Exploring the Performance of Spatial Stochastic Simulation Algorithms

Matthias Jeschke*, Roland Ewald, Adelinde M. Uhrmacher

Institute of Computer Science, University of Rostock, Joachim-Jungius-Str. 10, 18059 Rostock

Abstract

Since the publication of Gillespie's direct method, diverse methods have been developed to improve the performance of stochastic simulation methods and to enter the spatial realm. In this paper we discuss a spatial τ-leaping variant (Sτ) that extends the basic leap method. Sτ takes reaction and both outgoing and incoming diffusion events into account when calculating a leap candidate. A performance analysis shall reveal details on the achieved success in balancing speed and accuracy in comparison to other methods. However, performance analysis of spatial stochastic algorithms requires significant effort — it is crucial to choose suitable (benchmark) models and to carefully define model and simulation setups that take problem and simulation design spaces into account.

Key words: Stochastic Simulation, SSA, Performance Evaluation, τ-leaping, Next Sub-volume Method, Gillespie Multi-particle Method, Spatial τ-leaping, Performance Analysis

*Corresponding author

Email addresses: matthias.jeschke@uni-rostock.de (Matthias Jeschke), roland.ewald@uni-rostock.de (Roland Ewald), adelinde.uhrmacher@uni-rostock.de (Adelinde M. Uhrmacher)

Preprint

1. Introduction

Representing space in models is becoming of increasing interest, because todays advanced microscopy techniques allow the exact localization of molecules in different compartments, e.g., the cytoplasm or nucleus. In contrast to the well-stirred environment assumed by non-spatial methods (see, e.g., [13, 12, 15]), the interior of a cell is much more complex to capture: some structures are located in specific regions, e.g., the endoplasmic reticulum; molecules constantly bump into each other, which slows down the speed at which they travel through the cytosol (this effect is known as *molecular crowding*, cf. [17, 27]); and even in the dilute case, the reactant partners have to find each other before they can actually react.

The last decades brought forth a multitude of spatial stochastic simulation algorithms, e.g., [9, 21, 30], each having different characteristics: some are exact in the sense that they simulate every reaction and diffusion within the system, others may trade accuracy for execution speed by introducing additional approximations, and certain methods even use more than one technique to accomplish the task (e.g., [11]).

But developing an algorithm is merely the first step, only a *performance analysis* can establish its usefulness. There are two main approaches to do so: one is to manually deduce performance properties from an abstract description of the algorithm, the other is to empirically observe the performance of a concrete implementation on real hardware. Being undeniably useful, results of a theoretical analysis — like asymptotic complexity [24] — still may miss important aspects of algorithm performance in reality: the model of a modern

computer having, e.g., a multi-layered memory hierarchy (e.g., cf. [25]) and multi-core processor is only insufficiently captured by a sequential, unit cost Turing machine. Another problem is the dependence of algorithm performance not only on problem size but also on problem *structure*, i.e., *multiple* input features. This is usually neglected in theoretical performance analysis, as it often deteriorates tractability.

Additional care must be taken when the algorithm under scrutiny depends on *sub-algorithms*, e.g., an event queue [16, 19]. They are so common and ubiquitous in simulator development that they are often simply assumed "to be present", without considering them as independent variables that may have a strong impact on overall performance.

All in all, a solely theoretical investigation of algorithm performance does not appear to be sufficient [23] and needs to be complemented by empirical performance studies that directly measure the run time and other performance facets. We show that a wide range of different model characteristics and sub-algorithm implementations has to be considered in order to avoid biased results. To facilitate the former, we will discuss the benefits of using *benchmark models*: instead of searching for a real model that exhibits a certain feature, it may be a good alternative to define the characteristic of interest, e.g., the initial and steady state particle distribution, and build a flexible, parameterizable benchmark model for its representation. While the result does not represent a real system, it could still be worthwhile to consider because it facilitates comparability and scalability.

2. Background: Spatial Stochastic Simulation

Suppose a system containing a set of N chemical species $S = \{S_1, \ldots, S_N\}$ that can interact according to M reaction equations $R = \{R_1, \ldots, R_M\}$. For simplicity, let the volume of the system be a cube with side length λ. In the non-spatial case we assume that the particles of the species are distributed homogeneously within the volume — the system is *well-stirred*. However, sometimes this assumption does not hold; species could be present only in specific regions of space, e.g., near a membrane, or the time it takes for them to travel a certain distance cannot be neglected. One way to represent an *inhomogeneous* distribution is to discretize the volume into L smaller *sub-volumes* (SV), each one having a side length $\lambda_{sv} = \lambda / \sqrt[n]{L}$ and being well-stirred. The variable n shall denote the dimension of the system, so $n = 2$ represents a two-dimensional and $n = 3$ a three-dimensional volume.

Apart from participating in reactions, particles can also diffuse from a sub-volume into a neighboring site. This requires the assignment of *diffusion coefficients* to each species, which are stored inside an additional set $D = \{D_1, \ldots, D_N\}$. Furthermore, an $L \times 2n$ *connectivity matrix* $\mathcal{C}$ shall store the neighbors for each sub-volume, with the submatrix $\mathcal{C}_l \equiv \mathcal{C}[l; 1 \ldots 2n]$ holding the diffusion targets for particles leaving l.

With L sub-volumes, the *state* of the entire system can be represented by an $L \times N$ *state matrix* $\mathbf{X}$, with entry $x_{l,i}$ giving the number of S_i particles inside sub-volume l. Without knowing the velocity, energy and several other properties of all molecules at any time, reactions within sub-volumes and the movement of particles seem to take place at random. The evolution of the system, i.e., how the number of particles in each sub-volume changes over

time, can be therefore interpreted as a stochastic process $\mathbf{X}(t)$ over a matrix of random variables and $\mathbf{X}(t) = \mathbf{X}$ shall denote the current state; similar to the connectivity matrix, $\mathbf{X}_l$ represents the row vector $\mathbf{X}[l; 1 \dots N]$, i.e., the state of sub-volume $l \in [1, L]$.

Let us now focus on the reactions. Generally, an equation in R can be written as:

$$R_j : \sum_{i=1}^{N} v_{ji}^{r} S_i \xrightarrow{c_j} \sum_{i=1}^{N} v_{ji}^{p} S_i, \tag{1}$$

with $S_i \in S$ and $v_{ji}^{\{r,p\}}$ as stoichiometric parameters. The *stochastic rate constant* c_j depends on physical properties of the reactant species (e.g. molecular weight, size), the solvent, temperature, and other conditions [14]. It is defined such that $c_j dt$ gives the probability that during dt a *particular set* of reactant particles of R_j interact via a reactive collision inside the system. For convenience, if $v_{ji}^{r} = 0$ or $v_{ji}^{p} = 0$, then species S_i will not be listed as a reactant or product, respectively, in the equation. The *state change vector* $\mathbf{v}_j = (-v_{j1}^{r} + v_{j1}^{p}, \dots, -v_{jN}^{r} + v_{jN}^{p})$ summarizes the changes in the number of particles for each species if R_j takes place.

Based on the definition of c_j, the *propensity* $a_j^l(\mathbf{X})$ for reaction R_j is defined as $a_j^l(\mathbf{X}) = H_j^l(\mathbf{X}) c_j$, with $\mathbf{X}(t) = \mathbf{X}$ as the current state, $H_j(\mathbf{X})$ as the function calculating the number of reactant combinations for R_j in l, and $a_j^l(\mathbf{X})dt$ as the probability that there will be a reaction R_j during dt *somewhere* inside l. After executing R_j in sub-volume l the new state at time $t + \Delta t$ is $\mathbf{X}_l(t + \Delta t) = \mathbf{X}_l + \mathbf{v}_j$.

Stochastic simulation algorithms can be separated into *exact* and *approximative* variants. Exact algorithms, like the Next Sub-volume Method (NSM, [9]) simulate every single event that occurs inside the system. In a nutshell,

they find out *what* event type (reaction or diffusion) will take place next and *when* this will happen; with this information the algorithm can advance the time and update the state of the system. But with large propensities (due to high copy numbers of molecules and/or large stochastic rate constants or diffusion constants), the time step between successive events might decrease drastically, rendering the simulation progress very slow. Another problem arises when the model at hand is inherently *stiff* [cf. 6, p. 67 ff.], i.e., events occur on largely differing time scales, with fast events (e.g., reactions with a high propensity value) reaching equilibrium very quickly. In this case an exact algorithm would be busy executing only a subset of the events for most of the time while the remaining, e.g., slow reactions, would have a greater influence on the model dynamics. In contrast, approximative algorithms (e.g., the Gillespie Multi-particle Method GMPM, [30]), as the name already implies, trade accuracy for execution speed, making it possible to simulate larger models.

3. Spatial τ-leaping and the Sτ algorithm

Before we delve into this alternative simulation algorithm, we will first give a rough sketch of the basic leap idea, which has been introduced by Gillespie et al. [15] (and later improved by several authors, see, e.g., [4, 29, 34]) to speed up the simulation of biochemical reaction networks.

Suppose a single sub-volume l, whose state is given by $\mathbf{X}_l$, and ignore any outgoing or incoming diffusion events for the moment. Instead of simulating every single reaction that occurs inside it, as done by exact algorithms, τ-leaping performs "leaps" along the time line, calculating for each reaction the

number firings K_j during such an interval. The size of a leap, in the following denoted by τ, is restricted by the so-called *leap condition*, which states that the propensities $a_j^l(\mathbf{X}), j \in [1, M]$ have to remain nearly constant during τ. Gillespie showed that if this condition is fulfilled, an approximation for K_j can be given by a sample from the Poisson random variable $\mathcal{P}_j(a_j^l(\mathbf{X}), \tau)$ with mean and variance $a_j^l(\mathbf{X})\tau$.

Spatial τ-leaping methods additionally include diffusion events between sub-volumes into the calculation for a τ value. For example, Márquez-Lago and Burrage took binomial τ-leaping as the basis and proposed a spatial algorithm that performs local leaps: the sub-volume with the minimum next event time is selected and based on its current state the number of reaction and diffusion events taking place within it get calculated [26]. Though considering particles leaving the sub-volume, the method ignores the ones *entering* it during the leap, which could lead to inaccuracies.

While accounting for incoming particles, Rossinelli et al. [31] split the τ calculation for a single sub-volume to find a leap candidate for reaction and diffusion events independently; the final leap value is then the minimum of both. However, reaction and diffusion events take place concurrently, so both event types are *not independent* — an argument also raised in a recent publication by Iyengar et al. [21] which introduces a spatial τ-leaping variant very similar to the one developed for this paper.

Other authors also used the leap method to simulate spatially inhomogeneous systems, e.g., [35, 8]. An interesting hybrid algorithm was proposed recently [11], with the aim to speed up the simulation of the diffusion events. The distinct feature of this method is that it decides for each species and sub-

volume neighbor pair whether to use a macroscopic (numerical integration), approximative mesoscopic (τ-leaping), or exact mesoscopic (NSM) diffusion scheme for the current iteration.

3.1. Derivation of Sτ

A new reaction set R' is defined as

$$R' = R \cup \bigcup_{i=1}^{N} \left\{ R_{M+i} : S_i \xrightarrow{c_i} \emptyset \right\},$$

with $c_i = 2nD_i/\lambda_{sv}^2$. The additional reactions represent the diffusion of species into neighboring sub-volumes. The diffusion rate $\gamma_i^l(\mathbf{X})$ for species S_i can be calculated as $\gamma_i^l(\mathbf{X}) = c_i x_{l,i}$; unlike Iyengar et al., having only a single rate per species and sub-volume means we assume that diffusion is equally possible in all directions. Each new reaction R_{M+i} inside R' has a state change vector $\mathbf{v}_{M+i} = -\mathbf{1}_i^N$, i.e., the N-dimensional vector with $v_{M+i,i} = -1$ and $v_{M+i,j} = 0, j \neq i$. For each reaction $R_j \in R'$ the propensity $\xi_j^l(\mathbf{X})$ is:

$$\xi_j^l(\mathbf{X}) = \begin{cases} a_j^l(\mathbf{X}) & j \in [1, M] \\ \gamma_{j-M}^l(\mathbf{X}) & j \in [M+1, M+N]. \end{cases}$$

The change of the propensity for a given leap value τ can be written as:

$$\begin{aligned} \Delta\xi_j^l(\mathbf{X}) \triangleq & |\xi_j^l(\mathbf{X} + \mathbf{\Lambda}(\tau; \mathbf{X})) - \xi_j^l(\mathbf{X})| \\ \approx & \operatorname{tr}(\nabla \xi_j^l(\mathbf{X})^T \mathbf{\Lambda}(\tau; \mathbf{X})) = \sum_{l'=1}^{L} \sum_{i=1}^{N} \Lambda_{l',i}(\tau; \mathbf{X}) \frac{\partial \xi_j^l(\mathbf{X})}{\partial x_{l',i}}; \end{aligned} \tag{2}$$

here $\mathbf{\Lambda}(\tau; \mathbf{X})$ represents the state change from $\mathbf{X}$ to $\mathbf{X}(t + \tau)$ and the second line is a result of a Taylor expansion around $\mathbf{X}$. To satisfy the leap condition, we want the mean and variance of $\Delta\xi_j^l(\mathbf{X})$ to be bounded by

$\max\{\epsilon\xi_j^l(\mathbf{X}), c_j\}$, with ϵ as the *error control parameter*. Note that we use the modified requirement as proposed by Cao et al. in [4]; this is necessary because empty sub-volumes would produce a τ candidate which is zero – and the simulation would come to a halt. By focusing solely on sub-volume l, $\partial\xi_j^l(\mathbf{X})/\partial x_{l',i}$ is zero for all $l' \neq l$ and thus we need to consider only the l-th row vector of $\mathbf{\Lambda}(\tau;\mathbf{X})$, denoted by $\mathbf{\Lambda}_l(\tau;\mathbf{X})$, which tells us how the population in l changes during the leap. In the original non-spatial τ-leaping algorithm, $\mathbf{\Lambda}_l(\tau;\mathbf{X})$ only depends on the reactions inside sub-volume l. But with reaction-diffusion systems the state can now also be altered by diffusing particles that either enter or leave l. Taking this into account, the state change for l is:

$$\mathbf{\Lambda}_l(\tau;\mathbf{X}) = \sum_{j'=1}^{M+N} K_{j'}^l(\tau;\mathbf{X})\mathbf{v}_{j'} + \sum_{i=1}^{N} I_i^l(\tau;\mathbf{X})\mathbf{1}_i^N; \tag{3}$$

the rightmost term sums up the changes made by particles diffusing into SV l. As in the non-spatial τ-leaping algorithm, if a leap value τ satisfies the leap condition, the number of firings for events $R_{j'} \in R'$ inside l during τ can be approximated by a Poisson random variable $\mathcal{P}(\xi_{j'}^l(\mathbf{X}), \tau)$ with mean and variance $\xi_{j'}^l(\mathbf{X})\tau$, so

$$K_{j'}^l(\tau;\mathbf{X}) \approx \mathcal{P}(\xi_{j'}^l(\mathbf{X}), \tau). \tag{4}$$

The question is now how to approximate $I_i^l(\tau;\mathbf{X})$. For this, let us define an *incoming diffusion rate* as follows:

$$\beta_i^l(\mathbf{X}) = \frac{1}{2n}\sum_{k=1}^{2n} \gamma_i^{\mathcal{C}_{l,k}}(\mathbf{X})$$

Obviously, $\beta_i^l(\mathbf{X})$ depends on the diffusion rate $\gamma_i^{\mathcal{C}_{l,k}}(\mathbf{X})$ for species S_i in all neighbors $k \in [1, 2n]$ of SV l; multiplying it by the factor $1/2n$ results in the

rate *relative* to one neighbor.

If the leap condition can be satisfied for both the reaction propensities *and* the outgoing diffusion rates for all species and sub-volumes, then the number of incoming diffusion events can also be approximated with a Poisson random variable with mean and variance $\beta_i^l(\mathbf{X})\tau$, hence $I_i^l(\tau;\mathbf{X}) \approx \mathcal{P}(\beta_i^l(\mathbf{X}),\tau)$. Two things should be noted here: first, $\mathcal{P}(\beta_i^l(\mathbf{X}),\tau)$ is the sum of the independent Poisson random variables $\mathcal{P}(\gamma_i^{\mathcal{C}_{l,k}}(\mathbf{X})/2n,\tau), k \in [1,2n]$; we look at the states of all neighbors of SV l and approximate how many S_i particles will diffuse into l during a leap of size τ. In other words: if a sample of $\mathcal{P}(\gamma_i^l(\mathbf{X}),\tau)$ gives the number of S_i particles that will *diffuse out* of l (see eq. 4) and each of the $2n$ neighbors of l has the same probability of being selected as a target, then sampling from $\mathcal{P}(\gamma_i^l(\mathbf{X})/2n,\tau)$ provides the amount of particles that will *enter* one specific neighboring site. Secondly, after finding a value for τ, a sample f_i is only generated for $\mathcal{P}(\gamma_i^l(\mathbf{X}),\tau)$; then, for each of the f_i particles a target neighbor is selected from $\mathcal{C}_l$ with a point probability of $1/2n$.

Equation 3 can now be written as

$$\mathbf{X}_l(t+\tau) - \mathbf{X}_l \equiv \mathbf{\Lambda}^l(\tau;\mathbf{X}) \sum_{j'=1}^{M+N} \mathcal{P}(\xi_{j'}^l(\mathbf{X}),\tau)\mathbf{v}_{j'} + \mathbf{b}, \tag{5}$$

with the vector $\mathbf{b}$ defined as:

$$\mathbf{b} = \sum_{i=1}^{N} \mathcal{P}(\beta_i^l(\mathbf{X}),\tau)\mathbf{1}_i^N = (\mathcal{P}(\beta_1^l(\mathbf{X}),\tau),\ldots,\mathcal{P}(\beta_N^l(\mathbf{X}),\tau)).$$

Inserting equation 5 into 2 gives:

$$\Delta\xi_j^l(\mathbf{X}) \approx \sum_{i=1}^{N} \frac{\partial\xi_j^l(\mathbf{X})}{\partial x_{l,i}} \left(\sum_{j'=1}^{M+N} \mathcal{P}(\xi_{j'}^l(\mathbf{X}),\tau)v_{ij'} + b_i \right). \tag{6}$$

Writing the propensity change this way makes it very easy to explain what is calculated here: at first, we look at how the propensity depends on species S_i. If S_i is not a reactant of R_j, then $\partial\xi_j^l(\mathbf{X})/\partial x_{l,i}$ is zero and we do not have to consider the part inside the brackets. On the other hand, if reaction R_j depends on S_i, then we estimate how its state is changed during τ – which is given by the value inside the brackets: the first term sums up the changes made by reactions and outgoing diffusion events (again, $v_{ij'} = 0$ if S_i is neither a reactant nor product of $R_{j'}$), the second one counts how many S_i particles enter l.

Both the mean and the variance of the expected change in the propensity should be bounded by the value $\max\{\epsilon\xi_j^l(\mathbf{X}), c_j\}$. With equation 6, the estimates for both quantities are:

$$\left\langle \Delta\xi_j^l(\mathbf{X}) \right\rangle \approx \sum_{i=1}^{N} \frac{\partial\xi_j^l(\mathbf{X})}{\partial x_{l,i}} \left(\sum_{j'=1}^{M+N} \left\langle \mathcal{P}(\xi_{j'}^l(\mathbf{X}), \tau) \right\rangle v_{ij'} + \langle b_i \rangle \right)$$

$$\operatorname{var}\left\{ \Delta\xi_j^l(\mathbf{X}) \right\} \approx \sum_{i=1}^{N} \left(\frac{\partial\xi_j^l(\mathbf{X})}{\partial x_{l,i}} \right)^2 \left(\sum_{j'=1}^{M+N} \operatorname{var}\left\{ \mathcal{P}(\xi_{j'}^l(\mathbf{X}), \tau) \right\} v_{ij'}^2 + \operatorname{var}\{b_i\} \right)$$

The mean and variance for the Poisson distribution are $\left\langle \mathcal{P}\left(\xi^l, \tau\right) \right\rangle = \operatorname{var}\left\{ \mathcal{P}\left(\xi^l, \tau\right) \right\} = \xi^l\tau$.

$$\left\langle \Delta\xi_j^l(\mathbf{X}) \right\rangle \approx \sum_{i=1}^{N} \frac{\partial\xi_j^l(\mathbf{X})}{\partial x_{l,i}} \left(\sum_{j'=1}^{M+N} \xi_{j'}^l(\mathbf{X})\tau v_{ij'} + \beta_i^l(\mathbf{X})\tau \right) \equiv \mu_j^l\left(\mathbf{X}\right)\tau$$

$$\operatorname{var}\left\{ \Delta\xi_j^l(\mathbf{X}) \right\} \approx \sum_{i=1}^{N} \left(\frac{\partial\xi_j^l(\mathbf{X})}{\partial x_{l,i}} \right)^2 \left(\sum_{j'=1}^{M+N} \xi_{j'}^l(\mathbf{X})\tau v_{ij'}^2 + \beta_i^l(\mathbf{X})\tau \right) \equiv \left(\sigma_j^l(\mathbf{X})\right)^2 \tau$$

With the help of $\mu_j^l\left(\mathbf{X}\right)$ and $\left(\sigma_j^l(\mathbf{X})\right)^2$, an expression for calculating a τ candidate can be found:

$$\tau = \min_{l\in[1,L]} \left\{ \min_{j\in[1,M+N]} \left\{ \frac{\max\{\epsilon\xi_j^l(\mathbf{X}), c_j\}}{\left|\mu_j^l(\mathbf{X})\right|}, \frac{(\max\{\epsilon\xi_j^l(\mathbf{X}), c_j\})^2}{\left(\sigma_j^l(\mathbf{X})\right)^2} \right\} \right\}. \quad (7)$$

Having obtained a suitable τ, the next step would be to generate for each reaction R_j a sample from the Poisson distributed random variable (RV) $\mathcal{P}_j(\xi_j^l(\mathbf{X}), \tau)$. But this might lead to incorrect results. Poisson RV are unbounded and it could happen that the population of a species becomes negative as a consequence of too many reaction firings. Several methods have been published to prevent this, e.g., by using bounded binomial random variables [34, 7] or by classifying reactions into critical and non-critical and handling critical reactions differently [3] (the latter was chosen for the Sτ algorithm).

Most τ-leaping implementations switch to an SSA phase if τ is very small; the threshold for this decision is the time interval $\hat{\tau}$ until the next event will fire, which is approximately $1/\sum_{l=1}^{L}\sum_{j=1}^{M+N}\xi_j^l(\mathbf{X})$. If τ is smaller than a multiple θ of this value, e.g., $10\hat{\tau}$, then a number N_{SSA} of exact iterations is performed. Instead of calculating only small leap values, which is equivalent to a huge effort with only small gain, the algorithm tries to overcome the critical region in the state space by falling back to a much simpler and, in this case, often faster exact simulation.

3.2. A First Analysis

Apart from possible optimizations in the implementation, it also could be interesting to see how the basic algorithm behaves when faced with different problem instances. To do so, let us focus on the following two questions:

- Given a model, what are the conditions on the current state so that a Sτ leap is at least θ times larger than a single average NSM step $1/\sum_{l=1}^{L}\sum_{j=1}^{M+N}\xi_j^l(\mathbf{X})$?
- How does this value scale with the model size (e.g., with the number

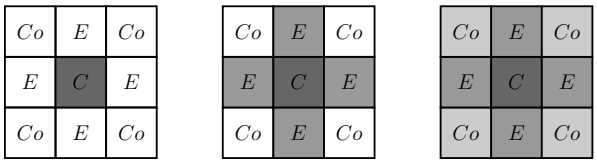


Figure 1: The "toy models" used for analyzing the spatial τ-leaping algorithm. (a) Only the center sub-volume contains A particles. (b) Some particles diffused into neighboring sub-volumes. (c) Each sub-volume now contains at least one A particle.

of sub-volumes or initial particles)?

While specific answers could be found for any model, it may be a better approach to start with small *benchmark problems* representing common model characteristics, e.g., *the spatial distribution of particles*. Figure 1 shows a simple 3×3 model with periodic boundaries and a single species A allowed to diffuse into every sub-volume. The problem is simplified by the assumption that each edge sub-volume (E) contains the same amount particles ($x_{E,A}$); this also applies for the corner sub-volumes (Co with $x_{A,Co}$). Initially, A particles are only present in the center (1a); as time progresses, they diffuse into the nearest neighbors (1b) and eventually reach the outermost sub-volumes (1c). In summary, this model represents the simplified transition of a system from an initially inhomogeneous to a well-stirred particle distribution.

Now the task is to analyze how the initial number of A particles present in the center, edge and corner SV for each of the scenarios shown in figure 1 influences the size of τ and to compare the result with the respective average NSM time interval. Having no reactions defined, the only entry for the vector

R' is the diffusion of A into neighboring sub-volumes, $R_1 : A \xrightarrow{c_1} \emptyset$, a first order reaction with $c_1 = 2nD_A/\lambda_{sv}^2$ and $a_1^l(\mathbf{X}) = c_1 x_{l,A}, l \in \{C, E, Co\}$; for simplicity, the diffusion constant D_A is set to $1\,\mathrm{s}^{-1}$ and the side length λ_{sv} to 1.[1] Furthermore, in a two-dimensional model ($n = 2$) each sub-volume has four neighbors, so taken all together the propensity function reduces to $a_1^l(\mathbf{X}) = 2nx_{l,A}\,\mathrm{s}^{-1} = 4x_{l,A}\,\mathrm{s}^{-1}$. Due to the simple structure of the model, it is not difficult to find the values for $\mu_1^l(\mathbf{X})$, and $\left(\sigma_1^l(\mathbf{X})\right)^2$:

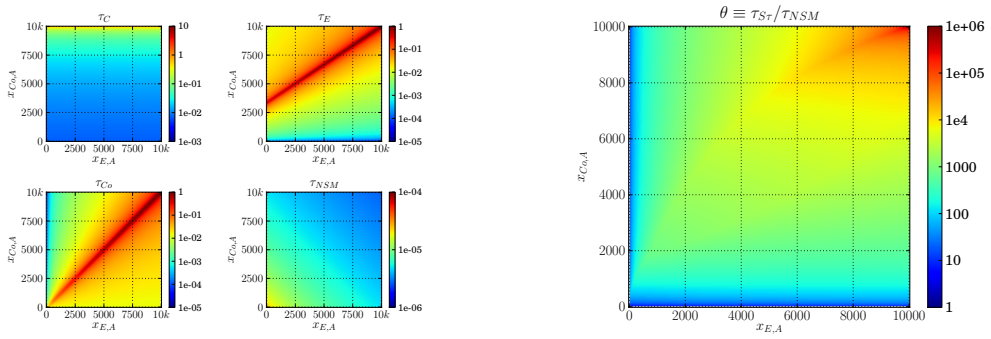
$$\begin{aligned}
\mu_1^C(\mathbf{X}) &= -16x_{C,A} + 16x_{E,A} & \left(\sigma_1^C(\mathbf{X})\right)^2 &= 64x_{C,A} + 64x_{E,A} \\
\mu_1^E(\mathbf{X}) &= -12x_{E,A} + 4x_{C,A} + 8x_{Co,A} & \left(\sigma_1^E(\mathbf{X})\right)^2 &= 80x_{E,A} + 16x_{C,A} + 32x_{Co,A} \\
\mu_1^{Co}(\mathbf{X}) &= -8x_{Co,A} + 8x_{E,A} & \left(\sigma_1^{Co}(\mathbf{X})\right)^2 &= 96x_{Co,A} + 32x_{E,A}.
\end{aligned}$$

Finally, getting a τ for each of the scenarios is simply a matter of inserting the results just presented into equation 7 and using an appropriate initial configuration of $\{X_C(0), X_E(0), X_{Co}(0)\}$.

For the following analysis the error control parameter ϵ has been set to 0.03. In the first scenario (1a, with $X_C(0) = [1, 10^4]$) τ_C will eventually converge to 0.0075 ($\epsilon/4$) and τ_E is always $1/x_{C,A}$ (in $\mu_1^E(\mathbf{X})$ and $\left(\sigma_1^E(\mathbf{X})\right)^2$, only the second terms are non-zero and the nominators in equation 7 are $c_1 = 4$ and $c_1^2 = 16$, respectively, so we have $\tau = \min\{4/(4x_{C,A}), 16/(16x_{C,A})\}$). The Next Sub-volume method takes a step of size $1/a_1^C(\mathbf{X}) = 1/4x_{C,A}$, so $\theta = 4$, i.e., a single leap is roughly equivalent to performing four NSM iterations — a sobering result, considering the complexity the Sτ algorithm.

[1] No units of length are given here as they are not relevant for now. Instead, D_A and λ_{sv} can be interpreted as being given *relative* to some unit length scale (e.g., if the scale is nm, then $D_A = 1\,\mathrm{nm}^2\mathrm{s}^{-1}$ and $\lambda_{sv} = 1\,\mathrm{nm}$).

For the next scenario (fig. 1b, $X_C(0) = 10^4$, $X_E(0) = [1, 10^4]$, $X_{Co} = 0$), the sites dominating the τ selection for most of the initial setups are the corner sub-volumes, similar to edges in the previous example. Both the center and the edges would allow larger jumps, but the size will eventually be restricted by $1/(2x_{E,A})$ (which is obtained in a similar way as it has been done for the first scenario), i.e., the particles diffusing into the corners, and thus as $X_E(0)$ is increased θ will converge to 10. In contrast to the former scenario there is a peak of about $\theta \approx 60$ for the ratio between $\tau_{S\tau}$ and τ_{NSM}, located at $\{10^4, 384, 0\}$. With increasing $x_{E,A}$ in the edge sub-volumes, the difference between particles coming from and leaving towards the center decreases, and so does $\left|\mu_1^E(\mathbf{X})\right|$. But at the same time the ratio $1/(2x_{E,A})$ gets lower, leading to smaller τ candidates for the corners; the peak therefore represents the optimum number of particles at the edge sites, i.e., the configuration yielding the maximum leap value. This is an important observation indicating that Sτ's performance (and the performance of spatial leap methods in general) is significantly dependent on the particle distribution: for the best run time, the difference between the amounts of particles moving into or out of a sub-volume has to be small, otherwise the large change in the diffusion rate would only permit small time intervals. It should also be remembered that a τ value is calculated over *all* sub-volumes; even if there is only one pair of adjacent sites having large differences in the population of a species, then this may dominate the size of the leap to a strong extent. Figure 2 shows the results for the third scenario (fig. 1c, $X_C(0) = 10^4$, $X_E(0) = [1, 10^4]$, and $X_{Co}(0) = [1, 10^4]$), which look much better. There are optimal setups for each of the τ candidates (the "red stripes" in fig. 2a), but the overall result also



(a) (b)

Figure 2: Leap and θ values for the third toy scenario (fig. 1c). Now both $X_E(0)$ and $X_{Co}(0)$ are independently varied in the interval $[1, 10000]$ while $X_C(0)$ is again fixed to 10^4. With a high overall population a single Sτ iteration is equivalent to performing several thousands of NSM steps.

looks quite promising: with large numbers of A particles in all sub-volumes, θ can range up to 10^5, so a single Sτ leap iteration encompasses $\approx 10^5$ NSM iterations. Even low populations allow considerable jump intervals compared to a single NSM step, e.g., $\theta \approx 54$ for $\{10^4, 350, 100\}$. But it should be emphasized at this point that a value of $\approx 10^5$ is *not* the attainable speed-up — this would only be the case if the time required to perform a single iteration is the same for NSM and Sτ, which is very unlikely due to a significant overhead inherent to the leap size calculation. As an example, if we found out that the time it takes to calculate a leap candidate and perform the jump corresponds to the execution of, e.g., 500 NSM steps, then Sτ should be faster by a factor of approximately 200 — at least for the type of models discussed in this example.

4. Performance Evaluation

All methods are implemented as plug-ins for the open-source modeling and simulation framework *JAMES II* [18]. It is written in Java 1.6 and is focused on delivering re-usable software components — so-called plug-ins — for simulation applications in general. A plug-in may rely on other plug-ins of a certain *type*, e.g., event queues, to fulfill auxiliary tasks. As these sub-algorithms may impact performance considerably, the *design space* of a spatial SSA approach should be explored by exchanging them (or changing parameters). Therefore, we speak of *simulation setups* — combinations of parameterized plug-ins for executing a simulation — instead of simulation algorithms. We restrict our study to *benchmark models*. Though not of immediate biological relevance, their application still has several advantages compared to real-world models (e.g., cf. to [20, p. 202] and [22, p. 219-220]): they improve comparability, provide more relevant parameters to be investigated, are easier to implement, and may allow to derive some analytical results to steer (or validate) the empirical performance analysis (see, e.g., the last section). A suitable benchmark model subsumes sets of more realistic models all exhibiting a specific property.

4.1. Performance Metrics

Not all simulation setups compute exact model trajectories: those containing Sτ or GMPM implementations give approximations instead. It is therefore necessary to evaluate both the execution speed and the accuracy of a setup. The execution speed of a simulation setup can be measured per single execution, i.e., by stopping the time it takes until the computation

is finished. The accuracy of the obtained results, however, can only be estimated by comparing result *distributions* of approximative and exact variants. What is considered a *good* performance ultimately depends on what the user regards to be a good compromise between execution speed and accuracy.

To make valid execution speed comparisons, our models are designed so that all trajectories impose similar computational load. Multiple replications are required nonetheless, to deal with the noise introduced by hardware, operating system, and virtual machine. Note that we are interested in the *relative* runtime performance of the proposed methods. While our Java implementations may not be on par with hand-crafted C code, general trends and findings should still be transferable to other circumstances.

The accuracy evaluation for the spatial SSA algorithms is based on a two-sample Chi-square test (see, e.g., [28]), which checks whether the *marginal frequency distributions* (i.e., estimations for the marginal probability density functions) for two samples are similar. During a *calibration step* we store samples of the model state at a prespecified time $\hat{t}$ into *sets* (in our case, 50 samples per set); here the NSM takes the role of the exact algorithm. Similar to the approach presented by Cao and Petzold [5], we use it to create a threshold for the actual comparison. A single $X^2_{\mathbf{p},\mathbf{q}}$ candidate is obtained by comparing two sets $\mathbf{p}$ and $\mathbf{q}$:

$$X^2_{\mathbf{p},\mathbf{q}} = \sum_{l=1}^{L} \sum_{i=1}^{N} \sum_{k=1}^{K} \frac{(\hat{p}_{l,i,k} - \hat{q}_{l,i,k})^2}{\hat{p}_{l,i,k} + \hat{q}_{l,i,k}}. \tag{8}$$

The values $\hat{p}_{l,i,k}$ and $\hat{q}_{l,i,k}$ are the relative frequencies for state k of species S_i in sub-volume l. With a total of 100 sets, 4950 X^2 candidates are generated. Based on these and the defined significance level (here, $\alpha = 0.05$) a threshold

value X_t^2 for rejection is calculated.

4.2. Sub-Algorithms & Hardware

We compare the performance of spatial τ-leaping with that of the Next Sub-volume Method (NSM) and Gillespie's Multi-particle Method (GMPM), both of which make use of an event queue. The set of event queues under scrutiny has been restricted to a simple list implementation (SIMPLE), a heap event queue (HEAP — proposed by Gibson and Bruck in [13]), an MList with an additional variant for faster event time updates (MLIST/MLISTRE — see [16]), and a TwoList implementation (TWOLIST — see [1]; we use the implementation from the `TwoList2` plug-in of *JAMES II*). A previous study [22] showed that random number generator (RNG) run time is negligible for the overall execution time when simulating well-stirred systems, as it is dominated by the execution time of the event queue implementation. Since we expect the same to hold for reaction-diffusion systems, we refrain from exploring the impact of different RNGs on execution time and use the implementation provided by the Java Base Libraries.

The performance measurements have been obtained from a Windows XP 64-bit workstation with two 2.5-GHz QuadCore Xeon Processors and 8 GB RAM. We never executed more than 6 simulation runs concurrently — two of the eight cores could therefore handle external load at any given time. Each sequential experiment was assigned to a dedicated core, which achieves a composite score of 528 for the (large) Java SciMark2 [33] benchmark (with JDK 1.7-beta 64-bit).

4.3. Benchmark Models

The test models for reaction-diffusion systems are specifically designed to represent different conditions of spatial inhomogeneities in species distribution. We use four model types: two variants of a *radial model*, the *molecule generator model*, and the *protein phosphorylation model*. Common to all is a cube-shaped volume with a variable number of sub-volumes per dimension, ranging from a small $5 \times 5 \times 5 = 125$ SV setup to a large one with $21 \times 21 \times 21 = 9261$ sub-volumes.

Radial Model. The first variant consists of only a single species A which can diffuse freely inside the system; it is initially located only in the center sub-volume. The interesting characteristic of this model is the transition from a complete inhomogeneous to a nearly uniform molecule distribution. The second variant includes two additional species and a single reaction rule:

$$R_1 : A + B \xrightarrow{c_p} C$$

Similar to the first, species A is located within the center sub-volume only. In contrast, species B is distributed homogeneously throughout the volume. Both A and B can diffuse freely. Though the two variants start with A inside a single sub-volume, the first one models its distribution over time inside an empty volume, whereas the second represents a point-sized injection of A into an area and the successive reaction with particles that are already present.

Molecule Generator Model (MolGen).

$$\begin{aligned} R_1 &: C \xrightarrow{c_p} C + A \\ R_2 &: A \xrightarrow{c_d} \emptyset \end{aligned} \tag{9}$$

At first glance the generator model resembles the first radial model variant, but there are several differences. While in the latter the number of A molecules is fixed for the entire simulation time, another type of molecule, C, now constantly produces A molecules in the center sub-volume. Reaction R_1 does not consume the generator molecule, so the C concentration is kept constant during the simulation interval. Furthermore, the A molecules degrade at a specific rate; instead of eventually ending up with almost the same concentration of A particles in every sub-volume, the generator model keeps an inhomogeneous distribution.

Protein Phosphorylation Model (Phospho).

$$\begin{aligned} R_1 &: P + K \xrightarrow{c_1} PP + K \\ R_2 &: PP + PH \xrightarrow{c_2} P + PH. \end{aligned} \tag{10}$$

Phosphorylation is the process of attaching a phosphate to another molecule, which often plays a crucial role in the activation and deactivation of enzymes. The model is based on the one used by Brown and Kholodenko in [2]: a protein P is phosphorylated by a membrane-bound kinase K and dephosphorylated by a cytosolic phosphatase PH. Particles for K and PH are only present in the leftmost and rightmost columns of the grid, respectively, while P and PP are allowed to diffuse freely.

4.4. Parameter Settings

We varied the following model parameters: number of sub-volumes, initial state of the system, distribution of particles, and diffusion constants stored in D. Adding more sub-volumes to a model should increase the run time for each of the algorithms under study. How the particles are initially distributed

may not be that relevant for the performance of both the NSM and GMPM. For example, it does not matter if there are 10000 particles in only one of 100 sub-volumes or 100 particles at each site, the average NSM time step $(1/\sum_{l=1}^{L}\sum_{j=1}^{M+N}\xi_j^l(\mathbf{X}))$ is the same for both scenarios. In case of the GMPM, the length of the intervals between diffusion events is entirely independent of the system's state and during the SSA phase the time until the next reaction takes place in any SV is approximately $1/\sum_{l=1}^{L}\sum_{j=1}^{M}\xi_j^l(\mathbf{X})$. However, spatial τ-leaping is a different subject: the closer a system is to a homogeneous distribution, the better Sτ should perform compared to an NSM execution (see sec. 3.2).

Furthermore, note that we only varied the diffusion but not the reaction constants; for all experiments the latter has been set to fixed values. We are aware that modifying the reaction constants likely has a serious impact on the performance of the algorithms as they are an integral part of the reaction propensity calculation. However, to limit the number of independent variables, we have decided to vary the *ratio* between diffusion and reaction constants instead.

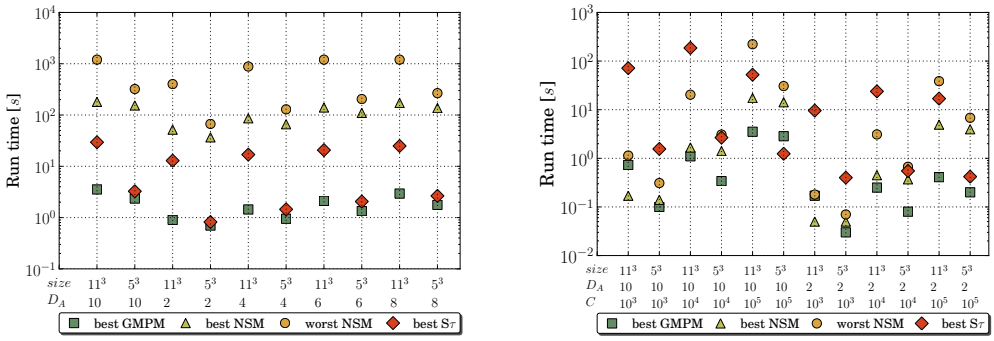
4.5. Performance Results

Overall Execution Speed. We studied more than 150 model setups by combining several *JAMES II* components to semi-automatically analyze algorithm performance: a performance database, a replication technique that allows to focus on the best-performing simulation setups, and a mechanism that selects a suitable simulation end time for a given model setup [10]. Executions that exceeded a predetermined timespan were aborted. If not stated otherwise, the maximal admissible duration for a single run was set to 20

minutes. Aborted runs are marked by an asterisk (*) in the graphs. Each run was replicated at least four times. We checked the standard deviation of all replicated run times, but found them to be in reasonable limits: for all runs longer than a couple of seconds, it was much below 10% of the execution time. For the sake of clarity, we omit the standard deviation and only show averaged results.

Both GMPM and NSM have been combined with the event queue implementations listed in the last section. Furthermore, we defined $7 \times 6 \times 5 = 210$ simulation setups for Sτ: all combinations of $n_c \in \{2, 3, 4, 5, 7, 9, 10\}$ (which is used to separate the set of reactions into non-critical and critical channels, see [3]), $\epsilon \in \{0.03, 0.034, 0.038, 0.042, 0.046, 0.05\}$ (the error control parameter), and $N_{SSA} \in \{10, 20, 30, 40, 50\}$ (the number of exact SSA iterations that will be executed if the leap value is too small). Several publications mention what parameter sets turned out to be suitable during their study. The values for this paper have been chosen to represent a common range of parameterizations, as suggested in various literature, e.g., [4, 32].

Figure 3a presents some rather encouraging performance results for Sτ, which consistently outperforms NSM but in turn is outperformed by GMPM. We expected GMPM to be faster than Sτ because its strong approximations cut much computational load. Another interesting aspect of the results in figure 3a is the impact of the event queue implementation on NSM performance. If the best Sτ setup is compared to the best NSM setup, it outperforms it usually by about one order of magnitude. If, however, we compare it to the *worst* NSM setup, the comparison gets even more impressive. This clearly shows two important points: a suitable event queue is essential for NSM



(a) (b)

Figure 3: Execution times for the Molecule Generator model. (a) Results for a model with 10^6 C particles in the central sub-volume, the diffusion constant D_A varies from 2 to 10 and the model has a size of either 5^3 and 11^3 sub-volumes (*size*). The simulation end time has been set to 3.0 s. (b) The same setup as in (a), with the amount of C particles varied between 10^3, 10^4, and 10^5.

efficiency, and not re-validating runtime performance with *different* implementations may introduce a strong bias to any performance comparison for this family of algorithms. For the simulation setups given above (and almost all others) we found the `MListRe` event queue to deliver the best results.

Finally, figure 3a also shows that the performance of Sτ strongly depends on the model size: the Sτ run times between otherwise equal model setups having 5^3 or 11^3 sub-volumes differ significantly. Apart from that, the general trend is that increasing D_A also increases the run times of all algorithms — this is straightforward, as an increased diffusion rate leads to more events within the same simulation time span. The diffusion rate $d_i^l(\mathbf{X}) = 2nD_i x_{l,i}/\lambda_{sv}^2$ does not only depend on D_i, but also on a) the side

length of a sub-volume and b) the number of particles inside it. To observe how the algorithms perform when faced with different diffusion rates, we varied two of these values and left λ_{sv} unaltered.

After observing the favorable run times of the new Sτ algorithm, it may seem safe to move forward and check its accuracy with respect to NSM. However, the point of experimentally studying a new algorithm is to explore the *regions* in the problem space where it is beneficial — delineating them implies to also find the regions where an algorithm performs *worse* than its competitors. Figure 3b shows the run times for the same setups of the Molecule Generator model, but now varying the number of C particles from 10^3 to 10^5 (instead of 10^6, as in fig. 3a). The results for $D_A \in \{4, 6, 8\}$ are similar. Here, it becomes apparent that Sτ does *not* always outperform NSM: indeed, it only does so when applied to relatively small (5^3 SVs) Molecule Generator models that contain many particles ($> 10^5$). Changing the diffusion constant D_A seems to have less impact on relative algorithm performance. It can also be seen that the impact of the event queue on NSM performance grows with model size, i.e., the difference between best and worst NSM simulation setup increases. Also note that for some model setups ($|C| = 1000$, size $= 11^3$) NSM is even *faster* than GMPM, although it does not use any approximations. This rather counter-intuitive finding again illustrates the importance of model features. Figure 3b also backs up what figure 3a already suggested, i.e., experiments that do not consider the event queue dependency of NSM are not meaningful: although the best Sτ setup outperforms the worst NSM setup in five out of twelve problems, it is in turn outperformed by the best NSM setup in three of these problems.

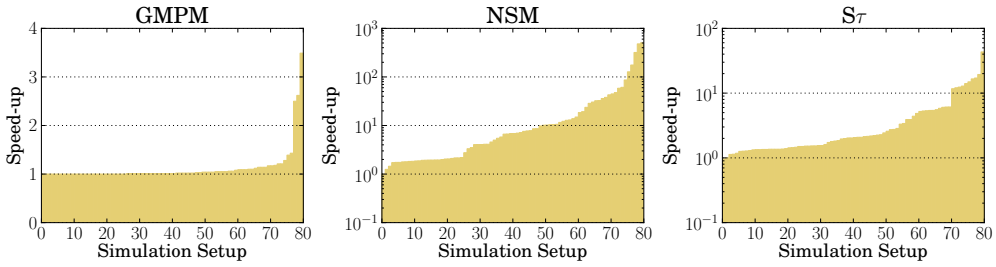


Figure 4: Sorted speed-ups attainable per algorithm for 79 Molecule Generator and Radial model set-ups. Note the different scales of the logarithmic y-axes.

Of course, the design space of simulation setups must not only be explored for NSM, but also for GMPM and Sτ. We analyzed the potential speed-up per reaction-diffusion algorithm, i.e., the execution time ratio between its fastest and its slowest simulation setup. This was done for 79 model setups of the Molecule Generator and Radial models; the results are shown in figure 4. The plots of the algorithms vary strongly from each other. GMPM run time, for example, does *not* depend strongly on the event queue that is used — it access it only infrequently and the performance difference between the best and the worst simulation setup with GMPM was usually less than 10% of the fastest execution time. For the problems we considered, GMPM with the TwoList queue performs best in ca. 60% of all setups; MListRe was second best (being optimal for ca. 20% setups). NSM, in contrast to GMPM, was again shown to be very sensitive to the choice of event queue (cf. fig 3a and 3b). For several models, selecting a proper event queue sped up execution *more than 50 times*. Here, the MListRe queue almost always worked best ($\approx 94\%$) and was very close to the optimum in all other cases. We therefore

restricted the design space explored in the following experiments to the NSM and GMPM setups that use the MListRe queue; for both algorithms it performs well on all model setups.

For Sτ, figure 4 shows that there are some cases in which its algorithm parameters have a huge impact on overall performance, leading to speedup higher than 20 — but for most cases the achievable speed-up is below 8. We further analyzed the impact of the individual parameters n_c, ϵ, N_{SSA}: of the 70 model setups for which the best Sτ setup needed more than 1 second to simulate, the optimal simulation setup was configured with $n_c = 2$ in 76% of the cases and with $n_c = 10$ in 21% of the cases. Hence, there seems to be a certain model property that makes choosing a large or small n_c value advisable. Further, we found out that the choice of ϵ has *little* impact on runtime performance: e.g., although ca. 84% of the best Sτ setups had ϵ set to 0.03, this is also true for the worst 70 % percent. The run times for a single model setup usually cluster around the different n_c values only. As we could neither make out any impact from N_{SSA}, we restrict the number of simulation setups for Sτ to four: a default one with $n_c = 10$, $\epsilon = 0.03$, and $N_{SSA} = 10.0$, and one adjusted setup for each parameter, i.e., one with $n_c = 2$, one with $\epsilon = 0.05$, and one with $N_{SSA} = 50.0$. All in all, we now have reduced the design space from over 200 simulation setups to just six: NSM + MListRe, GMPM + MListRe, and the four Sτ setups mentioned above.

Figure 5 shows algorithm performance on the Protein Phosphorylation model, where Sτ is clearly not the best choice. It barely manages to finish in *twice* the time that NSM execution requires. Here, GMPM is the fastest

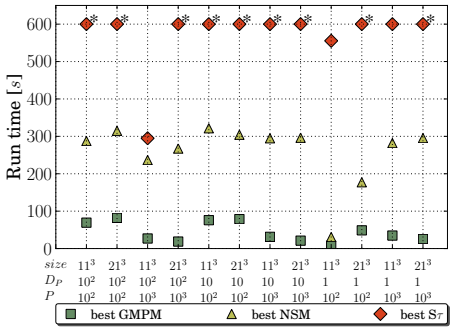
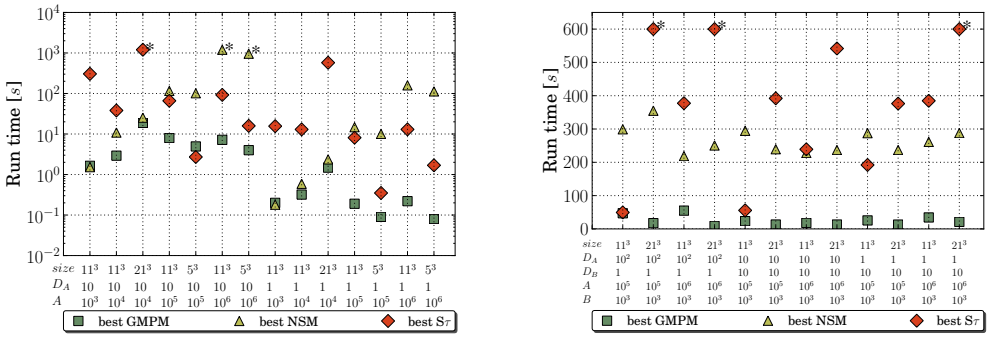


Figure 5: Algorithm performance on different set-ups of the phosphorylation model. Similar results were obtained for $|K| = |Ph| = 100$ instead of 1000. Simulation times vary across model set-ups, as their complexity differs strongly (they have been calibrated automatically [10] to let NSM run for $\approx 300s$, model setups with $d = 1, P = 100$ where so easy that even simulating until time 100 did not impose greater load for NSM). Unfinished executions were aborted after 10 minutes.

algorithm by far. While the various setups of the Phosphorylation model indicate where Sτ's performance is sub-optimal, algorithm performance on the Radial model is particularly interesting because it contains suitable scenarios for *both* NSM and Sτ. As figure 6a shows, both algorithms exceed the maximum allowed run time, and in both cases the respective other method was faster by roughly one order of magnitude. Again, Sτ could gain significant speed-up (over 47 in one case) when applied to small- and medium-sized models — but only if there are sufficiently many particles. It performed worst on models with a high number of sub-volumes. Similar results can be observed for the second variant of the Radial model; a subset of the results is depicted in figure 6b.



(a) (b)

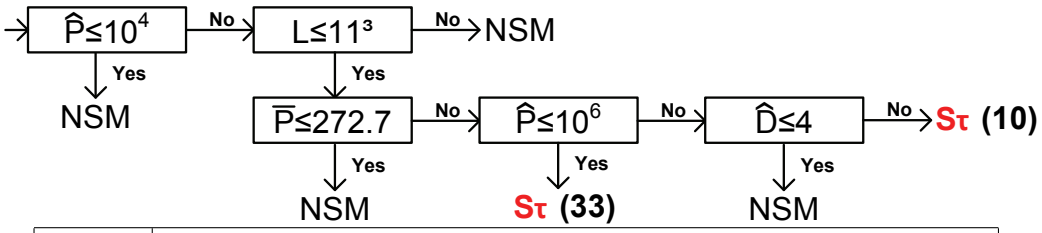
Figure 6: (a) Algorithm performance on different set-ups of the Radial model. Sτ outperforms NSM on small to medium models with sufficiently many particles. The simulation end time was set to 5 s. (b) Algorithm performance on different set-ups of the Radial 2 model.

Accuracy. One conclusion from the last paragraphs is: the GMPM is almost always the fastest algorithm, leaving its competitors often far behind. But the fastest algorithm is worth nothing if it fails to produce accurate results. Table 1 puts the results from the execution speed analysis into perspective. Using the test procedure introduced in section 4.1, the accuracy experiments show that, at least for the analyzed model setups, GMPM is far from capturing the spatial particle distribution accurately. The null hypothesis, stating that both NSM and GMPM sample from the same underlying distribution, is rejected for a significance level $\alpha = 0.05$ in each experiment. While GMPM fails to capture the spatial distribution for every test model, the spatial τ-leaping algorithm shows good performance, except for the molecule generator model. For the two radial variants and the phosphorylation model, Sτ matches the NSM outcome much better than its competitor.

			GMP		τ-leaping	
Model	Time	X_t^2	X_a^2	p_a	X_a^2	p
Radial, $D_A = 10$, $X_A(0) = 10^4$	0.1	168.84	923.4	0	162.97	0.49
	0.5	340.9	951.37	0	334.11	0.21
	0.9	354.02	418.17	0	343.18	0.43
Radial, $D_A = 1$, $X_A(0) = 10^5$	1	408.05	1657.71	0	401.23	0.47
	3	774.95	2652.98	0	770.54	0.13
	5	867.65	2662	0	855.94	0.39
Radial var. 2, $D_A = 10$, $D_B = 1$, $X_A(0) = 10^5$, $X_B(0) = 10^3$	0.1	1782.16	3129.69	0	1768.64	0.58
	0.2	1875.26	3183.37	0	1866.51	0.33
	0.3	1887.16	3127.39	0	1898.22	0.02
Phospho, $D_P = D_{PP} = 10$, $X_P(0) = 1.21 \cdot 10^5$, $X_K(0) = X_{PH}(0) = 100$	0.1	738.18	1099.98	0	732.85	0.19
	0.5	1112.22	1233.13	0	1099.12	0.34
	0.9	1240.17	1317.1	0	1218.55	0.77
MolGen, $D_A = 10$, $X_C(0) = 10^6$	0.1	146.91	371.06	0	761.47	0
	0.5	699.19	769.56	0	815.94	0
	0.9	967.69	989.91	0	960.96	0.21

Table 1: Accuracy results for five different reaction-diffusion models, evaluated at three time points each. The column X_t^2 lists the threshold values calculated during calibration; these are tested against the control values given in column X_a^2. The p-value for each model-algorithm combination is calculated using a significance level of $\alpha = 0.05$.

Performance Prediction. The accuracy results motivate to refocus on the run time performance of NSM and Sτ, as GMPM may often be too inaccurate. We therefore try to predict which algorithm is faster by applying the J48 decision tree learner from the WEKA machine learning toolkit [36, p. 159 et sqq.]. The learner is provided with features of the investigated model setups and their performance outcome (i.e., "NSM better" or "Sτ better"). The generated tree decides whether the best Sτ setup outperforms the best NSM



Symbol	Feature Description (see sec. 2)
$\hat{P}$	Maximum initial number of particles per SV: $\max_{i\in[1,L]}\{\sum_{j=1}^{N} X_j^i(0)\}$
L	Number of sub-volumes.
$\bar{P}$	Average initial number of particles: $\left(\sum_{i=1}^{L}\sum_{j=1}^{N} X_j^i(0)\right)/L$
$\hat{D}$	Maximum diffusion rate: $\max_{i\in[1,N]} D_i$

Figure 7: A decision tree to decide whether Sτ or NSM is faster. The root node is placed at the upper left corner. The tree misclassifies 12 of the 155 models (ca. 8%, checked with 10-fold cross-validation [36, p. 125]). The cases where Sτ is classified to be better are marked red, with the number of correctly classified examples given in parentheses.

setup on a model with given features. As shown in figure 7, the regions where Sτ outperforms NSM are characterized by small to medium numbers of sub-volumes ($L \leq 11^3$) and relatively many particles ($\bar{P} > 272.7$). This corresponds to our manual interpretation (e.g., see fig. 6a). Note that the numbers given in the tree (e.g., "$L \leq 11^3$") do not denote exact crossing points, as we conducted *factorial* experiments on the models: maybe Sτ also performs better for $L = 12^3$; we only checked for $L \in \{5^3, 11^3, 21^3\}$.

5. Discussion and Conclusion

We discussed the spatial leap method Sτ and compared its performance to existing algorithms in an extensive performance analysis. The spatial simulation algorithms considered in this study all have their strengths and weaknesses. GMPM's execution speed, for instance, was excellent for almost all model setups we considered; yet it lacks the accuracy of the other methods. In contrast, the NSM can be considered as being exact but slow for larger models. The performance of Sτ lies somewhere inbetween these two; it is generally more accurate than GMPM, but also takes longer to finish a single run. On the other hand, Sτ clearly outperforms NSM on various model setups (we observed a speed-up of up to 47), while still producing comparable results. The performance analysis allowed us to act upon the statements we made above: we showed that Sτ is unsuitable if a model exhibits certain parameters, e.g., only few particles are present.

Obtaining such results requires powerful tools both for recording the necessary data and for finding out which algorithm dominates in which region of the model feature space. By relying on plug-ins that can be easily exchanged even at run time, *JAMES II* allows to analyze an aspect that is usually neglected in simulator development and analysis: the interdependency between host and sub-algorithm(s). A striking example for this is the impact of the event queues on the overall execution speed for the NSM: simply replacing a queue could speed up the simulation by orders of magnitude (or do the opposite). But similar to the selection of a host-algorithm, a general statement about what queue implementation should be used cannot be made. Instead of picking real-world models for benchmarking, we purposefully designed ar-

tificial benchmark models that can be parameterized to capture a wide range of model characteristics, found in a wide range of systems. One advantage of this approach is the degree of control and knowledge an algorithm developer has when performing a performance study.

Future work. Our study hints at various directions for future research. The first one is a more in-depth exploration and consequently refinement of the Sτ algorithm. As an example, there might exist a state-dependent tipping point for some models, i.e., when the model reaches a certain state NSM should be replaced by Sτ. Secondly, the performance evaluation of Sτ is *not* exhaustive, although we evaluated more than 150 models and over 200 simulation setups: we did not consider the impact of different hardware architectures, the Java virtual machine, or the operating system. Finally, the accuracy analysis is still a compute-intensive and challenging task for which advanced statistical techniques will be explored in the future.

References

[1] John H. Blackstone, Gary L. Hogg, and Don T. Phillips. A two-list synchronization procedure for discrete event simulation. *Comm. ACM*, 24(12):825–829, 1981.

[2] Guy C. Brown and Boris N. Kholodenko. Spatial gradients of cellular phospho-proteins. *FEBS Lett.*, 457(3):452–454, September 1999.

[3] Y. Cao, D. T. Gillespie, and L. R. Petzold. Avoiding negative populations in explicit poisson tau-leaping. *J. Chem. Phys.*, 123:4104+, August 2005.

[4] Y. Cao, D. T. Gillespie, and L. R. Petzold. Efficient step size selection for the tau-leaping simulation method. *J. Chem. Phys.*, 124(4):044109, January 2006.

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

[6] Francois E. Cellier and Ernesto Kofman. *Continuous System Simulation.* Springer, March 2006.

[7] A. Chatterjee, D. G. Vlachos, and M. A. Katsoulakis. Binomial distribution based tau-leap accelerated stochastic simulation. *J. Chem. Phys.*, 122(2):024112, 2005.

[8] Abhijit Chatterjee and Dionisios G. Vlachos. Temporal acceleration of spatially distributed kinetic monte carlo simulations. *J. Comput. Phys.*, 211(2):596–615, 2006.

[9] J. Elf and M. Ehrenberg. Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. *Systems Biology, IEE Proceedings*, 1(2):230–236, 2004.

[10] Roland Ewald and Adelinde M. Uhrmacher. Automating the runtime performance evaluation of simulation algorithms. In *Proc. of the Winter Simulation Conference*, pages 1079–1091. IEEE Computer Science, 2009.

[11] Lars Ferm, Andreas Hellander, and Per Lötstedt. An adaptive algorithm for simulation of stochastic reactiondiffusion processes. *J. Comput. Phys.*, 229(2):343–360, January 2010.

[12] Thomas Fricke and Dietmar Wendt. The markoff-automaton - a new algorithm for simulating the time-evolution of large stochastic dynamic systems. *Int. J. Mod. Phys. C*, 6:277–306, Jul 1995.

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

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

[15] D. T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *J. Chem. Phys.*, 115(4):1716–1733, 2001.

[16] R.S. Goh and I.L. Thng. Mlist: An efficient pending event set structure for discrete event simulation. *International Journal of Simulation - Systems, Science & Technology*, 4(5-6):66–77, December 2003.

[17] D. Hall and A.P. Minton. Macromolecular crowding: qualitative and semiquantitative successes, quantitative challenges. *Biochim. Biophys. Acta.*, 1649(2)(1649):127–139, 2003.

[18] J. Himmelspach and A. M. Uhrmacher. Plug'n simulate. In *Proc. of the 40th Annual Simulation Symposium*, pages 137–143. IEEE Computer Society, 2007.

[19] Jan Himmelspach and Adelinde M. Uhrmacher. The event queue problem and pdevs. In *Proc. of the SpringSim '07, DEVS Integrative M&S Symposium*, pages 257–264. SCS, 2007.

[20] J. N. Hooker. Needed: An empirical science of algorithms. *Oper. Res.*, 42(2):201–212, 1994.

[21] Krishna A. Iyengar, Leonard A. Harris, and Paulette Clancy. Accurate implementation of leaping in space: The spatial partitioned-leaping algorithm. *J. Chem. Phys.*, 132(9):094101, 2010.

[22] M. Jeschke and R. Ewald. Large-scale design space exploration of ssa. In M Heiner and A M Uhrmacher, editors, *Proceedings of the CMSB 2008*, volume 5307 of *LNCS*, pages 211–230. Springer, 2008.

[23] D. Johnson. *Data Structures, Near Neighbor Searches, and Methodology*, chapter A theoretician's guide to the experimental analysis of algorithms, pages 215–250. American Mathematical Society, 2003.

[24] Donald E. Knuth. Big omicron and big omega and big theta. *SIGACT News*, 8(2):18–24, 1976.

[25] Anthony LaMarca and Richard E. Ladner. The influence of caches on the performance of sorting. In *Proc. of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 370–379. SIAM, 1997.

[26] T. Márquez-Lago and K. Burrage. Binomial tau-leap spatial stochastic simulation algorithm for applications in chemical kinetics. *J. Chem. Phys.*, 127(10):104101, 2007.

[27] A.P. Minton. Excluded volume as a determinant of macromolecular structure and reactivity. *Biopolymers*, 20:2093–2120, 1981.

[28] R. L. Plackett. Karl pearson and the chi-squared test. *International Statistical Review*, 51(1):59–72, 1983.

[29] M. Rathinam, L. R. Petzold, Y. Cao, and D. T. Gillespie. Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. *J. Chem. Phys.*, 119(24):12784–12794, 2003.

[30] V. J. Rodriguez, J. A. Kaandorp, M. Dobrzynski, and J. G. Blom. Spatial stochastic modelling of the phosphoenolpyruvate-dependent phosphotransferase (pts) pathway in escherichia coli. *Bioinf.*, 22(15):1895–1901, August 2006.

[31] Diego Rossinelli, Basil Bayati, and Petros Koumoutsakos. Accelerated stochastic and hybrid methods for spatial simulations of reaction-diffusion systems. *Chem. Phys. Lett.*, 451(1-3):136–140, January 2008.

[32] Werner Sandmann. Streamlined formulation of adaptive explicit-implicit tau-leaping with automatic tau selection. In *Proc. of the Winter Simulation Conference*, pages 1104–1112. IEEE Computer Science, 2009.

[33] Java Scimark. http://math.nist.gov/scimark2/. accessed 10/2010.

[34] T. Tian and K. Burrage. Binomial leap methods for simulating stochastic chemical kinetics. *J. Chem. Phys.*, 121(21):10356–10364, 2004.

[35] D. G. Vlachos. Temporal coarse-graining of microscopic-lattice kinetic monte carlo simulations via tau-leaping. *Phys. Rev. E Stat. Nonlin. Soft. Matter. Phys.*, 78(4):046713+, Oct 2008.

[36] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, October 1999.