)$. This logarithmic time complexity may provide a substantial improvement for simulating large models.

Table lookup search. The search for a candidate reaction can be reduced to constant time complexity by applying a table lookup method at the cost of an expensive pre-processing to build the lookup tables[22,23]. Although the table lookup search can be applied to standard SSA for selecting next reaction firings, the changes in propensities after each reaction firing require the lookup tables to be updated and makes the application of lookup search to SSA no more efficient than linear search. The downside of the lookup search is alleviated by RSSA where propensity bounds are used to select next reaction firings. RSSA updates the lookup tables infrequently, thus improving its amortized cost. We implemented and experimented a well-known lookup search, called the Alias method[24]. The theoretical foundation underlying the Alias method is a theorem stating that any discrete probability distribution over m probability values can be expressed as an equi-probable mixture of m two-point distributions. The m probabilities used in this case are $\overline{a_j}/\overline{a_0}$ for $j = 1 \ldots m$. The set-up of the Alias method requires to build two tables, each of size m, in which the first table, called cut-off table, stores the probability of the first values of the two-point mixtures and the second table, called alias table, contains the alias to the second parts of the mixtures[25]. The random number r_1 is first used to lookup the position of the equi-probable mixture. It is rescaled to select which part of the two-point mixture. These steps require only one comparison to choose the part of the two-point mixture and (at most) two table accesses to select the candidate reaction. The time complexity of the search is thus constant $T^{search}_{RSSA} = O(1)$. The generation of tables for the Alias method has complexity proportional to the number of reactions m, i.e., $T^{update}_{RSSA} = O(m)$[26].

C. Fluctuation interval control

A rejection test is applied on the selected candidate to ensure it fires with a correct probability. The acceptance of the candidate depends on the propensity bounds which can be adjusted indirectly through the fluctuation interval $[\underline{X}, \overline{X}]$. We should emphasize that the width of the fluctuation interval does not affect the correctness of the algorithm, but only affects the simulation performance.

We can define the fluctuation interval by a fluctuation parameter δ which could be a scalar value or a vector. If δ is a scalar value, we call it *uniform fluctuation* since all species uses the same parameter to the compute their fluctuation interval. By using the uniform fluctuation rate, the fluctuation interval will be defined (using vector notation) as $[\underline{X}, \overline{X}] = [(1-\delta)X(t), (1+\delta)X(t)]$. This approach has both advantages and disadvantages. On the positive side, the calculation of fluctuation interval is fast, requiring only vector computation. However, it does not allow a fine control for each species. If δ is a vector where each component δ_i defines the population bound for each single species in the state we call it *nonuniform fluctuation*. The population bound for species S_i is then defined as $[(1-\delta_i)X_i(t), (1+\delta_i)X_i(t)]$. In an implementation, a lookup table is used to store and retrieve the fluctuation parameters of species.

In some models, the population of some species may vary significantly during the simulation. The fluctuation parameters for such species should be changed adaptively to optimize the acceptance probability of the involved reactions. We call this approach *adaptive fluctuation*. For example, an absolute interval size (instead of a $\%$) can be preferred in case the population of a species is low (say e.g., less than 25). In order to exploit the adaptive interval control we set a threshold value λ on the population of species. During the simulation, if the population of a species S_i gets lower than the threshold value i.e., $X_i(t) < \lambda$, we will apply a fixed (absolute) fluctuation interval Δ. The population $X_i(t)$ of species S_i then is bound to the interval $[X_i(t)-\Delta, X_i(t)+\Delta]$. Otherwise, we will apply a fluctuation rate δ_i to define the population bound of species S_i. Thus, if $X_i(t) \geq \lambda$, the interval $[(1-\delta_i)X_i(t), (1+\delta_i)X_i(t)]$ is applied to bound the population of species S_i. Following this simple scenario, we extend the idea of adaptive fluctuation control to the models having many phases. A species S_i

is in phase k if its population is less than an upper threshold λ_i^k and greater than a lower threshold λ_i^{k-1}. Thus, if species S_i is bound to phase k, a fluctuation rate δ_i^k will be applied to derive the population bound for that species. This strategy allows the simulation to automatically adjust the fluctuation interval depending on the phase of the system.

IV. SIMULTANEOUS REJECTION-BASED ALGORITHM FOR SIMULATION ANALYSIS

A. Simulation analysis

The state X at a given time t is a random variable. Thus to have a reasonable estimation by simulation, we have to repeatedly perform independent simulations to generate realizations of $X(t)$. Let K be the number of simulations and respectively, let X^r with $r = 1 \dots K$ be the realizations of X obtained by repeatedly performing K independent runs of an exact simulation algorithm under the same simulation conditions. The statistical properties (e.g., mean, and variance) can be derived from the ensemble of K trajectories and these properties are ensured to approach the exact solution of CME as K approaches infinity.

Let $<X>$ be the sample mean and s^2 be the (unbiased) sample variance of state X at time t based on an ensemble of K independent simulations. We can compute these values by:

$$<X> = \frac{\sum_{r=1}^{K} X^r}{K} \tag{6}$$

and

$$s^2 = \frac{\sum_{r=1}^{K} (X^r - <X>)^2}{K-1} \tag{7}$$

By the law of large numbers, the sample mean and variance will asymptotically approach the true mean $\mathbb{E}[X]$ and variance $\mathrm{Var}[X]$ of the random variable X when K tends to infinity:

$$\mathbb{E}[X] = \lim_{K\to\infty} <X> \tag{8}$$

$$\mathrm{Var}[X] = \lim_{K\to\infty} s^2 \tag{9}$$

However, since the number of simulations K is limited, the convergence of the estimation is measured by the size of the confidence interval

$$d = \frac{\mathsf{z} s}{\sqrt{K}} \tag{10}$$

where z is a confidence level. If we fix the confidence level z, the probability that the true mean $\mathbb{E}[X]$ lies in the interval $[<X> - d, <X> + d]$ is $2\Phi(\mathsf{z}) - 1$ with Φ is the cdf of the standard normal distribution $\mathbb{N}(0,1)$. For instance, with the confidence level $\mathsf{z} = 1.96$, the probability that the true mean falls in $[<X> - 1.96s/\sqrt{K}, <X> + 1.96s/\sqrt{K}]$ is 95%. Thus, to reduce the confidence interval size given a fixed confidence level we have to increase the number of trajectories K.

Moreover, given an ensemble of K trajectories, we can infer the empirical distribution function (edf) (or histogram) of the species population. This is done by partitioning the population of species into bins and compute the occurring frequency of species in each bin. The edf of species will approach its pdf for large K. We can measure the statistical fluctuation introduced by stochastic simulation from edf. For instance, a statistical measurement is developed by Cao and Petzold[27] based on the distance between two empirical distributions to compare the accuracy of simulation algorithms. More specifically, given a fixed K, two algorithms have the same accuracy if the distance between two empirical distributions computed from two sets of independent realizations by these simulation algorithms is less than the so called *self distance*[27]. The self distance is the distance between the edfs computed from two sets of independent realizations derived from the same simulation algorithm. It is a random variable bounded by $\sqrt{4B/(K\pi)}$, where B is the number of bins.

B. Simultaneous Rejection-based Simulation Algorithm

In this section we present our new algorithm, called simultaneous rejection-based stochastic simulation algorithm (SRSSA), for generating multiple independent trajectories. The advantage of SRSSA is that the trajectories are generated simultaneously in a simulation run instead of many simulation runs. The independent trajectories generated by SRSSA are exact by exploiting the propensity bounds to select next reaction firings as in RSSA. For independent runs of RSSA the propensity bounds have to be replicated and separated for each simulation run. The propensity bounds in SRSSA, however, are only computed once and shared across the simulations. Since SRSSA uses the same propensity bounds across the realizations, it reduces the memory requirement to store the propensity bounds and improves its cache-friendliness. The recomputing of the propensity bounds in SRSSA when needed will be performed collectively in a single operation which further reduces the total number of propensity updates and improves the simulation time.

Let K be the number of trajectories and X^r be the system state of the r-th realization with $r = 1 \dots K$. Let a_j^r be the propensity of reaction R_j in the r-th realization. The key point of SRSSA is that it computes a lower bound $\underline{a_j}$ and an upper bound $\overline{a_j}$ for each reaction R_j such that $\underline{a_j} \le a_j^r \le \overline{a_j}$ for all $r = 1 \dots K$, and then uses these propensity bounds to select reaction firings for all K realizations. Thus, we only need to store m propensity bounds of m reactions independently of the number of realizations K. This feature is useful when we need to generate a large number of realizations for an online analysis of large reaction networks.

The propensity bounds $\underline{a_j}$ and $\overline{a_j}$ are derived by first defining a *global* fluctuation interval $[\underline{X}, \overline{X}]$ which bounds all possible populations of each species in all K states X^r with $r = 1 \dots K$. The algorithm then minimizes/maximizes the propensity function a_j on such a global fluctuation interval $[\underline{X}, \overline{X}]$. We define the global population bound for a species S_i by the following procedure. Let $X_i^{min} = \min(X_i^1, \dots, X_i^K)$ and $X_i^{max} = \max(X_i^1, \dots, X_i^K)$, re-

spectively, be the minimum and maximum population of species S_i in all K states. The chosen population interval $[\underline{X_i}, \overline{X_i}] = [(1-\delta_i)X_i^{min}, (1+\delta_i)X_i^{max}]$ will bound all populations of species S_i in K states, where δ_i is the fluctuation rate of this species. Repeating this procedure for all species in the state vector, we are forming a global fluctuation interval $[\underline{X}, \overline{X}]$ for these K states.

Knowing the lower bounds $\underline{a_j}$ and upper bounds $\overline{a_j}$, SRSSA selects reaction firings and updates the state X^r for the corresponding r-th realization with $r = 1 \dots K$ by applying the the rejection-based selection. The SRSSA algorithm is outlined in Algorithm 2.

SRSSA initializes the time t^r and initial state X^r for each $r = 1 \dots K$. It then derives the global fluctuation interval $[\underline{X}, \overline{X}]$ for all these K states and computes the propensity lower bound $\underline{a_j}$ and upper bound $\overline{a_j}$ for all reactions R_j. SRSSA maintains a set of species that should update their population bounds which is represented by the set UpdateSpeciesSet, initialized to an empty set. SRSSA also uses the Species-Reaction (SR) graph to retrieve which reactions should update propensity bounds when a species exits its population bound.

Inside the main simulation loop, the rejection-based selection will be continuously applied to select reaction firings and form trajectories. For the r-th realization, a candidate reaction R_μ is randomly selected with probability $\overline{a_\mu}/\overline{a_0}$. Then, the propensity ${a_\mu}^r$ is evaluated on the corresponding state X^r and used to validate this candidate reaction with acceptance probability ${a_\mu}^r/\overline{a_\mu}$. Note that the propensity lower bound $\underline{a_\mu}$ is still applied to avoid computing ${a_\mu}^r$ as much as possible. The selection of the reaction firing in the r-th realization is exact and independent of other realizations. If the reaction is accepted, the time t^r and state X^r are updated. This selection step is then repeated until a species population exits the global population interval (see line 8 - 28, Algorithm 2). Let S_i be the species whose population $X_i^r \notin [\underline{X_i}, \overline{X_i}]$ in the r-th realization. SRSSA adds this species S_i to the UpdateSpeciesSet. It then stops the current r-th realization and moves to the next realization.

Only when all K trajectories are stopped, new global population interval $[\underline{X_i}, \overline{X_i}]$ for all species $S_i \in$ UpdateSpeciesSet are redefined. This is the key difference between SRSSA and RSSA. RSSA has to redefine a new population bound as soon as a species exits its current population bound, while this step in SRSSA is postponed and performed once when all K simulationa are stopped. Then, SRSSA retrieves reactions for which propensity bounds have to be recomputed because they have reagent species that exit their population bounds (see line 33 - 39, Algorithm 2). This set of reactions affected by species S_i is extracted from the SR dependency graph and denoted by the set ReactionsAffectedBy(S_i). Thus, for each $R_j \in$ ReactionsAffectedBy(S_i), its new lower bound $\underline{a_j}$ and upper bound $\overline{a_j}$ is recomputed.

Algorithm 2: Simultaneous RSSA (SRSSA)

procedure: srssa
output: K independent trajectories of the reaction network
1: for each trajectory $r = 1 \dots K$, set initial time $t^r = 0$ and initial state $X^r = x_0$
2: build the species-reaction (SR) dependency graph $\mathcal{G}$
3: for each species S_i with $i = 1 \dots n$ define a bound $[\underline{X_i}, \overline{X_i}]$ such that $\underline{X_i} \leq X_i^1 \dots X_i^K \leq \overline{X_i}$
4: compute propensity bounds $\overline{a_j}$ and $\underline{a_j}$ for each reaction R_j with $j = 1 \dots m$
5: compute total upper bound propensity $\overline{a_0} = \sum_{j=1}^{m} \overline{a_j}$
6: **repeat**
7: set UpdateSpeciesSet $= \emptyset$
8: **for** (each trajectory $r = 1 \rightarrow K$) **do**
9: **repeat**
10: set $u = 1$
11: set $accepted =$ **false**
12: **repeat**
13: generate random numbers: $r_1, r_2, r_3 \sim U(0,1)$
14: select minimum index μ satisfied $\sum_{j=1}^{\mu} \overline{a_j} > r_1 \overline{a_0}$
15: **if** $(r_2 \leq (\underline{a_\mu}/\overline{a_\mu}))$ **then**
16: set $accepted =$ **true**
17: **else**
18: evaluate a_μ^r with state X^r
19: **if** $(r_2 \leq (a_\mu^r/\overline{a_\mu}))$ **then**
20: set $accepted =$ **true**
21: **end if**
22: **end if**
23: set $u = u \cdot r_3$
24: **until** $accepted$
25: compute firing time $\tau^r = (-1/\overline{a_0}) \ln(u)$
26: set time $t^r = t^r + \tau^r$
27: update state $X^r = X^r + v_\mu$
28: **until** (exists $X_i^r \notin [\underline{X_i}, \overline{X_i}]$) **or** ($t^r \geq T_{max}$)
29: **for all** (species S_i where $X_i^r \notin [\underline{X_i}, \overline{X_i}]$) **do**
30: set $UpdateSpeciesSet = UpdateSpeciesSet \cup \{S_i\}$
31: **end for**
32: **end for**
33: **for all** (species $S_i \in$ UpdateSpeciesSet) **do**
34: define a new $[\underline{X_i}, \overline{X_i}]$ such that $\underline{X_i} \leq X_i^1 \dots X_i^K \leq \overline{X_i}$
35: **for all** ($R_j \in$ ReactionsAffectedBy(S_i)) **do**
36: compute propensity bounds $\overline{a_j}$ and $\underline{a_j}$
37: update total upper bound sum $\overline{a_0}$
38: **end for**
39: **end for**
40: **until** ($t^r \geq T_{max}$ for all trajectories $r = 1 \dots K$)

V. NUMERICAL EXAMPLES

In this section we first report the performance of our efficient RSSA formulations in simulating large models. Then, we present the performance improvement of our new algorithm SRSSA. The models we considered in the performance comparisons are real biological processes. All the algorithms were implemented in Java and run on a Intel i5-540M processor. The implementation of the algorithms as well as the benchmark models are freely available at `http://www.cosbi.eu/research/prototypes/rssa`.

A. RSSA formulations performance on large models

Table I summarizes the models that we used for the benchmark. The models are chosen with varying network sizes and average number of propensity updates per reaction firing to observe the effects of both the search and update on the simulation performance. The number of reactions of models in the Table I spans from a few reactions as in the gene expression model (8 reactions) to an order of ten thousands reactions as in the B cell receptor signaling (24388 reactions). The average number of propensity updates after a firing of a reaction of the corresponding networks also increases from 3.5 to 546.66. A brief description of these models is in the following.

TABLE I: Summary of reaction models

Model	#Species	#Reactions	#Propensity Updates per Firing
Gene expression	5	8	3.5
Folate cycle	7	13	5
MAPK cascade	106	296	11.70
FcϵRI signaling	380	3862	115.80
B cell receptor signaling	1122	24388	546.66

The gene expression model is a type of regulatory pathway which plays a key role in the understanding of gene regulation mechanisms and functionality[28]. The result of gene expression is a collection of proteins encoded by the corresponding gene. Proteins are produced by two main consecutive processes: the transcription and then the translation. During the transcription process, the gene is copied to intermediate form called messenger RNA (mRNA). mRNA then binds to ribosomes to translate into the corresponding protein. We implement this model with 5 species and 8 reactions.

The folate cycle is a metabolic pathway which has a vital role in cell metabolism[29]. The result of this metabolism is the transfer of one-carbon units for methylation to produce methionine and synthesis of pyrimidines and purines. In the folate cycle, the tetrahydrofolate (THF) is catalysed to produce 5,10-methylene-THF which is subsequently either converted to 5-methyl-THF or 10-formyl-THF. The folate cycle completes when 5-methyl-THF is demethylated to produce methionine and THF. This model is composed of 7 species and 13 enzymatic reactions where their rates are modelled by the Michaelis-Menten kinetics[30,31].

The mitogen-activated protein (MAP) kinase (MAPK) cascade pathway describes a chain of proteins that cascade a signal from the cell receptor to its nucleus. It is stimulated when ligands, e.g., growth factors, bind to the receptor on the cell surface. The pathway is controlled through three main proteins kinases: MAPKKK, MAPKK and MAPK. First, the ligand activates MAPKKK. The activated MAPKKK phosphorylates MAPKK and subsequently activates MAPK through further phosphorylation. Finally, a cellular response e.g., cell growth is exhibited. We implement the MAPK model with 106 species and 296 reactions[32].

The FcϵRI signaling is used to model early events in high-affinity IgE receptor[33]. The signaling is initiated by ligand-induced receptor aggregation and results in a response from the immune system (e.g., allergic responses). This signaling is extensively studied in the literature[34]. We use the FcϵRI signaling model developed by Liu *et al.*[35] which contains 380 species and 3862 reactions.

The B cell receptor signaling model proposed in Barua *et al.*[36] studies the effect of protein Lyn and Fyn redundancy. This model was implemented with a rule-based modeling approach by including the site-specific details of protein-protein interactions. The reaction network generated from the model contains 1122 species and 24388 reactions.

Figure 1 compares performance of RSSA formulations with the Direct Method (DM), the Next Reaction Method (NRM) and the Partial-propensity Direct Method (PDM) and its variants including Sorting PDM (SPDM) and PDM with Composition Rejection (PSSA-CR) on the benchmark models. In order to run the partial-propensity approach (PDM, SPDM and PSSA-CR) with the folate cycle, we used a simplified version of the Michaelis-Menten rate and modified the rate computation. For RSSA, three variants of the search used for selecting the candidate reaction are considered: 1) the basic RSSA where linear search is applied, 2) RSSA with tree-based search (RSSA-Binary) and 3) RSSA with Alias lookup search (RSSA-Lookup). We also adaptively adjust the fluctuation interval of a species depending on its population. If the population of a species is less than 25, the absolute interval size $\Delta = 4$ is applied. Otherwise, the fluctuation rate $\delta = 10\%$ is applied. The performance of algorithms is averaged from 100 simulation runs. For each simulation run, the results are collected after 10^7 steps.

A conclusion from Fig. 1 is that our RSSA formulations achieve better performance than all other algorithms in all test cases, and outperform especially in large models. For instance, the speed-up gain by RSSA in comparison with DM, NRM, PDM, SPDM in simulating the FcϵRI signaling is 9, 8.6, 1.8 and 2, respectively. Furthermore, RSSA with an efficient search achieves a significant performance improvement when simulating large models. In simulating the FcϵRI signaling, RSSA-Binary and RSSA-Lookup is roughly 3 and 2.3 times faster than RSSA. In this benchmark, our efficient RSSA formulations perform better than PSSA-CR. In the FcϵRI signaling, the speed up gain of RSSA-Binary and RSSA-Lookup in comparison with PSSA-CR is 4.3 and 3.1, respectively. The detailed performance analysis is given below.

As shown in Fig. 1, DM and NRM is comparable for small models (i.e., the gene expression, the folate cycle) and is often faster than DM if the number of propensity updates is small. The speed-up gain by NRM is achieved by using a priority queue for selecting the next reaction firings and saving random numbers. The advantage of NRM becomes negative when the number of update propensities is large because of updating and maintaining the priority queue. For example, the percentage of update of NRM for the FcϵRI signaling is 94%, while in DM it is around 87%, thus NRM is only 7% faster than DM. For the B cell receptor signaling where the number of propensity updates after a reaction firing is very large, the update cost of NRM is at maximum and contributes

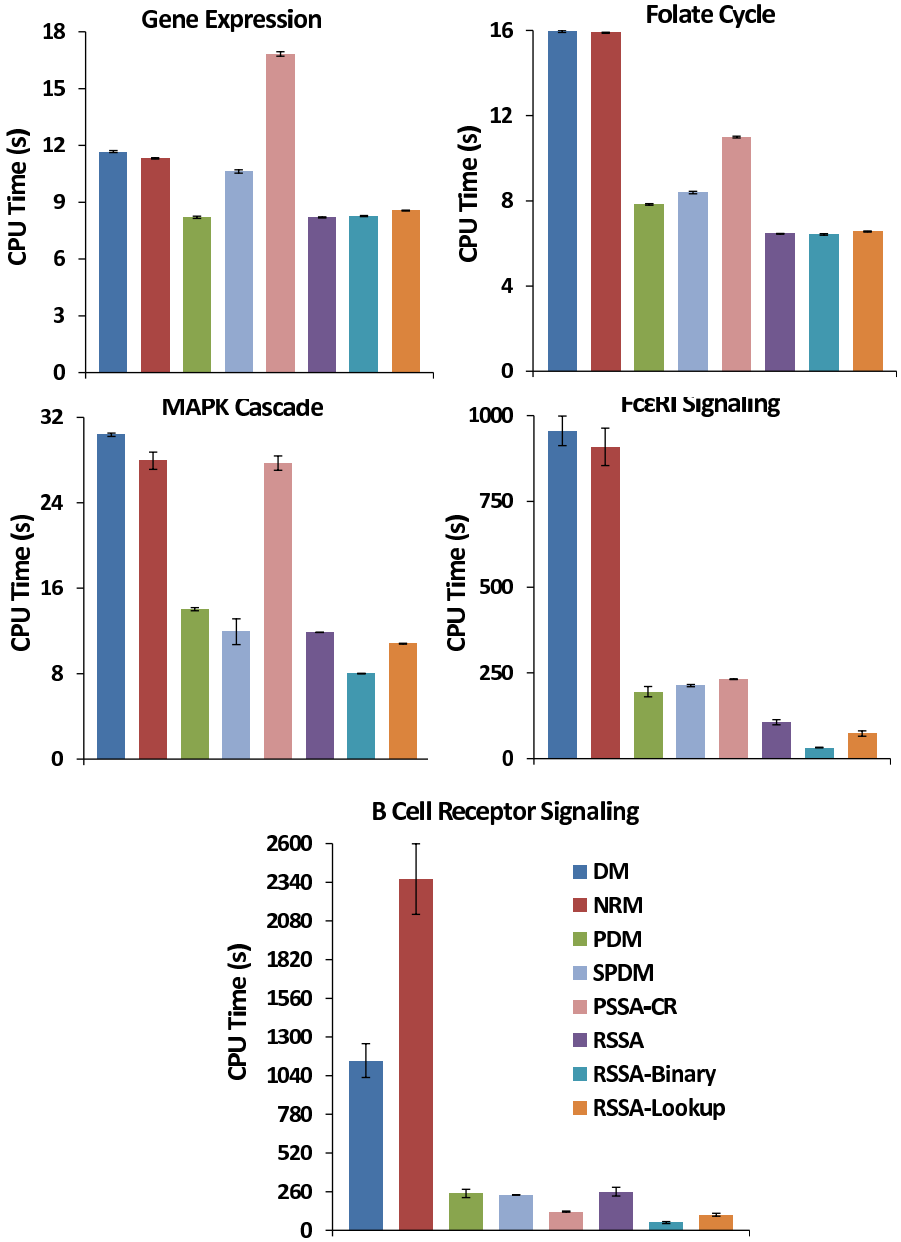

FIG. 1: Performance of RSSA formulations, DM, NRM and partial-propensity variants including PDM, SPDM and PSSA-CR. In order to be able to simulate the folate cycle with Partial propensity algorithms (PDM, SPDM and PSSA-CR) we used a simplified version of the Michaelis-Menten rate for this model and modified the rate computation procedure.

up to 99.5% of its total simulation time. In this case, NRM is 2.1 times slower than DM.

By reducing the update cost, PDM and RSSA efficiently accelerates the simulation. For the gene expression, the update cost of DM and NRM contributes 61% and 75% of its simulation time, respectively. By grouping common reactant and updating propensities in single operations, PDM reduces the update cost to nearly 45%. The update cost by RSSA is further reduced to only 10% of its total simulation time. Although this advantage of RSSA is limited by the rejection of candidate reaction where the acceptance probability of a candidate is around 90%. The result is that RSSA is 20% faster than DM and NRM. In simulating the gene expression model, RSSA and PDM have a comparable performance. For the folate cycle where Michaelis-Menten kinetics are applied, the performance gain by RSSA is significant because it efficiently handled the time-consuming Michaelis-Menten propensity. The rates of reactions with Michaelis-Menten kinetics are not constant but depend explicitly on their reactants, thus these rates have to be recomputed as soon as the state changes. RSSA only performs around 200 updates while other algorithms have to perform all 10^7 updates. The percentage of propensity update cost by DM and NRM thus contributes 73% and 85% of their simulation time, respectively. PDM re-

duces percentage of the update cost to 33% by grouping reactants. The update cost by RSSA is the best which contributes only 20% of the its simulation time. Thus, RSSA is around 2.5 times faster than DM, NRM and roughly 20% faster than PDM in simulating the folate cycle, even if this model is rather small. In simulating the large models by RSSA, a significant speed-up is also achieved. More specifically, the speed up gain by RSSA varies from 2 to 10 in comparison with DM, NRM and about 1.2 to 1.8 with PDM, SPDM.

A fast search procedure will make RSSA become more efficient in simulating large models. This is shown in the Fig. 1 where RSSA-Binary and RSSA-Lookup outperform all other algorithms for large models. For example, RSSA-Binary and RSSA-Lookup are 3.8 and 2.6 times, respectively, faster than the basic RSSA in simulating the B cell receptor signaling. For the B cell receptor signaling, PDM performance is comparable with RSSA; however, by applying a fast search algorithm a sharp improvement is achieved for RSSA. For instance, RSSA-Binary and RSSA-Lookup is, respectively, 3.7 and 2.3 times faster than PDM in simulating this model. Although the advantage of the composition-rejection of PSSA-CR makes it 2 times faster than PDM, the performance of PSSA-CR is still less significant in comparison with our efficient RSSA formulations. For example, RSSA-Binary is around 1.9 times faster than PSSA-CR. A similar speed up gain for RSSA-Binary, RSSA-Lookup is also obtained for simulating the MAPK cascade and the FcϵRI signaling. We note that the performance of PSSA-CR for all other models is very slow. An efficient search procedure, however, requires to build complex data structures that negate its efficiency for small models. For example, the basic RSSA is slightly faster than RSSA-Binary and RSSA-Lookup in simulating the gene expression.

We further compare performance of our efficient RSSA formulations with NFSim[37] on the B cell receptor signaling. NFSim is a network-free simulation algorithm. It keeps track of individual species in the model instead of species population. Figure 2 shows the performances of these algorithms applied to the B cell receptor signaling. For this model, we adjust the initial population of species by multiplying a scale factor (sf), which takes values from 1 to 1000, with a base value 300. The stopping time $T_{max} = 100$ is used in this performance comparison.

We have remarks from the Fig. 2. First, the memory requirement and simulation time of NFSim are increasing when increasing $sf = 1$ to 1000. In fact, the memory requirement for NFSim is growing linearly with the number of species in the system since it keeps track of each individual species. Thus, in case $sf = 1000$, the NFSim simulation has crashed due to lack of memory to keep track of all molecular species. We note that in the original model[36] the population of species is obtained by setting $sf = 1000$. In contrast, our RSSA formulations are still be able to simulate the system with a reasonable simulation time. Second, RSSA-Binary is faster than NFSim for all values of sf. For example, in case $sf = 100$, RSSA-Binary is roughly 2 times faster than NFSim. The speed up gain comes from the fast search and the low cost of update. For example, the search cost and update cost con-

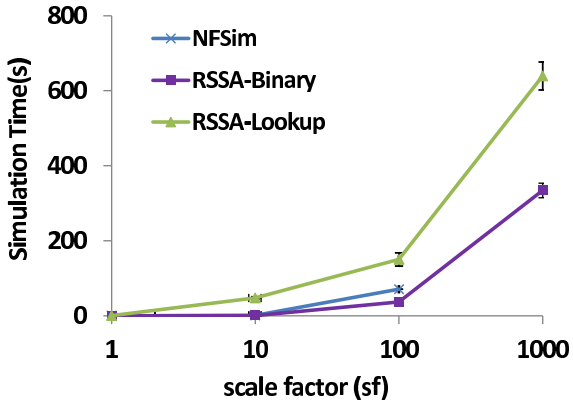

FIG. 2: Performance of efficient RSSA formulations and NFSim on B Cell Receptor Signaling model. NFSim crashes in case $sf = 1000$ due to lack of memory to keep track of all individual molecular species.

tributing to the simulation time of RSSA-Binary in the case $sf = 100$ are roughly 20% and 14%, respectively. Third, although the search time of Alias lookup is faster than tree-based search, the update of the supporting tables are rather expensive. Note that in RSSA-Lookup, the SR dependency graph is inapplicable because we have to rebuild the entire supporting tables when a species exits its population bounds. The update of RSSA-Lookup contributes up to 92% of its simulation time in case $sf = 100$, while the search contributes less than 4%. The result is that RSSA-Lookup has a lowest performance in this case.

B. SRSSA performance

In this section we focus on the performance of SRSSA with respect to RSSA and DM. We do not consider here NRM and PDM because we already showed that RSSA always performs better. We compare the simulation time while removing all the initialization and output writing.

The first experiment compares the simulation time of the algorithms on the gene expression. The simulation stopping time for all simulations is set to $T_{max} = 100$. SRSSA uses the same fluctuation parameters to define the fluctuation interval as RSSA. We compare the simulation time of algorithms to generate a total K independent trajectories. For RSSA and DM, we have to perform K independent runs of these algorithms. But with SRSSA, we will set to generate K independent trajectories simultaneously in one simulation runs. We vary the total number of trajectories K from 10 to 1000 to observe its effects on the performance of the algorithms. The performance is plotted in the Fig. 3.

The speed up gain of SRSSA with respect to RSSA and DM is about 16% and 33%, respectively, for all values of K. An explanation for this speed up gain follows. Starting with a small number of trajectories $K = 10$, we observe that there are $1.44x10^7$ reaction firings in simulating the gene expression model. Thus, DM has to perform $1.44\text{x}10^7$ updates. RSSA reduces the number of updates to $2.59\text{x}10^6$ and its up-

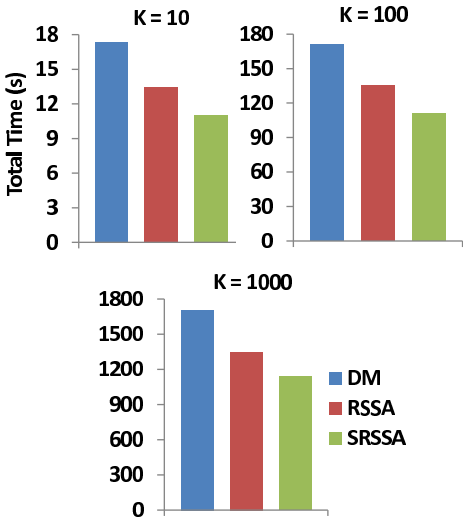

FIG. 3: Performance of SRSSA, RSSA and DM on Gene Expression model for generating K trajectories.

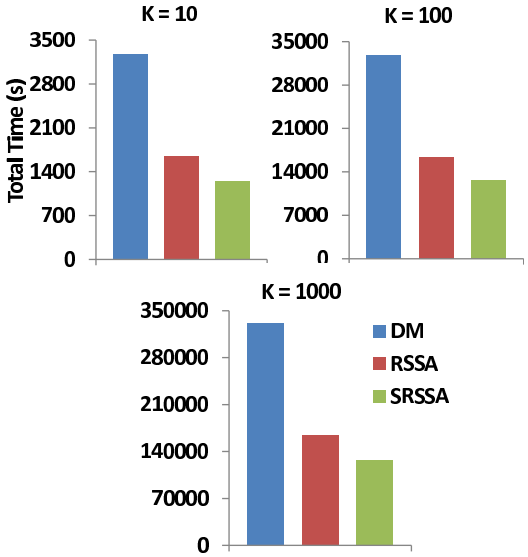

FIG. 4: Performance of SRSSA, RSSA and DM on Folate cycle for generating K trajectories.

date cost is about 25% of its total simulation time. By postponing and lumping together propensity updates into single operations, SRSSA further reduces the total number of updates to 400. The update cost of SRSSA contributes less than 1% of its total simulation time. This huge gain in the update cost compensates for the higher search time. This is because the acceptance probability of a candidate reaction in SRSSA is only around 78% which is lower than 93% of RSSA. For DM and RSSA, both the search and update costs will grow linearly with K since we have performed independent runs. However, with SRSSA where trajectories use the same propensity bounds for selecting the next reaction firings, only the search time increases with K while the update is nearly constant. The total number of updates in SRSSA is kept around 400 with the acceptance probability remaining at 78%. The result is that the same speed up gain is achieved by SRSSA even using $K = 1000$.

Figure 4 compares the performance of the algorithms on the Folate cycle with the simulation stopping time $T_{max} = 0.05$ and total number of trajectories K to be generated vary from 10 to 1000. As shown in the figure, SRSSA is better than RSSA and DM for all values of K. More specifically, the performance of SRSSA is roughly 30% faster than RSSA. The high speed up gain is because of further reducing the number of updates of the time-consuming Michaelis-Menten propensity. We observed that the update cost of RSSA is around 28% of its simulation time, while this percentage in SRSSA is less than 1%.

For the last experiment, we compare the performance of the algorithms with an Oscillator model. The Oscillator[38] is an artificial model which is a noise-induced system. It is used to observe the effect of large number of trajectories generated simultaneously on the performance SRSSA. This model has three reactions which are listed in Table II. The initial populations of species are: $\#A = 900$, $\#B = 500$ and $\#C = 200$.

For simulating the Oscillator model, we fix a total of $K = 1000$ trajectories to be generated for each algorithm. However, we set up for SRSSA to generate these trajectories in three different scenarios: 1) performing 100 runs of SRSSA with $N = 10$ trajectories generated simultaneously per run, 2) performing 10 runs of SRSSA with $N = 100$ trajectories generated simultaneously per run and 3) performing 1 run of SRSSA with $N = 1000$ generated simultaneously. The figure 5 shows performance of SRSSA with these three different settings in comparing with RSSA and DM.

Figure 5 shows that SRSSA is increasingly slow when increasing the number of trajectories generated simultaneously per run. For example, the performance of SRSSA in generating of $N = 1000$ trajectories simultaneously is nearly 1.9 times slower than RSSA. The reason for a poor performance of SRSSA in this setting is that the Oscillator model is noisy. The fluctuation in population of species is very large. In fact, the maximum signal-to-noise ratio in population of species, which is equal to the ratio of the mean over the standard deviation, is around 65%. SRSSA has to define a large interval to include all possible values of states of all trajectories. Even though the update cost is very low, the search becomes very costly due to a lot of rejections. Specifically, the acceptance probability of SRSSA reduces from 60% in case $N = 10$ to roughly 20% in case $N = 1000$. The search cost of SRSSA in case $N = 1000$ is thus 7 times slower than RSSA. This high cost negates the advantage of low update cost of SRSSA. In this model, a small number of simultaneous trajectories generated in a simulation run of SRSSA should be chosen. For example, in case $N = 10$ SRSSA is still 7% and 26%, respectively, faster than RSSA and DM.

VI. CONCLUSIONS

We studied efficient formulations to improve the search and update of the basic RSSA algorithm to efficiently simulate

TABLE II: Artificial Oscillator model

Reactions	Rate
R_1: A + B $\rightarrow$ 2B	$c_1 = 1$
R_2: B + C $\rightarrow$ 2C	$c_2 = 1$
R_3: C + A $\rightarrow$ 2A	$c_3 = 1$

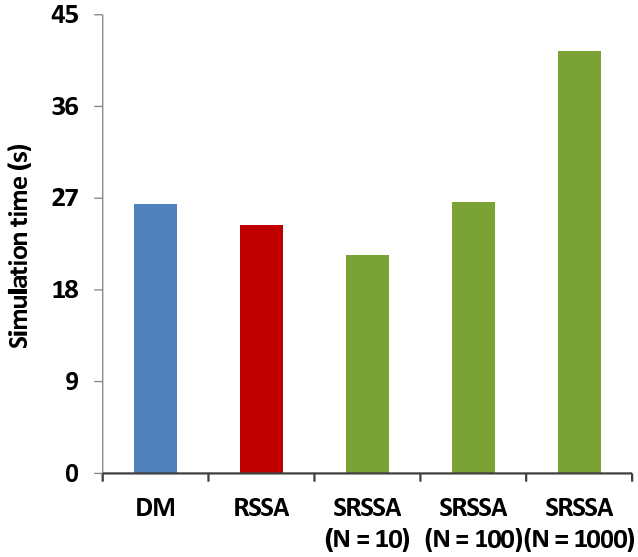

FIG. 5: Performance of SRSSA, RSSA and DM on Oscillator model for generating a total of $K = 1000$ trajectories. SRSSA generates these trajectories by 1) performing 100 runs with $N = 10$ trajectories generated simultaneously per run, 2) performing 10 runs with $N = 100$ trajectories generated simultaneously per run and 3) 1 run with $N = 1000$ trajectories generated simultaneously.

large scale biochemical reaction systems. We discussed different search procedures for selecting a candidate reaction. We proposed different mechanisms to control the fluctuation interval of the state. The proposed mechanism allows to control the fluctuation interval of each species and adaptively maintain it depending on the species population. We have implemented and experimented with these strategies. The choice of a suitable strategy for search and update of reactions in RSSA ultimately depends on the size and complexity of the underlying data structures. A simple search method (e.g., linear search) does not require any complex data structure, while having a low search performance; instead, some search procedures (e.g., binary search, Alias method) can have a fast speed, but require a complex data structure which is expensive to maintain. According to our experiments, complex methods should be applied on large models. In simulating large models, although the search of the Alias method is a constant, it requires to build the lookup tables, which cost is proportional to the number of reactions and negates the overall simulation performance. A possible approach for the future work to improve the cost of building lookup tables is to exploit the composition-rejection strategy. To integrate the composition-rejection strategy with RSSA, we group reactions into groups by their propensity upper bounds. A reaction in a group is selected by applying the composition-rejection search and then the rejection-based test to validate the selection. Each time a species whose population moves out of its fluctuation interval, we only need to update propensity bounds of affected reactions and move these reactions to their corresponding groups.

In this work we also proposed a new algorithm called simultaneous RSSA (SRSSA) for generating multiple independent trajectories to support the analysis of biochemical reaction systems. The advantage of SRSSA is that trajectories are generated simultaneously by one simulation run, instead of performing many simulation runs. SRSSA uses only a single set of propensity bounds across simulations to select reaction firings. The memory needed to compute and store the propensity bounds is thus independent of the number of trajectories. The selection of reactions to form trajectories of SRSSA, however, is exact by exploiting the rejection-based mechanism. The update of the propensities is lumped up together in one operation. For typical systems we can choose a large number of simultaneous trajectories to be generated in a simulation run of SRSSA; however, for noise-induced systems a small number of simultaneous trajectories for SRSSA, e.g., 10, should be chosen for a better performance.

[1] Daniel Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comp. Phys.*, 22(4):403–434, 1976.

[2] Daniel Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.

[3] Michael Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A*, 104(9):1876–1889, 2000.

[4] Yang Cao, Hong Li, and Linda Petzold. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *J. Chem. Phys.*, 121(9):4059, 2004.

[5] James McCollum and *et al.* The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *Comp. Bio. Chem.*, 30(1):39–49, 2006.

[6] S. Mauch and M. Stalzer. Efficient formulations for exact stochastic simulation of chemical systems. *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, 8(1):27–35, 2011.

[7] Tim Schulze. Efficient kinetic monte carlo simulation. *J. Comp. Phys.*, 227(4):2455–2462, 2008.

[8] James Blue, Isabel Beichl, and Francis Sullivan. Faster monte carlo simulations. *Phys. Rev. E*, 51(2):867–868, 1995.

[9] Hong Li and Linda Petzold. Logarithmic direct method for discrete stochastic simulation of chemically reacting systems. Technical Report `http://engineering.ucsb.edu/cse/Files/ldm0513.pdf`, 2006.

[10] Vo H. Thanh and Roberto Zunino. Tree-based search for stochastic simulation algorithm. In *Proc. of ACM-SAC*, 2012.

[11] Vo H. Thanh and Roberto Zunino. Adaptive tree-based search for stochastic simulation algorithm. *Int. J. Computational Biology and Drug Design*, 7(4):341–357, 2014.

[12] Alexander Slepoy, Aidan P. Thompson, and Steven J. Plimpton. A constant-time kinetic monte carlo algorithm for simulation of large biochemical reaction networks. *J. Chem. Phys.*, 128(20):205101, 2008.

[13] Sagar Indurkhya and Jacob Beal. Reaction factoring and bipartite update graphs accelerate the gillespie algorithm for large-scale biochemical systems. *PLoS ONE*, 5(1):8125, 2010.

[14] Rajesh Ramaswamy, Nlido Gonzlez-Segredo, and Ivo F. Sbalzarini. A new class of highly efficient exact stochastic simulation algorithms for chemical reaction networks. *J. Chem. Phys.*, 130(24):244104, 2009.

[15] Rajesh Ramaswamy and Ivo F. Sbalzarini. A partial-propensity variant of the composition-rejection stochastic simulation algorithm for chemical reaction networks. *J. Chem. Phys.*, 132(4):044102, 2010.

[16] E. Crampina, S. Schnella, and P. McSharry. Mathematical and compu-

tational techniques to deduce complex biochemical reaction mechanisms. *Progress in Biophysics and Molecular Biology*, 86(1):77–112, 2004.

[17] Vo H. Thanh, Corrado Priami, and Roberto Zunino. Efficient rejection-based simulation of biochemical reactions with stochastic noise and delays. *J. Chem. Phys.*, 141(13), 2014.

[18] Vo H. Thanh. *On Efficient Algorithms for Stochastic Simulation of Biochemical Reaction Systems*. PhD thesis, University of Trento, Italy. `http://eprints-phd.biblio.unitn.it/1070/`, 2013.

[19] Daniel Gillespie. A rigorous derivation of the chemical master equation. *Physica A*, 188(1-3):404–425, 2007.

[20] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. SIAM, 2009.

[21] David Huffman. A method for the construction of minimum-redundancy codes. In *Proc. of the IRE*, volume 40, pages 1098–1101, 1952.

[22] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986.

[23] Wolfgang Hormann, Josef Leydold, and Gerhard Derflinger. *Automatic Nonuniform Random Variate Generation*. Springer-Verlag, 2004.

[24] Alastair J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Trans. on Mathematical Software (TOMS)*, 3(3):253–256, 1977.

[25] Richard A. Kronmal and Arthur V. Peterson. On the Alias method for generating random variables from a discrete distribution. *The American Statistician*, 33(4):214–218, 1979.

[26] Michael D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Trans. on Software Engineering*, 17(9):972–974, 1991.

[27] Yang Cao and Linda Petzold. Accuracy limitations and the measurement of errors in the stochastic simulation of chemically reacting systems. *J. Comp. Phys.*, 212(1):6–24, 2006.

[28] Darren J. Wilkinson. *Stochastic Modelling for Systems Biology*. CRC Press, 2006.

[29] Lynn B. Bailey. *Folate in Health and Disease, 2nd Edition*. CRC Press, 2009.

[30] Michael C. Reed, Rachel L. Thomas, Jovana Pavisic, S Jill. James, Cornelia M. Ulrich, and H. Frederik Nijhout. A mathematical model of glutathione metabolism. *Theoretical Biology and Medical Modelling*, 5(8), 2008.

[31] Marco Scotti, Lorenzo Stella, Emily J. Shearer, and Patrick J. Stover. Modeling cellular compartmentation in one-carbon metabolism. *WIREs Syst Biol Med*, 5(3):343–365, 2013.

[32] Walter Kolch. Meaningful relationships: the regulation of the ras/raf/mek/erk pathway by protein interactions. *Biochem. J.*, 351(2):289–305, 2000.

[33] James R. Faeder and *et al*. Investigation of early events in fceri-mediated signaling using a detailed mathematical model. *J Immunol*, 170:3769–3781, 2003.

[34] Lily A. Chylek, David A. Holowka, Barbara A. Baird, and William S. Hlavacek. An interaction library for the fc?ri signaling network. *Front. Immunol.*, 5(172):1664–3224, 2014.

[35] Yanli Liu and *et al*. Single-cell measurements of ige-mediated fceri signaling using an integrated microfluidic platform. *PLoS ONE*, 8(3):60159, 2013.

[36] Dipak Barua, William S. Hlavacek, and Tomasz Lipniacki. A computational model for early events in b cell antigen receptor signaling: Analysis of the roles of lyn and fyn. *J. Immunol.*, 189:646–658, 2012.

[37] Michael W. Sneddon, James R. Faeder, and Thierry Emonet. Efficient modeling, simulation and coarse-graining of biological complexity with nfsim. *Nature Methods*, 8(2):177–83, 2011.

[38] Anne Condon, David Harel, Joost N. Kok, Arto Salomaa, and Erik Winfree. *Algorithmic Bioprocesses*. Springer, 2009.