EFFICIENT NON-SPATIAL AND SPATIAL SIMULATION OF BIOCHEMICAL REACTION NETWORKS

Dissertation

zur

Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

der Fakultät für Informatik und Elektrotechnik

der Universität Rostock

vorgelegt von

Dipl. Inf. Matthias Jeschke, geb. am 19. Dezember 1980 in Altdöbern

aus Rostock

Rostock, 15. Dezember 2010

urn:nbn:de:gbv:28-diss2010-0195-4

Principal Advisor:	Prof. Dr. rer. nat. habil. Adelinde M. Uhrmacher *University of Rostock, Germany*
External Reviewers:	Prof. Kevin Burrage *Oxford University, United Kingdom*
	PhD Koichi Takahashi *RIKEN Institute Yokohama, Japan*
Date of Defense:	19-OCT-2010

Acknowledgments

Writing this thesis would not have been possible without the help and support of many people. First and foremost I owe my sincerest gratitude to my supervisor, Prof. Adelinde M. Uhrmacher, who has supported me throughout the last three years with her motivation, patience, and knowledge while allowing me the room to work in my own way. With her experience she always did not only see the bigger picture, but was also able to paint it for others, which proved to be invaluable during the work in the research training group.

I am furthermore deeply indebted to Dr. Koichi Takahashi. His invitation to a reasearch visit at his lab gave me the opportunity to broaden my perspective on the research field and also left a lasting impression on the beauties of Japan. I am heartly thankful to Prof. Kevin Burrage, whose experience in the field lead to fruitful discussions about new ideas and concepts eventually realized as part of this work.

In my daily work I have been blessed with a friendly and cheerful group of fellow students, both from the research training and the modeling and simulation group. I would like to thank Roland Ewald for his support: it is not an understatement to say that his work made performing the numerous experiments a lot easier; without it, I would likely still struggle with the other (unofficial) member of the M&S group I wish to thank: *JAMES II*. In fact, I owe my gratitude to all the people who worked on this tool during the last years and I would not have come this far without them.

Finally, I thank my parents and my girlfriend for supporting me throughout my studies in Rostock and for providing me places to recharge my batteries.

This work was financially supported by the interdisciplinary DFG research training group dIEM oSiRiS and a research visit to the RIKEN institute, Yokohama, was funded by the JSPS.

Abstract

Reaction networks describe interactions between (molecular) species and their proper function is crucial for all living beings. Analyzing them *in vitro* is often aggravated by the spatial and temporal scale of the system: the size of species can range from nanometers to micrometers, some networks may fire in only a few nanoseconds while others require days to finish. As an alternative to real-world experiments, the system of interest (here: the reaction networks) can be abstracted to a model, which is then taken as the input for an experiment *in silico* — a simulation.

This dissertation is focused on algorithms for performing non-spatial and spatial stochastic simulations of reaction networks and on the exploration of improved or alternative approaches. A central topic is the evaluation and a subsequent comparison of algorithm performance in terms of execution speed and accuracy. A thorough discussion on the intricacies of a such an evaluation is followed by two large-scale studies which demonstrate the application of the elaborated concepts.

The second part of this work takes up the idea of temporal leap methods, extends it to the spatial realm, and introduces a variant for a parallel execution. Subjected to a theoretical and practical performance analysis, (parallel) spatial τ-leaping shows a strong dependence to both algorithm and model parameters, e.g., the particle distribution.

Finally, the realization of a multi-algorithm simulation is explored by introducing inter-rules that forge a link between the synchronized execution of subsidiary algorithms. A simple notation for the description of discrete-event simulations is extended to allow the representation of moving individuals that can interact with particles in a discretized volume. Despite being qualitative only, several example models and experiments show the feasibility of this approach.

Keywords: stochastic simulation, SSA, spatial simulation, evaluation, τ-leaping, multi-algorithm simulation

Zusammenfassung

Reaktionsnetzwerke beschreiben die Interaktionen zwischen (molekularen) Spezien und dieses oft komplexe Zusammenspiel ist essentiell für das Funktionieren aller Lebewesen. Die Analyse dieser Netzwerke *in vitro* wird häufig erschwert durch die vorhandenen Größenordnungen: die Spezien haben Durchmesser im Nano- bis Mikrometerbereich, einige Netzwerke feuern innerhalb von Nanosekunden, andere können für alle Schritte Tage benötigen. Eine Alternative zur Arbeit mit lebenden Zellen stellt das Erstellen von Modellen und das anschließende Experimentieren *in silico* dar — die Simulation.

Die vorliegende Dissertation beschäftigt sich mit Algorithmen für die nicht-räumliche und räumliche stochastische Simulation von Reaktionsnetzwerken sowie deren Verbesserung bzw. der Entwicklung neuer Ansätze. Ein zentrales Element hierbei ist die Evaluierung und die dadurch geschaffene Möglichkeit des Vergleichs der Performanz von Algorithmen in Hinsicht auf deren Ausführungsgeschwindigkeit und Genauigkeit. Die Arbeit formuliert und diskutiert Konzepte für solche Studien und wendet diese im Rahmen zweier Beispielevaluierungen an.

Im weiteren Verlauf der Dissertation wird die Idee hinter den sogenannten Leap-Methoden aufgegriffen, diese für räumliche Modelle erweitert und eine Variante für eine parallele Ausführung vorgestellt. Eine theoretische und praktische Evaluierung der Methode zeigt eine deutliche Abhängigkeit der Performanz sowohl von Algorithmen- als auch von Modellparametern.

Ein dritter Schwerpunkt ist die Realisierung einer Multi-Algorithmen Simulation durch die Einführung von Inter-Regeln, welche eine Interaktion zwischen verschiedenen Algorithmen erlauben. Basierend auf einer einfachen Notation zur Beschreibung von diskret-ereignisorientierten Simulationen wird ein mögliches Zusammenspiel zwischen Individuen und Populationen innerhalb eines diskretisierten Raumes diskutiert. Die Möglichkeiten, die dieses Konzept bietet, werden durch Beispielmodelle und -experimente demonstriert.

Schlagwörter: stochastische Simulation, SSA, räumliche Simulation, Evaluierung, τ-leaping, Multi-Algorithmen Simulation

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

$\mathbf{x}$	Vector
$\mathbf{X}$	Matrix
S, R	Set
X, Y	Random variable
$X(t), Y(t)$	Stochastic process of a single random variable
$\mathbf{X}(t), \mathbf{Y}(t)$	Stochastic process of a multivariate random variable
$\mathbf{k}_i^N$	A vector with N entries, the i-th being $\mathbf{k} \in \mathbb{R}$ and the rest 0, e.g., $\mathbf{1}_1^3 = (1, 0, 0)$
$\mathbf{k}_{i,j}^{L \times N}$	An $L \times N$ matrix, with entry (i, j) set to $\mathbf{k} \in \mathbb{R}$ and the rest to 0, e.g., $\mathbf{1}_{2,2}^{3\times 3} = \left(\begin{smallmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{smallmatrix}\right)$
$\mathbf{X}_l$	The l-th row vector of the matrix $\mathbf{X}$
$\tau \sim P(X)$	A sample from the distribution $P(X)$

1 Introduction

Simulation is, generally speaking, the process of extracting knowledge from a system by performing experiments on a representation, i.e., a model, of it. Though it would be preferable to experiment with the system itself, this is not always possible; it may, e.g., encompass time scales much larger than the life span of a single human (one could think of erosion processes) or be too dangerous or big to work with — or too small, as it is the case for biochemical reaction networks found in any living organism. A very simple, yet for the moment entirely sufficient, explanation of what is going on in these systems can be given in a few words: molecules, e.g., proteins, move through space and interact with each other, they can *react* into other molecules, which are in turn able to participate in other reactions. This all results in complex networks, evolved over millions of years — and which scientists try eagerly to understand, because this interplay between molecular entities is responsible for the proper *function* of every living systems on this planet. On the other hand, their *dysfunction* is believed to be the cause for many diseases, among them being, e.g., cancer and Alzheimer's. So understanding the dynamics of reaction networks does not only provide insights into how life "works", but also how to preserve it.

The motivation and benefits of investigating this specific type of system are clear — but the actual task turns out to be difficult at least. Not only the scale of the system poses problems, but also the conditions at which it can be observed. In-vitro and in-vivo experiments each have undeniable advantages: the former allows the analysis of how a selection of molecules interact in an isolated and confined environment — but *leaves out* the fact that networks are often interwoven and interactions take place in the presence of other entities, which may have an effect on the dynamics. In-vivo experiments, on the other hand, are very close to the real environment — but often hard to control, because *too much* is going on inside a cell simultaneously. Other factors not to be forgotten are money and time: preparing and cultivating cells can take several weeks, the costs for the entire process easily sum up to several hundred Euros, not including the actual hardware, e.g., microscopes, for which alone hundred

of thousands of Euros could be spent, depending on the field of application.

But there is a third, complementary type of experiments: *in-silico* — which brings the topic back to modeling and simulation (M&S). What the microscopes and Petri dishes are for a biologist are the computers for scientists working in the field of M&S. Just like wet-lab experiments, M&S offers a way to find answers to questions about the system under study. But in contrast to the former, it does so by working with a representation — a model, built with enough known information about the system to find out about yet unknown characteristics of it. However, a model alone is only half the battle; it requires a *simulation algorithm* to execute it. While an algorithm is built to support one specific type of models, it is possible that there exist several algorithms for the same type. To use the analogy of biologists again: depending on the questions they ask, they will use different methods and tools to find an answer, though their system of interest, e.g., a cell, stays the same. Some of the available alternatives are more suited than others, on some occasions it could be even necessary to try a completely different approach — in the end the job needs to be done, the less resources (e.g., time, money) required for this the better.

This actually summarizes the main intention of this dissertation quite well, which follows a path well-known to the scientific field: start by formulating questions and look at the data that is available to you. Then get an overview of the available tools and try to find out which are best suited for a given problem. If the results from this first step hint to possible improvements of certain methods, then evaluate them as well. It may even turn out that taking a step backwards and looking at the task from another perspective could open the gates to new research directions. This path from evaluation over improvement to exploration represents the recurrent theme connecting all the parts in the dissertation and is embedded in the context of biochemical reaction systems. Note that you likely cannot follow this way linearly: improvements and new approaches also require evaluations, which may again point to further optimizations and so on.

To get a feeling of what lies ahead, the central aspects and contributions of the work shall be briefly summarized in the following.

Evaluation When delving into a new research area, it is natural to first have a look at what is already there. A first step is often an intensive review of the literature published on the topic of interest. It helps to get an overview of the tools developed for a wide range of different model incarnations: for example, some may have been specifically

developed for reaction networks having many particles, others are well-suited for dealing with reaction sets including fast and slow reactions. If the models that should be simulated are known, then it might suffice to chose one of the alternatives and perform the experiments. However, a thorough evaluation can be especially important if the complexity of the models is still largely *unknown*: how many reactions and molecules will a model encompass and is it necessary to consider the spatial distribution of the latter? If yes, what should be the desired abstraction, i.e., suffices it to say "there are X particles in this component and Y in the other" or do the individual positions of all entities need to be tracked?

Furthermore, so-called "silver bullets", i.e., methods that supersede any other alternative for arbitrary models, do not exist. An evaluation study helps to identify strengths *and* weaknesses of algorithms; these results could then be used to give recommendations for methods that have proven to be, e.g., very fast or accurate for certain model types.

Chapter 5 is going to discuss the aspects and requirements of an evaluation study for algorithms working with reaction network models. It points to possible pitfalls during the evaluation and the subsequent analysis, argues for the use of artificial benchmark models to cover a wide range of characteristics found in real systems, and exemplarily shows a study defined for the simulation framework *JAMES II*. The application of concepts laid down there will be eventually demonstrated in Chapter 6 that compares the performance of non-spatial and spatial simulation algorithms

Improvement Analyzing the results from an evaluation may provide clues how to enhance a certain algorithm — as it was said, each method has its strengths and weaknesses. In contrast to exploration, which will be discussed below, an improved algorithm keeps the basic "modus operandi" of its ancestor, i.e., the steps performed to produce an output, but optimizes the operations involved in each step. This could be as simple as switching to a more efficient data structure to observe a considerable boost in execution speed or accuracy (Chapter 6 will come back to this). As an example and without going much into detail, each exact stochastic simulation algorithm first finds out the time and type of the reaction that will occur next and then "executes" it, i.e., the involved molecules gets removed and the products added. But the *implementation* of these steps can differ significantly between the numerous variants published over the last decades. Some make use of sorting algorithms to find the imminent event, others store the reactions in an auxiliary queue structure, which ensures that the next

dequeued reaction is the one that will fire next — nevertheless, each variant is not far away from the original.

If more than one computational resource is available, then it could also be beneficial to think about parallelizing an existing algorithm. Instead of having one machine doing all the work, the load is shared among a set of resources; each machine is only responsible for a fraction of the total calculations and therefore should need less time to finish — so the theory goes. As long as there are no dependencies between the calculations, e.g., no machine has to wait for results from another, a parallelization is straightforward; if this condition is not given, then things quickly can become much more intricate.

Chapter 4 presents an algorithm that extends the concepts behind the leap methods, a famous family of algorithms that trade accuracy for execution speed, to simulate reaction networks in a discretized space. A detailed derivation is followed by an analysis that identifies potential strengths and weaknesses of this method and discusses some further improvements, with a parallel implementation being one of those. Chapter 6 puts the algorithm in competition with two established variants and presents the results from a thorough empirical evaluation.

Exploration An improvement takes an existing method and optimizes it towards a more resource-efficient execution, e.g., the new algorithm either runs faster or requires less memory or both. In contrast, an exploration goes beyond this — it takes another perspective on the task, which is in M&S the representation of a system. It does so by modifying the set of assumptions made about the system; some could be added or removed, others maybe relaxed. For example, the aforementioned *leap methods* abandon the idea of executing one reaction at a time in favor of larger time jumps and an estimation of how often each reaction fired within the interval. The new assumption here is that reactions fire at a bulk, each single execution is no longer of interest. It was shown that the run time of a leap algorithm can be much shorter compared to an exact competitor — but at a price of a decreased accuracy, so a trade-off must be made.

Other techniques *combine* different methods in a unique way to compensate their individual weaknesses. Leap methods shine when the population is large, but they may run into problems if there are only a few molecules — so maybe they could be used *together* with exact algorithms for models that include molecule types present in high and low amounts.

Given a collection of molecules and the knowledge how they can interact, it could be explored, e.g., how they can be represented: maybe as individual entities, floating around in space, maybe only as a number counting how many of a given type are present — or maybe both alternatives at the same time. Or what about their position in space? Even in the case that molecules are just numbers, they still can be located in compartments and are able to move between them.

Chapter 7 discusses how to combine algorithms, in particular individual-based and population-based methods that operate on the same volume. An algorithm that implements this type of interplay will then be presented and analyzed in Section 7.7.

Before discussing the main concepts of this dissertation, the next chapter will establish a basic set of definitions, notations and tools used extensively in the further progress.

2 Background

This chapter is going to define the context of the presented work and shall equip the reader with the fundamental terms, definitions, and notations that will be used throughout the remaining sections. By doing so, it also sheds light on the title of the dissertation: it will introduce biochemical reaction networks and provide first ideas and concepts of how those can be modeled and simulated with or without assuming a spatially homogeneous environment.

2.1 Biochemical Reaction Networks

The cell is the building block of all known living beings and the dynamics taking place inside it are essential mechanisms that regulate vital functions of the organism. These dynamics are the result of complex interactions between species inside the cell. For the rest of this dissertation, a species can represent any intracellular entity whose specific nature varies from simple molecules consisting only of a few atoms, like hydrogen or water, to large, complex macromolecules, e.g., the DNA, that can be composed of millions of smaller molecular structures. Some of these species are able to interact with each other under certain conditions. They can participate in *chemical reactions*, i.e., processes which transform a set of input species — called the reactants — into a set of output species — the products. This is a high level abstraction; for the actual reaction process to happen, several conditions must be fulfilled, e.g., the reactants have to be close to each other, maintain the correct orientation, and the potential energy of the involved species must be larger than the activation energy of the reaction. But a single reaction is often only a small piece in a large puzzle. Most cell internal processes are regulated by an ensemble of reactions; only a specific sequence of firings leads to the designated effect. For such a sequence to exist, reactions have to share species, i.e., the products of one reaction are also reactants for another and vice versa.

Example 2.1.1 The canonical Wnt signaling pathway (cf. [Nus05, CCM09]), whose analysis, modeling, and simulation is one of the central subjects of the research training

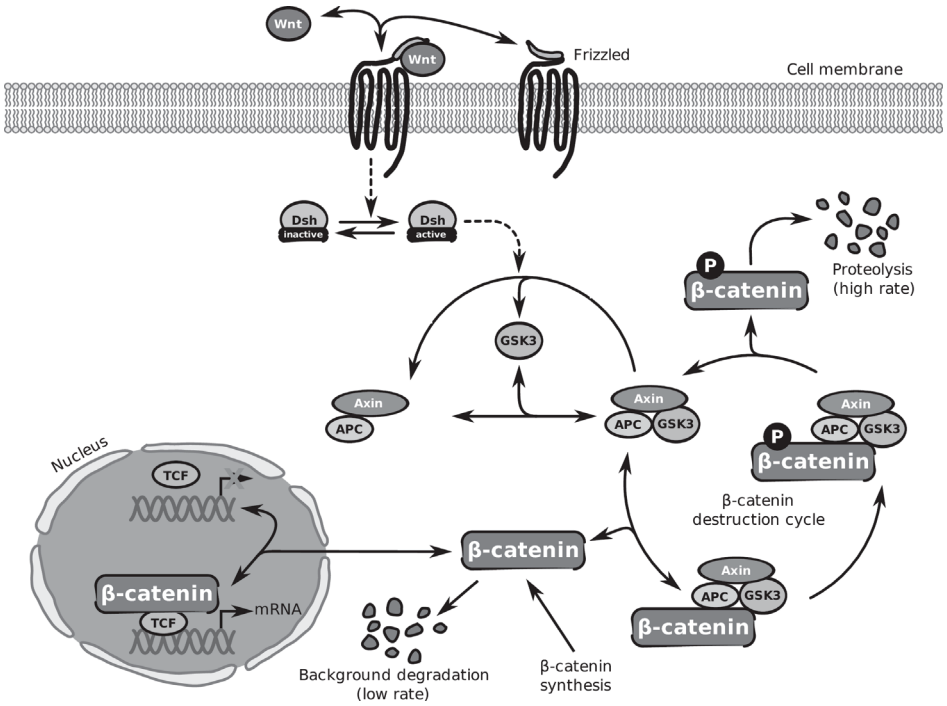


Figure 2.1: The canonical Wnt pathway.

group dIEM oSiRiS[1], is a good example to illustrate the complex interplay between species (see also Figure 2.1). It is focused around the regulation of the β-catenin protein concentration within the cytosol and nucleus. In the absence of a Wnt signal, β-catenin is constantly phosphorylated, i.e., a phosphate molecule is attached to its molecular structure, by the so-called destruction complex, a compound of proteins. This tags the β-catenin protein for ubiquitination, a process that eventually leads to its degradation. But if Wnt3A proteins are present outside the cell, they can bind to a specific receptor complex and the following reaction cascade deactivates the destruction complex, interrupting the process of β-catenin phosphorylation and degradation. As a result, β-catenin first accumulates in the cytosol and, after being shuttled, also in the nucleus where it can bind to the TCF protein and trigger the transcription of its target genes. The effect is twofold: a cell specific response to the Wnt signal, e.g., initiating the differentiation, and a negative feedback on the inducing signal itself. But with a deeper understanding of the pathway internals and participating species, scientists also began to assume that a defective signaling can be the cause for various diseases [MKFK04], including cancer [Pol00, BC00], kidney damage [TTO$^+$03], and schizophrenia [Miy99].

[1]Integrative Development of Modeling and Simulation Methods for Regenerative Systems

For example, a mutation of APC, which is part of the destruction complex, disables the degradation process of β-catenin and leads to a constant transcription of the TCF target genes, which in turn is assumed to promote the development of colorectal cancer [WSV+02].

The last part of Example 2.1.1 gives one reason why effort is spent into analyzing reaction networks: to find the cause and, if possible, a cure for diseases such as cancer or Alzheimer. Research in the *wet-lab*, i.e., the place where cell samples are cultivated and experimented with, is often cumbersome: experiments take a long time, they are usually cost intensive, and in the end only a small fraction of the cell culture might respond to a treatment as required.

Complementary to the work done in wet-labs are experiments performed in a *dry-lab*, i.e., inside a computer. Here the field of modeling and simulation provides powerful tools to support the analysis of reaction networks.

Definition 2.1.1 Analyzing a network of interacting species using *computational modeling and simulation* is the process of capturing the network's essential features and dynamics into a description that can be understood and processed, i.e., *simulated*, by a computer.

This definition will suffice for the moment; a more in-depth discussion on the terms "model" and "simulation" will follow in Section 2.2. In addition to lower costs and usually faster experiment executions, working with artificial representations of reaction networks offers full control over the experiment environment, which is nearly impossible to achieve in wet-labs. Temperatures can be fixed, there is no bias caused by inaccurate instruments or human errors, and the pathway of interest can be analyzed in isolation, without other networks influencing the species population. However, the last argument can also be used against modeling and simulation, as maybe essential or yet unknown interactions are not included in the model and thus results could differ significantly from the wet-lab data. Though this seems to be a vital point, it simply underlines a fundamental restriction of models: they are never complete [Sto01]. This should be kept in mind; modeling and simulation is an *iterative* process where results are verified against real data and models modified or extended if necessary.

Regarding the biological context, the main interest during this dissertation is the interplay between reactions and how they modify the species population over time; the internals of single reaction, e.g., details at the atomic level, will not be considered. Based on that, a first formal description of a biochemical reaction system shall be

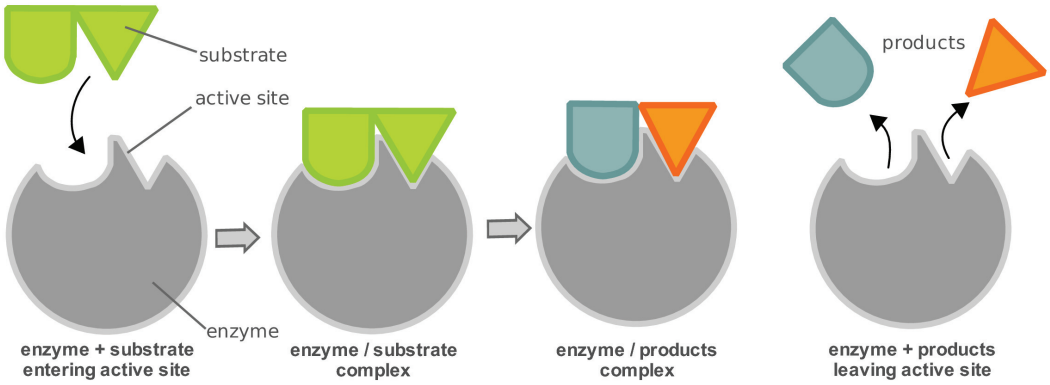


Figure 2.2: The enzyme-substrate catalyzation process. Here the speed for the dissociation of the substrate molecules (S) is increased by the enzyme (E). (Vickers, Tim. *Enzyme / substrate induced fit diagram*. 2006. Wikimedia Commons. Accessed: 11/10/09)

presented next. To make this part easier, a small reaction network — the catalyzation of reactions by enzymes — shall serve as an example throughout the rest of this chapter (Figure 2.2). An enzyme is a special type of protein that is able to catalyze, i.e., increase the rate of, chemical reactions between other molecules, called the substrates. The mechanisms behind the catalyzation effect vary between different enzyme types; one common way is to lower the energy needed for inducing the substrate reaction. The enzyme-substrate reaction starts with the binding of the substrate to the enzyme, forming a complex that can now either release the products or split up again into its constituent parts. In any case the enzyme is again available for the next substrate. Processes of this form have been extensively studied during the last decades using various modeling and simulation techniques, so they are a proper choice for introducing reaction networks.

The set $S = \{S_1, \ldots, S_N\}$ holds unique identifiers for all the species of interest; these can be, e.g., their names or some abbreviations. An index set S^i assigns a unique number $n \in \mathbb{N}_1$ to every species identifier. This allows an enumeration of the species in S using elements from S^i; for the rest of this dissertation the presence of such an auxiliary set is assumed whenever a set is accessed via indices.

Example 2.1.2 The species and index sets for the example network can be defined as $S = \{E, S, ES, P\}$ and $S^i = \{1, 2, 3, 4\}$, with the identifiers E for the enzymes, S substrates, ES for the intermediate enzyme-substrate complex and P for the product. With the help of S^i, the elements of S can be enumerated; for example, it is possible

to write either S_1 or E, both are references to the same species.

The state vector $\mathbf{x} = (x_1, \ldots, x_N)$ is responsible for keeping track of how many particles exist for each species; for now it is assumed that entries of $\mathbf{x}$ are simply integer numbers.

Example 2.1.3 The number of molecules for each of the species participating in the enzyme-substrate network is stored inside the vector $\mathbf{x} = (x_E, x_S, x_{ES}, x_P)$.

Reactions modify the state vector by removing reactant molecules and introducing the products into the system. Similar to the species set, $R = \{R_1, \ldots, R_M\}$ holds all reactions that can occur in the system. A single reaction $R_j \in R$ is essentially a mapping from reactant to product species and is defined as

$$R_j : \sum_{i=1}^{N} v_{ji}^r S_i \to \sum_{i=1}^{N} v_{ji}^p S_i, \tag{2.1}$$

with $S_i \in S$. The stoichiometric coefficients v_{ji}^r and v_{ji}^p give the number of molecules for species S_i that participate as reactants and products; they will be omitted in the further course if they are one. Furthermore, as both sums run over all species indices, a species will not be listed as reactant or product if v_{ji}^r or v_{ji}^p is zero, respectively. When a reaction R_j takes place, v_{ji}^r particles of species S_i are removed from the system while v_{ji}^p are added. The net effect of a reaction execution on the state $\mathbf{x}$ shall be summarized in the *state change vector* $\mathbf{v}_j$, a vector of length N with the i-th entry set to $v_{ji}^p - v_{ji}^r$.

Example 2.1.4 Returning to the enzyme-substrate example, the set R consists of three reactions:

$$\begin{aligned} R_{1,2} &: E + S \rightleftharpoons ES \\ R_3 &: ES \to E + P \end{aligned} \tag{2.2}$$

The double arrows indicate that the complex formation is a reversible reaction; $R_{1,2}$ can also be written as two separate reactions $R_1 : E + S \to ES$ and $R_2 : ES \to E + S$.

The term "reaction network" has been frequently used to describe a set of coupled reactions, but no formal definition was given so far. Having introduced both the species and reaction sets now allows the following graph theoretical interpretation.

Definition 2.1.2 A reaction network is a weighted *bipartite graph* $RN = (S, R, E)$, with the two vertex sets S and R and an edge set $E = S \times R \times \mathbb{N} \cup R \times S \times \mathbb{N}$.

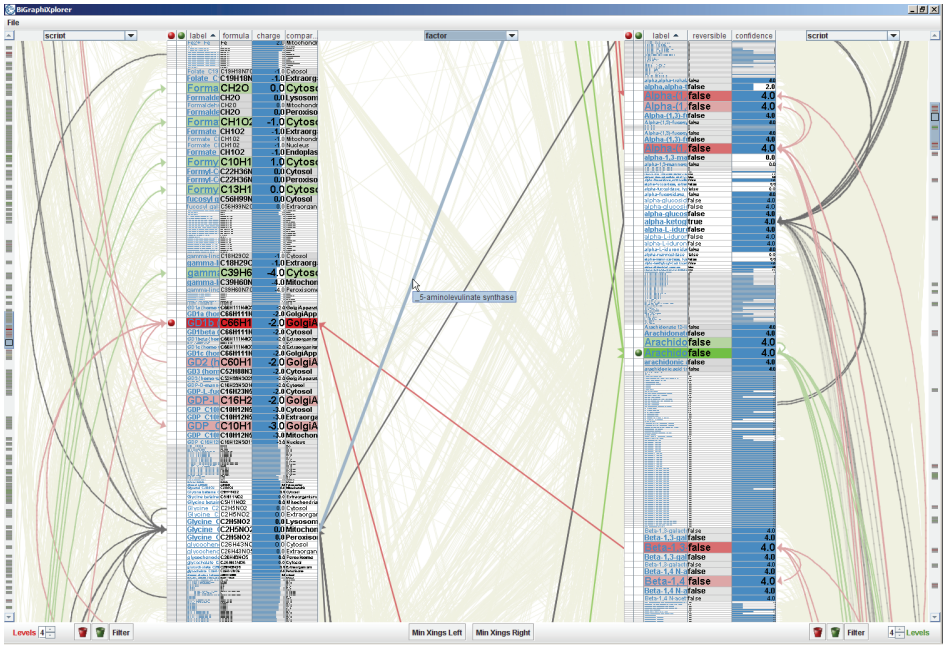


Figure 2.3: Visualization of a reaction network as a bipartite graph using separate columns for species (left) and reaction (right) nodes. A link exists between entries from the left and right column if the species participates either as a reactant (arrow from left to right) or as a product (arrow from right to left) in the reaction. ([SJUS08])

Let $\mathcal{R}_j \subseteq S$ and $\mathcal{P}_j \subseteq S$ represent the set of reactants and products, respectively, for reaction R_j. For $i \in [1, N], j \in [1, M]$, the edge set is defined as follows:

- $(S_i, R_j, v_{ji}^r) \in E$ iff $S_i \in \mathcal{R}_j$, i.e., S_i is a reactant of R_j, and
- $(R_j, S_i, v_{ji}^p) \in E$ iff $S_i \in \mathcal{P}_j$, i.e., S_i is created by R_j.

A bipartite graph is a special type of graph with two distinct vertex sets and a single edge set. Edges can only connect vertices from different sets; so in this case species nodes are always linked to reaction nodes and vice versa. A reaction can be easily identified by first looking for the appropriate node inside R and then following the incoming edges to determine its reactant species and the outgoing edges to find its products. The third entry of an edge tuple represents its weight, which is set to the stoichiometry of the reactant or product, respectively.

Example 2.1.5 Equation 2.3 and Figure 2.4 show the reaction network for the enzyme-

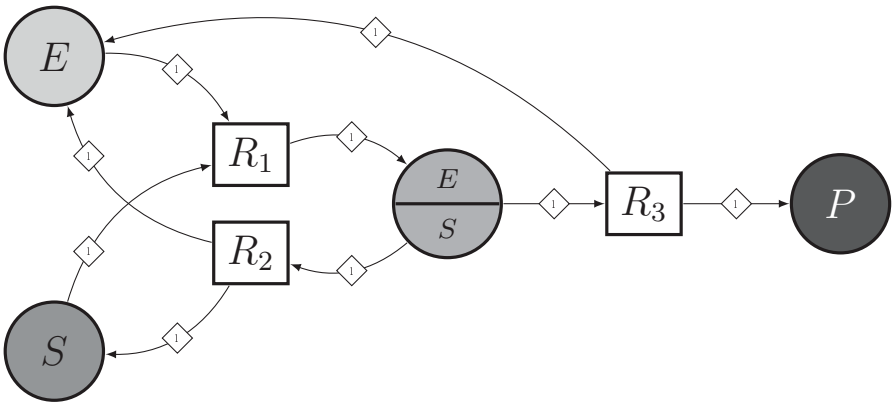


Figure 2.4: The enzyme-substrate reaction network as a bipartite graph. A single edge connects a vertex from the species set with a vertex from the reaction set. The small number associated to each edge denotes the stoichiometry for the species.

substrate catalyzation process.

$$\begin{aligned} RN_{es} =& (S, R, E) \\ S =& \{E, S, ES, P\} \\ R =& \{R_1, R_2, R_3\} \\ E =& \{(E, R_1, 1), (S, R_1, 1), (ES, R_3, 1), (ES, R_2, 1), \\ & (R_1, ES, 1), (R_2, E, 1), (R_2, S, 1), (R_3, E, 1), (R_3, P, 1)\}. \end{aligned} \tag{2.3}$$

Representing a reaction network as a graph is merely one method. Other authors, for instance, separate between atomic species and complexes [FH77]. Referring to Examples 2.1.2 and 2.1.4, in addition to the species set $\{E, S, ES, P\}$ a second set of complexes could be defined as $C = \{E + S, ES, E + P\}$ and reactions would now take place only between complex elements inside C. It should be noted that the original definition of C is slightly more complicated (it is actually a vector space of real valued functions on S, which assign stoichiometric values by multiplying an element of S with a real number and define complexes as the sum of S elements), but the simple interpretation given above suffices in this context.

An alternative to the *rule-based* interpretation of reaction networks, i.e., an explicit description of reaction rules, involves *communications* between individuals. Possible interactions are not written down as additional rules, but species now have the ability to express with whom they are able to communicate. This *individual-based* interpretation became particularly prevalent in the field of modeling languages during the

last decade [JLNU10, CH09], though first publications using it date back to the mid seventies [NVBDG77].

2.2 Preliminary Considerations About Modeling And Simulation

This section shall introduce terms and concepts regarding modeling and simulation (M&S) which are necessary for the understanding of the remaining chapters. A research field so broad and complex makes it difficult to cover it entirely within the narrow scope of a dissertation, so the discussion will be by far not complete and several topics, e.g., model languages, visualization, verification and validation, are left out entirely.

As it is often the case, definitions for ubiquitous terms such as "simulation" can be found in the literature of various research areas, e.g., in system simulation ([McL68, p. 3, p.6], [Gor77, p.5, pp. 17-18], [Riv72, pp. 109-110], [Sha98], [CG91, pp. 6-7]), game theory [WP03, p. 223], or medicine [Jef05]. With this abundance of interpretations it is a good approach to start with rather general definitions and subsequently refine them to fit the context of interest.

Definition 2.2.1 A *simulation* is an experiment performed on a model [KW78].

Definition 2.2.1 is remarkable for its simplicity: simulation is just a task applied on some entity. The task in this case is performing an experiment.

Definition 2.2.2 An *experiment* is the process of extracting data from a system by exerting it through its inputs [CG91].

Again, very simple but concise. According to Definition 2.2.2, performing an experiment is nothing more than presenting a specific input to some system and observing the generated output. It is not necessary to specify what a system is; all that is needed can be found in Definition 2.2.2. The only functions a system has to provide to be used as an experiment subject are taking inputs and generating outputs; nothing more is needed, the internals, i.e., what it does with the input and how it produces the output, are regarded as being hidden in a "black box". There are some other, more verbose definitions (e.g. [Zei84, Ste08, Gai79]), however, this interpretation fits the context of this work very well.

Looking at the previous definitions, the answer to the question about the nature of a model is, despite one key point, nothing of surprise.

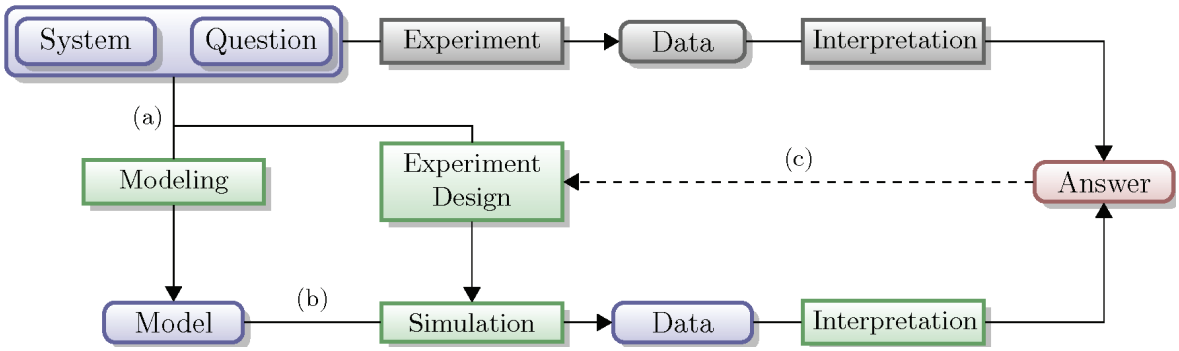


Figure 2.5: Using modeling and simulation as an alternative way to answer questions about a system. Both the system *and* the question to be answered are taken into account during the modeling and experiment design process (a). With the focus kept on the simulation part, it is assumed here that the model used for the simulation has been *verified* and *validated* (b). If the final answer does not provide the required information, a new iteration of experiment design and simulation starts (c). (reproduced and modified with permission from [Ste08])

Definition 2.2.3 A model for a system S and an experiment E is anything to which E can be applied in order to answer questions about S [CG91].

The interesting parts are the beginning and end of Definition 2.2.3. First of all, a model is always linked to a reference system; it is a replacement for the latter, created to provide answers about the substitute. Secondly, questions about a system's behaviour often cannot be answered by performing experiments with the system itself, because it might be too complex, expensive, or small to work with or operate on time scales which are difficult to observe (e.g., the time interval of interest is in the range of nanoseconds or encompasses several centuries). Returning to the initial motivation, the simulation of reaction networks, not all experiments a researcher would like to conduct are possible in praxis; two limiting factors already mentioned above are time and funding, others often depend on the questions asked. With standard wet-lab equipment it is, for example, nearly impossible to continuously trace every protein participating in a signaling pathway. This is where models enter the scientific stage. An important statement shall be repeated at this point: models are never complete, i.e., they are not intended to capture all features of its real counterpart; otherwise they would become the system and their main purpose meaningless. Instead, if a system turns out to be intractable for some experiments, a model can take its place and might provide the required answers. Figure 2.5 summarizes the relationship between simulation, experiment and model. It also shows the usual workflow: from defining the

initial question to modeling and simulation to data interpretation and the answer. As mentioned before, this is an iterative process; if the result is not as expected, then a new cycle starts by, e.g., refining the experiment setup.

Formulating the set of questions as part of the problem statement should be the first and most important step. They hint to system features which are essential and thus need to be represented in some way. On the other hand, properties of lesser importance could be ignored, reducing the model complexity. To sum up, they help to find an appropriate framework for the forthcoming modeling task.

The theory behind modeling is spread over various research fields, from the development of specific languages to express features of the system to model evaluation and visualization. As stated at the beginning of this section, an in-depth introduction to these topics is out of the scope for this dissertation and also not required. Instead, the approach followed here is more focused on the actual simulation than on modeling theoretical details. For simplicity, it is assumed that a system, e.g., a cannon shooting a ball, can be modeled using the following three sets:

- The first one holds constant parameters representing the system's time-invariant features. Examples are the weight and size of the ball or the length of the muzzle.
- A second set, called the *state*, collects all features which can *change over time*. Returning to the cannon ball, its position after leaving the cannon is a constantly updated state variable.
- State variables are modified according to *rules* stored in the third set. Differential equations describing how values change over time are one example and they can also be applied for the cannon system.

A language which can express these sets is referred to as a *modeling language* and an implementation translating the intended semantics into an output shall be called a *simulator*. If, for example, mathematics is regarded as the language and rules are coupled differential equations, a simulator would be any algorithm capable of solving them numerically, e.g., the well-known Euler method. Revisiting Definitions 2.2.1 to 2.2.3 now gives the following basic simulation scheme. During an initialization step the model is *parameterized* with the set of constant parameters and an initial state. While some run condition holds, e.g., a stop time or a final state has not been reached, the simulator cycles through the following steps: take the state variables as input, modify them, and output the new state.

State variables can be either defined on a discrete or continuous domain, depending on the *abstractions* made during modeling. A first simplification was assuming the existence of the three sets introduced above. Deciding for either domain can result in further abstractions; two examples shall illustrate this. Using integers to count objects or events seems natural: there can be only one, two, three etc. products or customer visits, never one and a half. But allowing only discrete state changes could be more difficult to model than switching to the continuous domain. Representing a state vector as a function with a continuous domain, e.g., time, and discrete codomain results in *non-differentiability* at any point the state changes from one value to another. A function is *non-differentiable* if it contains "jumps" and "gaps", i.e., there exist arguments for which the derivative is undefined. But if the discontinuities are treated as continuous transitions (or: abstracted to be continuous), then differential equations can be applied to model how the state changes within an infinitesimal interval; this is commonly done in several fields, e.g., social sciences [GT05], and can be regarded a cornerstone in general system simulation (cf., e.g., [CK06], which provides a good overview of this topic).

A case for abstracting in the other direction, i.e., from a continuous system to discrete model features, is the representation of an objects position, generally a three-dimensional vector with real valued coordinate entries for each dimension. Despite the fact that a digital computer is not able to handle continuous values — though the precision of floating point numbers is sufficient for most applications and can be further increased using arbitrary precision arithmetic (cf. [Knu97]) — modeling with this level of detail comes often at the expense of time and resources needed for the simulation. Taking a chessboard as a simple example, keeping track of a piece's position as real valued position vector would accurately reflect the real world, but is unnecessary; considering the field indices is not only easier to handle in a simulation but also closer to the actual game.

Another general abstraction involves the treatment of *time*. Peoples perception and interpretation of it differs, depending on the person who is asked; the smallest units in everyday life are commonly seconds or minutes, demographers might be more interested in processes that span months or years and biologist in dynamics at both very small (microseconds or nanoseconds) and very large (hours, days, months) scales. The point is: although time is continuous, it is only consciously perceived if linked to some *event* of interest. This is reflected in simulation, with a state change of a system taking the place of the event. Suppose that a system can be at any arbitrary time instant

in any of the state within a set $S = \{S_1, \ldots, S_K\}$. Transitions, $T_1, T_2, \ldots$ change the current state $P \in S$ to a successor $Q \in S$ and take place in intervals denoted by $\Delta t_1, \Delta t_2, \ldots, \Delta t_j \in \mathbb{R}$.

$$\cdots \quad \boxed{P \in S \to Q \in S} \xrightarrow{T_{j-1}\ \Delta t_j} \boxed{P \in S \to Q \in S} \xrightarrow{T_j\ \Delta t_{j+1}} \boxed{P \in S \to Q \in S} \quad \cdots \xrightarrow{T_{j+1}}$$

A simulation is called *continuous* if the number of transitions (events) within a certain time interval is infinite; it is equivalent to say that in a continuous simulation the individual time intervals Δt_j between transitions are infinitesimal small. As it was the case with state variables, a continuous representation of time is not possible on a digital computer (also due to the possibility of infinite state transitions). Instead, the simulation has to be broken down into discrete steps, as it is done, e.g., in numerical integration methods to solve differential equations: an initial step size is chosen and it is either kept constant or adapted for the remaining calculation steps. In continuous simulations the domains for the time, state variables, input, and output are continuous; models are described as sets of (partial) differential equations that represent how a variable changes within an infinitesimal small time increment.

More interesting for this dissertation are simulations where the number of state changes during an arbitrary time interval is finite (there are instances for which an interval could be zero; however, these are special cases and need to be treated with care, otherwise the simulation would come to a halt). Here the $\Delta t_j \in \mathbb{R}$ have either a fixed size — a *discrete-stepwise* simulation — or are calculated dynamically, usually based on the current state of the system, which is referred to as a *discrete-event* simulation where time is skipped forward until something of interest, i.e., an event, happens. It should be added that an explicit notion of time in a discrete-stepwise simulation may in some cases not be necessary: if only the sequence of state changes matters and not when they take place, then it suffices to consider the indices of the transition; time as such is then represented implicitly, however, it is later still possible to assign a time reference, e.g., the transition from one step to the next could represent a time span of a month or year. In contrast to continuous simulations, the domains of the state variables (and the respective inputs and outputs) can be either continuous or discrete.

Hybrid simulations are also possible by coupling the update schemes. Suppose a continuously simulated variable; if at some point in time its value exceeds a threshold, then an event is triggered and an output generated. For an outside observer it would look like a regular discrete-event simulation, while it is actually a hybrid, with state

variables updated continuously inside.

2.3 Simulation of Reaction Networks

Looking back at Definition 2.1.2, which introduced a first formal notion of the term "reaction network", it appears that it already comprises major elements of what has been termed a "model" in the previous section. The state is given by the vector $\mathbf{x}$ and the rules modifying it are stored as reactions inside R; additionally, the *state space* $\mathcal{S}$ of a model shall be defined as the set of all possible states the model can be in. The only part still missing before running a simulation is the actual simulator, which should decide what rules can be applied next and update the state accordingly. It is the part that encodes the semantics of the execution — which in turn are closely coupled to the *assumptions* made about how the original system should be represented. Some of those assumptions have been established right at the beginning: particles are abstracted to numbers that denote how many entities of a specific species are present; reactions are defined over species, they take a number of particles as inputs and transform them into products — a simple interpretation, leaving out the complex details of such a process, e.g., what happens at the level of atoms, because they do not provide additional information regarding the main question of interest: how does the number of particles change over time. In the following, several additional assumptions and their implications on the semantics shall be discussed.

For the most basic *non-spatial* simulation of a reaction network defined hitherto it is commonly assumed that the system is *well-stirred* and in *thermal equilibrium*. *Well-stirred* (or well-mixed) means that the particles inside the volume are distributed homogeneously, i.e., uniformly, on the time scale of the reactions. Though the firing of reactions lead to local and temporal inhomogeneities in the real system, those are removed by presuming that non-reactive collisions between the molecules drive the system back into a well-stirred state in a time much smaller than the typical interval between two reactions. *Thermal equilibrium* is reached when a change in the temperature can no longer be observed; as a consequence thereof, the velocities of all molecules are distributed according to the Maxwell-Boltzmann distribution.

Now assumptions and derived semantics can be seen as being connected via an "if-then" relationship. So why a well-stirred system with a constant temperature? **If** a system does have these characteristics, **then** its dynamics, i.e., the evolution of the state over time, can be well approximated by letting reactions fire at a certain

rate. This means that all individual characteristics of particles, e.g., their energy, shape, size, exact position, etc. do not have to be considered to represent a chemically reactive system as long as *reaction rates* can be calculated. The theory behind this is governed by the field of reaction kinetics. Each reaction is assumed to take place with an intrinsic speed, quantified by the *reaction rate constant*, which depends on the temperature, physical properties of the molecules, the viscosity of the solution, and so on; it is provided as part of the reaction equation:

$$R_j : \sum_{i=1}^{N} v_{ji}^{r} S_i \xrightarrow{k_j} \sum_{i=1}^{N} v_{ji}^{p} S_i, \tag{2.4}$$

A physically-grounded and very comprehensive derivation that lead to the existence of such constants based on the above assumptions can be found in [Gil77]. There are several factors influencing the speed at which a reaction occurs, one of which being the concentration of the particles: the more molecules are present, the higher is the rate. This relation finds its expression in the *law of mass action*, the most basic reaction kinetic.

That said, it is time to go back to the initial question: how to simulate the evolution of a systems state over time. A system changes its state whenever a reaction fires, which, taken the statements from the last paragraph, occurs at a certain rate. The relation between the change of a species state and the rates at which reactions fire can be expressed in form of an *ordinary differential equation* (ODE), the "classic" interpretation of reaction dynamics.

Example 2.3.1 Referring to the enzyme-substrate network, kinetic reaction constants are assigned to the reactions from Equation 2.2:

$$\begin{aligned} R_{1,2} &: E + S \underset{k_2}{\overset{k_1}{\rightleftharpoons}} ES \\ R_3 &: ES \xrightarrow{k_3} E + P \end{aligned} \tag{2.5}$$

The concentration of the enzyme (E) changes when either reaction R_1, R_2 or R_3 fires; R_1 decreases the concentration, so it negatively contributes to the overall rate and the reaction rate therefore has a negative sign, and R_2 and R_3 both increase the concentration and the corresponding rates have a positive sign. The rate for reaction R_1 at a time instant t is, according to the law of mass action, the product of the reactants concentrations, $x_E(t)x_S(t)$, times the reaction constant. The following equations are

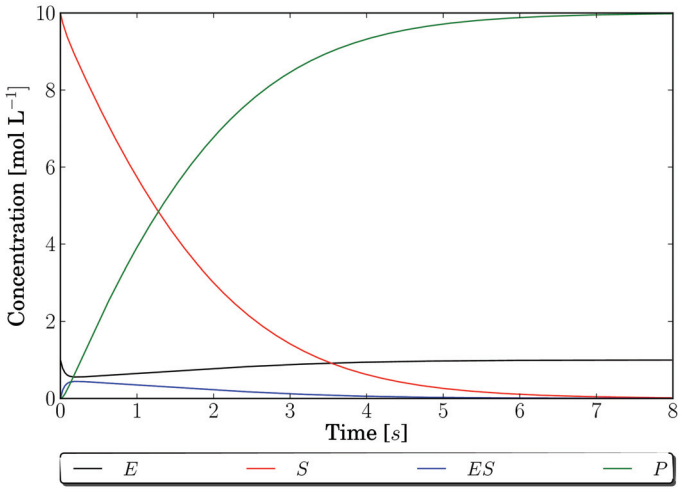


Figure 2.6: Plot of the solution for the enzyme-substrate ODE system. The reaction constants are $k_1 = 1\,\mathrm{L\,mol^{-1}\,s^{-1}}$, $k_2 = 1\,\mathrm{s^{-1}}$ and $k_3 = 10\,\mathrm{s^{-1}}$ and the system is initially in a state with the concentrations for the species set to $x_E(0) = 1\,\mathrm{mol\,L^{-1}}$, $x_S(0) = 10\,\mathrm{mol\,L^{-1}}$ and $x_{ES}(0) = x_P(0) = 0$.

the ODEs representing the dynamics of the network:

$$\begin{aligned}
\frac{dx_E(t)}{dt} &= -k_1 x_E(t) x_S(t) + (k_2 + k_3) x_{ES}(t) \\
\frac{dx_S(t)}{dt} &= -k_1 x_E(t) x_S(t) + k_2 x_{ES}(t) \\
\frac{dx_{ES}(t)}{dt} &= k_1 x_E(t) x_S(t) - (k_2 + k_3) x_{ES}(t) \\
\frac{dx_P(t)}{dt} &= k_3 x_{ES}(t)
\end{aligned} \tag{2.6}$$

Figure 2.6 shows how the state of the system changes for some example rate constants.

2.4 Stochastic Simulation

The solution to a system of ordinary differential equations represents a good approximation of the system dynamics when the species are present in high concentrations and a single reaction firing can be neglected. But when the concentration is very low, this firing could indeed make a difference. To illustrate this, a model for viral kinetics shall be taken as an example: a single viral template is introduced to some host

[SYSY02]. Such a template can either infect a cell — which then produces further template material, effectively spreading the infection — or the immune system of the host detects and removes it. Simulating this scenario with a set of ODEs showed that the infection always spreads throughout the host. In contrast, considering the race between the infection of the first cell and the detection of the template, a *stochastic* simulation revealed that the infection can be stopped before affecting further cells. So what is special about this type of simulation?

A simulation using ODEs is always *deterministic*: given an initial state, the evolution (or trajectory) is always the same, no matter how often the simulator (here: the numerical integration algorithm) is run. But reaction firings are not deterministic. Just because reactant particles are present does not mean that they actually react. For that to happen, they first have to move into close proximity and then collide with the correct orientation towards each other and enough energy to overcome the activation barrier — if any of those prerequisites is not given, then they simply bump off and drift away. Even if all factors influencing a reaction are known and considered, quantum-mechanical effects still make it impossible to predict whether and when a reaction will take place. In conclusion, reaction dynamics are subject to what is commonly referred to as *intrinsic* and *extrinsic noise* (e.g., cf. [TO01, SES02, ELSS02]: the former represents the uncertainty in the outcome of an encounter between reactant particles while the latter takes flucuations outside the network into account (e.g., the variation of a protein's concentration in a neighbor compartment, which influences how fast it is transported into into the observed volume). Intrinsic noise is further enhanced by a low copy number of reactive particles. Ordinary differential equations inherently assume that the state variables have a *continuous domain*. While for large populations, which are usually given as concentrations in mole per volume unit, with one mole having $\approx 6.022 \times 10^{23}$ particles of the measured species, the continuity assumption does not pose any problems, the case is different if only a few molecules are present, e.g., only some hundreds or thousands. Now the result of a numerical integration could be that for a time t there are 100.23 particles of some species inside the volume — although one would expect a discrete number.

As an example for all the points just mentioned, Figure 2.7 illustrates the difference between a continuous deterministic and a discrete stochastic simulation (details about *how* to perform a stochastic simulation will be given shortly). Note how the stochastic simulation already stops after about 3 s; all substrates have been converted into products.

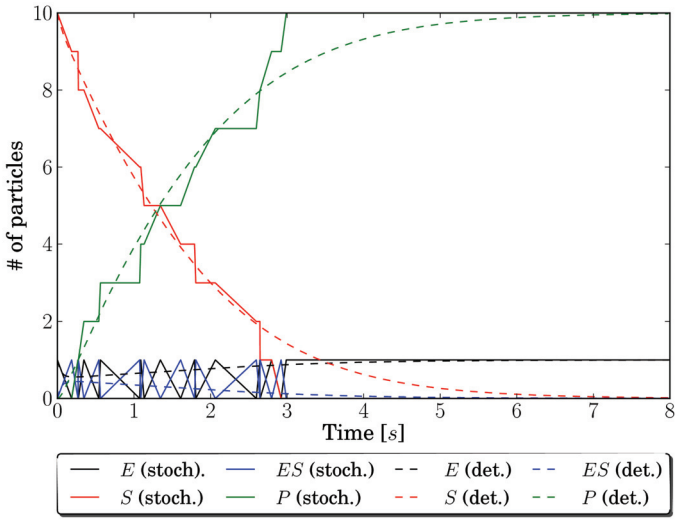


Figure 2.7: A single trajectory from the stochastic simulation of the enzyme-substrate reaction. The reaction constants and the initial state are adapted from the deterministic example with $V = 1\,\mathrm{L}$, so $c_1 = 1\,\mathrm{s}^{-1}$, $c_2 = 1\,\mathrm{s}^{-1}$, $c_3 = 10\,\mathrm{s}^{-1}$ and the initial state is set to $x_E = 1$, $x_S = 10$, $x_{ES} = x_P = 0$. Note that the unit of concentration has changed to number of particles — there are now 10 molecules of E instead of $10\,\mathrm{mol\,L}^{-1}$. The discrete-stochastic simulation makes jumps between the states while the ODE solutions are continuous. After about three seconds the last substrate has been converted into a product in the stochastic simulation.

To account for both stochasticity and discreteness, the evolution the state $\mathbf{x}$ over time, previously traced by a set of functions $x_i(t), i \in [1, N]$, shall now be represented by a *stochastic process* $\mathbf{X}(t)$. Stochastic processes are sequences of random variables (RV) whose order is defined by a (discrete or continuous) time index; as it is very likely to have more than one species in a model, the general case of multivariate RV shall be used for this introduction. $\mathbf{X}(t)$ is therefore defined by a sequence $(\mathbf{X}_t : t \in \mathbb{R})$ of *vectors* containing discrete random variables that represent the state of each species at some time t:

$$\mathbf{X}_t = (X_{1,t}, X_{2,t}, \ldots, X_{N,t}) \tag{2.7}$$

The distribution of $\mathbf{X}_t$ depends on what has been observed, i.e., on the history of *realizations* $(\mathbf{X}_{\hat{t}} = \mathbf{x}_{\hat{t}} : \hat{t} \in \mathbb{R}, 0 \leq \hat{t} < t, \mathbf{x}_{\hat{t}} \in \mathcal{S})$[2]:

$$p_{\mathbf{X}_t}(\mathbf{x}) = \mathrm{P}(\mathbf{X}_t = \mathbf{x} \,|\, (\mathbf{X}_{\hat{t}} = \mathbf{x}_{\hat{t}} : 0 \leq \hat{t} < t, \mathbf{x}_{\hat{t}} \in \mathcal{S})), \tag{2.8}$$

with

$$\mathrm{P}(\mathbf{X}_0 = \mathbf{x}) = \delta_{\mathbf{X}_0, \mathbf{x}_0}, \tag{2.9}$$

i.e., the probability mass at time $t = t_0$ is located at the initial values of the species $S_i, i \in [1, N]$. Note that for a single $X_{i,t}$, the *entire* previous state vectors are considered, not just the history of S_i; as a result, the random variables may be *correlated*, an aspect which will be relevant in Chapter 5.

If the history is restricted to only include the last state, i.e., if:

$$\mathrm{P}(\mathbf{X}_t = \mathbf{x} \,|\, (\mathbf{X}_{\hat{t}} = \mathbf{x}_{\hat{t}} : \hat{t} < t, \mathbf{x}_{\hat{t}} \in \mathcal{S})) = \mathrm{P}(\mathbf{X}_t = \mathbf{x} \,|\, \mathbf{X}_{pred(t)} = \mathbf{x}_{pred(t)}), \mathbf{x}_{pred(t)} \in \mathcal{S} \tag{2.10}$$

and $pred(t)$ is defined such that $\neg\exists \hat{t} : 0 \leq \hat{t} < t \wedge pred(t) < \hat{t}$, then the stochastic process has the *Markov property*. Applying a marginalization to the right hand side of Equation 2.10 gives the unconditional probability to be in state $\mathbf{x}$ at time t:

$$\mathrm{P}(\mathbf{X}_t = \mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{S}} \mathrm{P}(\mathbf{X}_t = \mathbf{x} \,|\, \mathbf{X}_{pred(t)} = \mathbf{y}), \tag{2.11}$$

Another special case are *time-homogeneous* processes; here the probability of a transi-

[2] $\mathbf{x}_{\hat{t}}$ is the configuration of the state variables as it was observed at $\hat{t}$, i.e., a realization of $\mathbf{X}_{\hat{t}}$.

tion from one state to a successor state is independent of the time, i.e.,:

$$\mathrm{P}(\mathbf{X}_t = x \,|\, \mathbf{X}_{pred(t)} = \mathbf{y}) = \mathrm{P}(\mathbf{X}_s = \mathbf{x} \,|\, \mathbf{X}_{pred(s)} = \mathbf{y}), t \neq s, \mathbf{y} \in \mathcal{S}. \tag{2.12}$$

In the following, most stochastic processes are assumed to be both Markovian and time-homogeneous, so the time reference is left out in some equations.

A *stochastic constant* c_j is assigned to each reaction R_j and replaces the kinetic constant k_j introduced above. This also brings along a shift from reaction rates to reaction *probabilities*: c_j also depends solely on physical properties of the reactant species, the solvent, the temperature, and other conditions, just like k_j, but $c_j\,dt$ gives now the probability that a *particular set* of reactant particles undergo reaction R_j during the infinitesimal time interval dt inside the system. Let $H_j(\mathbf{x})$ be the number of distinct reactant combinations of reaction R_j when the system is in state $\mathbf{x}$; $H_j(\mathbf{x})$ can be expressed as a product of binomial coefficients:

$$H_j(\mathbf{x}) = \prod_{i=1}^{N} \binom{x_i}{v_{ji}}, \tag{2.13}$$

with v_{ji} as the change in the particle number of species S_i caused by a firing of R_j. Applying the law of mass action leads to the probability that reaction R_j will take place *somewhere* inside the volume during dt:

$$\mathrm{P}(j; \mathbf{x}, dt) = H_j(\mathbf{x}) c_j dt. \tag{2.14}$$

Note that Equation 2.14 does *not* imply that each combination of reactant molecules has the same probability of participating in a reactive collision. This is clearly not possible as the molecules may be located far away from each other. In fact, $c_j\,dt$ represents the *average* probability of a reactive collision over all possible relative positions of a reactant set [GLP07]. Furthermore, Equation 2.14 is only valid if the size of the particles is negligible compared to the size of the volume — they are essentially points. In a recent publication Gillespie studied the question how to modify this equation to account for different reactant sizes [GLP07]. The authors restrict their calculations to the one-dimensional case because it is difficult to find a model that distributes finite-sized particles in a uniform and non-overlapping way for higher dimensions. However, they show that only minor modifications of Equation 2.14 are necessary to calculate the desired probability.

Zeroth-order ($O_j = 0$):	$R_j : \emptyset \xrightarrow{c} S$
First-order ($O_j = 1$):	$R_j : S_1 \xrightarrow{c} S_2$
Second-order ($O_j = 2$):	$R_j : S_1 + S_2 \xrightarrow{c} S_3$
	$R_j : 2S_1 \xrightarrow{c} S_2$
Third-order ($O_j = 3$):	$R_j : S_1 + S_2 + S_3 \xrightarrow{c} S_4$
	$R_j : 2S_1 + S_2 \xrightarrow{c} S_3$
	$R_j : 3S_1 \xrightarrow{c} S_2$

Table 2.1: The orders for the most basic reaction types.

As it may be guessed, both the stochastic and kinetic constants are closely linked to each other. To be exact, they are related via the volume V. Kinetic constants are relative to the unit volume because in the continuous deterministic regime the elements of the state vector are usually given in units of concentration, e.g., molarity, which denotes the number of molecules per liter of solution. In contrast, the state vector for a stochastic model contains discrete numbers of molecules for each species. So given a reaction R_j of order $O_j > 0$ (see Table 2.1), the stochastic constant c_j can be obtained from a given kinetic constant k_j by multiplying the latter with $1/V^{O_j-1}$.

Based on Equation 2.14, a *propensity* $a_j(\mathbf{x})$ for reaction R_j is defined as $a_j(\mathbf{x}) = H_j(\mathbf{x})c_j$.

2.4.1 The Chemical Master Equation

The dynamics of a system interpreted as a stochastic process having the Markov property can be expressed in form of the *chemical master equation* (CME), which is, in some sense, the stochastic equivalent to the set of ODEs used above. Let $\mathrm{P}(\mathbf{x}; t) \equiv \mathrm{P}(\mathbf{X}_t = \mathbf{x})$ be the function giving the probability that the system is in state $\mathbf{x}$ at time t (see last section); knowing it would completely characterize the state of the system for any point in time. An expression for this function can be derived by first looking at how this probability changes within a small time increment Δt.

$$\mathrm{P}(\mathbf{x}; t + \Delta t) \approx \mathrm{P}(\mathbf{x}; t)[1 - \sum_{j=1}^{M} a_j(\mathbf{x})\Delta t] + \sum_{j=1}^{M} \mathrm{P}(\mathbf{x} - \mathbf{v}_j; t) a_j(\mathbf{x} - \mathbf{v}_j)\Delta t$$

The first term on the right hand side gives the probability that the system is in state $\mathbf{x}$ at time t and no reaction will fire during Δt; the second term sums up for each reaction R_j the probability that the system is one firing of R_j away from reaching state $\mathbf{x}$ (i.e.,

the probability of being in a predecessor state $\mathbf{x} - \mathbf{v}_j$ at time t) and that this reaction will be executed during Δt. Rearranging the terms and taking the limit $\Delta t \to 0$ finally results in the CME, a set of coupled differential equations describing the evolution of the state probability density over time:

$$\begin{aligned}\frac{\partial \mathrm{P}(\mathbf{x};t)}{\partial t} &= \lim_{\Delta t \to 0} \frac{\mathrm{P}(\mathbf{x};t+\Delta t) - \mathrm{P}(\mathbf{x};t)}{\Delta t} \\ &= \sum_{j=1}^{M} \mathrm{P}(\mathbf{x}-\mathbf{v}_j;t) a_j(\mathbf{x}-\mathbf{v}_j) - \mathrm{P}(\mathbf{x};t) \sum_{j=1}^{M} a_j(\mathbf{x})\end{aligned}$$

It is fairly easy to write down the equation itself, but solving it can be difficult if not impossible — an equation is necessary for each of the (possibly infinite) states the system can be in. Despite this, it could be the case that not all states are actually reachable within the time interval $(t_0, t_f]$ of interest and, as has been shown by Macnamara et al. [MBS08], this can be exploited to find an approximate solution to $\mathrm{P}(\mathbf{x};t)$ at different time points $t \in (t_0, t_f]$ without the need for many SSA realizations.

But solving the CME, which is often intractable, is not the only way how to get the desired information from a stochastic processes.

2.4.2 The Stochastic Simulation Algorithm

First works in the field of stochastic simulation date back to the publications of Kurtz [Kur72], Bortz, Kalos and Lebowitz [BKL75] and Gillespie [Gil76, Gil77] from the early to mid-seventies. While Kurtz focuses on relating the stochastic interpretation of reaction dynamics to the deterministic one, Bortz et al. and Gillespie both propose a method to generate *trajectories* of a continuous-time Markov chain (CTMC) that can be described by a CME introduced in the previous section; the only significant difference between both approaches lies in the fields they were introduced to. While the former has been presented as a novel technique to study Ising spin systems, i.e., models for ferromagnetism, the latter was explicitly used to perform simulations of reaction networks. The underlying principles are basically the same and today the method is commonly referred to as *stochastic simulation algorithm (SSA)* (also *Gillespie's algorithm* in some publications).

Solving the CME gives the probability distribution for the state vector at an arbitrary point in time; this process is basically the stochastic equivalent to the analytical treatment of an ODE system. However, if either is not analytically tractable, then

Algorithm 2.1: Pseudo-code description of a template for discrete-event stochastic simulation algorithms.

Parameters:
t (global time)
Input:
$RN = (S, R, E)$ (reaction network)
$\mathbf{x} = (x_1, \ldots, x_N)$ (initial state)
$[t_{start}, t_{end}]$ (simulation interval)

1 $t \leftarrow t_{start}$;
2 **while** $t < t_{end}$ **do**
3 $\quad \tau, \mathbf{k} \leftarrow SelectNextReactions(\mathbf{x}), \tau \in \mathbb{R}, \mathbf{k} = (k_1, \ldots, k_M)$;
4 $\quad t \leftarrow t + \tau$;
5 $\quad \mathbf{x} \leftarrow \mathbf{x} + \sum_{j=1}^{M} k_j \mathbf{v}_j$;

numerical methods are required to find a solution. Instead of dealing with the CME, stochastic simulation algorithms can generate single trajectories through the state space that represent *realizations* of the underlying Markov process. Reactions between the species are seen as events that modify the state vector at discrete time points. If there is a way to determine the type of the next event, i.e., the reaction that is going to fire next, and the time when this will occur, then a basic simulation scheme could look like the one shown in Algorithm 2.1. All that is needed here is an algorithm for $SelectNextReactions(\mathbf{x})$. Two equivalent variants, called the *Direct Method* (DM) and the *First Reaction Method* (FRM), have been presented by Gillespie in [Gil77]. To derive the second one (see Algorithm 2.2), it shall be remembered that the underlying stochastic process is a continuous-time Markov chain. As such, being in some state $\mathbf{x}$ at time t, the next possible states are the ones that can be reached by a single firing of any reaction in R. The time it takes until R_j will fire next is an exponentially distributed random number with rate parameter λ set to $a_j(\mathbf{x})$ — a direct consequence of the memoryless property of the process. All the FRM variant needs to do is to calculates for each R_j a sample τ_j from $\mathrm{Exp}(a_j(\mathbf{x}))$ (e.g. via inverse transform sampling, so $\tau_j = -\ln U(0,1)/a_j(\mathbf{x})$) and execute the reaction with the minimum next event time. But this requires in any case the generation of M samples — which is why the FRM is generally considered as being inferior to the other alternative, the DM.

This method first calculates the time when *something* will happen and afterwards

Algorithm 2.2: Pseudo-code description for the $SelectNextReactions(\mathbf{x})$ variant of the FRM. The return value $\mathbf{k} = \mathbf{1}_{\mu}^{M}$ is an M-dimensional vector with $k_{\mu} = 1$ and $k_j = 0, j \neq \mu$.

Parameters: None
Input: $\mathbf{x}$ (current state)

1 **for** $j \in [1, M]$ **do**
2 update propensity $a_j(\mathbf{x})$;
3 **for** $j \in [1, M]$ **do**
4 $\tau_j \leftarrow \dfrac{-ln(U(0,1))}{a_j(\mathbf{x})}$;
5 $(\tau, \mu) \leftarrow (\min_{j\in[1,M]}\{\tau_j\}, \operatorname{argmin}_{j\in[1,M]}\{\tau_j\})$;
6 **return** $\tau, \mathbf{1}_{\mu}^{M}$

Algorithm 2.3: Pseudo-code description for the DM's $SelectNextReactions(\mathbf{x})$ method.

Parameters:
$a_0(\mathbf{x}) = 0$ (sum of propensities)
Input: $\mathbf{x}$ (current state)

1 **for** $j \in [1, M]$ **do**
2 update propensity $a_j(\mathbf{x})$;
3 $a_0(\mathbf{x}) \leftarrow a_0(\mathbf{x}) + a_j(\mathbf{x})$;
4 $\tau \leftarrow \dfrac{-ln(U(0,1))}{a_0(\mathbf{x})}$;
5 determine μ as smallest integer satisfying
6 $\displaystyle\sum_{j=1}^{\mu} \frac{a_j(\mathbf{x})}{a_0(\mathbf{x})} > U(0,1), \mu \in [1, M]$;
7 **return** $\tau, \mathbf{1}_{\mu}^{M}$

what event actually will take place. It utilizes the fact that the sojourn time in a state $\mathbf{x}$ is exponentially distributed with rate parameter $a_0(\mathbf{x}) = \sum_{j=1}^{M} a_j(\mathbf{x})$, so it takes a time $\tau \sim \text{Exp}(a_0(\mathbf{x}))$ until any of the reactions from R will fire. After this time has expired the system changes from state $\mathbf{x}$ to $\mathbf{x} + \mathbf{v}_j$ with probability $a_j(\mathbf{x})/a_0(\mathbf{x})$. Algorithm 2.3 summarizes the main steps performed in the DM. Note that instead of M samples from the uniform distribution $U(0, 1)$, the DM only requires two: the first one to get the interval τ and the second for finding the reaction that is about to fire.

Compared to a toolbox, the DM and FRM are the hammer and screwdriver: the basic instruments that can be found in most of the simulation software released over the last years. They are simple to implement and do their job, one reaction after another; the latter is actually the reason why they are also called *exact* algorithms, each generated trajectory is a sample from the underlying stochastic process.

2.5 Spatial Stochastic Simulation

With the advent of high-resolution microscopy, the demand for stochastic algorithms that can also handle inhomogeneous particle distributions grew — simulating the evolution of a system not only over time, but also in space was considered as the next step to deepen the understanding of reaction networks.

For the systems considered so far, space has been treated only implicitly, i.e., the molecules of all species are distributed homogeneously inside the given volume, without considering individual positions. But for some biological systems this approximation does not reflect the inherent dynamics properly. One example is signaling pathways.

In the Wnt pathway introduced in Section 2.1 the signal from outside the cell causes a reaction cascade across the cell membrane, the cytosol and the nucleus, which eventually triggers the transcription process. But before any reaction can take place its reactants have to "find" each other, i.e., they have to move into close proximity. As a consequence, the signal is not only transported over time but also through space. To capture this, models for *reaction-diffusion systems* include an explicit treatment of space. The most accurate representation would consider each molecule as an individual that moves continuously through the volume. But this accuracy often goes along with high computational requirements which limit the number of molecules that can be present within the volume and also the simulation time interval. And it could be asked: is this detail even necessary to answer the questions about the system?

One way to reduce the computational effort is to discretize space into smaller sub-

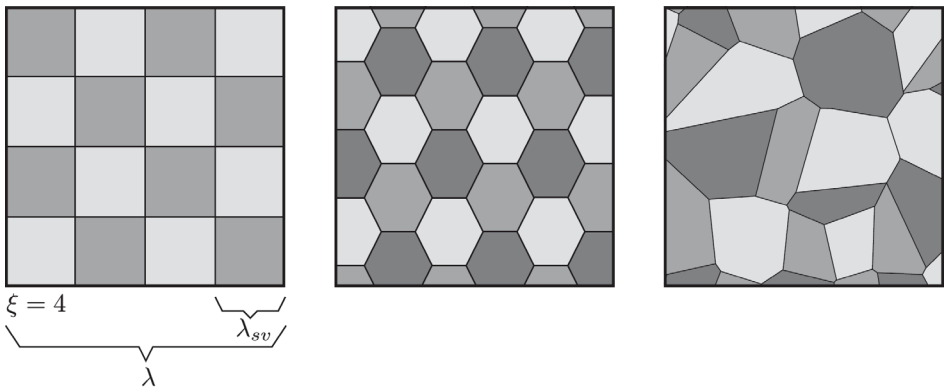


Figure 2.8: Some examples for a 2D spatial discretization. From left to right: Sub-volumes as squares, regular hexagons and Voronoi regions.

domains, commonly termed *sub-volumes* in the literature (cf. [Gil76]). Objects inside these regions can still be either individuals or anonymous parts of a population, but their movement is restricted to jumps from one sub-volume to a neighbor. The sub-volumes can have an arbitrary shape as long as their union covers the entire volume, though it is often convenient to use a square (2D) or cubic (3D) representation (see Figure 2.8). Their size depends on the desired spatial resolution: a very fine-grained discretization, i.e., the area (or volume in higher dimensions) of a single region approaches zero while their number increases, permits an almost continuous motion of the molecules. Large sub-volumes, on the other hand, can be used to model entire compartments, e.g., the exterior of a cell, the cytosol and the nucleus.

In the course of this dissertation, only very simple reaction-diffusion systems shall be studied where the species are contained inside an n-dimensional hypercube with side length λ and a volume $V = \lambda^n$; in the planar case ($n = 2$) the container is therefore a square and in the three-dimensional space a cube. Let ξ denote the discretization parameter; the original cube-shaped volume is then sub-divided into $L = \xi^n$ sub-volumes, each having a side length of $\lambda_{sv} = \lambda/\xi$ and a volume of $V_{sv} = \lambda_{sv}^n$ (cf. Figure 2.8). Each of those smaller volumes is a well-stirred environment of its own: they contain a number of particles which can react just like it was introduced previously. The only addition is that particles are now allowed to leave a site and *diffuse* into a neighbor; from the perspective of a single sub-volume, some molecules suddenly "disappear" while others "show up".

It has been shown [Gil76] quite early that the task of simulating such a spatial model can be reformulated to make use of an existing non-spatial simulation algorithm, e.g.,

the DM or FRM from the last section. Given a reaction network RN, the fist step is to create copies of the species and reactions such that each sub-volume basically has its own sets. With N species and L sub-volumes, a new set $\hat{S}$ is defined on the basis of the existing species:

$$\hat{S} = \{S_1, S_2, S_3, \ldots, S_{N(l-1)+i}, \ldots, S_{NL}\}, \tag{2.15}$$

with species $S_{N(l-1)+1}$ to $S_{N(l-1)+N} = S_{Nl}$ "belonging" to sub-volume l. Similarly, the state vector $\mathbf{x}$ is expanded as well:

$$\hat{\mathbf{x}} = \{x_1, x_2, x_3, \ldots, x_{N(l-1)+i}, \ldots, x_{NL}\}. \tag{2.16}$$

After defining the state of the model, the next step is to look at the rules that can modify it. It is not only important to specify what species variables are changed, but also *where*. A reaction previously defined for only one volume can now fire in any of the L sub-volumes, which makes it necessary copy a rule L times, but with each copy taking care of the state changes in only one sub-volume. Reactions in the general form:

$$R_j : \sum_{i=1}^{N} v^r_{j,i} S_i \xrightarrow{c_j} \sum_{i=1}^{N} v^p_{j,i} S_i, \tag{2.17}$$

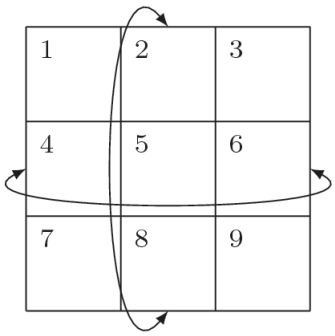
are replaced in R by the following L subsidiary reactions:

$$\bigcup_{l=1}^{L} R_{j,l} : \sum_{i=1}^{N} v^r_{j,N(l-1)+i} S_{N(l-1)+i} \xrightarrow{c_j} \sum_{i=1}^{N} v^p_{j,N(l-1)+i} S_{N(l-1)+i}; \tag{2.18}$$

note that the state change $v_{j,N(l-1)+i}$ only affects the state variable at position $N(l-1)+i$ in $\mathbf{x}$.

But reactions are not the only rules that have to be considered; particles can diffuse between sub-volumes, as mentioned above. An auxiliary structure, an $L \times 2n$ *connectivity matrix* $\mathbf{C}$, is defined that keeps track about the neighbors of each sub-volume (cf. Figure 2.9), with $\mathbf{C}_l = \mathbf{C}[l; 1 \ldots 2n]$ denoting the submatrix (in this case: vector) consisting only of the entries in l-th row. Then a "diffusion" reaction expressed in form of a rule may look like the following:

$$R_{i,l,k} : S_{N(l-1)+i} \xrightarrow{D_i} S_{N(\mathbf{C}_{l,k}-1)+i}. \tag{2.19}$$



$$\mathbf{C} = \begin{pmatrix} 3 & 4 & 2 & 7 \\ 1 & 5 & 3 & 8 \\ \cdots & \cdots & \cdots & \cdots \\ 8 & 3 & 7 & 6 \end{pmatrix}$$

Figure 2.9: Small 3×3 model and the corresponding connectivity matrix. It is assumed that the connectivity relation between the sub-volumes is toroidal, i.e., whenever a particle would leave the volume, it diffuses into the sub-volume directly located on the opposite side.

The rule removes one $S_{N(l-1)+i}$ particle and introduces it to the k-th neighbor of sub-volume l. Taking all species and sub-volumes into account adds a total of $NL2n$ reactions to R — which only describe the movement of the species between the sub-volumes. The diffusion constant D_i is a measure for the speed at which species S_i moves through the volume. An expression for the propensity function $a_{i,l,k}(\mathbf{x})$ associated with the diffusion reaction $R_{i,l,k}$ can be derived from Fick's first law [Fic55], which relates the flux J, i.e., the amount of substance diffusing through an area A during a unit time interval, to the speed of the particles and the concentration difference along the movement axis; so, for one spatial dimension, the flux is:

$$J = -DA\frac{\partial C(x)}{\partial x}. \tag{2.20}$$

To see how this can be used for deriving $a_{i,l,k}(\mathbf{x})$, a small system is assumed, consisting only of two sub-volumes, 1 and 2, with side length λ_{sv} and a single species S_1 with diffusion constant D_1 (ref. Figure 2.10). Each sub-volume contains only S_1 molecules, so the state matrix can be therefore represented by the vector $\mathbf{x} = (x_1, x_2)^T$. How does the propensity function $a_{1,1,2}(\mathbf{x})$ behave for the diffusion reaction $R_{1,1,2} : S_1 \xrightarrow{D_1} S_2$? As a first step, Equation 2.20 is approximated by a finite difference equation:

$$J \approx DA\frac{[C(x + \Delta x) - C(x)]}{\Delta x}. \tag{2.21}$$

For the well-stirred sub-volumes 1 and 2, the concentration difference is the difference in molecule numbers divided by the volume $V_{sv} = \lambda_{sv}^3$ and Δx is set to λ_{sv}, the center-

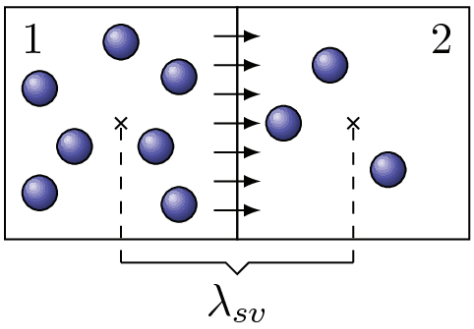


Figure 2.10: Example 2×1 model used to derive the propensity function $a_{1,1,2}(\mathbf{x})$ for the diffusion reaction $R_{1,1,2} : S_1 \xrightarrow{D_1} S_2$.

to-center distance between the two sites. Inserting this into Equation 2.21 finally gives:

$$a_{1,1,2}(\mathbf{x}) \equiv J \approx D\lambda_{sv}^2 \frac{[x_1 - x_2]}{\lambda_{sv} V_{sv}} = D\frac{[x_1 - x_2]}{\lambda_{sv}^2}. \tag{2.22}$$

But diffusion shall only be allowed in the direction from high to low concentration, so the propensity for an arbitrary diffusion reaction $R_{i,l,k}$ is defined as:

$$a_{i,l,k}(\mathbf{x}) = \begin{cases} D\dfrac{\left[x_{N(l-1)+i} - x_{N(\mathbf{C}_{l,k}-1)+i}\right]}{\lambda_{sv}^2} & \text{if } x_{N(l-1)+i} - x_{N(\mathbf{C}_{l,k}-1)+i} > 0, \\ 0 & \text{else.} \end{cases}, \tag{2.23}$$

The state change vector for reactions of this type has a single -1 at position $N(l-1)+i$ and a 1 at $N(\mathbf{C}_{l,k} - 1) + i$, so for short:

$$\mathbf{v}_{i,l,k} = -\mathbf{1}^{NL}_{N(l-1)+i} + \mathbf{1}^{NL}_{N(\mathbf{C}_{l,k}-1)+i} \tag{2.24}$$

Everything is now prepared for a non-spatial simulation of a spatial model using either the DM or FRM. Though not explicitly built for dealing with space, these basic algorithms can still be used if the reaction network is expanded as described above; from the perspective of an SSA, it just generates trajectories of a very large non-spatial model. However, representing spatial models like this does have some serious drawbacks. The size of the reaction network increases drastically when even only a few species, reactions, or sub-volumes are added — which also means more work for an algorithm. Furthermore, it is not very intuitive: there should be only N species, not LN, and reactions basically have the same effect, no matter in what sub-volume they

take place, so the expansion here should also be not necessary. Nevertheless, the fact remains that it *is* possible this way and the last thirty years saw several new algorithms, specifically tailored towards executing spatial models, which, more or less, are built upon the basics laid out in this section. The next chapter will present some of these and also discuss how to find a better representation of a reaction-diffusion system.

The Selection of λ_{sv} It has been long argued that for real systems λ_{sv}, i.e., the grid spacing, cannot be chosen arbitrarily small because the representation of the dynamics as a *reaction-diffusion master equation* (RDME), which is basically a generalization of the CME for spatially discretized systems, would break down if λ_{sv} is either smaller than the mean free path between collisions or larger than the mean free path between reactions [BM96] (the mean free path measures how far a particle can travel on average before colliding or reacting with another molecule). These restrictions have been re-evaluated lately by Sjoberg et al. [SBE09], with the result that at least the lower bound can be further decreased, allowing sub-volume sizes that come close to microscopic length scales; however, doing so requires an adaptation of rate constants. Regarding this topic, the dissertation is more focused on the algorithms that are able to simulate non-spatial and spatial reaction networks, and less on their application to explicit real world systems. Forthcoming chapters primarily use artificial benchmark models (the decision for this type of models will also be discussed) that allow to analyze how algorithms behave when faced with certain model parameters, e.g., high reaction constants, slow diffusions, many sub-volumes or sub-volumes with different λ_{sv}; so emphasis is laid on the question how the performance of an algorithm changes if, e.g., the diffusion constants increase by an order of magnitude or the number of sub-volumes is doubled. Despite this, the results obtained by Sjoberg et al. are especially important for Chapter 7 where an algorithm is introduced that uses a variable grid spacing.

2.6 Simulation Using *JAMES II*

The *JAMES II* (**JA**va-based **M**ultipurpose **E**nvironment for **S**imulation) framework [HU07b] has been created to ease the development of modeling and simulation methods. In contrast to its successor, which was specifically designed for agent-based modeling, *JAMES II* is targeted at a more general audience: it currently supports several formalisms, e.g., variants of DEVS [ZPK00], the π calculus [Mil99], and Cellular Automata, and offers various implementations of corresponding simulation algorithms.

The architecture of *JAMES II* is based on a concept called *plug'n simulate*, which enables users to create plug-ins for a number of plug-in types, such as models, simulators, random number generators, or event queues — just to name a few. Over the years the library of plug-ins grew constantly, encompassing now over 500 implementations; all together the framework consists of more than 6000 classes with nearly 450.000 lines of code. The main functions, like defining experiments and executing them, are collected inside the *JAMES II* core, which represents the heart of the framework. Plug-ins can be seen as extensions to this core functionality; for example, to implement a new stochastic simulation algorithm all a user has to do is to implement the algorithm as a plug-in for the PROCESSOR plug-in type and specify that this plug-in supports a certain type of model (e.g., Species-Reaction models). If additional sub-algorithms are needed, e.g., random number generators or event queues, than the instances are either created explicitly by the user or by *JAMES II*; in the latter case the REGISTRY, which is a repository for all registered plug-ins, is queried to find an appropriate implementation.

Models in the context of *JAMES II* are instances of *model classes*, which in turn are again realized as plug-ins. Simple languages for an easier creation of model instances for non-spatial and spatial stochastic simulations have been developed, but as this work is focused on the simulation part they will not be discussed in detail here.

JAMES II has been used intensively for this dissertation. All of the algorithms discussed in the evaluation study found in Chapter 6 (and several more) have been implemented as plug-ins for the framework. Furthermore, the study also makes use of advanced techniques for experiment execution, e.g., parallel simulation replication [ELU09] and automated runtime performance evaluation [EU09]. A more in-depth description of *JAMES II* can be found in the dissertation of Jan Himmelspach [Him07] and the software itself is available for download at `www.jamesii.org`.

2.7 Summary

The aim of this chapter was to provide the basic definitions, notations, and terms used throughout the dissertation. Starting from a biological perspective, Section 2.1 motivated the use of simulation, i.e., performing experiments on a model of a system, as a way to complement the work done in the wet-lab. Without a formal definition of what a model actually is, a preliminary abstract representation for a real-world biochemical reaction network has been introduced, making this the entry point into the topic of modeling and simulation (M&S) that followed.

Section 2.2 was a discourse into the field of M&S and provided the basic definitions for "simulation", "experiment", and "model"; these terms and several more are going to be used regularly in upcoming chapters.

The reaction network definition from the first section was re-visited and extended in Section 2.3. Reactions are the rules that modify the state variables; it was discussed how to include time by assigning reaction rate constants that quantify the intrinsic speed at which those reactions fire. When it comes to the actual simulation, there are several fundamental alternatives (solving differential equations, generating trajectories of stochastic processes), all of which assume a homogeneous distribution of the molecules inside the volume, i.e., a well-stirred system.

Discretizing the volume into smaller sub-units, called sub-volumes, allows a renunciation from the well-stirred assumption. It has been shown in Section 2.5 how a spatial model, where particles are allowed to diffuse between sub-volumes, can be reformulated such that a non-spatial simulation algorithm is able to perform the simulation task.

Finally, Section 2.6 introduced *JAMES II*, the simulation framework that has been extensively used for this dissertation.

3 Stochastic Simulation Algorithms: A Small Survey

From the early beginnings thirty years ago to the present day there has been — and still is — an active community behind the work on stochastic simulation of biochemical reaction systems. Existing methods get revisited again, discussed in the light of larger, more complex models which motivate the development of better, improved variants. There are, for example, numerous extensions of the basic Direct Method introduced in the last chapter; some of these descendants are again used as the foundation for others, so an entire family tree of methods exist. Speaking of a family tree, at some points a new seed gets planted that develops into a separate research direction; one example are the so-called *leap-methods* that introduce further assumptions on the dynamics of the underlying system.

On other occasions links are forged between research areas, either by developing a method that closes the gap or by a combination of techniques from different fields. For instance, starting with discrete stochastic exact algorithms (like the DM) it is possible to successively add assumptions that further approximate the real solution and finally arrive at the continuous deterministic reaction rate equations, the system of ODE's already mentioned previously.

The following sections will present some of the work that has been done in the field of stochastic simulation — in a sense, these can be seen as "branches" of different family trees, some of which are quite new while others have grown over almost three decades. To stay focused, only methods and concepts relevant for the further progress of this dissertation shall be discussed in more detail; alternative variants will be mentioned as well, but much more briefly.

3.1 Beyond The Origins

Taken as starting points, the DM and FRM (see Section 2.4.2) can be seen as "grandfathers" of modern exact stochastic simulation algorithms. Since their publication a number of improved and optimized variants have been presented; still conserving the basic modus operandi — calculate the time and type of the next reaction, then update the state and proceed with the next event — these methods are aimed at optimizing time-consuming operations or performing them in parallel on multiple processing units. Before introducing some of them, a different perspective on the representation of reaction network dynamics shall be briefly discussed. Four years before Gillespie presented his paper, Kurtz [Kur72] studied the relationship between deterministic and stochastic models for chemical kinetics. The main difference to the stochastic representation described in the previous chapter is that in the framework used by Kurtz the state of the system at any time t is defined by a vector of random variables representing the number of firings for each reaction (he also shows how to obtain the number of particles from this vector) and whose evolution is goverend by a *Poisson process*. While this may be confusing and less intuitive at the beginning, it turns out that starting from this premise it requires only a small step to arrive at the so-called *leap methods* that will be discussed below. The work of Kurtz was later taken up again by, e.g., Anderson [And07, And08].

3.1.1 The Next Reaction Method

A well-known representative is the Next Reaction Method (NRM) by Gibson & Brucks [GB00], who took the FRM and modified it to yield shorter execution times. As a reminder, the basic steps the FRM performs are: update all propensities, calculate for each reaction the interval until the next occurrence, take the minimum thereof, and finally execute the respective event. Almost every step has been refined towards a better performance. The first one, updating the propensities, is now limited to *dependent reactions*. Looking at the definition of $a_j(\mathbf{x})$ reveals that it does not have to depend on the entire state vector but only on the state of the reactant species. It can be therefore concluded that the propensity of a reaction R_j does not change unless a) R_j fires or b) another reaction fires that shares at least one species with the reactant set of R_j. This insight resulted in the introduction of an additional data structure — the dependency graph G_D.

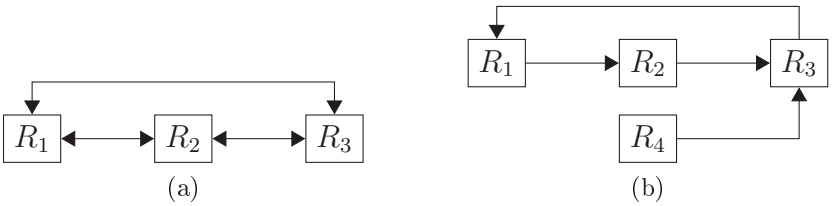


Figure 3.1: The dependency graphs for the a) enzyme-substrate reaction network and b) the small example network defined by the set of reactions in Equation 3.1.

Definition 3.1.1 A *reaction dependency graph* $G_D = (I_R, E)$ consists of the reaction index set I_R and an edge set that is defined as follows. Let $\mathcal{R}_j \subseteq S$ and $\mathcal{P}_j \subseteq S$ be the sets of reactant and product species for reaction R_j. Then there exists a directed edge (j, k) in the graph, i.e., R_j *depends on* R_k, iff:

a) R_j and R_k share reactant species, i.e., $\mathcal{R}_j \cap \mathcal{R}_k \neq \emptyset$ or

b) R_k has products species that are also reactants for R_j, i.e., $\mathcal{R}_j \cap \mathcal{P}_k \neq \emptyset$.

The graph can be easily created during the initialization of the algorithm. Once a reaction has been executed, all dependent reactions are identified and their propensities recalculated. The effectiveness of using the graph depends on the model: if the outgoing order, i.e., the number of edges originating from a node, is large for the majority of the reaction nodes, then more reactions have to be updated in each iteration and the benefit of the dependency graph becomes less noticeable.

Example 3.1.1 The dependency graph for the enzyme-substrate reaction example is shown on the left side of Figure 3.1. Here a firing of any reaction requires an update of all other reactions, so for this example the dependency graph does not provide any improvements over algorithms not using this structure. However, the scenario is quite different for the following set of reactions:

$$\begin{aligned} R_1 &: A + B \rightarrow C \\ R_2 &: D + E \rightarrow A \\ R_3 &: C \rightarrow D + F \\ R_4 &: F \rightarrow G \end{aligned} \tag{3.1}$$

The associated graph is depicted in Figure 3.1b. Executing R_3 requires an update of two other propensity values, while for any other event only a single additional update is necessary.

Algorithm 3.1: Pseudo-code description for the NRM's *SelectNextReactions*($\mathbf{x}$) method. Note that the NRM uses absolute instead of relative times, so if t_μ is the smallest next event time stored inside the event queue and t the current time, then $t_\mu - t$ is the interval until the next event will occur. During initialization, with $\mathbf{X}(0) = \mathbf{x}_0$, the event queue EQ is filled with pairs $(-\ln(U(0,1))/a_j(\mathbf{x}), R_j)$.

Parameters:
EQ (event queue)
G_D (dependency graph)
$\mu_l \leftarrow \infty$ (last executed reaction)
Input:
$\mathbf{x}$ (current state)
t (current time)

if $\mu_l \neq \infty$ **then**
  $D_{\mu_l} \leftarrow \{\nu : \exists(\nu, \mu_l) \in G_D \cup \mu_l \neq \nu\}$;
  update next event times for all $R_j, j \in D_{\mu_l}$:
    $\forall \nu \in D_{\mu_l} : t_\nu \leftarrow \left(\frac{a_\nu(\mathbf{x} - \mathbf{v}_{\mu_l})}{a_\nu(\mathbf{x})}\right)(t_\nu - t) + t$;
  requeue pairs $(t_\nu, R_\nu), \forall \nu \in D_{\mu_l}$ into EQ;
  update propensity $a_{\mu_l}(\mathbf{x})$ and calculate next event time for μ_l
    $t_{\mu_l} \leftarrow t + -ln(U(0,1))/a_{\mu_l}(\mathbf{x})$;
  enqueue pair (t_{μ_l}, R_{μ_l});
dequeue reaction R_μ from EQ with the smallest next event time t_μ;
$\mu_l \leftarrow \mu$;
return $t_\mu - t, \mathbf{1}_\mu^M$

Gibson & Bruck furthermore showed that calculating the next event time does not necessarily require generating new random numbers, which is considered by the authors as one performance bottleneck. A prerequisite for this is switching from relative to absolute event times. If events have a relative time stamp, then a reference time point is needed to determine when the event will take place. In both the DM and FRM τ is always given relative to the current time t; the next event time for the selected reaction R_μ is $t + \tau_\mu$.

The NRM, on the contrary, calculates for each R_j the next event time t_j and stores the time-reaction pair (t_j, R_j) inside an *event queue*. Algorithms of this type ensure that at any time the top most event, i.e., the item that will be dequeued next, has the minimum (or maximum, depending on the sorting preference) value for some sorting criterion, which is usually the time stamp of the event, yet it can be any of its attributes.

After a reaction firing, the propensities of all dependent reactions need to be updated, but doing so invalidates their stored next event times. Now the usual step would be to draw fresh random numbers and determine the interval until each of the reactions will fire again. Gibson & Bruck prove that by making the transition to absolute times the next event time t_j can be recalculated based on the current value and the updated propensities, without making calls to the RNG. If R_μ is the executed reaction at $t = t_\mu$ and $D_\mu = \{\nu : \exists(\nu, \mu) \in G_D \cup \nu \neq \mu\}$ the set holding the indices of reactions depending on R_μ, then

$$t_\nu = \left(\frac{a_\nu(\mathbf{x} - \mathbf{v}_\mu)}{a_\nu(\mathbf{x})}\right)(t_\nu - t) + t, \nu \in D_\mu. \tag{3.2}$$

The only random number required is used to determine when R_μ will fire next, which is done in the same way as it was for the FRM: by sampling from $Exp(a_\mu(\mathbf{x}))$ (see Section 2.4.2). Algorithm 3.1 summarizes the Next Reaction Method in pseudo-code description.

3.1.2 The Optimized and Logarithmic Direct Methods

A prevalent argument against the NRM is targeted at the more complex algorithmic description and the need for additional data structures, e.g., an event queue implementation and a storage for the dependency information. Cao et al. [CLP04] presented an optimized version of the DM (the Optimized Direct Method, or ODM for short), that updates only dependent reactions similar to the NRM, but also uses an additional enhancement: the reactions are sorted by decreasing propensity values, which can speed up finding the reaction that will be executed next. In the original DM, individual probabilities $a_j(\mathbf{x})/a_0(\mathbf{x})$ are summed up until the sum exceeds a generated uniform random number. The index of the last reaction whose probability was added is selected as the imminent reaction. This can be very inefficient if there are a lot of reactions that only contribute with small propensity values to the overall sum. The ODM records the amount of firings per reaction during an initial warm-up phase to calibrate itself. It then sorts all reactions by their propensity, in descending order. This way higher propensities are summed up first, which makes it more likely that the iteration stops after only a few summations.

Other authors also identified the search for the reaction that will fire next as one critical bottleneck of the original DM. Even including the ODM's modifications, the upper bound complexity for locating the imminent reaction is still $O(M)$ — in the worst case the sum runs over all reaction indices. Furthermore, relying solely on a pre-

simulation to find the ordering can lead to a non-optimal result if the reaction execution behavior of the model changes over time, i.e., from some time point onwards reactions that only had a low firing rate first start getting executed more often, yet they are still near at the end of the sequence determined during the pre-simulation. But interpreting the task as a *search problem*, i.e., find the index $j^* \in [1, M]$ in a list of partial sums $(a_1(\mathbf{x}), \sum_{j'=1}^{2} a_{j'}(\mathbf{x}), \ldots, \sum_{j'=1}^{M} a_{j'}(\mathbf{x}))$ for which $\sum_{j'=1}^{j^*-1} a_{j'}(\mathbf{x}) < U(0,1)a_0(\mathbf{x}) \leq \sum_{j'=1}^{j^*} a_{j'}(\mathbf{x})$, opens up a wide range of special algorithms that perform this lookup very efficiently. This idea finds its application, among others, in the Logarithmic Direct Method [LP06]; its name is derived from the average complexity required to locate the index when using a binary search algorithm: $O(\log M)$. Another algorithm, the Sorting Direct Method [MPC+06], is built upon the ODM and *dynamically* sorts the reactions by decreasing propensity: after a reaction has fired, it is moved up in the hierarchy, so even reactions that are "silent" at the beginning but active later on will eventually appear near the start of the list.

It shall be noted that the basic idea for this type of algorithms (using a search to find the next reaction) has been already proposed in 1995 by Fricke & Wendt [FW95] in the context of reaction-diffusion systems. Here sub-volumes are classified according to their propensity (called "reactivity" in the paper) and in the first step during each iteration the class containing the volume that will execute an event next is found using linear selection (by adding up the propensities, just like it is done in the DM), but with the twist that the classes also get sorted by decreasing reactivity — which makes it more likely to find a class after only a few summations in subsequent iterations. The next two steps select a sub-volume from the chosen class and then the event (diffusion or reaction) that is going to take place.

3.1.3 Parallel Variants

The algorithms discussed so far are most often implemented as a sequential execution of operations, i.e., they only perform a step after the previous one has finished. With the advent of multi-core processing units, e.g., dual or quad-core CPUs or GPUs (graphics processing units), which enable a concurrent execution of tasks, developers return to some of the existing algorithms and try to adapt their implementations to the new hardware. As it turned out, the FRM, considered as being inferior to the DM, is a suitable candidate for this. Figure 3.2 illustrates the differences between a sequential and parallel implementation of this method. The former variant always runs over all

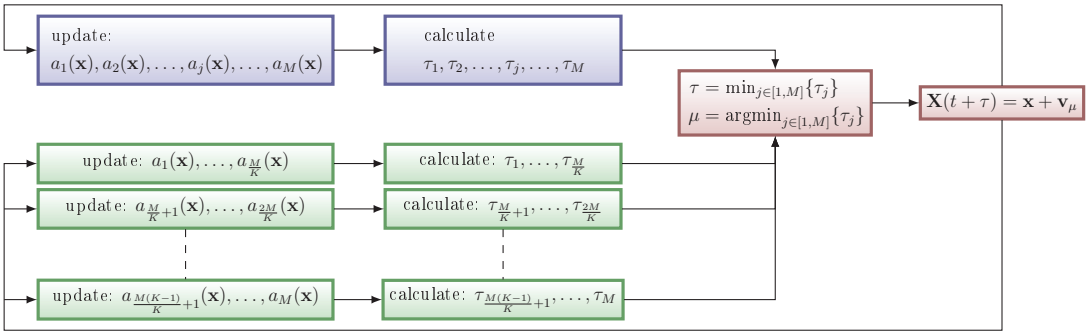


Figure 3.2: Sequential versus parallel FRM execution. The sequential algorithm (top, blue) first calculates all propensities and then the τ candidates. In contrast, a parallel variant (bottom, green) partitions the set of M reactions across $K \leq M$ processing units, which perform the updates and calculations independently from each other. After finishing their task, they send back the results and the minimum time interval is determined.

reactions twice, first to update the propensities and then to find the minimum next event interval. It should be noted here that Dittamo & Cangelosi [DC09] modified the original algorithm, primarily to reduce its memory requirement, but it also avoids the second loop (see Algorithm 3.2). The FRM has to store N state variables, M propensities and M τ candidates, which does not look like much but if memory is limited (as it is the case for GPUs, details follow below) every additional object to store can be critical.

Regardless of the sequential variant, the entire work is done by a single processor. If the number K of available processing units is larger than one, e.g., $1 < K \leq M$, then it is possible to distribute the work load across these resources, as has been show, e.g., by Dittamo & Cangelosi [DC09] and Niu et al. [NZC+07]. Doing so only requires a partition of R into K subsets, each having at most $\lceil M/K \rceil$ entries. Now these sets and a reference to the state vector are assigned to each processor; they independently perform all the steps listed in Algorithm 3.2 (with the loop of the k-th processor running over the interval $[\lceil M(k-1)/K+1 \rceil, \lceil kM/K \rceil], k \in [1, K]$) and send their results back to some central unit, which collects the returned data into a new set with k τ candidates. The final steps are similar to the original: find the minimum among these candidates, execute the appropriate reaction and update the current time.

This data level decomposition (each processor basically runs the same code but calculates the minimum τ only for the reactions assigned to it) is characteristic for a

Algorithm 3.2: Alternative $SelectNextReactions(\mathbf{x})$ variant for the FRM, which avoids the second loop.

Parameters:
$\tau_{min} \leftarrow \infty$ (minimum τ candidate)
$\mu_{min} \leftarrow \infty$ (index of reaction having minimum τ)
Input: $\mathbf{x}$ (current state)

```
1 for j ∈ [1, M] do
2     update propensity a_j(x);
3     τ_j ← −ln(U(0,1))/a_j(x);
4     if τ_j < τ_min then
5         τ_min ← τ_j;
          μ_min ← j;
6 return τ_min, 1^M_μmin
```

"parallelization inside a simulation run"[1]. The processors have to send their individual results back to the central unit during each iteration; this overhead in message processing should be much less than the actual computation to gain an advantage over a sequential execution. The better the processors are connected, e.g., all are located close together on the same chip, the more likely it is that this overhead can be neglected. In contrast, with increasing distance the messages will also take longer until they reach their destination; the step from a multi-core processor to a local network of computers may result in losing the benefits of a parallel execution.

A point which has not been mentioned so far is the parallel generation of random numbers. Simply using one RNG per processing unit may lead to correlations between the sequences of different generators if they have been parameterized without caution. Dittamo & Cangelosi [DC09] avoid this problem by using a special RNG implementation which dynamically creates parameters for all parallel instances. Executing their modified FRM variant on a GPU results in a speed-up of about 2 for the heat shock response model of E.Coli. consisting of 28 species and 61 reactions. As it was mentioned above, adapting the FRM for running on a GPU proved to be a challenging task; the very fast on-chip memory has a size of only 16 KB, so it was necessary to somehow reduce the storage requirement of the original version.

An alternative parallel execution approach has been presented by Niu et al. [NZC+07]; unfortunately, they do not provide any further information about how the processors

[1] referred to as a "parallelism across the method" approach in [TB05]

are connected (they utilize the Java-based JADE framework for multi-agent systems, which can distribute a simulation across several machines) or how random numbers are generated in their DSSA. They report a maximum speed-up of about 4 for a large, 5000 reaction decay-only model and 6 processors; tests with the same number of units and a MAPK model (86 species, 300 reactions) showed a decrease in execution time by a factor of 2.5.

An alternative to "parallelization inside a simulation run" is called "parallelization across a simulation"[2]. Many runs, i.e., trajectories, are needed to derive characteristics of the underlying stochastic process. Instead of having one processor calculating all replications, each unit simulates a single run and returns the required data to the central server at the end of the time interval. Communication only takes place after a processor has finished an entire run, so the number of messages sent through the network is much smaller than for a "parallelization inside a simulation run". Such an approach has been presented by Tian & Burrage [TB05] and is based on OpenMP [Ope], an API for shared-memory multi-processing. Interestingly, they do not use a parallel RNG at all; a predetermined amount of numbers is generated and stored in a shared data structure before each processors executes one or more SSA steps. Hence a simulation run is interrupted more or less frequently to wait for the RNG (running on a single unit) to fill up the data structure again (a more dynamic variant could produce a set of numbers and pause the processors if all have been consumed). Also performing experiments with the MAPK model, the speed-up reported is about 4.5 using 10 processors.

In a later publication by Petzold & Li [PL09] the same basic principle of parallel replicated runs has been implemented on a graphics processing unit. Random numbers are generated directly on the GPU using a multi-threaded version of the widely applied Mersenne twister algorithm. The results are impressive: the authors were able to accelerate the execution by a factor of 200 for both tested models, the first one being a simple decay dimerization model and the second a spatially inhomogeneous system.

3.2 Leap Methods

Realizing that any exact algorithm suffers from the same drawback of becoming very slow in case of large populations or high reaction constants, a second family of methods, termed *approximative* algorithms, originated from the τ-*leaping* algorithm that have been published in 2001. As the name suggests, these methods do not provide an

[2] referred to as "parallelism across the simulation" in [TB05]

exact sample of the state space; instead, they sacrifice accuracy for execution speed by making additional assumptions about the dynamics.

3.2.1 The τ-leaping Algorithm

The basic, non-spatial τ-leaping algorithm has been introduced by Gillespie et al. [Gil01] to speed up the simulation of large biochemical reaction networks having many species with high populations. In a nutshell, instead of simulating every single reaction that occurs inside the system, as done by exact algorithms, the τ-leaping algorithm performs "leaps" along the time line, calculating for all reactions how often they fire during each interval. A more formal way to illustrate the basic difference between exact and τ-leaping algorithms is to have a look at how the state is updated; the general equation has been already introduced in Section 2.4.2 (Algorithm 2.1):

$$\mathbf{X}(t+\tau) = \mathbf{x} + \sum_{j=1}^{M} k_j \mathbf{v}_j. \tag{3.3}$$

The M-dimensional vector $\mathbf{k}$ stores how often each reaction is allowed to fire. Exact methods handle every reaction execution as an independent event; if the execution of R_μ is scheduled next, then $\mathbf{k} = \mathbf{1}_\mu^M$, i.e., it is a zero vector with only the μ-th entry set to 1. In contrast, τ-leaping variants calculate a "leap" τ usually being larger than the interval until the next event, hence more than one reaction can fire within $[t, t+\tau]$ — this also means that $\mathbf{k}$ can now have more than one entry set to a non-zero value. Each $k_1, k_2, \ldots, k_M$ is no longer either zero or one, but a result of some function (or, maybe more precisely, a sample from the random variable) $K_j(\tau; \mathbf{x})$ that depends on the size of the leap and the current state of the model[3]. The difficulty in finding an expression for these functions arises from the fact that they are (most likely) dependent random variables – executing any reaction modifies the state and therefore also *changes the propensity of all dependent reactions*. But if all RV are *assumed* to be independent, then the k_j could be sampled individually from univariate distributions. This transition from dependent to independent RV is the first essential step towards an approximative

[3] In fact, the k_j for the exact case can be written as results of functions. Given a sample u from the uniform distribution $U(0,1)$, then:

$$k_j = K_j(\mathbf{x}) = \begin{cases} 1 & \text{if } \operatorname{argmin}_{J \in [1,M]} \left(\sum_{j'=1}^{J} a_{j'}(\mathbf{x}) / a_0(\mathbf{x}) > u \right) = j, \\ 0 & \text{else.} \end{cases}$$

simulation scheme. To see that the idea behind τ-leaping is based on the assumptions already made for exact methods, it is necessary to have a closer look at the latter.

Suppose that at time t_0 the reaction network is in state $\mathbf{x}$. The last chapter defined the probability that a reaction R_j will fire during the next interval $[t_0, t_0 + \Delta t]$ as $a_j(\mathbf{x})\Delta t$ for small Δt (Equation 2.14). Based on this, a hypothetical exact simulation algorithm could check for successive intervals $[t_0+i\Delta t, t_0+(i+1)\Delta t], i \in \mathbb{N}_0$ whether any $R_j, j \in [1, M]$ will fire. Because the checks are performed independently of each other and nothing changes the state between two intervals, this corresponds to a sequence of *Bernoulli trials* with probability $a_j(\mathbf{x})\Delta t$ of observing a "success" (i.e., a firing) and $(1 - a_j(\mathbf{x})\Delta t)$ of observing a "fail" (no firing). Trials are performed until a success is observed for some $R_\mu, \mu \in [1, M]$; like in any exact method (Algorithm 2.3), the state is then altered by executing R_μ, the propensities are updated and the sampling process is started again.

Instead of performing the individual checks it is also possible to directly obtain the index $\hat{i}_j$ of the interval $[t_0 + \hat{i}_j\Delta t, t_0 + (\hat{i}_j + 1)\Delta t]$ during which the first firing of R_j is observed — it is a sample from the discrete *geometric distribution* with parameter $p = a_j(\mathbf{x})\Delta t$. If now $\Delta t \to 0$, the geometric distribution can be replaced by its continuous analogue, the *exponential distribution*, which gives the time interval until R_j fires next; taking the minimum over all $j \in [1, M]$ gives the imminent reaction. Comparing this simulation scheme with Algorithm 2.2 reveals that it simply describes Gillespie's First Reaction Method.

Tau-leaping is very similar to what has been just described. Starting with $\mathbf{x}$ at time t_0, an observer may realize that the sequence of reaction firings at times $t_1, t_2, \ldots t_T$ changes the state only insignificantly, i.e., $|\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i)| \leq \epsilon, \forall i \in [0, T]$ for some small ϵ. With the difference between the new and old state being negligible small, it is assumed that $a_j(\mathbf{x})\Delta t$ is constant for the interval $\tau = t_T - t_0$. It has been asked above how long it takes until the first reaction will fire; this time the interval is known but not how often each reaction has fired. Again, Bernoulli trials can be performed for successive intervals $[t_0 + i\Delta t, t_0 + (i + 1)\Delta t]$ until $(i + 1)\Delta t = \tau$; the number k_j of successes for each reaction then determines how often R_j is executed at $t + \tau$. As before, the individual trials can be avoided by generating samples from the *Binomial distribution*, which models the number of successes observed in $n = i + 1$ trials if each has a success probability of $p = a_j(\mathbf{x})\Delta t$; if $\Delta t \to 0$, the Binomial distribution converges to the continuous *Poisson distribution* with mean and variance $a_j(\mathbf{x})\tau$.

However, the size of τ is not known *a priori*; the only information given is the

state at time t_0. Nevertheless, what *is* known is that the length of the interval should be chosen such that the state does not change significantly; if a τ can be calculated that does not violate this *leap condition*, then the number of reaction firings can be obtained by sampling from a Poisson distribution $\text{Pois}(a_j(\mathbf{x})\tau)$. But what does the vague phrase "does not change significantly" mean? Tau-leaping has been developed to handle models with large populations or high rate constants. The more particles are present, the smaller is the impact of a single reaction and, further increasing the population, also of several reactions executed as a bulk. So the answer to the question above is: it depends on the current state of the model. First publications, e.g., [Gil01], proposed $\epsilon a_0(\mathbf{x})$ as the threshold for restricting the propensity change, with ϵ as an error control parameter. But this turned out to be too restrictive: suppose summing up only high propensities and finally adding a very small one — the allowed change for this last reaction would be very high, so in fact the leap should be rejected. Later improvements defined a threshold based on individual propensities or bounded the changes in the species populations such that the leap condition is satisfied [CGP06]. The original τ selection equation shall be given here, however, it may look slightly different for other variants of this technique; a more detailed derivation of this function will be given in the next chapter that introduces a spatial extension to the leap method:

$$\tau = \min_{j \in [1,M]} \left\{ \frac{\epsilon a_0(\mathbf{x})}{|\mu_j(\mathbf{x})|}, \frac{\epsilon a_0(\mathbf{x})^2}{(\sigma_j(\mathbf{x}))^2} \right\} \tag{3.4}$$

with $\mu_j(\mathbf{x})$ and $(\sigma_j(\mathbf{x})^2$ defined as:

$$\begin{aligned} \mu_j(\mathbf{x}) &\equiv \sum_{j' \in \mathcal{J}} a_{j'}(\mathbf{x}) \sum_{i \in [1,N]} \frac{\partial a_j(\mathbf{x})}{\partial x_i} v_{j'i} \\ (\sigma_j(\mathbf{x}))^2 &\equiv \sum_{j' \in \mathcal{J}} a_{j'}(\mathbf{x}) \sum_{i \in [1,N]} \frac{\partial^2 a_j(\mathbf{x})}{\partial x_i^2} v_{j'i}^2 \end{aligned} \tag{3.5}$$

For now, let $\mathcal{J} = [1, M]$.

Having found a suitable τ, the next step would be to generate for each reaction R_j a sample from $\text{Pois}(a_j(\mathbf{x}), \tau)$. But this may lead to incorrect results. Poisson RVs 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 [TB04, CVK05] or by classifying reactions into critical and non-critical and handling critical reactions

differently [CGP05b]. As the τ-leaping variants implemented and studied for this dissertation are based on the latter approach, it shall be explained in more detail. A reaction is considered critical if one of its reactants is only n_c firings away from depletion. After separating the indices of critical and non-critical reactions into two disjoint sets J_c and J_{nc}, the leap calculation in Equation 3.5 only runs over $\mathcal{J} = J_{nc}$, producing a first τ candidate τ_{nc}. The second candidate τ_c is determined in the same way as for the DM: $\tau_c = -ln\, U(0,1)/a_0^c(\mathbf{x})$, with the sum $a_0^c(\mathbf{x})$ running only over the critical reactions in J_c. If $\tau_c < \tau_{nc}$, then a single critical reaction will be executed; the method to find out which one is again the same as for the DM (Algorithm 2.3). Regardless of whether a critical reaction fires or not, how often each non-critical reaction will be executed is sampled from a Poisson RV parameterized as described above.

Most τ-leaping implementations switch to an SSA phase if τ_{nc} is very small; the threshold for this decision is the time interval until the next reaction will fire, which is approximately $1/a_0(\mathbf{x})$. If τ_{nc} is smaller than a multiple of this value, i.e., $\gamma/a_0(\mathbf{x})$ (with γ usually set to 10), then a number N_{SSA} of exact iterations is performed. This is essentially an admission to the higher computational cost required to find a value for τ_{nc}. A possible simple improvement avoids the fixed value for γ by considering how long it took to calculate a leap candidate. Let t_T be the time required to find a τ_{nc} and t_S an estimation for the time it takes to calculate the step size an exact algorithm. Then the ratio t_T/t_S gives the factor that denotes how much longer it took to find the leap value. Multiplying this value with $1/a_0(\mathbf{x})$ essentially approximates the length of the interval simulated with an exact algorithm in t_T time units — if the outcome is larger than τ_{nc}, then τ-leaping should be abaondoned for the moment. A similar condition can be defined by looking at the ratio between τ_{nc} and $\tau_S = 1/a_0(\mathbf{x})$: if $\tau_{nc}t_s/\tau_S < t_T$, then simulating an interval of size τ_{nc} takes less time with an exact algorithm than with τ-leaping. Using the time it takes to finish the task of calculating a leap candidate as an additional criterion considers different *implementations* of the same algorithm; two variants of the same method could differ, e.g., in the data structures used to represent species and reactions. In conclusion, 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 more simpler and, in this case, often faster exact simulation.

Algorithm 3.3: Pseudo-code description for the $SelectNextReactions(\mathbf{x})$ method used in the τ-leaping algorithm. $CalculateNonCriticalTau()$ can be implemented, e.g., according to Equation 3.4. Additionally, the template from Algorithm 2.1 has to be extended to test the final state if there are negative entries; if yes, then τ is halved and a new $\mathbf{k}$ calculated.

Parameters:

$\epsilon \in \mathbb{R}$	(leap condition parameter)
$\gamma \in \mathbb{R}$	(SSA threshold parameter)
$N_{SSA} \in \mathbb{N}$	(number of total SSA iterations)
$h \leftarrow 0$	(number of remaining SSA iterations)
$n_c \in \mathbb{N}$	(critical reaction threshold)
$SelectNextReactions(\mathbf{x})^*$	(an implementation from Algorithms 2.3, 2.2, or 3.1)

Input: $\mathbf{x}$ (current state)

for $j \in [1, M]$ **do**
  update propensity $a_j(\mathbf{x})$;
if $h > 0$ **then** // check for additional SSA iterations
  $h \leftarrow h - 1$;
  return $SelectNextReactions(\mathbf{x})^*$;
else
  $J_c \leftarrow \{j : j \in [1, M] \wedge \min_{i \in [1,N], v_{ji} < 0} \lceil x_i / |v_{ji}| \rceil < n\}$;
  $J_{nc} \leftarrow [1, M] \setminus J_c$;
  $\tau^{nc} \leftarrow CalculateNonCriticalTau(\mathbf{x}, J_{nc}, \epsilon)$;
  if $\tau^{nc} < \gamma / a_0(\mathbf{x})$ **then**
    $h \leftarrow N_{SSA} - 1$;
    return $SelectNextReactions(\mathbf{x})^*$;
  $a_0^c(\mathbf{x}) \leftarrow \sum_{j \in J_c} a_j(\mathbf{x})$;
  $\tau^c \leftarrow -ln(U(0,1))/a_0^c(\mathbf{x})$;
  if $\tau^{nc} < \tau^c$ **then** // no critical reaction fires
    $\tau \leftarrow \tau^{nc}$;
    $\mu_c \leftarrow \infty$;
  else // one critical reaction fires
    $\tau \leftarrow \tau^c$;
    $\mu_c \leftarrow$ smallest integer j satisfying $\sum_{j \in J_c} a_j(\mathbf{x})/a_0^c(\mathbf{x}) > U(0,1)$;
  create $\mathbf{k} \in \mathbb{N}^M$ with

$$k_j \leftarrow \begin{cases} 0 & j \in J_c, j \neq \mu_c, \\ 1 & j \in J_c, j = \mu_c, ; \\ \text{sample from } \mathrm{Pois}(a_j(\mathbf{x}), \tau) & j \in J_{nc} \end{cases}$$

  return $\tau, \mathbf{k}$

3.2.2 k_α-leaping

An alternative leap method, called k_α-leaping, has been presented along with τ-leaping in [Gil01]. While it was previously the task to first find a τ that satisfies the leap condition and then calculate for each reaction how often it will fire, the order of these operations is now reversed — at least for a specified reaction $R_\alpha \in R$. The idea is to determine the maximum number k_α of times R_α can be executed before the resulting state change will violate the leap condition for one of the reactions in R and then use this result to find τ and the $k_j, j \in [1, M], j \neq \alpha$. If it is assumed that the propensities are nearly constant during the leap, then the time it takes for k_α firings of R_α is distributed according to an Erlang distribution (a special type of a Gamma distribution, where the shape parameter k is an integer), so $\tau \sim \mathrm{Erl}(k_\alpha, a_\alpha(\mathbf{x}))$. Having found τ, the remaining k_j are again sampled from $\mathrm{Pois}(a_j(\mathbf{x}), \tau)$, just like in the τ-leaping algorithm. Both k_α-leaping and τ-leaping are equivalent variants of the same basic technique: accumulate reactions as long as the leap condition is satisfied. However, the developers of k_α-leaping argue that it often could be more convenient to use the alternative τ-leaping (as a reminder, both methods have been presented in [Gil01]): with the latter it is possible to perform a leap that precisely ends at some predetermined time point t', at which, e.g.,, a new species is introduced into the system. If the current time is t then τ-leaping can be performed until $t + \tau > t'$; instead of executing this leap, τ is set to $t' - t$, so the simulation jumps exactly to t' and the event scheduled at this time can be executed. But basing τ on the number of firings for one reaction makes it much more difficult to handle such scenarios. However, several authors took the idea of k_α-leaping and presented improved offsprings of this technique. R-leaping [ACK06] and the K-leap [CX06] method are two examples; in contrast to the original, both restrict the number of firings for *each* reaction channel, not just a single R_α. Evaluation studies show that they are not just faster than τ-leaping but also able to produce more accurate results, which makes them interesting alternatives.

3.2.3 Implicit τ-leaping

The leap methods presented so far are essentially *explicit* solvers, i.e., they take the current state as an input and calculate how it will evolve over the next leap τ — very similar to explicit numerical integration algorithms used to find a solution for a system of differential equations, like Euler's method. As this, they are susceptible to *stiff systems*, which are generally characterized by having well-separated fast and slow time

scales, with the fast ones being stable. Stiffness may be present in systems including reversible reactions, like the earlier mentioned enzyme-substrate reaction network (see Equation 2.5); if $k_2 >> k_3$, i.e., the substrate unbinding is much faster than the actual transformation into the product, then the system exhibits a stiff behavior [CGP05a]. Reactions R_1 and R_2 simply undo each other and shortly after starting a simulation run the populations of S and ES will quickly reach some equilibrium state (the number of particles fluctuate around a mean value) and stay nearly constant until eventually an ES reacts into a product P and an enzyme E.

When faced with such systems, explicit methods can become unstable, i.e., they start oscillating around the equilibrium, which eventually can lead to large errors in the computation (see, e.g., [RPCG03]). One way to prevent this is to reduce the maximum allowed step size to the time scale of the fastest reactions; however, this would limit the performance of the algorithm drastically.

Luckily, as stiff systems are also a prominent problem in numerical analysis, better techniques more suited for this type of systems have been developed over time, which are generally summarized as *implicit* methods. Instead of looking only at the current state, algorithms of this type take the still unknown future state also into account when, in the case of stochastic simulation of reaction networks, determining how often each reaction fires during an interval. However, while implicit leap methods, like the one presented by Rathinam et al. in [RPCG03], overcome the stability problems, they require additional steps to find a leap candidate, with the result that they are slower than explicit variants in case of a non-stiff system.

But what if it is unknown whether some model is stiff or not? Or maybe it is non-stiff for some time, but then becomes stiff as time progresses and initially infrequent reactions start to fire more often. Adaptive methods offer one solution [CGP07, San09b]. As was mentioned above, the presence of reversible reactions and large differences between rate constants may hint to a potentially stiff system. An adaptive algorithm somehow has to determine what method (explicit or implicit) is better suited for the next leap based on the current state of the system. Cao et al. propose to make this decision based on the sizes of an explicit and implicit leap: both are calculated according to Equation 3.4, but the set $\mathcal{J}$ is different in Equation 3.5: it is replaced with J_{nc} for the explicit and J_{nc}/J_{pe} for the implicit case. Here the set J_{pe} contains the indices of reactions which are reversible and in partial equilibrium, i.e., that have almost the same propensity as the respective complementary reaction; the idea behind implicit leaping is to "ignore" these reactions during τ calculation as their firings would only

have a marginal impact on the overall state change during the leap. If now the explicit leap value is much smaller than the implicit one (e.g., by a factor of 100), then the system is considered to be stiff and the state change is calculated implicitly.

3.3 Algorithms for Spatially Inhomogeneous Systems

The first attempt to simulate spatially inhomogeneous systems, as presented in Section 2.5, essentially reformulated the task to utilize non-spatial algorithms: each sub-volume gets their own set of species and diffusions are modeled as unimolecular reactions. While possible, creating models this way can be cumbersome if the number of sub-volume is increased. For the following discussion on spatial stochastic simulation algorithms, a more convenient description for reaction-diffusion systems shall be introduced.

Each sub-volume contains a number of particles for each of the N species defined in the reaction network. With L sub-volumes, each one being a well-stirred environment of its own, the state of the *entire* system can be represented by combining all state variables into an $L \times N$ *state matrix* $\mathbf{X}$, which essentially replaces the state vector $\mathbf{x}$:

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,N} \\ x_{2,1} & x_{2,2} & \dots & x_{2,N} \\ \dots & \dots & \dots & \dots \\ x_{L,1} & x_{L,2} & \dots & x_{L,N} \end{pmatrix}. \tag{3.6}$$

The evolution of the system, i.e., how the number of particles in each sub-volume changes over time, is again interpreted as a stochastic process $\mathbf{X}(t)$, but this time defined over a *matrix of random variables*; $\mathbf{X}(t) = \mathbf{X}$ shall denote the current state and, 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 $l \in [1, L]$. If not stated otherwise, the set R only contains the rules regarding the reactions between species, but not the equations describing their diffusion; the latter is handled implicitly by the algorithms and the modeler only has to provide the set of diffusion constants $D = \{D_1, \dots, D_N\}$ along with S and R.

3.3.1 Next Sub-volume Method

The Next Sub-volume Method [EE04] can be roughly described as a spatial algorithm incorporating techniques from both the DM and NRM. Instead of time-reaction pairs,

each sub-volume together with its next event time is inserted into an event queue. Events are no longer only reactions, but can also be diffusions between neighboring sites. They are processed in a hierarchical manner by first finding out *where* the next event will occur and then specifying *what* type (reaction or diffusion) it is going to be.

If $\mathbf{X}(t) = \mathbf{X}$ is the current of the system, $a_0^l(\mathbf{X})$ the propensity sum and $d_0^l(\mathbf{X})$ the sum of diffusion rates for sub-volume l, then sampling from $\text{Exp}(a_0^l(\mathbf{X}) + d_0^l(\mathbf{X}))$, i.e.,

$$\tau_l = \frac{-\ln U(0,1)}{a_0^l(\mathbf{X}) + d_0^l(\mathbf{X})}, \tag{3.7}$$

gives the time interval until either a reaction inside or a diffusion originating from sub-volume l takes place. At the start of an iteration, the sub-volume l^* with the minimum next event time τ_{l^*} is taken from the queue. Each event type has a probability proportional to its rate sum, hence

$$P\{\text{event is reaction}\} = \frac{a_0^{l^*}(\mathbf{X})}{(a_0^l(\mathbf{X}) + d_0^{l^*}(\mathbf{X}))} \tag{3.8}$$

and

$$P\{\text{event is diffusion}\} = \frac{d_0^{l^*}(\mathbf{X})}{(a_0^{l^*}(\mathbf{X}) + d_0^{l^*}(\mathbf{X}))}; \tag{3.9}$$

deciding *what* will occur is then equivalent to testing, e.g., whether

$$P\{\text{event is reaction}\} > U(0,1). \tag{3.10}$$

If true, i.e., the event is a reaction, the index $\mu \in [1, M]$ of the imminent reaction is found in the same way as it is for the DM. Executing R_μ first adds its state change vector to the state of the SV, then both $a_0^{l^*}(\mathbf{X})$ and $d_0^{l^*}(\mathbf{X})$ get updated and finally Equation 3.7 calculates a new next event time.

Otherwise, i.e., in case of a diffusion, each species $S_i, i \in [1, N]$ has probability

$$P_{diff}\{S_i; l^*\} = \frac{x_{l^*,i} D_i}{\lambda_{sv}^2 d_0^{l^*}(\mathbf{X})} \tag{3.11}$$

that during τ one of its particles moves from the selected sub-volume to a neighbor. Just like finding the index μ for the imminent reaction, the index ι of the diffusing

species can be determined by finding the smallest integer satisfying

$$\sum_{i=1}^{\iota} \frac{x_{l^*,i} D_i}{\lambda_{sv}^2 d_0^{l^*}(\mathbf{X})} > U(0,1), \iota \in [1, N]. \tag{3.12}$$

A particle of S_ι is allowed to diffuse freely into any of the sub-volume's neighbors, each of which having therefore the same probability $1/2n$ of being selected as the diffusion target; finding it is simply a matter of randomly picking an index from the neighbor list $\mathcal{C}_{l^*}$. Assuming that SV l' has been chosen, the final step updates both states $\mathbf{X}_{l^*}$ and $\mathbf{X}_{l'}$, the values for $a_0^{l^*}(\mathbf{X}), a_0^{l^*}(\mathbf{X}), d_0^{l^*}(\mathbf{X})$, and $d_0^{l^*}(\mathbf{X})$, and recalculates the event intervals τ_{l^*} and $\tau_{l'}$.

3.3.2 Gillespie Multi-Particle Method

The Gillespie Multi-Particle Method [RKDB06] uses operator splitting to clearly separate between reaction and diffusion events, with the latter taking place as a bulk and in predetermined intervals. The time τ_i between two diffusion events of species $S_i \in S$ is approximately the same needed by a single molecule to cross a sub-volume of side length λ_{sv}, so

$$\tau_i = \frac{\lambda_{sv}^2}{2nD_i}, \tag{3.13}$$

with $D_i \in D$ as diffusion coefficient for species S_i and n as the dimension of the system, e.g., $n = 3$ for a 3D volume. As a consequence, diffusion in GMPM is independent of the system's state; all molecules of S_i will be propagated at fixed time points $\tau_{i,k} = k\tau_i, k \in \mathbb{N}_1$. During initialization, the species are inserted into an event queue according to their first event time $\tau_{i,1}$. An iteration at time t starts by receiving the species $\hat{S}$ with the smallest next diffusion time $\hat{\tau}$ from the queue. The only events that can occur during $[t, t+\hat{\tau}]$ are reactions, so an SSA variant runs locally in every sub-volume for this interval. At $t + \hat{\tau}$, the algorithm switches to the diffusion phase in which all particles of species $\hat{S}$ are simultaneously distributed across neighboring sub-volumes. Finally, the current time is updated, i.e., $t = t + \hat{\tau}$, the next diffusion event for $\hat{S}$ is scheduled at $t + \hat{\tau}$ and a new iteration starts.

The models simulated with the GMPM in [RKDB06] differ heavily in their complexity, which leaves the impression that the algorithm can be used as a multi-purpose method. As a first test case GMPM has been applied to a small gene expression model characterized by having only slow moving particles and thus diffusion limited reactions.

Algorithm 3.4: A short pseudo code description of the Next Sub-volume Method.

Parameters:
t (global time)
EQ (event queue)
Input:
RN (reaction network)
D (diffusion constants)
$\mathcal{C}$ (connectivity matrix)
$\mathbf{X}(0)$ (initial state)
$[t_{start}, t_{end}]$ (simulation interval)

$t \leftarrow t_{start}$;
for $l \in [1, L]$ **do** // initialization
  $a_l \equiv a_0^l(\mathbf{X})$;
  $d_l \equiv d_0^l(\mathbf{X}) \leftarrow 2n \sum_{i=1}^{N} D_i x_{l,i} / \lambda_{sv}^2$;
  $s_l \equiv s_0^l(\mathbf{X}) \leftarrow a_l + d_l$;
  store a_l, d_l, s_l;
  $\tau_l \leftarrow -\ln U(0,1)/s_l$;
  enqueue (τ_l, l) into EQ;
while $t < t_{end}$ **do**
  take $\mathbf{X}(t) = \mathbf{X}$ as current state;
  dequeue (τ, l) from EQ, get a_l, d_l, s_l;
  if $U(0,1) < a_l/(s_l)$ **then** // reaction event
    determine μ as smallest integer satisfying
      $\sum_{j=1}^{\mu} a_j^l(\mathbf{X})/a_l > U(0,1), \mu \in [1, M]$;
    $\mathbf{X}_l(t+\tau) \leftarrow \mathbf{X}_l + \mathbf{v}_\mu$;
    update: a_l, d_l, s_l;
    $\tau_l \leftarrow -\ln U(0,1)/s_l$;
    enqueue (τ_l, l) into EQ;
  else // diffusion event
    determine ι as smallest integer satisfying
      $\sum_{i=1}^{\iota} x_{l,i} D_i / (\lambda_{sv}^2 d_l) > U(0,1), \iota \in [1, N]$;
    sample index i from interval $[1, 2n]$, get neighbor $l' \leftarrow \mathcal{C}_{l,i}$;
    $\mathbf{X}_l(t+\tau) \leftarrow \mathbf{X}_l - \mathbf{1}_\iota^N$;
    $\mathbf{X}_{l'}(t+\tau) \leftarrow \mathbf{X}_{l'} + \mathbf{1}_\iota^N$;
    update: a_l, d_l, s_l, $a_{l'}$, $d_{l'}$, $s_{l'}$;
    $\tau_l \leftarrow -\ln U(0,1)/s_l$, $\tau_{l'} \leftarrow -\ln U(0,1)/s_{l'}$;
    enqueue (τ_l, l), requeue $(\tau_{l'}, l')$;

The larger second model is a pathway which is part of the glycolysis and involves 17 species that can interact via 20 reactions; the species vary in their diffusion speed and the set of reactions contains both slow and fast channels. Compared to microscopic and macroscopic simulations the algorithm has shown to yield good overall results.

GMPM seems to be an efficient alternative to the NSM, considering that it avoids calculating single diffusion events but processes them in a bulk. It will be interesting to see how this approximation affects the outcome of a simulation, especially how well it captures the spatial distribution of the particles.

It should be noted that GMPM, in contrast to the spatial τ-leaping algorithms discussed in the following chapter, does not have any *algorithm parameters* that can be varied to make it either faster or more accurate (cf. the parameter ϵ in Section 3.2.1). The only way a user can influence the execution of the algorithm is to change *model parameters*, e.g., making λ_{sv} smaller or increasing the diffusion constants lowers the interval between the diffusion phases. But this can be interpreted as using an entirely new model instead of performing a simulation of the same one with different simulation parameters. This should be kept in mind when analyzing the results presented in Chapter 6.

3.3.3 Parallel Variants

As was mentioned in Section 3.1.3, the idea of a parallel simulation is to distribute the work among available computational units; it is reasoned that if each processor has to take care only of a small part of the problem, then their concurrent execution should be faster than a sequential simulation of the entire model. The methods discussed in this sections demonstrated that it is possible to achieve a speed-up for non-spatial biochemical reaction systems by letting certain tasks, e.g., the recalculation of the reaction propensities, run in parallel (i.e., a "parallelization inside a simulation run"). However, the practical application of this concept to *spatial* systems seems much more difficult, as was shown, e.g., in [JPE+08]. The authors took the NSM (Section 3.3.1) and partitioned the collection of sub-volumes into disjoint subsets which then got assigned to the processors as work load. Thus each of the latter essentially executed the algorithm only for the sub-volumes in the set it was given to; it was expected that this would speed up the simulation, like as has been observed for the non-spatial case. But it turned out that the effort to *synchronize* all processors is significant — up to the point that a parallel execution was much slower than a sequential one. One reason for this is the

Algorithm 3.5: Pseudo code description for the Gillespie Multi-Particle Method (GMPM). Note that an update vector is used as temporary storage for the particles that diffuse into other sub-volumes. This is necessary, because these particles cannot be added to a neighbor's state unless the latter has been processed (otherwise the neighbor site would not use the current, but an already modified state vector).

Parameters:

t	(global time)
EQ	(event queue)
$SSA(RN, \mathbf{x}, t_{start}, t_{end})$	(an SSA variant, e.g., DM, FRM, or NRM)
$(k_1, \ldots, k_L)$	(update vector)
n	(dimension of the system)

Input:

RN	(reaction network)
D	(diffusion constants)
$\mathcal{C}$	(connectivity matrix)
λ_{sv}	(side length of a SV)
$\mathbf{X}(0)$	(initial state)
$[t_{start}, t_{end}]$	(simulation interval)

$t \leftarrow t_{start}$;
for $i \in [1, N]$ **do** `// initialization`
  $\tau \leftarrow \frac{\lambda_{sv}^2}{2nD_i}$;
  enqueue (τ, i) into EQ;
while $t < t_{end}$ **do**
  dequeue (τ, i) from EQ;
  for $l \in [1, L]$ **do**
    run $SSA(RN, \mathbf{X}_l, t, t + \tau)$ locally in SV l during $[t, t + \tau]$;
    for $k \in [1, 2n]$ **do** `// neighbor diffusion`
      get neighbor $l' \leftarrow \mathcal{C}_{l,k}$;
      sample number of S_i particles diffusing from l to l':
        $k \sim \text{Binom}(x_{l,i}, k/2n)$;
      update local state: $x_{l,i} \leftarrow x_{l,i} - k$;
      adapt update vector: $k_{l'} \leftarrow k_{l'} + k$;
  for $l \in [1, L]$ **do** `// diffusion processing`
    update state with incoming S_i particles stored in update vector:
      $\mathbf{X}_l(t + \tau) \leftarrow \mathbf{X}_l + \mathbf{k}_{\mathbf{l}i}^M$;
  $t \leftarrow t + \tau$;

stochastic nature of the problem: diffusions between sub-volumes take place randomly and require an update of both the source and target site. What is no problem between sub-volumes contained on the same subset becomes quite intricate if source and target are hosted by different processors. To see this it shall be remembered that, from a global perspective, the events (reactions and diffusion) have to be processed in the order of increasing time-stamps, otherwise a causality error would spoil the simulation results. Now causality is guaranteed for events processed within a processor — but, due to the concurrent execution, not necessarily for events that are sent between them, i.e., the information about diffusing particles. There are two ways to make sure that events are only processed with increasing time-stamps. The first alternative involves a negotiation between the units: to put it simply, all processors have to agree on a time interval during which it is safe to execute internal events without the risk of a "straggler" event arriving from another processor (a straggler is an event with a time-stamp smaller than the current time of the recipient). A prerequisite is that each unit knows when it will send out events next — a task which is impossible if the inter-event times are, e.g., an exponentially distributed random variable. Hence the just described *conservative* synchronization seems not applicable to reaction-diffusion systems.

The second scheme allows causality errors to occur if a processor can recover from them. Processors send each other events as soon as they occur; if a straggler arrives, then the execution stops, the state of the system right before the time-stamp of the straggler is restored (a so-called "rollback" is performed, e.g., by loading a previously stored state from memory or reverse-executing processed events until the correct time point is reached), the interrupting event executed, and the algorithm continues its work. In comparison to the conservative approach there is no negotiation required; however, if processors have to rollback frequently, then the cost for maintaining the causality may exceed the effort spent with the actual simulation and the overall performance deteriorates — which was one result discussed in [JPE+08]. But later a second publication showed that a speed-up is indeed possible [DM08]. The authors based their work also on the NSM and partitioned the sub-volume set similar to [JPE+08], but used custom-built low-level implementations for the synchronization and inter-thread communication. With these improvements they were able to decrease the run-time of a simple predator-prey model by factors between 1.22 and 8.06, depending on the model size.

3.4 Multi-X Methods

The term "multi-x" shall refer to a wide range of methods that operate on different levels of resolution (*multi-resolution*) or scales (*multi-scale*) simultaneously or utilize more than one technique to accomplish a task (*multi-algorithm*, also referred to as *hybrid algorithms*). These attributes are not necessarily mutually exclusive; for example, in some cases a method uses a particular algorithm to simulate parts of the system at a certain level of resolution, hence it is both multi-scale and multi-algorithm.

The motivation to build a multi-x method often stems from the observation that each method has its strengths and weaknesses; it is therefore reasoned that a combination of algorithms may compensate for the weaknesses and eventually perform better in some cases than a single one. Section 3.2.1 introduced one alternative how to handle stiff systems within the framework of the leap methods. Implicit leaping is able to dampen the fluctuations caused by fast reversible reactions which significantly restrict the maximum step size of explicit methods. While a stiff system is characterized by having reactions executed at different time scales, the leap methods still treat all reactions basically the same, regardless whether they are fast or slow. This is different for the slow scale stochastic simulation algorithm (ssSSA) [CGP05c]. It separates both species and reactions into fast and slow components. This is a two-step process: first R is partitioned into two sets, the first one containing all reactions having an expected next event time much smaller than each member of the second set. A species is then considered as "slow" if its state is only changed by slow reactions (from the second set) and it is "fast" when reactions from both sets can alter the state. This initial separation is later refined with respect to two essential requirements. The stochastic process which governs the evolution of the fast species (termed "virtual fast process" because the influence of slow reactions is neglected) must be stable in the sense that the probability of a certain configuration of the fast state variables at some time $t+\Delta t$, given an initial state at t, does not change if Δt approaches infinity. In other words: starting from an initial probability distribution for the fast state variables, after some time it converges to a "stable distribution", i.e., it ceases to change with increasing time. The second requirement is closely linked to the first one: the partition of R must be chosen such that the time until the distribution is stabilized (i.e., until the firing of the fast reactions does no longer change the estimation of the fast state variables) is much smaller than the expected time of the next slow reaction. If these two conditions can be fulfilled, then a multi-scale stochastic simulation algorithm can be constructed

which selects and executes the slow reactions similar to the DM and updates the fast variables using a sample from the converged virtual fast process.

What is not mentioned in the paper is if it is possible to use ssSSA in an adaptive manner. In the original publication the partition is done during the initialization and takes the initial state to separate slow and fast reactions and species. But some systems may undergo transitions from non-stiff to stiff dynamics; here it would be interesting to see if the proposed algorithm can be modified to allow a re-partitioning during execution.

A different direction was taken by Takahashi et al. [TKHT04] with their multi-algorithm, multi-timescale method implemented as part of the E-Cell simulation environment. They follow the "no silver bullet" philosophy, which means that algorithms have both strengths and weaknesses and it is unlikely that one method supersedes all others for all possible problem instances (see also Chapter 6). Central to the approach proposed by Takahashi et al. is a coordinator algorithm (which is called meta-algorithm in the publication) which coordinates the execution of *steppers*, independent algorithms that are responsible for updating a subset of the state variables. Steppers are classified into discrete-event, discrete-time, and continuous variants; the coordinator successively selects the stepper having the minimum next step time and executes it. If it is a continuous stepper, then the state of its variables is updated using a numerical integration algorithm which considers the contributions from all steppers that are also able to change these variables. A discrete-event stepper implements, e.g., one of the non-spatial SSA introduces previously and modifies its variables as usual by adding the state change vector of the selected reaction. After processing a stepper, the coordinator notifies all dependent modules to update internal parameters such as the next event time; the dependency is defined such that a stepper S_1 requires an update after S_2 has been selected if S_1 has read access to variables that can be modified by S_2.

This concept of a multi-algorithm simulation has proven to be faster than even a pure deterministic treatment. One remarkable advantage is the modularity: new stepper modules can be added easily and different variants exchanged without difficulty — a point which will be taken up again in a later chapter.

These were just two examples of non-spatial multi-x methods, more can be found in the literature [WLV07, MBS08]; but research in this direction is also progressing for reaction-diffusion systems. Kalantzis [Kal09] presented a hybrid simulation algorithm that partitions events (reaction and diffusion) into fast and slow subsets and updates

the species population using both PDEs for the fast and SSA for the slow events. Bayati et al. [BCK08], developed a multi-resolution spatial algorithm that supports sub-volumes of inhomogeneous size. Up to now the model volume has been discretized into sub-volumes with a fixed side length λ_{sv}. This way regions that may be of lesser interest, e.g., inside the cytosol, are represented with the same spatial resolution as, e.g., the area near a membrane where finer spatial details could be more important. The algorithm presented by Bayati et al. now allows to selectively modify the spatial resolution: the discretization is higher near membranes and other areas where inhomogeneity is assumed to play an important role, but can be significantly lower elsewhere — which also reduces the effort for the simulation. Changing the grid resolution causes disparities in both the reaction propensities and diffusion rates, an important aspect also discussed and quantitatively analyzed in the paper. A similar technique, but with a different motivation, will be used in Chapter 7.

The multi-x methods mentioned in this section have been specifically developed for the use with reaction networks or reaction-diffusion systems and represent only a very small subset of the work that is done in this direction. However, they should give a first impression that there are further alternatives to the algorithms given in previous sections for the simulation of biochemical reaction systems.

3.5 Summary

The survey presented in this chapter is by far not complete, but if it was able to invoke the impression that there are a lot of choices and alternatives when it comes to the stochastic simulation of reaction networks, then it achieved its main goal.

The DM and FRM from Chapter 2 can be seen as the basic algorithms from which several optimized variants have originated. A central addition is the reaction dependency graph, which nowadays can be found in the majority of algorithms; it limits the number of propensity updates after a reaction has fired. Another potential bottleneck has been identified and tackled by several authors: the reaction selection process. As this is basically a search problem, the knowledge from this field has been integrated into the algorithm development.

To counteract the performance loss of exact algorithms (which simulate each event independently) in case of reactions with high propensities, a new strain of methods have emerged: the leap algorithms (Section 3.2). The most prominent member is τ-leaping: a time interval constrained to the leap condition, which requires the changes

in the propensities during this interval to be bounded by an error control parameter, is determined and then used to calculate how often each reaction fired during the jump. Some variants restrict the number of firings instead of the leap size; others have been specifically developed to handle stiff systems.

Just like the algorithms for well-stirred systems, spatial methods have also evolved over the last years (Section 3.3). One of the best known variants is the exact Next Sub-volume Method used in the MesoRD tool [HFE05], which stores the sub-volumes according to their next event time into an event queue and subsequently processes the sites where the next event (a diffusion or reaction) takes place. As an alternative, the Gillespie's Multi-Particle Method, an approximative operator-split algorithm, avoids single diffusion events and distributes the particles of a species in predetermined time intervals; between each diffusion phase a non-spatial SSA runs locally in each sub-volume and takes care of the reactions.

Yet another family of algorithms is added if parallel methods are considered as well (Sections 3.1.3 and 3.3.3); they utilize more than one computational resource to accomplish the simulation task, which seems a worthwhile approach given that the number of processors per chip is steadily increasing. Two techniques have been introduced: "parallelization inside a simulation run" and "parallelization across the simulation": in the former case the individual processors have to send and receive event messages frequently, e.g., when a particle diffuses between two sub-volumes that are hosted by different workers. In the latter type the workers need to communicate much less or even not at all; some tools, e.g., *JAMES II*, offer the distribution of an ensemble of replications among the computational units, with each unit then running a sequential variant of the simulation algorithm to generate one trajectory. Different publications report speed-ups between 2 and 8 for the first type and up to 200 for the second type.

The last section in this chapter was dedicated to multi-x approaches, i.e., methods that operate on multiple scales or resolutions or use more than one algorithm to simulate a model. They follow the "separation of concerns" principle: for example, fast reactions are treated differently than slow ones to decrease the overall run time of the simulation. Most often some kind of coordinator is used to steer the execution of distinct methods, each responsible for a specific part of the model.

4 The Spatial τ-leaping Algorithm

The topic of approximative spatial stochastic simulation algorithms was intentionally left out in the survey of the last chapter. Non-spatial leap methods avoid executing each reaction individually; if the leap condition is fulfilled for some time interval τ, then the number of times each reaction fires during τ can be sampled from a Poisson random variable. As long as a single leap comprises enough reaction events the execution can be much faster than an exact variant (a more in-depth analyze of the performance of τ-leaping will follow in a later chapter).

In this chapter the leap principle is taken into the spatial realm. It is first motivated why a step in this direction could be promising; being not the first attempt, already existing approaches will be discussed as well. The main part of this chapter is then dedicated to the derivation and analysis of the spatial τ-leaping algorithm.

4.1 Problem Statement

All exact stochastic simulation algorithms, regardless whether a variant considers space or not, simulate each event individually, one at a time. As already defined earlier, for a model with L sub-volumes and N species, the $L \times N$ matrix

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,N} \\ x_{2,1} & x_{2,2} & \dots & x_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{L,1} & x_{L,2} & \dots & x_{L,N} \end{pmatrix} \tag{4.1}$$

denotes the state of the model and $\mathbf{X}(t)$ the stochastic process governing its evolution over time. With $\mathbf{X}(t) = \mathbf{X}$ being the state at time t, the entries of the row vector $\mathbf{X}_l \equiv \mathbf{X}[l; 1, \dots, N]$ are the numbers of molecules currently present in sub-volume l. In general, the average time interval between two events is the inverse of $\sum_{l=1}^{L}(a_0^l(\mathbf{X}) + d_0^l(\mathbf{X}))$; each term sums up the total propensity and diffusion rate over all reactions and

species (the second rate quantity is set to zero for non-spatial algorithms having only a single sub-volume). Due to this inverse relationship, the time interval can become very small if the sum increases. Returning to the bottom, the propensity and diffusion rate functions are defined as

$$a_j^l(\mathbf{X}) = c_j H_j^l(\mathbf{X}) = c_j \prod_{i=1}^{N} \binom{x_{l,i}}{v_{ji}} \tag{4.2}$$

$$d_i^l(\mathbf{X}) = 2n \frac{D_i}{\lambda_{sv}^2} x_{l,i}. \tag{4.3}$$

In terms of probability, $c_j dt$ gives the chance for any of the $H_j^l(\mathbf{X})$ independent trials to have a positive outcome. Higher values for this constant therefore make it more likely to observe a "hit", i.e., a reaction in this case, when testing each of the $H_j^l(\mathbf{X})$ reactant groups. On the other hand, given a fixed c_j the same effect can also be achieved by increasing the number of trials, which means nothing more than adding reactant particles to the system. A similar argumentation line can be given for the diffusion equation. Note that here the probability $2nD_i dt/\lambda_{sv}^2$ that a particle will leave a sub-volume within the next dt depends on two constants: the diffusion "speed" D_i and the side length λ_{sv} of the sub-volume; making the particles faster or the sub-volume smaller both increases the chance that a particle will diffuse away. Figure 4.1 gives an example how model parameters influence the size of the time step calculated by the non-spatial Direct Reaction method. Raising the population from initially 100 to 10000 particles decreases the average and median time intervals approximately by the same order of magnitude; the minimum interval is even lowered by a factor of 10^5 in the dense case.

Just like the DM, the NSM is an exact spatial stochastic simulation algorithm and its execution speed thus suffers if any of the above parameters increases significantly. Things look even worse for models which include, e.g., not only some ubiquitous species present in large amounts, but whose diffusion constants are high as well; it is very likely that most of the time required to simulate those types of models is spent on propagating particles of less interesting species.

There are several alternatives available, developed with a similar motivation in mind that lead to the development of the non-spatial τ-leaping algorithm (Section 3.2.1). The GMPM looks promising, though concerns may be expressed about the spatial accuracy, i.e., how well it captures the distribution of the particles in space. Particle displacement is an all-or-nothing step: in each diffusion phase all particles of a sub-volume are distributed among its neighbors, leaving it empty if it does not receive new

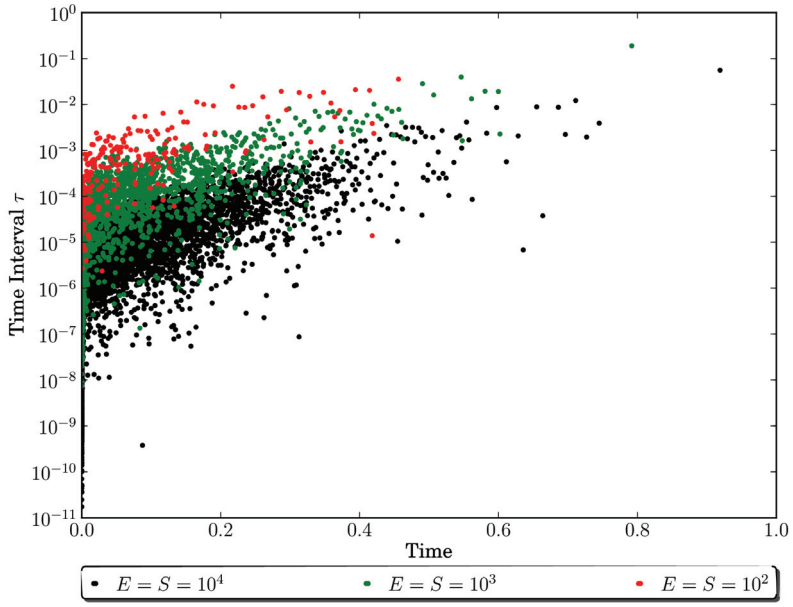


Figure 4.1: Time steps calculated by the DM for different parametrizations of the enzyme-substrate model. Red (green, black) dots show the interval values for a single run of a model with initially 100 (1000, 10000) E and S particles present. The smallest, largest, average and median time intervals for each parameter set are: $(2.35 \times 10^{-6}, 0.035, 1.982 \times 10^{-3}, 5.5 \times 10^{-4})$ for the 100 particle model, $(7.4 \times 10^{-9}, 0.19, 3.6 \times 10^{-4}, 2.9 \times 10^{-5})$ for the 1000 particle model, and $(1.1 \times 10^{-11}, 0.12, 4.3 \times 10^{-5}, 2.0 \times 10^{-6})$ for the 10000 particle model.

entities — which occurs when all neighbors did not contain any own particles; this scenario will be further discussed in the next chapter.

Another approximative method is the binomial spatial τ-leaping algorithm (BStau, [MLB07]). Based on the NSM, it allows more than one event per iteration; BStau is essentially a local binomial τ-leaping, calculating for each sub-volume an individual leap value. Among those the minimum is taken as the time increment for the current iteration. Using the τ-leaping approach, it is determined for the sub-volume having the minimum time stamp how often each reaction fires and how many particles diffuse into neighboring sub-volumes during this interval. But calculating the leap value does not consider particles diffusing *into* a sub-volume; clearly being an additional approximation, it is not stated if this has a noticeable impact on the accuracy.

In a later publication Rossinelli et al. [RBK08] also presented a spatial leap algorithm, this time based on the modified τ-leaping algorithm by Cao et al. [CGP06]. While accounting for incoming particles, the authors 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. [IHC10] which introduced a spatial τ-leaping variant very similar to the one that will be derived below.

Other authors also used the leap method to simulate spatially inhomogeneous systems, e.g., [Vla08, CV06]. An interesting hybrid algorithm was proposed by Ferm et al. [FHL10], 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. Like GMPM, it is also based on operator splitting, however, it differs from the former in that the time increment τ is calculated at each iteration instead of doing this only during initialization for each species.

What follows in the next section is a detailed derivation of the spatial τ-leaping algorithm (Sτ).

4.2 Derivation

For simplicity, the introduction to the spatial τ-leaping algorithm follows closely the steps made by Gillespie to derive the original τ-leaping idea [Gil01]. Before starting,

some basic preparations have to be made. A new reaction set R' is defined as

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

The additional reactions represent the diffusion of species into neighboring sub-volumes, with $c_i = 2nD_i/\lambda_{sv}^2$; their propensity is calculated similar to unimolecular reactions, i.e., $a^l_{M+i}(\mathbf{X}) = d^l_i(\mathbf{X}) = c_i x_i$. Each new $R_{M+i} \in R'$ has a state change vector $\mathbf{v}_{M+i} = -\mathbf{1}^N_i$, i.e., the N-dimensional vector with $v_{M+i,i} = -1$ and $v_{M+i,j} = 0, j \neq i$.

What was only mentioned briefly in Section 3.2.1 shall now be elaborated in more detail: how to find a value for the leap τ without violating the leap condition. As a reminder, the latter restricts the size of τ to be small enough that the propensity change $|\Delta_\tau a^l_j(\mathbf{X})|$ of each reaction is insignificant during the leap; expressed more formally, the leap condition states that

$$|\Delta_\tau a^l_j(\mathbf{X})| \leq \max\{\epsilon a^l_j(\mathbf{X}), c_j\}, j \in [1, M+N]. \tag{4.4}$$

Note that the threshold is based on the individual propensities, a necessary modification of the original condition ensuring that reactions having a small propensity do not fire more often than actually intended [CGP06]; the constant c_j is a lower bound on the change of each propensity function and avoids that $\Delta_\tau a^l_j(\mathbf{X})$ (and, as a consequence, also τ^l_j) approaches zero if $a^l_j(\mathbf{X}) \to 0$.

Due to its dependence on the state, the propensity of R_j changes whenever the population of its reactants is modified; so given $\mathbf{X}$, an expression for the left hand side of Equation 4.4 is found by taking the difference between the current propensity and the updated one after the leap:

$$\begin{aligned} \Delta_\tau a^l_j(\mathbf{X}) \triangleq & |a^l_j(\mathbf{X} + \mathbf{\Lambda}(\tau; \mathbf{X})) - a^l_j(\mathbf{X})| \\ \approx & \operatorname{tr}(\nabla a^l_j(\mathbf{X})^T \mathbf{\Lambda}(\tau; \mathbf{X})) = \sum_{l'=1}^{L} \sum_{i=1}^{N} \Delta_\tau x_{l',i} \frac{\partial a^l_j(\mathbf{X})}{\partial x_{l',i}}; \end{aligned} \tag{4.5}$$

$\Delta_\tau x_{l,i}$ is the entry in $\mathbf{\Lambda}(\tau; \mathbf{X})$ at position (l, i) and the second line is a result of a Taylor expansion around $\mathbf{X}$. The matrix $\mathbf{\Lambda}(\tau; \mathbf{X})$ represents the state change from $\mathbf{X}$ to $\mathbf{X}(t+\tau)$; however, only the propensity change $\Delta_\tau a^l_j(\mathbf{X})$ is of interest, so $\partial a^l_j(\mathbf{X})/\partial x_{l',i}$ is zero for all $l' \neq l$ and thus only the l-th row vector of $\mathbf{\Lambda}(\tau; \mathbf{X})$, denoted by $\mathbf{\Lambda}_l(\tau; \mathbf{X})$, needs to be considered, which gives the change in the population of sub-volume l during

the leap.

In the original non-spatial τ-leaping algorithm, $\mathbf{\Lambda}_l(\tau; \mathbf{X})$ only depends on how often each reaction fired during τ:

$$\mathbf{\Lambda}_l(\tau; \mathbf{X}) = \sum_{j=1}^{M} K_j^l(\tau; \mathbf{X}) \mathbf{v}_j. \tag{4.6}$$

But with reaction-diffusion systems the state can now also be changed by diffusing particles that either leave or enter l. While the first case is simply captured by replacing R with R', the second requires an additional vector representing how many particles of each species diffuse from l's neighbors into it. Thus Equation 4.6 is replaced with

$$\mathbf{\Lambda}_l(\tau; \mathbf{X}) = \sum_{j=1}^{M+N} K_j^l(\tau; \mathbf{X}) \mathbf{v}_j + \sum_{i=1}^{N} I_i^l(\tau; \mathbf{X}) \mathbf{1}_i^N. \tag{4.7}$$

The difference to the original τ-leaping state update is the additional term: it sums up the changes made by particles diffusing into SV l. Similar to the non-spatial τ-leaping algorithm, if a τ satisfies the leap condition, the number of firings for an event $R_j \in R'$ inside l can be approximated by a Poisson random variable $\text{Pois}(a_j^l(\mathbf{X}), \tau)$ with mean and variance $a_j^l(\mathbf{X})\tau$, so

$$K_j^l(\tau; \mathbf{X}) \sim \text{Pois}(a_j^l(\mathbf{X}), \tau). \tag{4.8}$$

However, knowing how to deal with the first sum is just half the truth; it still needs to be shown how to approximate the second function $I_i^l(\tau; \mathbf{X})$, i.e., the number of particles diffusing into l. As a first step, an *incoming diffusion rate* $\beta_i^l(\mathbf{X})$ shall be defined as

$$\beta_i^l(\mathbf{X}) = \frac{1}{2n} \sum_{k=1}^{2n} a_{M+i}^{\mathbf{C}_{l,k}}(\mathbf{X}).$$

Notice the factor $1/2n$: the diffusion rate $a_{M+i}^{\mathbf{C}_{l,k}}(\mathbf{X})$ for S_i in sub-volume k is calculated over all possible sides a particle can leave k ($2n$ for a spatial dimension of n), but now only the direction towards l is of interest.

If the leap condition is satisfied for both the reaction propensities *and* the outgoing diffusion rates for all neighbors of l, then the number of incoming particles can also be

approximated with a Poisson random variable with mean and variance $\beta_i^l(\mathbf{X})\tau$:

$$\begin{aligned} I_i^l(\tau;\mathbf{X}) &= \sum_{k=1}^{2n} \text{Pois}(\frac{1}{2n} a_{M+i}^{\mathbf{C}_{l,k}}(\mathbf{X}), \tau) \\ &\sim \text{Pois}(\frac{1}{2n} \sum_{k=1}^{2n} a_{M+i}^{\mathbf{C}_{l,k}}(\mathbf{X}), \tau) \\ &\sim \text{Pois}(\beta_i^l(\mathbf{X}), \tau); \end{aligned} \tag{4.9}$$

the transition from the first to the second line applies a well-known property of Poisson random variables: the sum of N Poisson RV, each one having a rate parameter λ_i, is again a RV that also follows a Poisson distribution with $\lambda = \sum_{i=1}^{N} \lambda_i$. What is done here is basically to 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 $\text{Pois}(a_{M+i}^l(\mathbf{X}), \tau)$ gives the number of S_i particles that will *diffuse out* of l (see Equation 4.8) during τ and each of the $2n$ neighbors of l has the same probability of being selected as a target, then sampling from $\text{Pois}(a_{M+i}^l(\mathbf{X})/2n, \tau)$ gives the amount of particles that will *enter* one specific neighboring site. However, the algorithm samples only from $\text{Pois}(a_{M+i}^l(\mathbf{X}), \tau)$; then, for each particle that leaves l a target neighbor is selected from $\mathcal{C}_l$ with a point probability of $1/2n$.

Returning to Equation 4.7, inserting Equations 4.8 and 4.9 and defining the vector $\mathbf{b}$ as

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

gives

$$\mathbf{\Lambda}_l(\tau;\mathbf{X}) = \sum_{j=1}^{M+N} \text{Pois}(a_j^l(\mathbf{X}), \tau)\mathbf{v}_j + \mathbf{b}. \tag{4.10}$$

Now putting this result into Equation 4.5 provides an approximation for the propensity change, given a leap value τ:

$$\Delta_\tau a_j^l(\mathbf{X}) \approx \sum_{i=1}^{N} \frac{\partial a_j^l(\mathbf{X})}{\partial x_{l,i}} \left(\sum_{j'=1}^{M+N} \text{Pois}(a_{j'}^l(\mathbf{X}), \tau) v_{ij'} + b_i \right). \tag{4.11}$$

Writing the propensity change this way makes it very easy to explain what is calculated here: at first it is determined how the propensity depends on species S_i. If S_i is not a reactant of R_j, then $\partial a_j^l(\mathbf{X})/\partial x_{l,i}$ is zero and is it not necessary to look at the part

inside the brackets. On the other hand, if reaction R_j depends on S_i, then the state change of the latter during τ is estimated — 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.

Equation 4.11 does not look very useful: still depending on random variables, how to constrain this expression according to the leap condition from Equation 4.4? The answer is to relax the condition and only require the mean and the variance of the expected change $\Delta_\tau a_j^l(\mathbf{X})$ to be bounded by $\max\{\epsilon a_j^l(\mathbf{X}), c_j\}$:

$$\left\langle \Delta_\tau a_j^l(\mathbf{X}) \right\rangle \leq \max\{\epsilon a_j^l(\mathbf{X}), c_j\} \wedge \operatorname{var}\left\{\Delta_\tau a_j^l(\mathbf{X})\right\} \leq \max\{\epsilon a_j^l(\mathbf{X}), c_j\}^2, j \in [1, M+N]. \tag{4.12}$$

So all that is left to do is to substitute the random variables in Equation 4.11 with their respective means and variances — which is very convenient for Poisson RV, as $\langle \mathrm{Pois}(\lambda, \tau) \rangle = \operatorname{var}\{\mathrm{Pois}(\lambda, \tau)\} = \lambda\tau$.

$$\begin{aligned} \left\langle \Delta_\tau a_j^l(\mathbf{X}) \right\rangle &\approx \sum_{i=1}^{N} \frac{\partial a_j^l(\mathbf{X})}{\partial x_{l,i}} \left(\sum_{j'=1}^{M+N} \left\langle \mathrm{Pois}(a_{j'}^l(\mathbf{X}), \tau) \right\rangle v_{ij'} + \langle b_i \rangle \right) \\ &\approx \sum_{i=1}^{N} \frac{\partial a_j^l(\mathbf{X})}{\partial x_{l,i}} \left(\sum_{j'=1}^{M+N} a_{j'}^l(\mathbf{X}) \tau v_{ij'} + \beta_i^l(\mathbf{X}) \tau \right) \equiv \mu_j^l(\mathbf{X})\, \tau \end{aligned} \tag{4.13}$$

$$\begin{aligned} \operatorname{var}\left\{\Delta_\tau a_j^l(\mathbf{X})\right\} &\approx \sum_{i=1}^{N} \left(\frac{\partial a_j^l(\mathbf{X})}{\partial x_{l,i}} \right)^2 \left(\sum_{j'=1}^{M+N} \operatorname{var}\left\{\mathrm{Pois}(a_{j'}^l(\mathbf{X}), \tau)\right\} v_{ij'}^2 + \operatorname{var}\{b_i\} \right) \\ &\approx \sum_{i=1}^{N} \left(\frac{\partial a_j^l(\mathbf{X})}{\partial x_{l,i}} \right)^2 \left(\sum_{j'=1}^{M+N} a_{j'}^l(\mathbf{X}) \tau v_{ij'}^2 + \beta_i^l(\mathbf{X}) \tau \right) \equiv \left(\sigma_j^l(\mathbf{X})\right)^2 \tau \end{aligned} \tag{4.14}$$

Finally, inserting $\mu_j^l(\mathbf{X})\,\tau$ and $\left(\sigma_j^l(\mathbf{X})\right)^2 \tau$ into Equation 4.12, dividing by $\mu_j^l(\mathbf{X})$ and $\left(\sigma_j^l(\mathbf{x})\right)^2$, respectively, and taking the minimum of both gives a single τ candidate for R_j in sub-volume l. Doing this for every reaction in all sub-volumes eventually results in the leap value:

$$\tau = \min_{l \in [1,L]} \left\{ \min_{j \in [1, M+N]} \left\{ \frac{\max\{\epsilon a_j^l(\mathbf{X}), c_j\}}{\left|\mu_j^l(\mathbf{X})\right|}, \frac{\max\{\epsilon a_j^l(\mathbf{X}), c_j\}^2}{\left(\sigma_j^l(\mathbf{X})\right)^2} \right\} \right\}. \tag{4.15}$$

The derivation so far represents the core of the algorithm, i.e., the steps performed in the *CalculateNonCriticalTau*() method of Algorithm 4.1; an actual implementation would include, e.g., further optimizations or modifications such as replacing the Poisson with binomial random variables or re-formulating the leap condition to bound the allowed change in the species concentration. The latter shall be discussed briefly because the spatial τ-leaping algorithm used in the forthcoming performance study (Chapter 6) is based on it.

Each propensity function depends on the number of reactant particles present at the current time point, a fact which is, loosely speaking, exploited by the alternative τ selection procedure: instead of focusing on the dependent variable, i.e., the propensity functions in Equation 4.4, it should also be possible to restrict the change in the argument, i.e., the state $\mathbf{X}$. Or, more formally, find a leap value for sub-volume l subject to the condition:

$$\Delta_\tau x_{l,i} \leq \max\{\epsilon_i x_{l,i}, 1\}, \forall i \in [1, N], \forall l \in [1, L] \tag{4.16}$$

that is also a valid choice under the original leap condition. Notice the individual error control parameters ϵ_i: a reaction can have more than one reactant species, but its propensity change $\Delta_\tau a_j^l(\mathbf{X})$ still has to be bounded by $\max\{\epsilon a_j^l(\mathbf{X}), c_j\}$. Finding a general expression for ϵ_i is difficult, but approximations can be given by analyzing the fundamental reactions of order up to three [CGP06]. It turned out that setting the error parameters to $\epsilon / g_i^l(\mathbf{X})$ is sufficient to ensure that the leap condition still holds, with

$$g_i^l(\mathbf{X}) = h(i) + \frac{h(i)}{n(i)} \sum_{j=1}^{n(i)-1} \frac{j}{x_{l,i} - j}. \text{ [San09b]} \tag{4.17}$$

The values for $h(i)$ and $n(i)$ are taken over the set $R^i \subseteq R$ of all reactions having S_i as a reactant: $h(i)$ denotes the highest order of any reaction inside R^i and $n(i)$ the maximum v_{ji} over all $R_j \in R^i$ with order $h(i)$. Having found ϵ_i, Equation 4.10 provides an approximation for the left hand part of Equation 4.16. As before, the mean and variances should be bounded, so:

$$\begin{aligned} \langle \Delta_\tau x_{l,i} \rangle &\approx \sum_{j=1}^{M+N} \left\langle \text{Pois}(a_j^l(\mathbf{X}), \tau) \right\rangle v_{ij} + \langle b_i \rangle \\ &\approx \sum_{j=1}^{M+N} a_{j'}^l(\mathbf{X}) \tau v_{ij} + \beta_i^l(\mathbf{X}) \tau \equiv \mu_i^l(\mathbf{X})\, \tau \end{aligned} \tag{4.18}$$

$$\begin{aligned} \operatorname{var}\{\Delta_\tau x_{l,i})\} &\approx \sum_{j=1}^{M+N} \operatorname{var}\left\{\operatorname{Pois}(a_j^l(\mathbf{X}),\tau)\right\}(v_{ij})^2 + \operatorname{var}\{b_i\} \\ &\approx \sum_{j=1}^{M+N} a_j^l(\mathbf{X})\tau\,(v_{ij})^2 + \beta_i^l(\mathbf{X})\tau \equiv \left(\sigma_i^l(\mathbf{X})\right)^2\tau \end{aligned} \tag{4.19}$$

Compared to Equation 4.14, it is much easier to calculate the required values for the τ selection procedures, which changes slightly to

$$\tau = \min_{l\in[1,L}\left\{\min_{i\in[1,N]}\left\{\frac{\max\left\{\epsilon x_{l,i}/g_i^l(\mathbf{X}),1\right\}}{\left|\mu_i^l(\mathbf{X})\right|}, \frac{\max\left\{\epsilon x_{l,i}/g_i^l(\mathbf{X}),1\right\}^2}{\left(\sigma_i^l(\mathbf{X})\right)^2}\right\}\right\} \tag{4.20}$$

4.3 Analysis

This section provides a more detailed look at implementation issues and internals of the spatial τ-leaping algorithm. Before testing the method "in the field", i.e., conducting an empirical performance analysis (see Chapter 6) for several model instances, it could be worthwhile to first identify potential performance bottlenecks, both from an implementation as well as a theoretical point of view.

Algorithm 4.1 summarizes the main steps performed by Sτ, including the check for critical reactions to avoid negative populations. It is a straightforward representation: almost the entire derivation done in the previous section is condensed into the single call to $CalculateNonCriticalTau(\mathbf{X}_l, I_{nc}^l, \epsilon)$ (line 6), with $\mathbf{X}_l$ denoting the row vector holding the current state of sub-volume l, I_{nc}^l as the set of indices for non-critical events (reactions and outgoing diffusions), and ϵ providing the upper bound in the relative propensity changes $\Delta_\tau a_j^l(\mathbf{X})/a_j^l(\mathbf{X}), j \in I_{nc}^l$. The remaining operations essentially first write the state changes (i.e., the calculated entries of $\mathbf{\Lambda}(\tau;\mathbf{x})$) to a *temporary* storage and eventually apply them to the actual state matrix. This is necessary due to the sequential iteration over all sub-volumes and the occurrence of diffusion events. Let l be the current sub-volume for which lines 15 to 24 are about to be executed. It is not possible to update the state vector of a sub-volume k selected as the target for a diffusion event from l directly because this (new) state would be used in k's iteration to determine how often each reaction will fire and how many particles will leave k during τ — an approximation based on the wrong state.

It could happen that a state is driven negative by too many reaction firings (just as

Algorithm 4.1: Pseudo-code description for the spatial τ-leaping algorithm. This represents a short and condensed version, though it already considers critical and non-critical reactions to avoid negative populations; some additional details can be found in the τ-leaping algorithm (Algorithm 3.3).

Parameters:
$\epsilon \in \mathbb{R}$ (leap condition parameter) $N_{SSA} \in \mathbb{N}$ (number of total SSA iterations)
$\gamma \in \mathbb{R}$ (SSA threshold parameter)

Input:
RN (reaction network) $\hat{\mathbf{X}}$ (update matrix)
$\mathbf{C}$ (connectivity matrix) $[t_{start}, t_{end}]$ (simulation interval)
$\mathbf{X}(0)$ (initial state matrix)

1 $t \leftarrow t_{start}$;
2 **while** $t < t_{end}$ **do**
3 **for** $l \in [1, L]$ **do** `// calculate τ candidates`
4 update propensities;
5 get critical and non-critical reaction sets: I_c^l, I_{nc}^l;
6 $\tau_{nc}^l \leftarrow CalculateNonCriticalTau(\mathbf{X}, I_{nc}^l, \epsilon)$;
7 $\tau_c^l \leftarrow -ln(U(0,1))/\sum_{j \in I_c^l} a_j^l(\mathbf{X})$;
8 $\tau_{nc} \leftarrow \min_{l \in [1,L]}\{\tau_{nc}^l\}$;
9 $\tau_c \leftarrow \min_{l \in [1,L]}\{\tau_c^l\}$;
10 $l' \leftarrow \text{argmin}_{l \in [1,L]}\{\tau_c^l\}$;
11 **if** $\tau_{nc} < \gamma / \sum_{l=1}^{L} \sum_{j \in M+N} a_j^l(\mathbf{X})$ **then**
12 perform N_{SSA} NSM iterations;
13 continue with next iteration;
14 $\tau \leftarrow \min\{\tau_{nc}, \tau_c\}$;
15 **for** $l \in [1, L]$ **do** `// calculate update matrix`
16 **if** $\tau = \tau_c \wedge l = l'$ **then**
17 sample reaction index μ from I_c^l;
18 **if** $\mu > M$ **then**
19 diffusion: update particle counts in $\hat{\mathbf{X}}_l$ and random neighbor $\hat{\mathbf{X}}_k, k \in \mathbf{C}_l$;
20 **else**
21 reaction: update particle count in $\hat{\mathbf{X}}_l$;
22 generate samples $s_j \leftarrow \text{Pois}(a_j^l(\mathbf{X}), \tau), j \in I_{nc}^l$;
23 diffuse s_j particles to neighbors $k \in \mathbf{C}_l$ for all $j > M, j \in I_{nc}^l$;
24 execute events: update particle counts in $\hat{\mathbf{X}}_l$ and $\hat{\mathbf{X}}_k, k \in \mathbf{C}_l$;
25 **for** $l \in [1, L]$ **do** `// apply update matrix`
26 $\mathbf{X}_l(t + \tau) \leftarrow \mathbf{X}_l + \hat{\mathbf{X}}_l$;
27 **if** $\exists x_{l,i} < 0, i \in [1, N]$ **then**
28 undo changes for $l^* \in [1, l]$;
29 $\tau_{nc} \leftarrow \tau_{nc}/2$;
30 goto line 11;
31 $t \leftarrow t + \tau$;

in the non-spatial variant, cf. Section 3.2.1). In this case the algorithm needs to undo the changes made in line 26 for all already processed sub-volumes, e.g., by subtracting the $\hat{\mathbf{X}}_l$ from the respective states.

However, as it is very cost-intensive to iterate over all sub-volumes three times (as it is done in Algorithm 4.1), a first optimization could be targeted at reducing the number of necessary loops. One alternative variant of the above algorithm relocates the update process from the third loop into the first one. Before calculating a τ candidate for sub-volume l, the respective entries from the update matrix are applied to the state of l *and* any neighbor that has not been updated so far. Going back to Algorithm 4.1, the last loop can be removed and the following additional steps get added after line 3:

```
1 if l needs update then
2     X_l(t + τ) ← X_l + X̂_l;
3     update propensities;
4     mark l as updated;
5 for k ∈ C_l do
6     if k needs update then
7         X_k(t + τ) ← X_k + X̂_k;
8         update propensities;
9         mark k as updated;
```

The τ value is the result from the *previous* iteration. Modifying the original algorithm this way entirely avoids one loop, but it has to be considered that while the current time has been updated at the end of an iteration, the state matrix has not; the system's state at $t + \tau$ is therefore only valid *after* finishing the first loop. But this should be no problem in practice as it is just a matter of adapting the condition when to read out the current state, e.g., for visualization purposes. Just like in Algorithm 4.1, it is also necessary to revert the changes if during the update any state variable is driven negative.

Another technique incrementally makes copies of the state vectors and is based on two observations. First of all, the *current* state is required for calculating the state change values, unaffected by any incoming particles. Algorithm 4.1 ensures this condition by keeping the entire state matrix untouched within the second loop; instead, it writes the results into a temporary structure. Secondly, each sub-volume l has $2n$ neighbors,

whose indices are stored in the connectivity vector $\mathbf{C}_l$; particles from l can only reach those adjacent sites during τ, any other of the $L-2n$ sub-volumes remains untouched. As a direct consequence thereof, l can only be selected as the target for a diffusion event by any of its $2n$ neighbors — in other words, the possible "interactions" of l with other sub-volumes are limited to a very small subset of $[1, L]$.

Now linking both observations gives rise to a modified update processes. Figure 4.2 depicts a small, two-dimensional 5×5 grid with the sub-volumes labeled column-wise; this image shall help to demonstrate the approach, which operates according to the following scheme:

y \ x	0	1	2	3	4
0	1	6	11	16	21
1	2	7	12	17	22
2	3	8	13	18	23
3	4	9	14	19	24
4	5	10	15	20	25

Figure 4.2: A simple 5×5 grid.

Initialization: Start with sub-volume $l_0 = 1$, store the current states of all its neighbors inside an auxiliary set C and perform steps 16 to 23 from Algorithm 4.1. Notice that a neighbor's state is allowed to be modified during those steps because the original population has been stored in C.

Iteration: Proceed with sub-volume $l_{i+1} = l_i + 1$ and store the state of a neighbor $l^* \in \mathbf{C}_{l_{i+1}}$ in C *only* if $l^* > l_{i+1}$ (all sub-volumes with a lower id have been already processed and their states can be updated without risk). Then take (but not remove) the stored state $\mathbf{X}_{l_{i+1}}$ from C and use it to perform steps 16 to 23 from Algorithm 4.1.

As an example, the initial set C for the 5×5 grid is $\{\mathbf{X}_2, \mathbf{X}_5, \mathbf{X}_6, \mathbf{X}_{21}\}$. After determining how many particles left or reacted inside sub-volume 1, the counter increases to $l_1 = 2$ and the neighbor states of the corresponding sub-volume are stored, i.e., $C \cup \{\mathbf{X}_3, \mathbf{X}_7, \mathbf{X}_{22}\}$. Now the stored state $\mathbf{X}_2$ can be taken as an input for calculating the number of reaction firings and outgoing diffusion events. Starting with the third column, the first sub-volumes can recalculate their propensities: after 11 has been processed, all neighbors of sub-volume 6 have calculated the state updates, so 6 is now allowed to update its propensities and check whether there are negative state entries; if yes, then the process has to be aborted and the original states restored using the entries from C. Note that it is also possible to perform these steps at the beginning of a new iteration.

Timing studies showed, at least for the implemented variants and models under study, that the overall execution time is nearly unaffected by the choice of the update

method; using an update matrix turned out to be slightly faster than making copies of the state vectors $\mathbf{X}_l$, so this technique has been used for the study in Chapter 6.

Apart from possible optimizations, it also could be interesting to see how the basic algorithm behaves when faced with different problem instances. However, before delving into the analysis it must be clear what questions can be answered with a pure theoretical approach. The derivation made in the previous section ultimately resulted in an estimation for a value τ compliant with the leap condition. Given a simple model, this value can be calculated easily and compared with the average step size made by the Next Sub-volume Method — which gives a first (yet rough) impression about the attainable speedup. To see how this can be useful, Algorithm 4.1 shall be again used as the starting point. Lines 11 to 13 read:

```
1 if τ_nc < γ / Σ_{l=1}^{L} Σ_{j∈M+N} a_j^l(X) then
2     perform N_SSA NSM iterations;
3     continue with next iteration;
```

It is checked whether the τ candidate for the non-critical reactions is at least γ-times larger than the average NSM step size $1/\sum_{l=1}^{L}\sum_{j\in M+N} a_j^l(\mathbf{X})$; if not, then τ-leaping is abandoned for the moment and a number of NSM steps performed instead. This is a common procedure found in several τ-leaping algorithms; not every state allows a large τ value and before making tiny jumps that would require much effort to calculate, the more simple (yet in this case more efficient) Next Sub-volume Method is used (or any other exact algorithm).

That said, it could be worthwhile to analyze the following two question:

- Given a model, what are the conditions on the current state so that a Sτ leap is at least γ times larger than a single average NSM step?
- How does this value scale with the model size?

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* (a more in-depth discussion about benchmark models will follow in Section 5.3). Figures 4.3a to 4.3c show a simple 3×3 model with reflective 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)

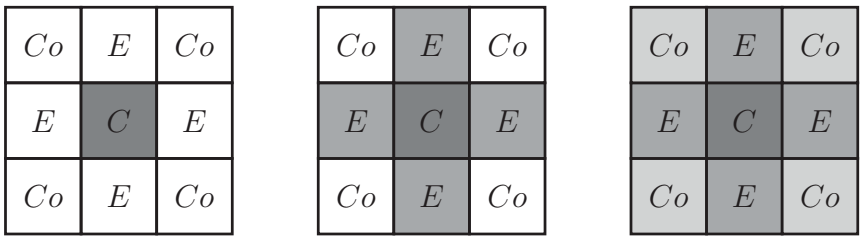


Figure 4.3: The "toy models" used for analyzing the spatial τ-leaping algorithm. (a) Only the center sub-volume contains A particles. (b) Some particles diffused into neighboring sub-volumes. (c) Each sub-volume now contains at least one A particle.

contains the same amount particles ($x_{E,A}$); this also applies for the corner sub-volumes (Co and $x_{Co,A}$). Initially, A particles are only present in the center (Figure 4.3a); as time progresses, they diffuse into the nearest neighbors (Figure 4.3b) and eventually reach the outermost sub-volumes (Figure 4.3c). 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 the dependence of τ to the initial number of A particles for each of the scenarios shown in Figure 4.3 and to compare the result with the respective average NSM time interval. Using the modified formula from Equation 4.20, the size of a leap calculated with the Sτ algorithm is

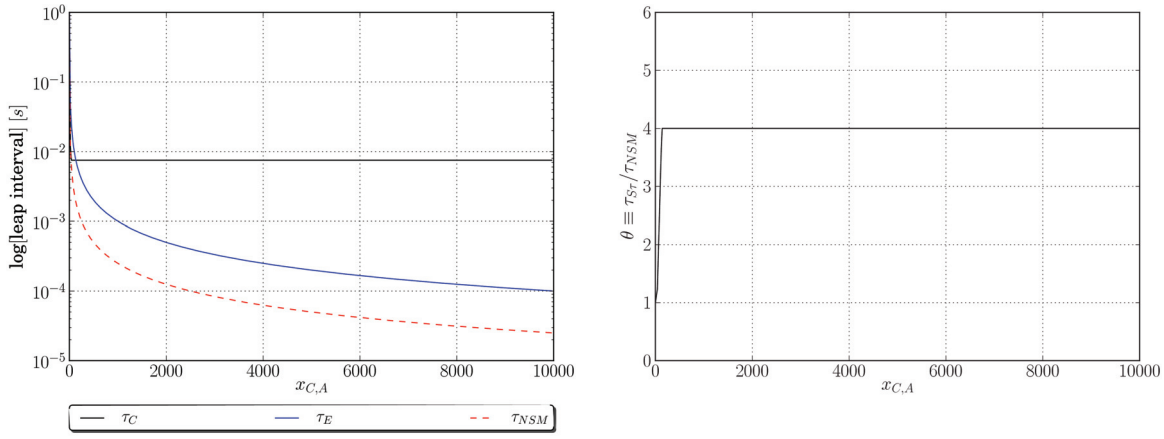
$$\tau = \min_{l\in\{C,E,Co\}} \left\{ \frac{\max\left\{\epsilon x_{l,A}/g_A^l(\mathbf{X}),1\right\}}{\left|\mu_A^l(\mathbf{X})\right|}, \frac{\max\left\{\epsilon x_{l,A}/g_A^l(\mathbf{X}),1\right\}^2}{\left(\sigma_A^l(\mathbf{X})\right)^2} \right\}. \tag{4.21}$$

Due to the simple structure of the model, it is not difficult to find the values for $g_A^l(\mathbf{X}), \mu_A^l(\mathbf{X})$, and $\left(\sigma_A^l(\mathbf{X})\right)^2$. Having no reactions defined, the only entry for the vector R' is the diffusion of A into neighboring sub-volumes,

$$R_1 : A \xrightarrow{c_A} \emptyset, \tag{4.22}$$

a first order reaction with $c_A = 2nD_A/\lambda_{sv}^2$ and $a_1^l(\mathbf{X}) = c_A x_{l,A}, l \in \{C,E,Co\}$; 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,

[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 $\lambda_{sv} = 1\,\mathrm{nm}$).



(a) (b)

Figure 4.4: Leap and θ values for the first toy scenario having only A particles inside the center sub-volume.

so taken all together the propensity function reduces to $a_1^l(\mathbf{X}) = 2nx_{l,A}\,\mathrm{s}^{-1} = 4x_{l,A}\,\mathrm{s}^{-1}$.

Species A only occurs in R_1, so according to Equation 4.17 $g_A^l(\mathbf{X}) = 1, l \in \{C, E, Co\}$. The only values yet to be found are $\mu_A^l(\mathbf{X})$ and $\left(\sigma_A^l(\mathbf{X})\right)^2$, but with a propensity as defined above both are easy to write down:

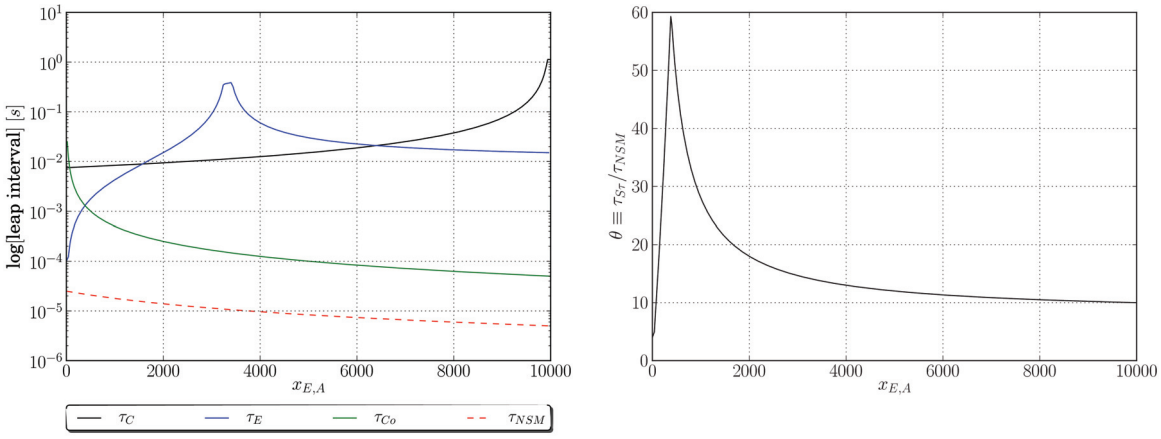
$$\mu_A^C(\mathbf{X}) = -4x_{C,A} + 4x_{E,A} \qquad \left(\sigma_A^C(\mathbf{X})\right)^2 = 4x_{C,A} + 4x_{E,A} \tag{4.23}$$

$$\mu_A^E(\mathbf{X}) = -3x_{E,A} + x_{C,A} + 2x_{Co,A} \qquad \left(\sigma_A^E(\mathbf{X})\right)^2 = 5x_{E,A} + x_{C,A} + 2x_{Co,A} \tag{4.24}$$

$$\mu_A^{Co}(\mathbf{X}) = -2x_{Co,A} + 2x_{E,A} \qquad \left(\sigma_A^{Co}(\mathbf{X})\right)^2 = 6x_{Co,A} + 2x_{E,A}. \tag{4.25}$$

Finally, to get a τ from Equation 4.21 for each of the scenarios is simply a matter of inserting the results just presented and using an appropriate initial configuration of $\{x_{C,A}, x_{E,A}, x_{Co,A}\}$. Figure 4.4 shows the minimum leap values for the center and edge sub-volumes and the average NSM step size for the first toy scenario, with an initial population $\{[1, 10000], 0, 0\}$[2] and $\epsilon = 0.03$. As expected, τ_C will eventually converge to 0.0075 ($\epsilon/4$) and τ_E is always $1/x_{C,A}$. The Next Sub-volume method takes a step of size $1/(a_1^C(\mathbf{X}) + 4a_1^E(\mathbf{X}) + 4a_1^{Co}(\mathbf{X})) = 1/4x_{C,A}$, so $\theta = 4$, i.e., a single leap is roughly equivalent to performing four NSM iterations — a sobering result, considering the complexity of the Sτ algorithm. But this example also hints to another optimization: the corner sub-volumes do not have any influence on the leap value, hence they can

[2] $\{[1, 10000], 0, 0\}$ shall actually denote the set of configurations $\{\{1, 0, 0\}, \{2, 0, 0\}, \ldots, \{10000, 0, 0\}$.



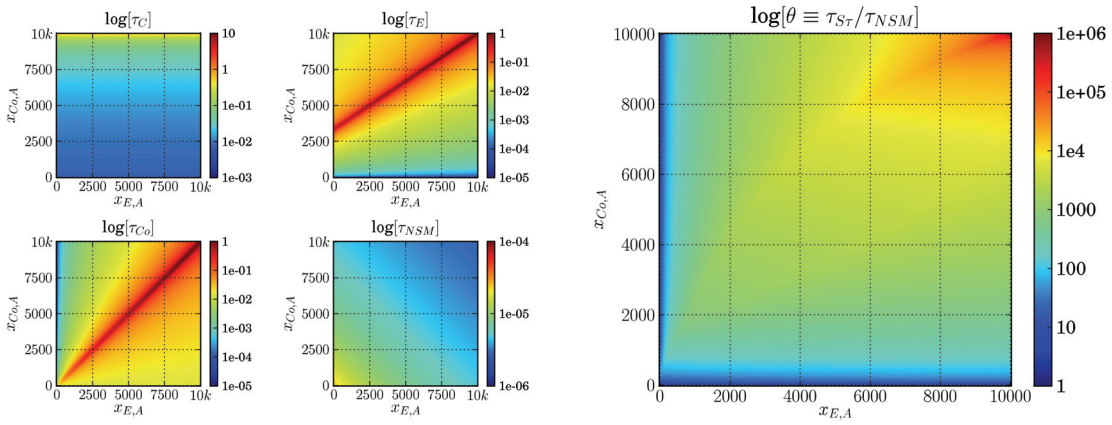
(a) (b)

Figure 4.5: Leap and θ values for the second toy scenario. The number of particles inside the center sub-volume is fixed (10000) and the A population is varied in the interval $[1, 10000]$ for the edge sites ($x_{E,A}$.

be ignored during the calculation (which generally applies to *every* sub-volume that is empty and whose neighbors also do not contain any particles).

Results for the next scenario (Figure 4.3b) are shown in Figure 4.5, with $\{10000, [1, 10000], 0\}$ as configurations and ϵ again set to 0.03. 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 $1/(2x_{E,A})$, i.e., the particles diffusing into the corners, and thus as $x_{E,A}$ is increased θ will converge to 10. In contrast to the former scenario there is a peak of about $\theta = 60$ for the ratio between $\tau_{S\tau}$ and τ_{NSM}, located at $\{10000, 380, 0\}$; it occurs at the intersection between τ_E and τ_{Co} (see Figure 4.5a). 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 $\left|\mu_A^E(\mathbf{X})\right|$. 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. The strange shape of the curve for τ_E in figure 4.5a can be explained by looking at the values for the mean and variance: at $x_{E,A} \approx 3333$, the expected mean change $\left|\mu_1^E(\mathbf{X})\right|$ reaches its minimum at approximately 1, so one leap candidate would be $\tau_E \approx 100\,\mathrm{s}$; however, the final leap value is capped by the expected variance $\left(\sigma_1^E(\mathbf{X})\right)^2$ to $\approx 0.375\,\mathrm{s}$.

For the third and final scenario the initial number of particles inside the edge and



(a) (b)

Figure 4.6: Leap and θ values for the third toy scenario. Now both $x_{E,A}$ and $x_{Co,A}$ are independently varied in the interval $[1, 10000]$ while $x_{C,A}$ is again fixed to 10000. With a high overall population a single Sτ iteration is equivalent to performing several thousands of NSM steps.

corner sub-volumes are varied independently, i.e., the configuration set is defined as $\{10000, [1, 10000], [1, 10000]\}$ (Figure 4.6). There are optimal setups for each of the τ candidates (the "red stripes" in Figure 4.6a) and the overall result also looks quite promising: with large numbers of A particles in all sub-volumes, θ is about 10^5, so a single Sτ leap iteration encompasses $\approx 10^5$ NSM iterations. Even low populations still allow considerable jump intervals compared to a single NSM step, e.g., $\theta \approx 54$ for $\{10000, 350, 100\}$. 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 a single Sτ iteration takes almost as long as an NSM iteration, which is very unlikely. As an example, if the implementation is such that a leap candidate can be found and the jump executed within the time it takes to perform, e.g., 500 NSM steps, than Sτ should be faster by a factor of approximately 200.

To sum this section up, the Sτ algorithm offers several starting points for optimizations, e.g., calculating the state update and applying it can be interwoven in several ways. Being a spatial algorithms, it is especially interesting to analyze how it behaves for different particle distribution patterns, three of which having been discussed on the previous pages. While Sτ may have problems with states that contain a mix of empty and non-empty sub-volumes, its performance should get better compared to NSM if a certain minimum population is present at all sites.

4.4 Parallel Extension

Even with only two loops, the overhead of processing each sub-volume twice could be significant for models hosting thousands or millions of them. Looking at the operations performed in the first loop, the only data accessed are the state vectors for the current sub-volume and its neighbors and the set of reactions. Furthermore, the order in which the sub-volumes are processed does not matter, the loop iterations can in fact be executed independently from each other; all that is needed are the τ candidates from each site. As a consequence, if more than one processing unit is available (e.g., a multi-core CPU, a cluster, or a network of computers) then it is possible to split up this work and distribute it among them.

Section 3.1.3 already introduced two parallelization methods: parallelization inside a simulation run and across a simulation. While the latter technique is generally applicable to reduce the execution time required for an *ensemble* of stochastic runs (hence the Sτ algorithm can be used without making any changes), the former represents a modification of the original algorithm aimed at making a *single* run more efficient. The basic idea for a parallel extension of Sτ, roughly outlined above, can be summarized as follows. If L is the number of sub-volumes and K processing units are available, then $[1, L]$ is sub-divided into K sets with at most $\lceil L/K \rceil$ entries each. All of those units need access to the state matrix and the reaction vector; how this is achieved depends on the actual implementation, e.g., in a shared memory environment the data structure representing the state can be directly read and written to by all processes whereas copies need to be send around if memory is only private to one unit.

Assuming the availability of K processors, lines 3 to 10 from Algorithm 4.1 can be replaced by the following abstract description of a parallel τ calculation (see also Figure 4.7):

---

1 **for** $k \in [1, K]$ **do**
2 | create parallel process $P_k\left([\lceil L(k-1)/K + 1 \rceil, \lceil kL/K \rceil], \mathbf{X}, R, \epsilon\right)$;
3 | start process P_k;
4 wait until all processes returned their respective tuple $(\tau_{k,nc}, \tau_{k,c}, l'_k)$;
5 $\tau_{nc} \leftarrow \min_{k \in [1,K]}\{\tau_{k,nc}\}$;
6 $\tau_c \leftarrow \min_{k \in [1,K]}\{\tau_{k,c}\}$;
7 $\hat{k} \leftarrow \operatorname{argmin}_{k \in [1,K]}\{\tau_{k,c}\}, l' \leftarrow l'_{\hat{k}}$;

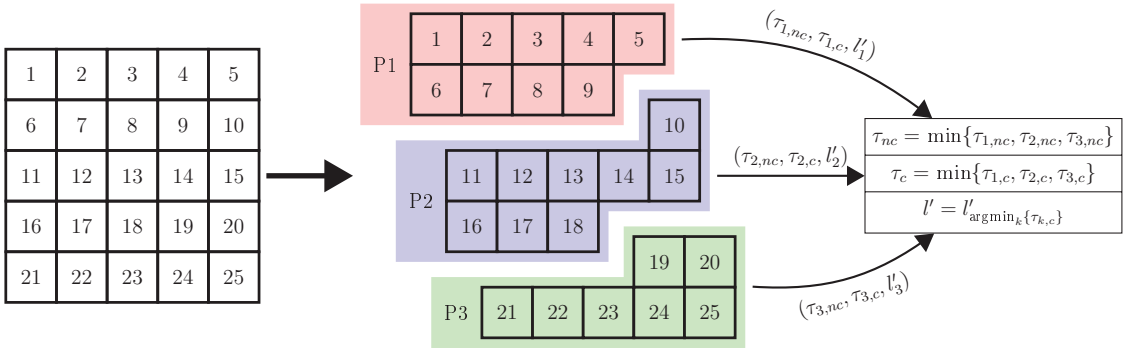


Figure 4.7: Distributing the τ calculation among 3 processing units P_1 to P_3. With 25 sub-volumes in total, each unit takes care of at most $\lceil 25/3 \rceil = 9$ sites. They independently determine the leap values and send them back to the main process where the global interval is taken as the minimum of thereof.

A process P_k simply gets the set L_k of indices for all sub-volumes it is responsible for and then performs all the task done inside $CalculateNonCriticalTau(\mathbf{X}, I^l_{nc}, \epsilon)$ for each $l \in L_k$. Finally, it reports back the τ values for both the non-critical and critical reaction sets and the index of the critical sub-volume, i.e., the site having the minimum critical τ. Getting a reference to the state matrix $\mathbf{X}$, P_k does not only have access to the state vectors for all $l \in L_k$, but also knows the current population of sites not included in its sub-volume index set — which is required to estimate how many particles are going to enter a sub-volume having neighbors not in L_k.

At this point it makes sense to distinguish between two possible implementations, the first one using shared, the second one only private memory. While the partition of L sub-volumes into the K lists L_k can be done once and the result send to the processes during initialization, it seems to be redundant to provide the *entire* state matrix. Being the only data structure that will change over time it has to be constantly updated on every processor. This should pose no problem for a shared memory environment, but could be critical if the matrix has to be sent across a network during each iteration. One way to reduce the load is to send only those state vectors that are actually necessary, i.e., every $\mathbf{X}_l$ with $l \in L_k$ *and* the set

$$\bigcup_{l \in L_k} \{\mathbf{X}_m : m \in \mathbf{C}_l \wedge m \notin L_k\}, \tag{4.26}$$

i.e., states for sub-volumes which are neighbors to others but not in L_k.

With the first loop parallelized, the next step is to look at the second one and see if

a similar replacement can be made there as well. But the problem here is again to find an update process that ensures that all state changes have been calculated based on the *current* state of a sub-volume. As it turns out, both optimizations discussed in the previous section can be applied with minor modifications. The adaptation of the first one — write the changes to an auxiliary matrix and relocate the state update into the first loop — is straightforward. Lines 15 to 24 in Algorithm 4.1 are substituted with:

```
1 for k ∈ [1, K] do
2     create parallel process P_k ([⌈L(k − 1)/K + 1⌉, ⌈kL/K⌉], X, l′, τ, R);
3     start process P_k;
4 wait until all processes returned their respective update matrix X̂_k;
```

Now the processes running during the τ calculation (see the first pseudo code snippet from this section) are slightly modified as well: a process P_k gets the following set as an additional parameter:

$$\{\hat{\mathbf{X}}_k\} \cup \bigcup_{k' \in K/k} \{\hat{\mathbf{X}}_{k'} : \exists l \, \exists m \, (l \in L_k \wedge m \in L_{k'} \wedge m \in \mathbf{C}_l)\}, \tag{4.27}$$

i.e., its own update matrix and all updates for neighbors not managed by P_k. Again, the processors have to check for negative state variables and abort the process if one is detected; before returning, they restore the original state using the information from the update matrix. This is a very abstract description of how to parallelize parts of the Sτ algorithm and details are up to the actual implementation. For example, if memory is shared among processors then only a single update matrix is necessary; all units get a reference to it and are able to directly read and write entries in a synchronized way.

The second variant makes copies of the current states instead of storing the changes made to them. A general adaptation for a parallel execution requires a communication between the processors. To see this, sub-volumes 9 and 10 from Figure 4.7 are taken as examples. Both are managed by different units; if P_2 starts the update with 10, then it is possible that particles diffuse into 9, which is hosted by another process P_1. Now P_2 can send a message to P_1, announcing a state change for 10; if 10's current state is not an element of the auxiliary set, it gets added to it and the state modified according to P_2's message. Note that a processor can only finish after receiving a "done" message from all neighboring units; though it may have updated its last sub-volume, particles

can still arrive from other processors. Communication can be avoided entirely if the sets C and Co (which holds the indices of the last sub-volume processed by each P_k) and the state matrix are shared among the units. The steps from the sequential variant need to be adjusted as well:

Initialization: Set $Co_k = -1$, start with sub-volume $l_0 = L_{k,0}$, and store its state in C.

Iteration: Proceed with the current sub-volume l_i and set $Co_k = i$. Store the state of each neighbor $l^* \in \mathbf{C}_{l_i}$ in C only if either of the following conditions is true:

- $l^* \in L_k$ and $l^* > l_i$
- $l^* \in L_{k'}$ and $l^* > Co_{k'}$

Take the stored state from C, use it to perform the steps 16 to 23 from Algorithm 4.1, and set $l_{i+1} = l_i + 1$.

The update technique just described assumes that it is possible to quickly determine the owner of a sub-volume l, i.e., the P_k with $l \in L_k$. Whether this is done using, e.g., some formula or a lookup table is again dependent on the actual implementation. At the end of an iteration, all sub-volume states have been updated; at the beginning of the next iteration it can be checked whether there are negative state entries.

It should be remarked that working with shared memory often requires means to synchronize the access to variables that can be read or written to concurrently by several processes (e.g., the counter Co_k or the data structure representing the current state), otherwise inconsistencies can occur. For example, while changing a variable a process gets interrupted before he can write the result; during this interrupt, another process also modifies the variable, but this change gets later overwritten by the result from the interrupted process. But being concentrated on the general procedure and due to the fact that nowadays many programming languages offer those features via semaphores or locking mechanisms, details in that direction will not be discussed in the course of this dissertation.

4.5 Summary

The algorithm presented in this chapter extends the basic principle of the leap methods into the spatial realm. Sτ considers both incoming and outgoing diffusion events and includes reactions and diffusion into the τ calculation. A detailed derivation shows how

to find the leap candidate for a sub-volume and use the minimum thereof to determine how often each reaction fired and how many particles of each species diffused into a neighboring site.

Section 4.3 provided an analysis that was aimed at further optimizations of the algorithm. Two alternative implementations have been discussed which avoid the third loop over all sub-volumes that was originally necessary to finally apply the state changes calculated before. This process can either be relocated to the beginning of the loop for determining the leap candidate or, with the help of additional data structures, combined with the state change calculations.

Apart from these optimizations, it has also been investigated how the size of the leap depends on the particle number and distribution. A preliminary conclusion can be drawn from the results: the closer Sτ gets to a homogeneous particle distribution, the larger is the ratio between the leap size and a single step taken by the NSM.

The final section discussed a parallel extension to Sτ. Each processor operates only on a subset of all sub-volumes and thus may require less time to find a leap candidate and calculate the state update; if this assumption is true is going to be analyzed in a following performance study.

5 Evaluating the Performance of Stochastic Simulation Algorithms

Both the background section and the survey list several stochastic simulation algorithms and variants thereof, either exact or approximative, and even more can be found when reading further literature from this field, e.g., [STP08, Sol09, SV05]. The spatial τ-leaping algorithm presented in the last chapter, for example, is one out of several methods for the simulation of spatial biochemical reaction networks — the NSM, GMPM, or Binomial spatial τ-leaping are all viable alternatives. So how "good" is Sτ compared to those? But maybe this is too specific; more generally, the question should be: how "good" is one stochastic simulation algorithm compared to some others?

Both developers and users have a strong interest in the evaluation of algorithms. The former group needs a way to compare a newly designed method to competing approaches. For what problems is it better suited, where are its weaknesses, but also: how could it be further improved? The second group, more interested in applying these methods to their specific problems, may require some guidance through the "huddle" of available algorithms. If it can be stated for what models it makes sense to use a certain method and in which cases it does not, then the task of choosing a simulator can be simplified — at least to a certain extent.

But what does "evaluation" in general and in the context of stochastic simulation algorithms in particular actually mean and what are the intricacies of performing one? This chapter will discuss these topics; it is not a straightforward guide how to conduct such a study, it rather focuses on selected aspects and points to some pitfalls that may be encountered. The concepts presented here will then be applied in two case studies, one for non-spatial and one for spatial simulation algorithms, in the next chapter.

5.1 Problem Statement

Definition 5.1.1 Evaluation is the systematic acquisition and assessment of information to provide useful feedback about some object [TD06].

Being very general, Definition 5.1.1 has to be put into the context of this work: the objects under observation are simulation algorithms and the information that shall be acquired and assessed are measurements for *performance facets*, discussed in more detail in the next section, which help to compare different methods with each other. With this in mind, an evaluation can be regarded as a special type of experiment: not the outputs of the algorithms are the central element of the study, but *how* they are produced. It involves measuring, e.g., how fast or accurately the system generates the results and the amount of resources it requires to fulfill the task.

Publications about new algorithms usually start with a motivation for the development, then proceed with the derivation of the method and eventually perform an evaluation study, which shall demonstrate the superiority over selected competitive variants. There are two main approaches to do the latter, often used in conjunction: 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.

Theoretical analysis of algorithm complexity is a cornerstone of computer science; it allows to characterize problem complexity and to gain deep insights into the nature of both algorithms and computational problems. It is a valuable tool to measure the *intrinsic* efficiency of an algorithm, i.e., how it performs independent from implementation or hardware. But what is undeniably an advantage can also be regarded as a weakness, because analysis techniques such as asymptotic complexity [Knu76] might miss important aspects of algorithm performance in reality. This is because the full complexity of modern hardware is difficult to model theoretically [SF01]; for example, Turing machines, where memory access is sequential and all basic operations have equal cost, are too simple to capture algorithm performance on hardware supporting a multi-layered memory hierarchy (e.g., cf. [LL97]), parallel execution (e.g., GPUs, cf. [PAYS09]), and so on. Additionally, algorithms can be non-monolithic, i.e., they could be dependent on the presence of other algorithms. If an algorithm A requires during execution functions provided by another algorithm B, then A is said to be *dependent* on B and B shall be called a *sub-algorithm* with respect to A. Well-known examples for sub-algorithms in simulation are *random number generators* (RNGs) [LK99] and

event queues [GT03, HU07a]. Both types are so common 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. The limitations of a theoretical treatment also apply to all sub-algorithms and their interplay with the host algorithm — which makes analysis even harder.

The alternative, empirical performance studies that directly measure the run time and other properties, suffer from different drawbacks. Executing algorithm implementations on physical hardware is as close at it can get to reality, but it is unfeasible to consider all possible setups. Furthermore, such studies involve a lot of effort, as the corresponding experiments have to be designed and executed carefully. Still, the input space is only marginally covered: models are needed, which naturally can represent only a small fraction of the problem space, and many algorithms require parameters for which appropriate values have been found.

Assessing the performance impact of sub-algorithms may appear simple at first glance, but is often cumbersome: the evaluation of all available setups and their correlation with overall performance is needed. An otherwise slow simulator could well be working quite efficiently in the presence of a suitable sub-algorithm; so in principle it is necessary to re-evaluate algorithm performance whenever a new implementation or a new sub-algorithm is available.

A good summary about the advantages and disadvantages of theoretical and empirical studies is given in [DI00]. All in all, the two approaches should go hand in hand [Joh03] to get the best out of both worlds.

But the results from thorough evaluation does not only tell a developer or user where the algorithm shines, they may also point to scenarios where it could run into problems; maybe some model features force it to make only very small steps (as it is the case with explicit τ-leaping and stiff systems, see Section 3.2.3) or it does not provide the required accuracy (which could be the case for all approximative variants). This could not only stir future development and provide a starting point for other researchers, but also ease the work of selecting an algorithm for a given problem. Having maybe only little experience in the field of simulation, most users have to rely on the implementations provided by their tools of choice; they are more interested in "getting the job done" than in the internals of the underlying algorithms. But without at least some information about the strengths and weaknesses of available alternatives they could be stuck with slow running, resource devouring methods that are entirely not suited for their task.

That said, the evaluation principles presented in the next sections will especially

focus on requirements for *empirical* studies, as these are more common in the field of SSA. Note that this discussion is specifically tailored towards evaluating stochastic simulation algorithms and only covers a small area in the wide field of *Experimental Algorithmics*, which was first proposed by McGeoch [McG96] and later reviewed by, e.g., Demetrescu [DI00] and Moret [Mor02].

5.2 Performance Facets

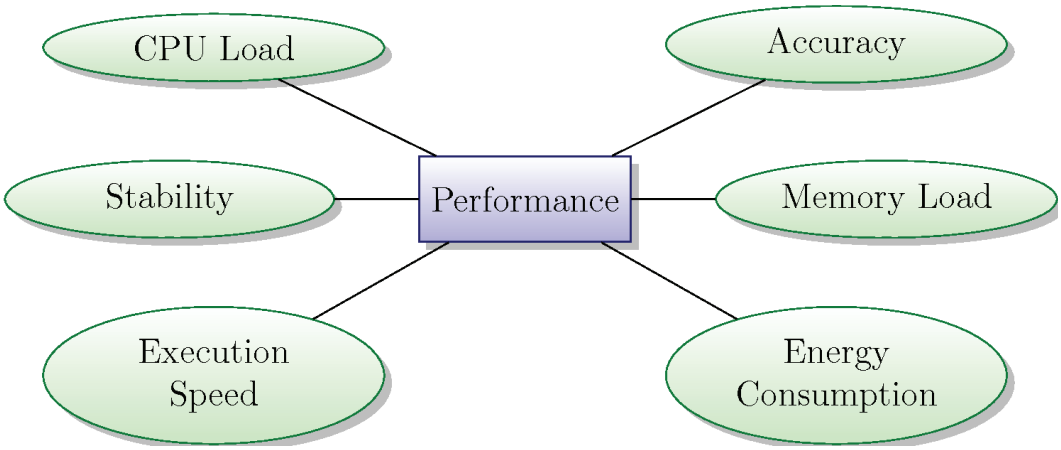


Figure 5.1: Performance facets for empirical evaluation studies.

The phrase "algorithm performance" appeared frequently in the last section, but nothing has been written so far about what *performance* actually means.

Definition 5.2.1 *Algorithm performance* is a general term for describing both the quantity ("how much") and quality ("how good") of the work done by an algorithm compared to the required resources (e.g., time, CPU, memory).

According to Definition 5.2.1, speaking of "the performance of an algorithm" is imprecise; any statement about performance has to include the aspect or *facet* which has been studied.

If evaluation is the general assessment of information, then facets specify their particular type. Depending on the type, a difference can be made between facets studied in a theoretical or empirical analysis. The former considers the algorithm itself, i.e., the sequence of instructions to perform a specific task, and is interested in answering questions such as, given a large input, how long it will take asymptotically for the algorithm to finish — which is the subject of the already mentioned asymptotic complexity

analysis. An empirical analysis, in contrast, depends on a concrete implementation running on real hardware, so the facets of interest include measurements for, e.g.,

- *execution speed* and *accuracy*, both will be discussed in more detail below,
- *CPU* and *memory load*, the two main limiting resources of computers, or
- *network load*, which is important for any algorithm operating across several machines.

Facets are not only used for comparing algorithms, but they are also central to the field of algorithm optimization (not to be confused with optimization algorithms, i.e., methods for finding an argument that maximizes or minimizes a target function). Here the evaluation results are taken as an input for the refinement of the algorithm; its execution could be geared towards being, e.g., faster in general or for certain problem instances, less resource-demanding or more accurate. However, optimization is often limited by the diametric relationship between some facets. Minimizing both CPU and memory load at the same time can only be done to a certain limit; from then on each further modification in either direction will have to be balanced by making sacrifices at the opposite side (the time-memory trade-off, cf., e.g., [Sta03]). For example, to save computation time, values frequently used and originally calculated with a time-consuming procedure during run time could be pre-calculated and stored within some data structure, e.g., a map or list. This also works for the other direction, saving memory by performing additional computations.

Similarly, algorithms which are fast *and* accurate for a variety of problem instances are also very unlikely. Being exact comes often at the cost of a slow execution; simulation methods calculating every single event, e.g., DM or FRM, generally need longer to finish than approximative methods such as τ-leaping. Still, it should be added that this is not always the case; a later section will revisit this statement and present conditions for which it does not hold.

At the end of the day it is the user that has to weigh one facet over another — or, more specifically, one algorithm over another. There could be several alternatives (as is the case for the stochastic simulation of chemical reactions), each one having their own strengths and weaknesses: some might be more tailored for a fast execution of larger models, others for a very detailed simulation at a lower scale. To ease the task of selecting a method, exhaustive evaluations, both theoretically and empirically, are a vital instrument.

In the course of this dissertation, the diametric performance facets execution speed and accuracy will be studied in detail for multiple non-spatial and spatial stochastic simulation algorithms. The following two paragraphs shall provide an overview of the techniques applied for their evaluation.

5.2.1 Execution Speed

Performing simulations means working with two different time references: the *simulation time* and the *wall-clock time*. The former specifies the time interval for which the system is simulated, e.g., usually nanoseconds or microseconds for reaction networks or hours, days or weeks for meteorological models; the latter measures how long it takes for the algorithm to finish its task. There is usually no connection between both references: a simulation can run for days before finally having processed a single second in the time frame of the system (cf. [RKS98]); on the other hand, simple estimations about population dynamics over several years can be done within milliseconds.

Definition 5.2.2 Execution speed relates the wall-clock time to the simulation time. The smaller the wall-clock time required to process a certain simulation time interval, the higher is the execution speed of the algorithm.

That said, measuring execution time is, to put it simply, nothing more that starting a timer at the beginning of a simulation and stopping it at the end. Although this can be done easily, a meaningful speed comparison still requires some precautions, especially when dealing with stochasticity. Firstly, the computed trajectories might differ in their characteristics, potentially biasing the results of a single execution. Therefore, multiple replications have to be executed and averaged execution times need to be considered. For example, the model studied in [SYSY02] can evolve via two different dynamics: either the viral template gets degraded shortly after the simulation starts or it infects the first cell, reproduces itself and starts attacking neighboring cells. A run of the first scenario finishes much more quickly than a run of the second one; without the template, no further reactions will take place, so the next event time would be infinity, i.e., the algorithm can already terminate. As a consequence, depending how often the first case occurs, the distribution of observed running times may vary significantly, which should be considered during result analysis.

Secondly, the hardware, operating system (OS), and (in case of Java) the virtual machine all introduce additional stochastic noise and bias. Their states are hard to control, apart from keeping their configuration fixed throughout the experiments and

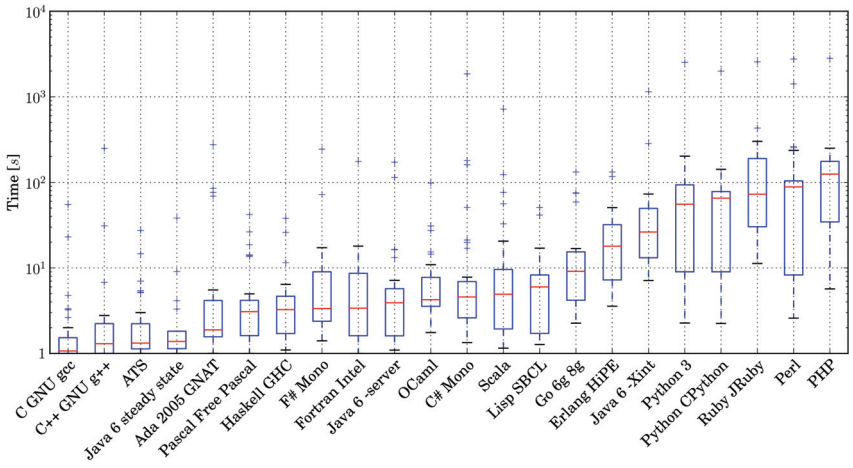


Figure 5.2: Benchmark results for selected programming languages, taken from the *Computer Language Benchmark Game website*[1]. The tests measure the run time, size and memory allocation of several programs. This chart analyzes the run time differences as follows: for each benchmark test T the run time t_l^T for each language is normalized to $t_l^T / \max t_l^T$. The median (red) and quartiles (25% and 75%) are then calculated for a language over all benchmarks. Outliers are shown as small crosses.

avoiding any unusual load (e.g., that may come from processes running in the background). How they influence the performance of a specific implementation is non-negligible and non-trivial to assess (e.g., cf. [VDB04]). To counter these effects, i.e., to reduce the bias, the design spaces of each algorithm can be explored, i.e., testing various algorithm and parameter combinations, each with multiple replications to account for noise and unobserved variables (e.g., cache state, etc.). The amount of required simulation runs motivates techniques for sequential experiment design [Rob52] (where the further progress of an experiment is based on what has been observed previously), in order to efficiently explore the algorithms' runtime distributions [ELU09], and the usage of a dedicated performance database [EHU08].

Related to the second point is the discussion about what algorithm implementation to use for a fair evaluation study. According to Johnson [Joh03], the best option is to obtain the source code from the authors and let the algorithms run against each other on a local machine. Doing so represents an *absolute* comparison, as it leaves the original code untouched; each implementation is the same as the one used for the primary publication of the algorithm. But this approach is open for debate: though a single hardware setup for all experiments avoids one source for bias, the performance differences between programming languages just may add another one. Figure 5.2

illustrates benchmark results for a couple of languages, collected by the *Computer Language Benchmark Game* website[1]. This chart provides only a rough and by far not exhaustive comparison, still it suffices to support the message.

An alternative, listed by Johnson as the second best option, is the development of own implementations for the algorithms under scrutiny, preferably in one language. This *relative* comparison may actually be the better choice. Everything up to the programming language is the same for each test candidate, so the evaluation is more focused on the actual algorithmic operations. Furthermore, an algorithm is not tied to a specific language (just as it is not tied to hardware, OS etc.); it only enumerates the steps to perform a specific task. Re-implementing these steps should be possible based on the description and pseudo-code provided in the publication; if any problems are encountered, the original authors can still be asked for help. Once done, the basic assumption of a relative comparison can be formulated as: if the algorithm was faster by some factor as a competitor implemented in the same language, then it should also be faster by a similar factor after re-implementing both in another language (given that no bias has been introduced by the person who implemented the algorithms). This dissertation follows the second option: all algorithms have been implemented as plug-ins for *JAMES II*. Java may no yield the same peak performance as compiler-optimized C or C++ code, but, as argued above, this is not required for an evaluation.

5.2.2 Accuracy

While execution speed is usually measured in the time or number of steps an algorithm needs to finish a task, it may be more difficult to evaluate the quality of the produced results. To begin with, a standard textbook definition for accuracy may look like the following:

Definition 5.2.3 Accuracy specifies how close a calculation comes to the true value.

Depending on the subject under study, the true value can be anything from a numerical value to a target function or even just the expectation about the correct outcome. As a first example, the most simple method to judge quality is using qualitative means. Plots of raw output data, e.g., trajectories, may help to get a first impression about how good the results are. The true values taken here could be a plot of the real solution or some constraints defined on the output. Those constraints can be formulated,

[1] `http://shootout.alioth.debian.org`

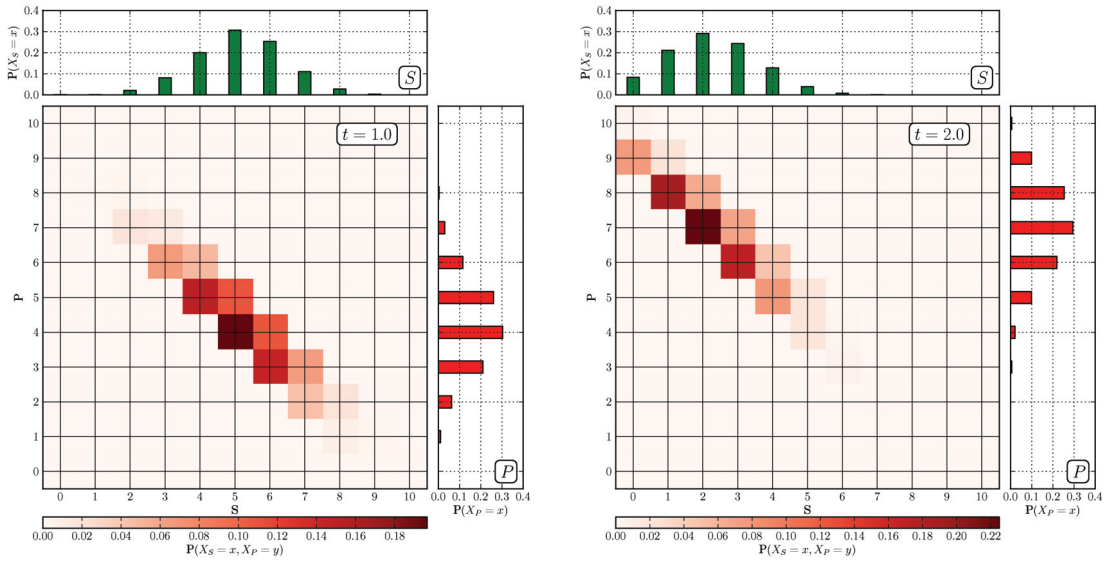


Figure 5.3: Probability distribution and population histograms (for the substrate S and product P) of the enzyme-substrate model from Equation 2.2, measured at time points $t = 1.0$ (left) and $t = 2.0$ (right) and based on 2000 replications.

e.g., for a species population: if it is expected that the number of molecules should only decrease over time but never increase and yet the latter can be observed, then something must be wrong with either the model or the simulation algorithm.

In the field of SSA, the true value is the time-dependent state probability function $\mathrm{P}(\mathbf{x}; t)$ introduced in Section 2.4.1. The task at hand is to evaluate the accuracy of stochastic simulation algorithms whose output is a single path through the state space. More formally, an SSA trajectory is one realization of the underlying Markov process, which is completely characterized by $\mathrm{P}(\mathbf{x}; t)$. If the time point is fixed to an arbitrary $\hat{t}$, performing replicated runs of a stochastic simulation algorithm and storing the state vector at $\hat{t}$ provides an approximation $\tilde{\mathrm{P}}(\mathbf{x}; \hat{t}) \approx \mathrm{P}(\mathbf{x}; \hat{t})$ in form of a histogram (see also Figure 5.3 for an example). Having found $\tilde{\mathrm{P}}(\mathbf{x}; \hat{t})$, the question is: how well did the algorithm capture the dynamics of the system, i.e., how close is this approximation to the exact solution $\mathrm{P}(\mathbf{x}; \hat{t})$? The first problem encountered when attempting to answer this question is that $\mathrm{P}(\mathbf{x}; \hat{t})$ may be unknown, which is likely for most of the more complex models. If this is the case, then the only alternative is to approximate it as well and consider the solution $\tilde{\mathrm{P}}(\mathbf{x}; \hat{t})$ as the true value. There are basically two ways how to get $\tilde{\mathrm{P}}(\mathbf{x}; \hat{t})$: the first one is to directly solve it for simpler systems (see e.g., [JU10]) or to calculate it with methods such as the sliding windows approach [WGMH10] or finite state projections [MBS08]; the second one is an approximation

via replicated runs, but using only *exact* SSA variants (like the NRM or ODM). In this dissertation the second alternative is applied: in contrast to the first one, it does not require additional information from the user. While the authors in [WGMH10] argue that their method is faster than running the model several thousand times, it is, e.g., not clear what shape the window should have for a certain model or if this can be determined automatically. One advantage of the second alternative is that it is basically applicable to any type of model without making any assumptions about how the state will evolve over time. Still, a lot of replications are necessary; to ease the task, *JAMES II* offers a convenient way to perform replications on multiple processors concurrently, which allows to utilize a network of computers or a cluster [ELU09]. But the question remains: how many replications are actually necessary to compute meaningful statistics? For this dissertation the number has been fixed but this is clearly not the best solution. Sandmann [San09a] proposed a method for limiting the number of replications necessary to narrow down the confidence interval for the variable of interest (e.g., the population mean) by calculating the interval for successive replications until its half width is below a certain (relative or absolute) threshold. This approach is currently tested in form of a replication criterion for *JAMES II* and may be extended in the future to allow an accuracy comparison of different simulation algorithms that does not rely on a fixed number of trajectories.

The central idea is quite simple: based on collected samples $\{\mathbf{y}^1, \ldots, \mathbf{y}^O\}$ and $\{\mathbf{x}^1, \ldots, \mathbf{x}^O\}$ of state vectors observed at an arbitrary time point $\hat{t}$ for both a test (approximative) and a reference (exact) algorithm, the former is considered as being accurate w.r.t. the latter if $\tilde{\mathrm{P}}(\mathbf{y}; \hat{t}) \sim \tilde{\mathrm{P}}(\mathbf{x}; \hat{t})$[2]. But how to perform this comparison? With two sample sets, one for exact and one for the approximative algorithm, the comparison essentially involves measuring the *difference* between the underlying distributions, either represented by the histograms (approximates the probability density function) or the empirical distribution functions (edf, approximates the cumulative distribution function). If there is only a single species defined in the model, then one way to do this is to use a two-sample statistical test, e.g., the Kolmogorov-Smirnov, X^2, or Student's t-test.

Cao & Petzold [CP06] devised an adapted technique, which does not rely on any existing test but is based on the underlying theory and also measures the distance between the histograms and the edfs. However, the distance alone does not say anything

[2] O state vectors at $\hat{t}$ are stored for both algorithms. It is then tested whether the distributions the sample vectors came from are similar.

about how good the approximation actually is; for this, a threshold is necessary — defined by the authors as the self-distance, i.e., the distribution distance between two sample sets generated by the *same* algorithm. In other words: if there are sample sets from two *different* algorithms (e.g., DM and τ-leaping) and the measured distance between them is smaller than the distance between two sets generated by *one of these* algorithms (e.g., DM), then the null hypothesis that the distributions sampled by both algorithms are similar would be accepted.

If multiple species are defined, then the state is a vector of random variables, which makes testing for the null hypothesis much more difficult than for the univariate case. The following discussion is made to provide concepts how this can be done using an iterative scheme; however, several issues are still open and future work is required.

To test for $\tilde{\mathrm{P}}(\mathbf{y};\hat{t}) \sim \tilde{\mathrm{P}}(\mathbf{x};\hat{t})$ either requires a multivariate analysis, e.g., Hotelling's T^2 test (see, e.g., [She07]), or an iterative method that tests whether:

$$\tilde{\mathrm{P}}(\{x_i : i \in I\};\hat{t}) \sim \tilde{\mathrm{P}}(\{y_i : i \in I\};\hat{t}), I \in \mathcal{P}(N),$$

with $\mathcal{P}(N)$ as power set of $\{1,\ldots,N\}$. Instead of comparing the entire set of state variables right from start, the latter begins with the one-dimensional marginal distributions ($k = 1$):

$$\begin{aligned}
\tilde{\mathrm{P}}(x_1;\hat{t}) &\sim \tilde{\mathrm{P}}(y_1;\hat{t}) \\
\tilde{\mathrm{P}}(x_2;\hat{t}) &\sim \tilde{\mathrm{P}}(y_2;\hat{t}) \\
&\vdots \\
\tilde{\mathrm{P}}(x_N;\hat{t}) &\sim \tilde{\mathrm{P}}(y_N;\hat{t});
\end{aligned}$$

each single comparison can be done by an arbitrary statistical test. Should all of these pass, i.e., the *null hypothesis* H_0 stating that the test and reference samples for each species come from the same universe cannot be rejected, it then proceeds with the next higher dimensions, i.e.,:

$$\begin{aligned}
&k = 2 : \tilde{\mathrm{P}}(x_i, x_j;\hat{t}) \sim \tilde{\mathrm{P}}(y_i, y_j;\hat{t}), i \in [1,N], i < j \leq N \\
&k = 3 : \tilde{\mathrm{P}}(x_i, x_j, x_m;\hat{t}) \sim \tilde{\mathrm{P}}(y_i, y_j, y_m;\hat{t}), i \in [1,N], i < j \leq N, j < m \leq N \\
&\vdots \\
&k = N : \tilde{\mathrm{P}}(\mathbf{x};\hat{t}) \sim \tilde{\mathrm{P}}(\mathbf{y};\hat{t});
\end{aligned}$$

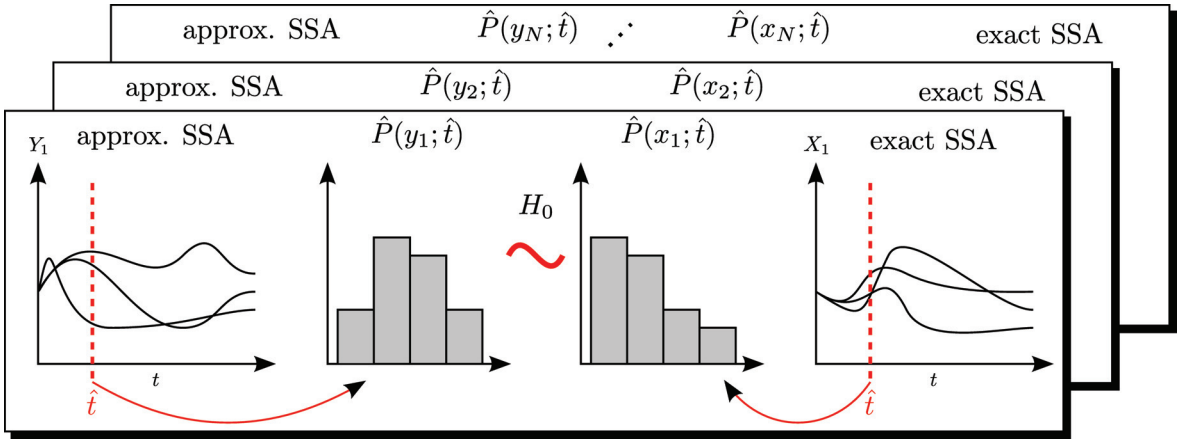


Figure 5.4: First iteration of the accuracy evaluation using replicated runs. The distributions $\tilde{P}(y_i;\hat{t})$ and $\tilde{P}(x_i;\hat{t})$ for each species $S_i \in S$ are compared individually. If any of the N tests rejects the null hypothesis (both distributions are similar), then the evaluation is stopped; otherwise the next iteration will start, testing for $\tilde{P}(y_i, y_j;\hat{t}) \sim \tilde{P}(x_i, x_j;\hat{t}), i \in [1, N], i < j \leq N$.

the final step eventually includes the entire state vector. During any of the N iterations one of the tests may reject H_0 at the given significance level; this suffices to show that also the initial assumption $\tilde{\mathrm{P}}(\mathbf{y};\hat{t}) \sim \tilde{\mathrm{P}}(\mathbf{x};\hat{t})$ does not hold and the method can be stopped at this point.

Alternatively, the $\binom{N}{k}$ tests per iteration can also be condensed into a single one by adapting an existing statistical test procedure. This direction has been explored for the accuracy evaluation of spatial stochastic algorithms; the goal is to get a single test statistic per k that tells the user if the species distribution over all sub-volumes is similar for the test and reference algorithm. As it is often unknown how this distribution will look like, an adapted variant of the non-parametric two-sample Chi-square test (see, e.g., [Pla83]) was chosen as the test procedure because it does not make any assumptions in this regard: given the two histograms for sets of test and reference samples, it checks whether the observed number of points in each bin is similar. Before the actual evaluation can start, the method requires a calibration phase in which a threshold for rejecting the null hypothesis H_0 (test and reference samples come from the same distribution) is calculated using replicated runs of an exact spatial algorithm, e.g., the Next Sub-volume Method; this approach of finding a threshold for deciding whether to reject or accept H_0 is similar to what has been done in [CP06]. During this initialization, Rep_{tot} replications are distributed among Rep_s sets, each one having therefore Rep_{tot}/Rep_s entries. A single $X^2_{p,q}$ candidate is obtained by comparing two

of these sets p and q:

$$X^2_{p,q} = \sum_{l=1}^{L} \sum_{i=1}^{N} \sum_{m=1}^{K} \frac{(\hat{p}_{l,i,m} - \hat{q}_{l,i,m})^2}{\hat{p}_{l,i,m} + \hat{q}_{l,i,m}}. \tag{5.1}$$

The values $\hat{p}_{l,i,m}$ and $\hat{q}_{l,i,m}$ are the relative frequencies for state m of species S_i in sub-volume l. With Rep_s sets in total, a pairwise comparison results in $Rep_s(Rep_s - 1)/2$ X^2 candidates; depending on the chosen significance level α, one of these candidates is selected as the threshold. To get this value, a vector of candidates is defined such that for any index pair (i, j) with $i < j$ the relation $X^2_i \leq X^2_j$ is true. Then the entry at index $Rep_s(1 - \alpha)$, denoted by X^2_t, is used in the following evaluation to decide whether the null hypothesis is rejected or not.

After this initial step, which essentially used samples from just one algorithm to approximate a Chi-square distribution, it is then tested if the $X^2_{\tilde{p},\tilde{q}}$ value for a test set $\tilde{p}$ and reference set $\tilde{q}$ is a sample from this very distribution. All that is needed is to compare the threshold with this value, i.e., reject H_0 if $X^2_{\tilde{p},\tilde{q}} > X^2_t$. Should this be the case, then the p-value (the probability of finding a X^2_t at least as extreme as the calculated value) can be obtained by finding the fraction of candidates which are higher than $X^2_{\tilde{p},\tilde{q}}$. Note that the current implementation of this test runs over *all* sub-volumes, which may not be necessary and could lead to long run times for models with a large number of sites. The replications performed to find the threshold during the calibration phase could be used to limit the number of sub-volumes to the ones having, e.g., a minimum number of particles present, which means that the comparison is focused in the regions in space where the majority of the particle mass is located.

While it seems a lot of effort — in the worst case, the first methods requires $\sum_{k=1}^{LN} \binom{LN}{k}$ checks for a model with N species, with $L = 1$ for the non-spatial case, while the maximum for the second is the number of iterations, i.e., LN – the fact that the computation may already finish after only a few iterations makes these methods still worthwhile to consider. Additionally, this property can also be exploited to reduce the number of replications that have to be performed. Starting with only a limited number of trajectories, the method runs until either a test fails or the amount of replications is no longer sufficient to compare higher dimensional distributions; in the latter case new simulation runs are performed, their results added to the sample set and the process continued. However, what sounds simple is actually rather difficult: how many additional replications are required for the k-th iteration? This could be answered using a multivariate power analysis and considering data (means, standard deviations, correlations) provided by already generated state vectors (see, e.g., [DNZ01]), but still may

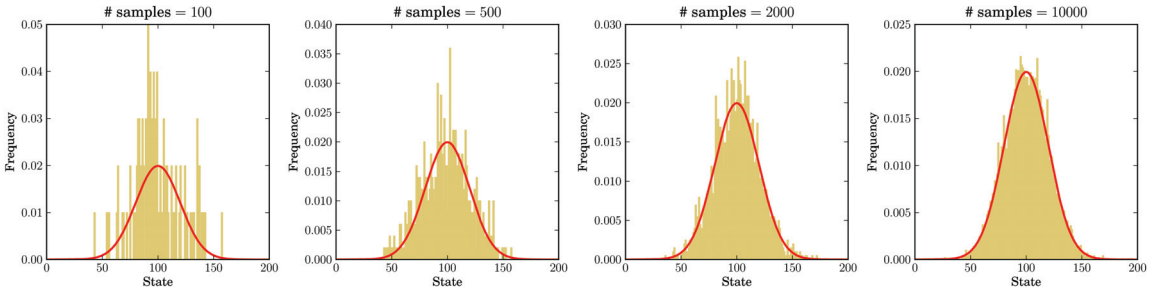


Figure 5.5: Example histograms for a species' state at some time t if the distribution is roughly Gaussian with $\mu = 100$ and $\sigma = 20$. The red line shows the appropriate probability density function. If the number of samples (replications) is increased, then the shape of the histogram gets more closer to the continuous probability density function.

be a complex task; as estimating the amount has not been resolved yet, the accuracy tests in Chapter 6 are limited to $k = 1$.

If this number can be found, then trajectories can be generated on demand instead of collecting a large number right from the start to conduct an N-dimensional statistical test.

Example 5.2.1 It is known that the particle numbers of six species S_1 to S_6 at time t are all "roughly" distributed according to a normal distribution with $\mu = 100$ and $\sigma = 20$ (note that the number of particles for $S_i, i \in [1, 6]$ is a *discrete* variable, so the distributions would also be discrete, but a normal distribution is defined over a *continuous* domain; "roughly distributed" means in this context that the histogram, i.e., the empirical probability distribution, can be approximated by a normal distribution; see also Figure 5.5). This means that for each S_i over two third (68%) of the samples fall on one of the 40 states ranging from 80 to 120 and nearly all of them ($\approx 95\%$) are located between the states 60 and 140 — so, approximately 1360 samples get distributed among 40 states and, assuming no correlations between the species, 2000 replications should be sufficient to get a good approximation of the underlying distributions. However, if the particle numbers of two species S_i and S_j *are* correlated, then it is necessary to also test the two-dimensional normal distribution $P(x_1, x_2; t)$; here the number of states within one and two standard deviations $\sigma = (20, 20)$ from the mean $\mu = (100, 100)$ increases to 1600 and 6400. Again, with 2000 replications 1360 samples now fall into a range of 1600 states — which already may be too few to get a picture of the two dimensional distribution distribution. And in in the case that the state variables of *all six* species are not independent, then 40^6 states are located within one standard

deviation — about four billion.

Admittedly, for high-dimensional distributions it makes sense using *binning* to reduce the number of states. Looking at Example 5.2.1 again, a possible binning could be defined for each species by:

$$B_1(x_i < 60), B_2(60 \leq x_i < 70), \ldots, B_9(130 \leq x_i < 140), B_{10}(140 < x_i), \tag{5.2}$$

with a B_i holding the number of samples that fall within its range. Thus a sample for one species is assigned to one of ten possible bins; if $N = 6$, the total number of states now to consider is 10^6 — compared to 40^6, this is a reduction by a factor of 2^{12}. Note that this is only an example; a more sophisticated strategy for deciding the bin width can be found, e.g., in [LY06]. However, a common requirement for the Chi-square test is that there is a minimum number of observations for each bin (usually more than 5), otherwise the test is not valid.

Common to all the discussed methods is that the sample sets for iterations which compare multi-variate distributions may require a pre-processing (if $k = 1$, then only the individual species distributions are compared and their state values can be directly taken as the input set). For $k > 1$, an event (in the context of probability) is defined by the values of k state variables. As an example, with $k = 2$ and N species, two events $e_1 = (x_1, x_2, \ldots, x_N)$ and $e_2 = (y_1, y_2, \ldots, y_N)$ are equal iff $x_1 = y_1 \cup x_2 = y_2$; any additional state variable is not considered. To ease the evaluation, a unique number can be assigned to each event and the set of all identifiers for an event class $\{x_i : i \in I\}, I \in \mathcal{P}(N)$ is taken as the input for the $k = |I|$ dimensional accuracy evaluation, thus modifying the accuracy analysis to a comparison of event distributions (see also Figure 5.6 for an example).

5.3 Benchmark Models

Having discussed *what* should be analyzed during an evaluation — the performance facets — the next step is to shed some light on the *inputs* for such a study: the models. The most obvious choice are representations of real world systems, e.g., the enzyme-substrate catalyzation process used as an example throughout this dissertation or a signaling pathway. Information about these models can often be found by scanning through the literature; especially the enzyme-substrate reaction network has been the subject of a variety of publications because similar dynamics can often be found in

state sample				event id			
E	ES	S	P	$\{E, ES\}$ $k=2$	$\{E, S\}$ $k=2$	$\{P, S\}$ $k=2$	$\{E, ES, P\}$ $k=3$
0	1	4	5	1	1	1	1
1	0	5	5	2	2	2	2
0	1	2	7	1	3	3	3
0	1	3	6	1	4	4	4
0	1	7	2	1	5	5	5
1	0	5	5	2	2	2	2
0	1	4	5	1	1	1	1
0	1	3	6	1	4	4	4
1	0	5	5	2	2	2	2
0	1	3	6	1	4	4	4
0	1	4	5	1	1	1	1
0	1	3	6	1	4	4	4
0	1	3	6	1	4	4	4
0	1	5	4	1	6	6	6
1	0	2	8	2	7	7	7
1	0	6	4	2	8	8	8
1	0	3	7	2	9	9	9
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

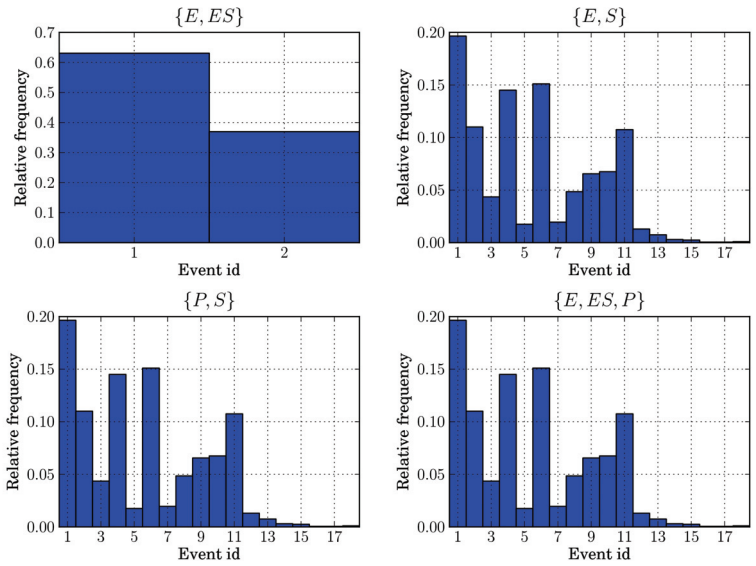


Figure 5.6: Example sample set and event identifier assignment for the enzyme-substrate model. The state is recorded at a predetermined time point ($t = 1.0$ in this case) and a unique event id is assigned to each of those samples. For example, considering the pair $\{E, ES\}$ there can only be two event ids, corresponding to the two states the enzyme can be in: either it is free ($E = 1, ES = 0$) or a substrate is bound to it ($E = 0, ES = 1$). The table also suggests that there is no difference between the ids for the event classes $\{E, S\}$, $\{P, S\}$, and $\{E, ES, P\}$, so testing the distribution for either one may be sufficient for the evaluation (see also Figure 5.7 for a state space analysis of the enzyme-substrate reaction network); this is additionally supported by the histograms for the event frequencies shown at the bottom (2000 replications).

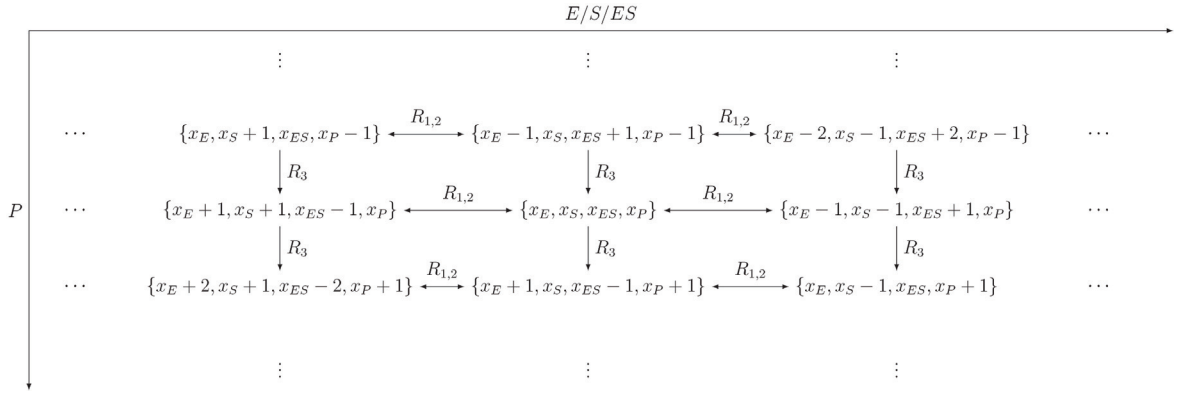


Figure 5.7: State space analysis of the enzyme-substrate model from Equation 2.2. The task is to show that the values of two state variables are sufficient for uniquely identifying state. To do so, an arbitrary state is taken as a starting point and it is observed how the number of particles changes when each of the reactions fires exactly once. If reaction R_1 gets executed, then one enzyme and one substrate molecule are converted into the compound ES; R_2 simply reverses this process. R_3 decreases the amount of ES particles and adds the enzyme and one product to the state vector. Between two rows the state vectors differ in the number of product particles, which changes only via reaction R_3; within a row, each vector is unique in the particle amount for either of the three remaining species E, S, or ES. Taking both observations together, one state can be unambiguously identified by a pair (x_P, x_i), with $i \in \{E, S, ES\}$.

more complex reaction networks (the destruction complex from the Wnt pathway can be seen as one example: it binds to β-catenin and either attaches a phosphate or again releases the protein, which is essentially captured by the set of reactions defined in Equation 2.2). They are undeniably helpful — being designed with a real world system in mind, they are as close as it can get to a typical problem instance for any algorithm implementation. But there are two sides to every coin: those models do not only represent "a typical", but at the same time also "just one" problem instance. Admittedly, this is not a concern for studies that are focused on getting insights to a system, but testing the efficiency of algorithms with such "static" models (a fixed parameter set based on empirical data) may result in biased assessments caused by limiting the problem space to only a small area. Five models selected are merely five points in the parameter space spanned by, e.g., one axis for the number of initial particles present, one for the number of reactions, a third representing the degree of interdependency between reactions, and so on.

Most models can be categorized according to several characteristics inherent to them. Some have just been mentioned, others include, e.g., the distribution of species in space or dynamics such as oscillations or multistability (depending on the initial seed of the RNG, the model ends up in one of several possible steady states), just to name a few. Considering this, the evaluation task may shift from observing an algorithm's behavior when faced with a couple of static models towards analysing the influence of model characteristics on their performance. Models that have been specifically designed to represent typical characteristics shall be called *benchmark models*. Though not of immediate biological relevance, their application still has several advantages compared to real-world models. Additionally to those already mentioned, they may offer:

- **Comparability**: Agreeing on a fixed set of models (e.g., two of the models used to study the performance of non-spatial algorithms have been presented by Cao et al. [CLP04]) makes it possible to compare the runtime performance of the realizations. Large deviations may indicate subtle implementation errors or the dependency on additional, yet undiscovered factors (such as specific hardware or compiler optimizations).

- **Ease of Implementation**: Due to their simple structure, the synthetic models can be generated automatically and are easily prepared for any SSA implementation. This is important for re-validating the results with other implementations.

- **Scalability**: The synthetic models are built to scale. Their size can be varied by,

e.g., adjusting the number of species N, which allows us to analyze the efficiency of a simulator when the problem size grows. There are many algorithms that perform very well on small problems, but become more and more inefficient when the problem size is increased. Scalable models help to find the problem size for which an algorithm is most efficient.

- **Parameterization**: Parameters of synthetic models allow us to investigate algorithm performance on *classes* of real-world models. For example, adjusting one parameter of the cyclic chain system, a model defined in detail in the results chapter, controls the degree of interdependency between reactions, i.e., how many propensity updates are necessary after executing a reaction, which in turn influences the effectiveness of a dependency graph.

- **Analytical treatment**: It is usually much easier to derive analytical results for synthetic models, since they exhibit a regular and simple structure. This may guide the developers in case of invalid results and could facilitate result analysis in general.

Despite all of these points, the algorithms should eventually work with models of real-world systems. Anyhow, chances are that these models include features already represented by some benchmark models, i.e., the latter subsume sets of more realistic models all exhibiting a specific property. The experimental results for benchmark models may therefore still facilitate the selection of a suitable algorithm for the concrete problem at hand. Considering the degrees of freedom in implementing and configuring any of the methods discussed so far, e.g., with respect to event queue, RNG, or τ-leaping parameters like ϵ or n_c (cf. Section 3.2.1), the recorded performance data and its analysis could eventually lead to selection rules that can be applied *automatically* [EHU08]. This would greatly improve the usability of SSA methods for users with other backgrounds than simulation.

The section shall be concluded with an example for a small benchmark model, called the *Molecule Generator Model* or MolGen for short. It consists of two species, the generator C and the product A, and its dynamics are defined via the following two reactions:

$$\begin{aligned} R_1 &: C \xrightarrow{c_p} C + A \\ R_2 &: A \xrightarrow{c_d} \emptyset. \end{aligned} \tag{5.3}$$

Species C is able to produce a molecule of A, which gets degraded by the second reaction R_2. What are the characteristics of this model? Both reactions are antagonistic with respect to A: R_1 increases its population, while R_2 decreases it, and it is not difficult to show that the number of A particles eventually converges to an average of $(c_p/c_d)x_C$, with x_C as number of C particles. This model does not seem to be very helpful, yet it gets interesting if the transition is made from a well-stirred to a spatially inhomogeneous system. Due to its simple structure, it also allows an easy steady state analysis in this case, i.e., to find the state at which the number of A particles in every sub-volume does not change any longer. According to the law of mass action, the change in concentration for A is:

$$\frac{dx_{l,A}}{dt} = c_p x_{l,C} V^{-1} - c_d x_{l,A} V^{-1} - \frac{2nD_A}{\lambda_{sv}^2} x_{l,A} + \frac{D_A}{\lambda_{sv}^2} \sum_{k=1}^{2n} x_{k,A} = 0. \tag{5.4}$$

The values $x_{l,A}$ and $x_{k,A}$ represent the number of particles in sub-volume l and its k-th neighbor, respectively. Rearranging the terms finally results in:

$$x_{l,A} = \frac{1}{c_d V^{-1} + 2nD_A \lambda_{sv}^{-2}} (c_p x_{l,C} V^{-1} + \frac{D_A}{\lambda_{sv}^2} \sum_{k=1}^{2n} x_{k,A}). \tag{5.5}$$

Figure 5.8 shows both the steady state solution of equation 5.5 and a numerical solution of equation 5.4 for diffusion coefficients $D_A = 1.0$ (top) and $D_A = 10.0$ (bottom), a fixed number of C particles located in the center sub-volume, and a side length of 1. The outer volume boundaries are assumed to be reflective, i.e., whenever a particle wants to leave the model volume, it bounces off of the border and stays inside the source sub-volume.

Results from this analytical treatment could guide subsequent performance experiments. For instance, focusing on certain time intervals (e.g., observe only the transition to the steady state for the Molecule Generator model) does not only save execution time. Assuming the existence of a characteristic state transition pattern during this simulation period, it also provides information about an algorithm's performance when confronted with this type of state dynamics — it might turn out that switching to an alternative could be a better option.

To sum up this discussion: benchmark models can be a valuable tool especially for evaluation studies, as they do not try to represent a real world system but instead have been designed with specific model characteristics in mind, which may be found in

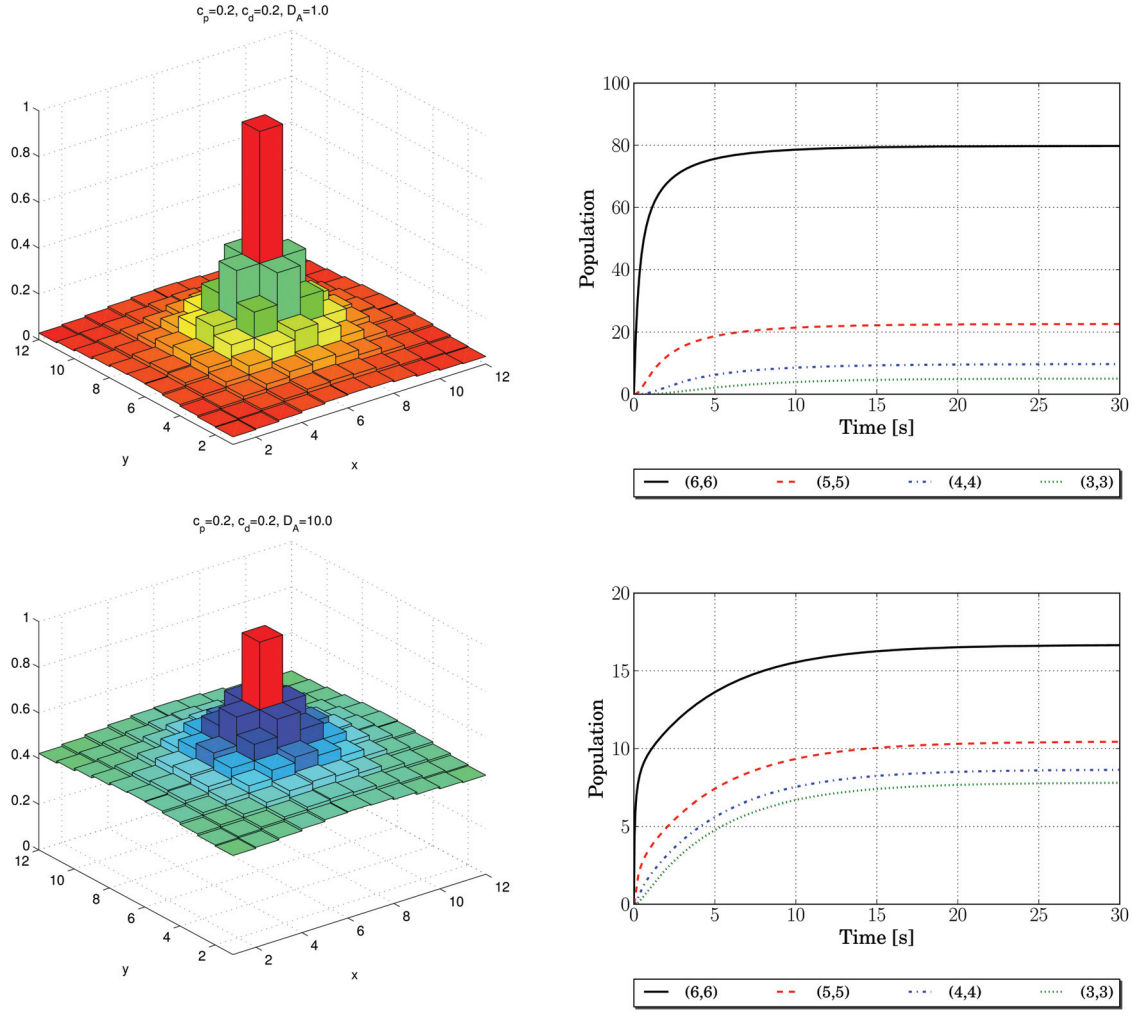


Figure 5.8: Steady state solutions for the concentrations of A particles (scaled to the interval $[0, 1]$) in a 2D 11×11 molecule generator model (colors represent the concentration). The parameters are set to $c_p = 0.2$, $c_d = 0.2$, $X_C(0) = 1000$, $D_A = 1.0$ (top), and $D_A = 10.0$ (bottom). The plots on the right side show the concentrations over time for the four sub-volumes with coordinates $(6, 6)$ (the center), $(5, 5), (4, 4)$, and $(3, 3)$.

a wide range of relevant models. However, they are not a replacement for real world models, but rather being complementary to them. The best way to perform such an evaluation may be to start with highly abstract benchmark models that narrow the problem space well covered by the algorithm under scrutiny. Based on the results from this first step, more realistic models located in this region are then added to the model set, checking whether the method is also applicable under real conditions.

5.4 Bringing It All Together: The Evaluation Study

This part shall finally tie together all the points made so far in this section. As it was stated right at the beginning, an evaluation can be seen as a special type of an experiment, which itself has been defined as the process of presenting inputs to a system and observing the generated outputs. Just like the definition of a simulation essentially replaced the term "system" with "model" in the last sentence, it is now appropriate to use "algorithm implementation" as the substitute: *experiments on algorithms* and the subsequent *output analysis* are the central steps of a performance evaluation.

Experiments gather information about the subject of interest by feeding it some input and collecting the output. If the system under study is an algorithm, the input is a set of *problem instances*, with each instance consisting of

- a (benchmark) model,
- a set of model parameters, and
- an algorithm setup, i.e., values for parameters controlling the behavior of an algorithm, e.g., ϵ or N_{SSA} (the threshold for switching to an exact SSA execution) in case of a τ-leaping variant, but also the set of sub-algorithms, if required, and their respective parametrization.

As it would be tedious to define each problem instance by hand, several techniques exist to either automate or assist the user in this task. *Parameter scanning* takes intervals or predefined lists with values for each parameter and creates instances by combining entries from them. This can be done, for example, full factorial: having two lists $\mathbf{l}_1 = (l_{11}, l_{12}, \ldots, l_{1L})$ and $\mathbf{l}_2 = (l_{21}, l_{22}, \ldots, l_{2K})$, a parameter tuple is defined as $(l_{1i}, l_{2j}), i \in [1, L], j \in [1, K]$, so every element from $\mathbf{l}_1$ is paired with each $\mathbf{l}_2$ entry. Though considering every possible combination, this design quickly generates a large number of instances; with a set of N parameters and $K_i, i \in [1, N]$ list entries for the

ith element, there are $\prod_1^N K_i$ possible parameter combinations that have to be executed — and this does not include any additional replications required for a stochastic simulation. Alternatives to reduce the number of instances may consider only tuples (l_{1i}, l_{2i}) (which is straightforward if $L = K$, otherwise the missing entries could be set to some default value), fractions of a full factorial combination, or are based on the Plackett-Burman design for multifactorial experiments [PB46]; all of these designs are already supported in *JAMES II*.

In addition, besides the generation of problem instances, stopping criteria have to be defined and applied to all runs. The pseudo-code listings for all algorithms presented so far require a start and an end time to be given as parameters; a single run then updates the internal time variable and stops the computation if it exceeds the end time. Though this is an obvious choice for a stopping criterion, it is just one of many possible conditions. Instead of waiting until the simulation run reaches a certain time point, it could also exit the main loop after an observed variable reached equilibrium or a given threshold. One criterion exceptionally useful especially in evaluation studies is the wall-clock time allowed for an algorithm to finish one problem instance from a given set. Formulating rules such as "stop the execution after 20 minutes, regardless how far the algorithm progressed", permits us to "censor", i.e., prematurely abort, runs that exceed a maximum time window and thus sets a fixed limit for the length of a single run. This may reduce the overall experiment execution time considerably if the set of problem instances contains configurations with highly varying run times; Chapter 6 gives several examples for such a condition.

Stochasticity adds another layer of complexity to experiment design, as a single run of an instance alone may not tell the full story, yet performing hundreds or thousands of replications for a large number of problems could take a long time. Finding a balance between providing meaningful results and the time spent to produce them is often hard to achieve. If the task is, for example, to find the best algorithm setup for one model and a fixed set of parameters, equally running each instance a predetermined number of times may result in wasting a large fraction of the total experiment time for processing the slowest (and thus uninteresting) setups. To avoid this scenario, but still be confident in the outcome (a list with the best setups), only promising configurations are executed more often, which decreases the run time variances and therefore provides more reliable results. No further replications are spent on any setup showing a bad performance after maybe one or two runs. *JAMES II* offers this technique in the form of an *AdaptiveSimulationRunner* [ELU09]; the forthcoming studies presented in the

next chapter make heavy use of this component.

The topics discussed up to now include experiment preparation and execution, so the pieces still missing are output observation and analysis. Before making any statements about an algorithm's performance, e.g., how fast or accurate it is, the data backing up any empirical evaluation has to be collected. And even before that it has to be clear *what* data should be observed at all. Comparing the execution speed, for example, requires information about how long it took an algorithm on average (or maximum / minimum) to finish one or all replications for a single setup. The output of interest here is execution time: a timer is started at the beginning of the computation (including initialization) and stopped when the task is done, i.e., the abort criterion fulfilled. Note that care must be taken if the execution is subject to a warm-up phase, as it is, e.g., the case with applications running inside a Java virtual machine (JVM): the source code is first compiled to bytecode, which is then loaded by the JVM; this extra time should not be counted against the algorithm. *JAMES II* relies on the well-known observer and instrumenter patterns, which are responsible for specifying *what* and *who* should be observed, respectively. For example, to get the data for an accuracy analysis, i.e., samples of state vectors for a given set of snapshot time points, an observer would be attached to the simulator via an instrumenter and gets notified whenever the model state has been changed. It then compares the current time with the next snapshot time; if the former exceeds the latter, the observer writes the last stored state into a preconfigured data sink (e.g., a file or database).

Wrapping the above up, listings 5.1 and 5.2 show two simple experiment definitions in *JAMES II*. Without going much into detail, the first one represents a full factorial model parameter scanning for testing how an algorithm scales when increasing the model size: the number of initial enzyme and substrate particles for the enzyme-substrate catalyzation process is varied in a range from one to ten (lines 12, 14). Setting the replication count to 100 (line 6) results in a total of 10000 runs for the entire experiment. To reduce the execution time, line 31 instructs *JAMES II* to adaptively assign more replications to promising algorithms; an *AdaptiveSimulationRunner* also distributes the replications among available computational resources (e.g., CPUs on a multi-core chip), so this is one implementation of a "parallelization across a simulation", which has been discussed briefly in Section 3.1.3.

The second listing actually creates several experiments and executes them sequentially. In this example the impact of auxiliary data structures on the performance of an algorithm is observed. Every combination of an event queue and a random number

generator offered by *JAMES II* (lines 3 and 5) is tested and the execution times (100 replications for each setup) stored for further analysis (line 37).

```
1  public static void main(String[] args) throws Exception {
2    // Basic setup
3    BaseExperiment e = new BaseExperiment();
4    e.setModelLocation(new URI("file-sr://localhost/home/user/models/↩
5      EnzymeSubstrate.sr"));
6    e.setRepeatRuns(100) ;
7    // Add model parameters that should be modified during the experiment
8    // Full factorial: E = (1,2,...,10), S = (1,2,...,10)
9    List<ExperimentVariable<Integer>> variables = new ArrayList<↩
10     ExperimentVariable<Integer>>();
11   variables.add(new ExperimentVariable<Integer>("E", 1,↩
12     new IncrementModifierInteger(1, 1, 10)));
13   variables.add(new ExperimentVariable<Integer>("S", 1,↩
14     new IncrementModifierInteger(1, 1, 10)));
15   exp.setupVariables(variables);
16   // Set-up database connection
17   SimSpExPerspective.setDbConnectionData(new DBConnectionData(↩
18         "jdbc:mysql://localhost/param_scanning", "user", "password",↩
19         "com.mysql.jdbc.Driver"));
20   // Choose simulator (optional)
21   e.getParameters().getParameterBlock().addSubBl(
22     ProcessorFactory.class.getName(), ↩
23     NextReactionProcessorVarBFactory.class.getName());
24   // Set stopping criterion: the simulation interval shall be [0,5]
25   ParameterBlock stopParameters = new ParameterBlock(↩
26     SimTimeStopFactory.class.getName()).
27     addSubBl(SimTimeStopFactory.SIMEND, 5.0);
28   e.getParameters().getParameterBlock().addSubBl(
29     SimulationRunStopPolicyFactory.class.getName(), stopParameters);
30   // Use all available cores for parallel replication
31   e.setSimRunnerFactory(new AdaptiveSimulationRunnerFactory());
32   // Set-up performance database recorder
33   PerfDBRecorder perfDBRecorder = new PerfDBRecorder();
34   e.getExecutionController().addExecutionListner(perfDBRecorder);
35   // Execute experiment
36   perfDBRecorder.start();
37   e.execute();
38   perfDBRecorder.stop();
39 }
```

Listing 5.1: Example experiment definition for model parameter scanning.

This section shall be concluded by briefly giving the hardware specifications for the experiment testbed. All performance measurements have been conducted on a Windows XP 64-bit workstation with two 2.5-GHz QuadCore Xeon Processors and 8 GB RAM. To minimize bias from external load, no more than 6 simulation runs are executed concurrently — two of the eight available cores were idling at any given time, and could therefore handle any load from the operating system etc. Furthermore, each sequential experiment was assigned to a single dedicated core, to further reduce bias from thread-switching and caching. A single core of the workstation achieved a

composite score of 528 for the (large) Java SciMark2 [Sci] benchmark, executed with the 64-bit version of the JDK 1.7 (beta).

5.5 Summary

This chapter discussed several central aspects of conducting a performance study. It has been defined what "performance" in the context of algorithms means: it is a collective term for a number of facets, e.g., CPU, memory, or network load, each of which can be analyzed for itself — thus comparing the performance of algorithms actually involves comparing certain facets. Two of these facets are the subject of almost all evaluations found in the literature on SSA: execution speed, i.e., how long it takes for the algorithm to finish a certain simulation time interval, and accuracy, i.e., how close the produced results come to the true value. Quantifying the first can be done by measuring the elapsed time since the start of the simulation run; however, differences in the underlying hardware and algorithm implementations have to be considered; furthermore, algorithms often depend on certain parameters or sub-algorithms, so analyzing their design space is a must.

The accuracy comparison has been based on the fact that the results of two stochastic simulation algorithms are similar if the null hypothesis H_0 that the generated trajectories are samples coming from the same distribution cannot be rejected. A first approach uses statistical tests to decide whether to accept H_0 or not; it has to be considered that the state variables may be correlated, so an iterative test procedure has been presented that starts with the marginal distributions and continues with higher dimensional random variables if H_0 is not rejected. An alternative method was devised for spatial algorithms that is based on a Chi-square test and outputs a single test statistic per iteration.

Section 5.3 discussed the advantages and disadvantages of using benchmark models instead of representations for real-world problems in an evaluation study. Though not of direct biological interest, benchmark models have several benefits, e.g., they are parameterizable and can be designed to reflect specific model characteristics (fast or slow diffusions or reactions, homogeneous or inhomogeneous particle distribution etc.).

The last section finally brought all of the mentioned aspects together and outlined the basic ingredients of a performance evaluation study. Further points are added for consideration, e.g., how to balance between the number of replications and the confidence in the results or how to define stopping criteria for simulations; one way

to achieve a compromise for the former is to use an adaptive simulation runner that executes more replications for algorithms whose results (execution speed, accuracy) seem promising. However, the time invested to perform a more detailed analysis for the good algorithms comes at the cost of limiting the number of runs for slow configurations; in other words: the interest in those is dropped early during the evaluation and the resources are relocated to more interesting setups.

The chapter concluded with two examples for how to define experiments using the *JAMES II* framework.

```
1  public static void main(String[] args) throws Exception {
2    List<EventQueueFactory> eqfList = SimSystem.getRegistry().↩
3      getFactories(EventQueueFactory.class);
4    List<EventQueueFactory> rngfList = SimSystem.getRegistry().↩
5      getFactories(RNGGeneratorFactory.class);
6
7    // Set-up database connection
8    SimSpExPerspective.setDbConnectionData(new DBConnectionData(↩
9           "jdbc:mysql://localhost/alg_scanning", "user", "password",↩
10          "com.mysql.jdbc.Driver"));
11
12   for (EventQueueFactory eqf : eqfList)
13     for (RNGGeneratorFactory rngf : rngfList) {
14       // Basic setup
15       BaseExperiment e = new BaseExperiment();
16       e.setModelLocation(new URI("file-sr://localhost/home/user/models/↩
17         EnzymeSubstrate.sr"));
18       e.setRepeatRuns(100);
19       // Set up algorithm parameters
20       ParameterBlock parameters = new ParameterBlock(↩
21         NextReactionProcessorVarBFactory.class.getName()).↩
22         addSubBl(RNGGeneratorFactory.class.getName(),↩
23           rngf.getClass().getName()).↩
24         addSubBl(EventQueueFactory.class.getName(),↩
25           eqf.getClass().getName());
26       // Choose specific simulator (optional)
27       e.getParameters().getParameterBlock().addSubBl(
28         ProcessorFactory.class.getName(), parameters);
29       // Set stopping criterion: the simulation interval shall be [0,5]
30       ParameterBlock stopParameters = new ParameterBlock(↩
31         SimTimeStopFactory.class.getName()).
32         addSubBl(SimTimeStopFactory.SIMEND, 5.0);
33       e.getParameters().getParameterBlock().addSubBl(
34         SimulationRunStopPolicyFactory.class.getName(), stopParameters);
35       // Use all available cores for parallel replication
36       e.setSimRunnerFactory(new AdaptiveSimulationRunnerFactory());
37       // Set-up performance database recorder
38       PerfDBRecorder perfDBRecorder = new PerfDBRecorder();
39       e.getExecutionController().addExecutionListner(perfDBRecorder);
40       // Execute experiment
41       perfDBRecorder.start();
42       e.execute();
43       perfDBRecorder.stop();
44   }
45 }
```

Listing 5.2: Example experiment definition for sub-algorithm scanning.

6 Performance Evaluation Studies for Non-spatial and Spatial Simulation Algorithms

Several stochastic simulation algorithms have been implemented as plug-ins for *JAMES II*, which now offers a wide range of methods that can be exploited to simulate either spatial or non-spatial reaction network models. But diversity is a two-sided coin: though it provides increased flexibility and may equip the user with a good set of tools to handle the majority of his problems, he still has to *select* one of those alternatives before starting any simulation runs. Without some measurement or assessment of how good an algorithm is when faced with a certain problem instance, the inexperienced user is left alone and may be overstrained by the tool originally built to support his work.

But the diversity does not stop at the level of simulation algorithms. Only a few publications go into details about sub-algorithms, e.g., what event queue or random number generator has been used for the study. But is it justified to leave those details out? *JAMES II* provides the capabilities to find answers to this question. Algorithms can be parameterized, but not only by giving values for numerical features, e.g., the error control parameter in τ-leaping variants; entire sub-algorithms, implemented as plug-ins and independently executable, can be exchanged without additional effort, which makes it convenient to assess their impact on the performance of the host algorithm.

Based on the concepts outlined in Chapter 5, the next sections will present the results of performance studies conducted for a selection of those algorithms. The overall structure is the same for both the non-spatial and spatial case. A set of benchmark models is introduced first, each one representing certain characteristics of real world models. Those are then taken as the input for the actual experiments, whose aim is to evaluate the speed at which the algorithms perform their tasks and the quality of the

produced results.

6.1 Performance Analysis for Non-spatial Algorithms

The algorithms under scrutiny are the classic Direct Reaction Method (DM) and its optimized variant (ODM), the First Reaction Method (FRM) and its spiritual successor, the Next Reaction Method (NRM), as well as the modified τ-leaping version presented by Cao et al. in [CGP06]. Each method needs a source of random numbers; additionally, the NRM uses an event queue to keep track of the reaction that is about to fire next.

Six *JAMES II* plugins for random number generation have been used: the default Java RNG, a custom implementation of a linear congruential generator (LCG) with the same parameters as Java's RNG, the Mersenne Twister [MN98], a recursion-with-carry generator (Marsaglia's mother of all RNGs [Mar95]), ISAAC (a cryptographically secure RNG, [Jen96]), and RANDU, which is a classical LCG that is not of practical relevance any more, due to its strong correlations. Generating random numbers for stochastic simulations is not a trivial task, since the pseudo-random numbers may correlate and therefore bias a stochastic simulation [Hel98, Gra93]. This is particularly dangerous when RNGs are poorly initialized [MWKA07]. However, the SSA variants do not rely on high-dimensional tuples of random numbers, so that correlations should be very rare. Therefore, studying the runtime performance was the main focus of the RNG-related investigations. Another important aspect of RNGs in the context of stochastic simulation is the size of their seed, as it limits the maximal number of trajectories that can be generated [Mar03]; for the following evaluation, all RNG seeds are of type `long`.

Though more are actually available, the set of event queues has been restricted to a simple list implementation (SIMPLE), a heap event queue (HEAP — proposed by Gibson and Bruck in [GB00]), an MList with an additional variant for faster event time updates (MLIST/MLISTRE — see [GT03]), and a TwoList implementation [BHP81] (here the implementation from the `TwoList2` plug-in of *JAMES II* is used).

6.1.1 Benchmark Models

Evaluation studies are often presented alongside new SSA variants to illustrate the merits of the method. Unfortunately, most evaluations only consider a rather narrow

spectrum of benchmark models, e.g., a single model in [GB00]. This brings about a potential bias that arises from focusing on such models that are easy to solve for the proposed algorithm. In [CLP04], Cao et al. introduce two benchmark models for comparing a new SSA variant to existing ones, with the emphasis laid on answering the question how the execution speed scales with the initial state and the number of reactions and their interdependency. For the sake of comparability and reproducibility, both models are also included here. In total, three models are used as part of this study: the *Linear Chain System* (LCS), the *Totally Independent System* (TIS), and a *Cyclic Chain System* (CCS); the first two, LCS and TIS, were taken from the mentioned publication [CLP04]. Note that for the experiments the structure of a model, i.e., the number species and reactions and the dependency between the latter, and the initial state has been changed, but not the reaction constants; this was done to limit the number of independent variables that have to be varied during a simulation.

Totally Independent System (TIS)

$$R_i : S_i \xrightarrow{c_i} \emptyset,\ i \in [1, N]$$

The default parameters are $N = 600$, $c_i = 1.0$, and $X_i(0) = 10000, i \in [1, N]$. Each reaction changes only the amount of its own reactant, hence there are no dependencies between reactions. As a result, only a single propensity update is required in each iteration, independent from the number of species present in the model; any algorithm using a dependency graph structure should have an advantage compared to methods that re-evaluate every propensity.

Linear Chain System (LCS)

$$R_i : S_i \xrightarrow{c_i} S_{i+1},\ i \in [1, N-1]$$

If not specifically given, the parameters are set to $N = 601$, $c_i = 1.0, i \in [1, N]$ and $X_1(0) = 10000$. The N species can react via $N - 1$ reaction channels, with the product of reaction R_j participating as reactant for R_{j+1}. This model describes a loosely coupled system with all reactions except for R_{N-1} affecting two propensity values — a firing of R_{N-1} only requires an update of $a_{N-1}(\mathbf{x})$, while for any other reaction $R_j, j \in [1, N-2]$, an additional update of $a_{j+1}(\mathbf{x})$ is necessary. Considering this, the impact of a dependency graph should increase with the number of species.

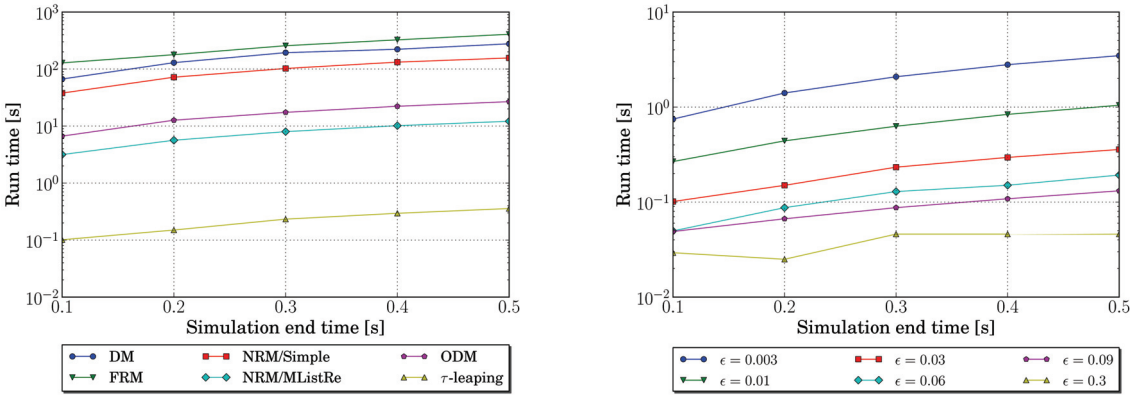
Cyclic Chain System (CCS)

$$R_i : \sum_{j=0}^{k} S_{(i+j) \bmod N+1} \xrightarrow{c_i} \sum_{j=k+1}^{2k+1} S_{(i+j) \bmod N+1},\ i \in [1, N], k \in [0, \lfloor N/3 \rfloor]$$

This model can be additionally parameterized with a value for k, which determines the coupling of the system such that a reaction execution affects $\min\{3k+2, N\}$ propensity values. Per default, $k = 2$, $N = 10$, and $X_i(0) = 10000, i \in [1, N]$, so for this setup a firing causes an update of 8 other reaction propensities. Two special cases are models with $k = 0$, which represents a loosely coupled LCS (with an additional reaction $S_{N-1} \xrightarrow{c_{N-1}} S_1$), and $k = \lfloor N/3 \rfloor$, a totally coupled system in which each propensity needs to be updated if any reaction fires. As a result, the impact of the dependency graph can be adjusted by using different values for k; smaller k generate models that should perform better with algorithms using the graph, while for larger k the benefit decreases as more and more reactions need to be updated. Due to the structure and the initial state of the default model, the τ-leaping algorithm is expected to yield the best results: at the start of the simulation, the propensities for each reaction are the same and if each reaction fires once, the state will not change — in other words, the system has reached its equilibrium state, which allows τ-leaping to make large jumps.

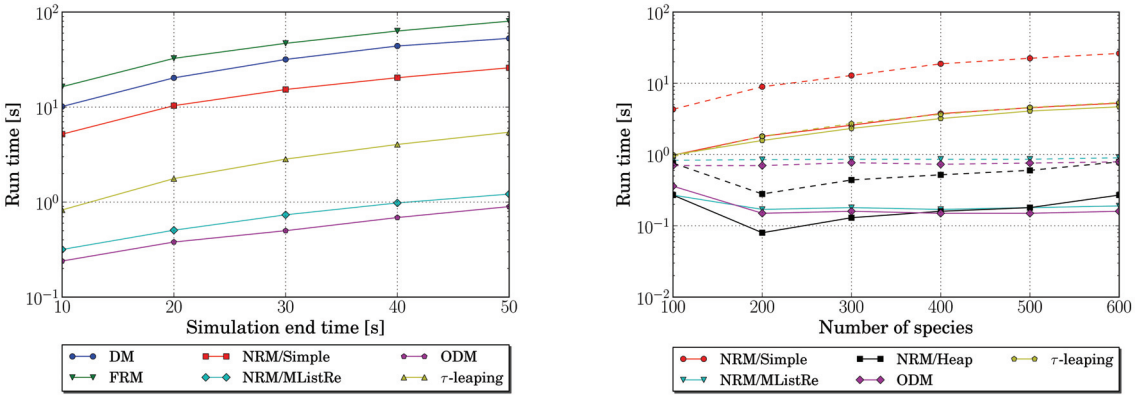
6.1.2 Results

Overall Execution Speed Figures 6.1 to 6.4 summarize the results from performance experiments that measured the execution speed of algorithms when faced with different model setups. The τ-leaping algorithm performs best for the default TIS and CCS models, while both the ODM and the NRM with the MListRe have shorter execution times in case of the LCS model. Figure 6.1 suggest a classification of the algorithms into three groups: the τ-leaping algorithm with execution times ranging from 0.1 s to 0.35 s, the ODM and NRM/MListRe in the mid-range from 3.16 s to 26.8 s and the slowest configurations, the DM, FRM, and NRM/Simple, with execution times starting around 37.85 s seconds and going up to 407.5 s. The speedup from using the best configuration compared to the second best is about 24, from best to worst it is even three orders of magnitude. One important parameter for each leap method is ϵ, the error control parameter. A more detailed discussion will follow below, but Figure 6.1b already shows how the run time increases for larger values of ϵ. In general, this can be observed for every benchmark model.



(a) (b)

Figure 6.1: Results for the *Totally Independent System*. (a) The τ-leaping algorithm performs best with this type of model, followed by the NRM parameterized with the MListRe event queue. (b) While, as expected, lower values for ϵ decrease the execution speed of τ-leaping, it still executes faster than the best exact variant.



(a) (b)

Figure 6.2: Results for the *Linear Chain System*. (a) Both the ODM and the NRM with the MListRe event queue outperform the τ-leaping algorithm. Problematic for the latter are species near the end of the particle distribution "wave", which contribute with τ candidates that are just above the threshold before switching to an exact SSA variant. Similar to the results from the TIS, there is an impressive gap between the performances of the NRM variants; the speedup at a simulation end time of 50 s is about 21 when using the MListRe. (b) Run times for different model configurations. Continuous and dashed lines represent results for models with initially 2000 and 10000 S_1 particles, respectively. Though τ-leaping is inferior to both NRM and ODM, it shows that it scales better with the number of initial particles than the other algorithms.

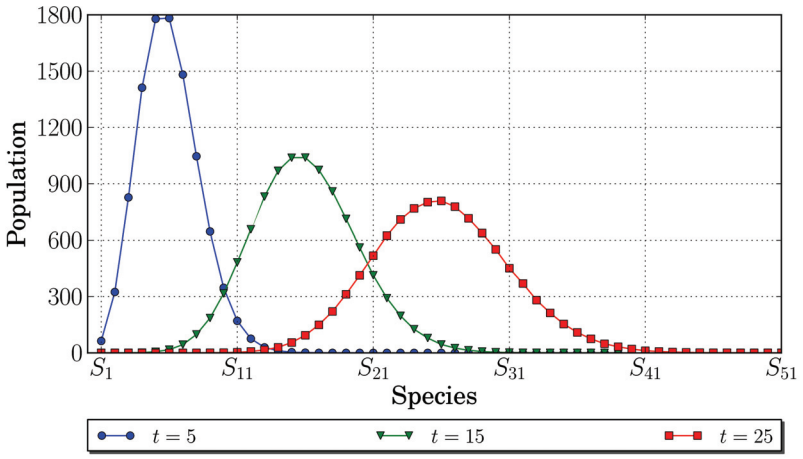
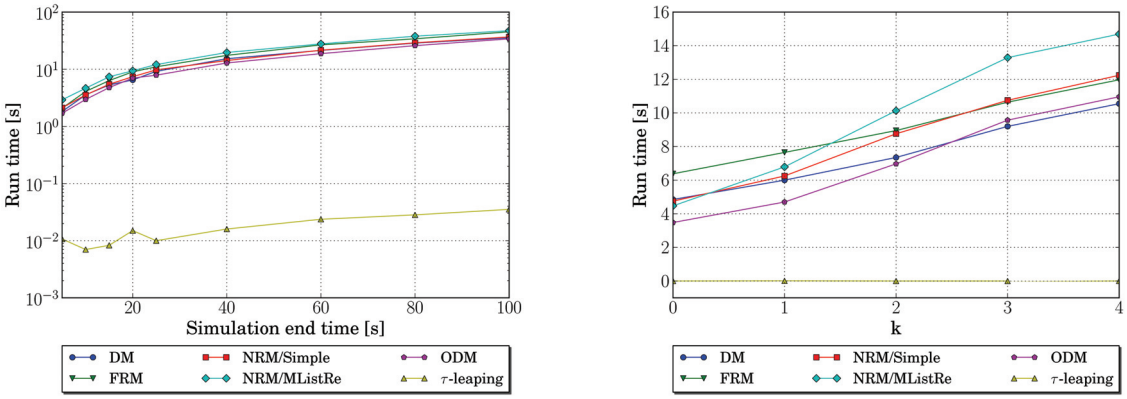


Figure 6.3: Particle distribution of the LCS model after 5 s, 15 s, and 25 s, averaged over 100 replications. The initial number of S_1 particles is 10000.



(a) (b)

Figure 6.4: Results for the *Cyclic Chain System*. (a) The τ-leaping algorithm performs best for this model type. Surprisingly, the NRM combined with the MListRe event queue, which has proven to be a very good configuration for other models, is slower than any other setup; it is even outperformed by the DM and FRM. (b) As expected, with a larger value for k the benefit of using a dependency graph vanishes. The turnover points are $\hat{k} = 1$ for NRM/MListRe and NRM/Simple and $\hat{k} = 3$ for the ODM. The τ-leaping algorithm is not affected by the parameter; in fact, after each leap all propensities are updated anyway.

Figure 6.2a shows significant gaps between three groups, the first one consisting of the ODM and the NRM/MListRe, the second of the τ-leaping algorithm, and the third one of the DM, NRM, and the NRM/Simple configuration. On average, a simulation run is about 5 times slower when switching from the ODM to τ-leaping; this factor increases to 40 if an algorithm from the third group (DM, FRM, NRM/Simple) is selected instead. Noticeable is the poor performance of τ-leaping compared to the other methods. Looking at the τ candidates reveals that the minimum value is always calculated for a species near the end of the state "wave" (see Figure 6.3). For example, at $t \approx 25$ species S_{37} to S_{39} restrict the leap to a length which is slightly larger than $10/a_0(\mathbf{x})$ (ten times the average time until the next reaction firing) — which means that τ-leaping *constantly* operates near the border between an approximative and exact execution because this "wave", although it flattens with increasing simulation time, is always present.

For the standard CCS (Figure 6.4) the τ-leaping algorithm clearly outperforms any other configuration by several orders of magnitude. The standard deviation for the slower setups (ODM, NRM/MListRe, NRM/Simple, DM, FRM) ranges from 0.47 s for an end time of 5 s to 6.04 s for 100 s, so switching between algorithms in this set does not provide any significant speedup.

Sub-algorithm and Parameter Dependence The sub-algorithm of interest here is the event queue implementation for the NRM. For the TIS model (Figure 6.1) the execution time is decreased by a factor of about 12.5 when the MListRe is used instead of the much more simple sorted list. A similar result is shown in Figure 6.2b for the LCS model; the speedup in this case is approximately 20. This looks impressive and based on these results one could conclude that the MListRe seems to be the best choice for the NRM. But Figure 6.4 provides a counter example for this statement: it is in fact the *worst* setup for this model type with the default parameters $k = 2$ and 10 species. The reason for this is the high number of reactions that have to be updated after each execution; 8 out of 10 propensity values need to be recalculated and, in case of the NSM, the respective reactions requeued with new next event times — which is apparently more expensive for the MListRe than it is for the simple list.

Regarding the dependence of the execution speed to the choice of the random number generator: Figure 6.5 shows that at least for the utilized benchmark models the difference between the RNG implementations is not significant. Generating random samples was assumed to be one performance bottleneck of stochastic algorithms [GB00], how-

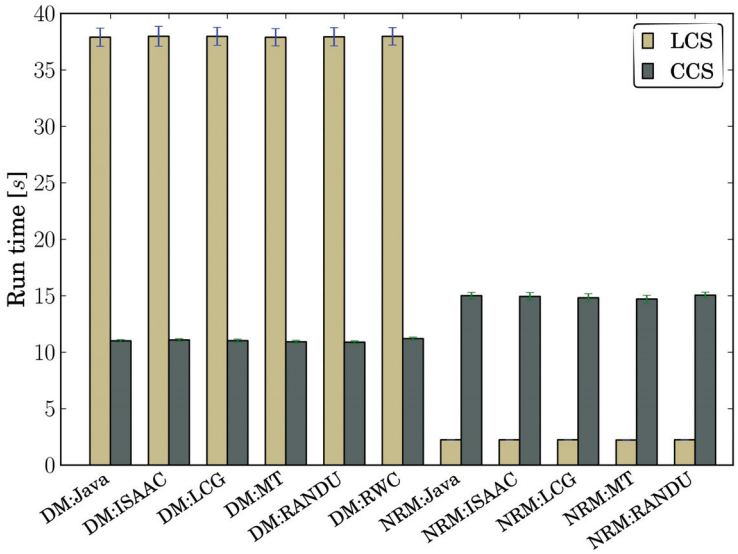


Figure 6.5: Comparing the performance of different RNG implementations.

ever, the results indicate that its influence is only marginal compared to the impact of, e.g., the event queue structure.

The only algorithm requiring a set of additional parameters is τ-leaping. Figure 6.6 shows the execution times for different parameter values; apparently, run time is very sensitive to changes in the error control parameter and n_c (the threshold for deciding whether a reaction is critical or not). Making the former smaller by a factor of ten nearly increases the run time five-fold. On the other hand, setting ϵ to 0.3, i.e., ten times the default value, halves the execution time. Those results could have been expected: ϵ is essentially defining the leap condition by providing the upper bound of the allowed relative propensity change, i.e., for a reaction R_j it must hold that $\Delta_\tau a_j(\mathbf{x})/a_j(\mathbf{x}) < \epsilon$. Increasing this bound permits larger changes without violating the leap condition and, as a result thereof, also larger time steps.

The parameter n_c is used for testing whether a reaction should be considered as being critical or not, based on the current state of its reactants. With a larger value for n_c, more reactions are considered critical and τ-leaping would gradually approach an exact variant with only one reaction getting executed per iteration — this effect can be seen in the last setup of Figure 6.6. But as has been written earlier, smaller n_c could lead to negative state variables after a reaction fired more often than reactants are actually available — while this could be avoided by decreasing the leap value and re-sampling the number of firings, it also means additional operations that all add up

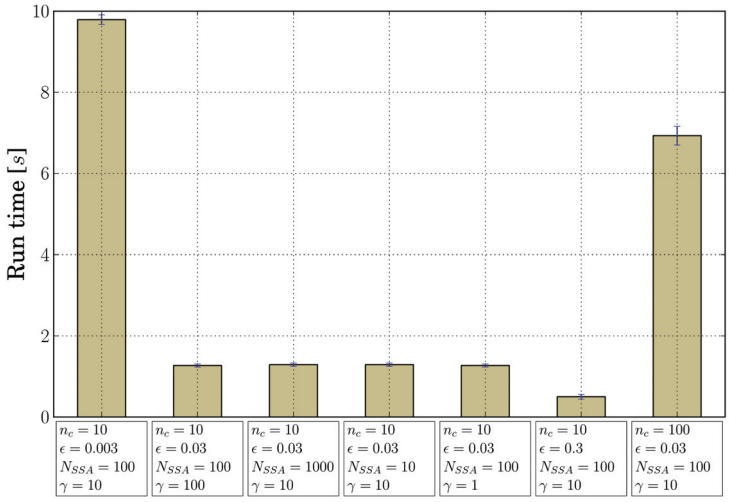
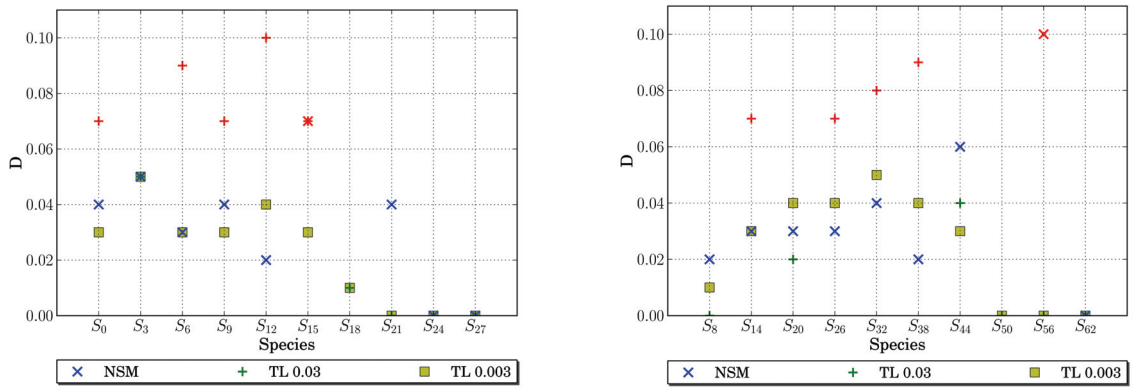


Figure 6.6: Results for τ-leaping algorithm applied to the LCS, with different values for its parameters. Overall execution speed is especially dependent on the error control parameter ϵ and n_c (the latter is used to separate the reactions into critical and non-critical).

to a longer execution time.

Model Parameter Dependence Taking only Figure 6.2a, the ODM seems to be the best choice when simulating models of the LCS type. But these are only results for a single set of model parameters ($N = 600, X_1(0) = 10000$); for Figure 6.2b the number of species and the initial particle count for species S_1 are varied, showing that the performance of τ-leaping is almost independent from these parameters. In contrast, the execution time is increased by nearly a factor of 4 for all other algorithms when the initial concentration of S_1 is raised from 2000 to 10000 particles.

The last paragraph already mentioned the poor performance of the NRM/ MListRe configuration, which could be caused by looking up dependent reactions if most of the propensity values needs to be updated anyway; Figure 6.4b now provides further evidence for this assumption. For the default parameters with 10 species, the maximum value for k is $\lfloor 10/3 \rfloor = 3$. The NRM/MListRe setup is slightly faster than the NRM/Simple and DM for $k = 0$, i.e., an LCS. Let $\hat{k}$ denote the value for k at which the execution time drops below either the DM or FRM time; for both NRM variants, $\hat{k} = 1$, for the ODM the turnover point is reached at $\hat{k} = 3$, i.e., a totally coupled system. So given this example, the DM should be preferred over the NRM as the choice for an



(a) (b)

Figure 6.7: The KS-Test statistic D for several species of the default LCS model at $t = 5$ (a) and $t = 25$ (b). The value is an upper bound for the distance between the empirical distribution functions sampled by the ODM / NRM ($\times$), ODM / τ-leaping with $\epsilon = 0.03$ ($+$), and ODM / τ-leaping with $\epsilon = 0.003$ (square). A red color denotes the cases where the KS-Test rejects the null hypothesis that the samples come from the same underlying distribution. The significance level α is 0.05.

exact algorithm if at least 5 reactions out of 10 need to be updated after an arbitrary firing; for a totally coupled system the DM even outperforms the ODM.

Accuracy The LCS model was used for the accuracy studies because of its simple dynamics: at the beginning of a simulation run the particle distribution over all species takes the form of a spike, located at S_1, so molecules are initially only present for this species. Over time S_1 particles get converted into S_2 molecules, S_2 into S_3 and, more generally, S_i into $S_{i+1}, i \in [1, N]$; as a result, the distribution is "flattened" and its mean is shifted towards S_N (Figure 6.3). Note that the evaluation is based on comparing empirical probability distribution functions obtained by replicated runs of an exact method (the ODM) and the test methods (τ-leaping and NRM); see Section 5.2.2 for details. However, due to the simplicity of the LCS model (only monomolecular reactions) it should be emphasized that it is also possible to find a solution for its CME directly using a method presented in [JU10].

Figure 6.7 and Table 6.1 summarize the outputs of the two-sample KS-Test, with the ODM taken as the exact algorithm. As expected, the NRM accurately captures the state distribution for the majority of the observed species at both time points. However, the null hypothesis is rejected in four cases (S_{15} and S_{18} for $t = 5$, S_{50} and

t = 5							t = 25						
	NRM		τ-leaping $\epsilon = 0.03$		$\epsilon = 0.003$			NRM		τ-leaping $\epsilon = 0.03$		$\epsilon = 0.003$	
S	p	D	p	D	p	D	S	p	D	p	D	p	D
S_0	0.5	0.04	0.01	0.07	0.83	0.03	S_8	0.91	0.02	1	0	1	0.01
S_3	0.26	0.05	0.16	0.05	0.12	0.05	S_{14}	0.72	0.03	0.01	0.07	0.65	0.03
S_6	0.72	0.03	0	0.09	0.83	0.03	S_{20}	0.69	0.03	0.99	0.02	0.37	0.04
S_9	0.29	0.04	0.01	0.07	0.72	0.03	S_{26}	0.76	0.03	0.02	0.07	0.47	0.04
S_{12}	0.98	0.02	0	0.1	0.57	0.04	S_{32}	0.47	0.04	0	0.08	0.16	0.05
S_{15}	0.01	0.07	0.03	0.07	0.65	0.03	S_{38}	1	0.02	0	0.09	0.54	0.04
S_{18}	0	0.43	1	0.01	1	0.01	S_{44}	0.1	0.06	0.29	0.04	0.65	0.03
S_{21}	0.26	0.04	1	0	1	0	S_{50}	0	0.48	1	0	1	0
S_{24}	1	0	1	0	1	0	S_{56}	0	0.1	1	0	1	0
S_{27}	1	0	1	0	1	0	S_{62}	1	0	1	0	1	0

Table 6.1: Outputs from the two-sample KS-Tests to compare the accuracy of the NRM and τ-leaping algorithms (with ϵ set to 0.03 and 0.003) at time points $t = 5$ and $t = 25$ for the default LCS model. The listing shows the p values and the KS statistic D; the null hypothesis is rejected if p is smaller than the significance level α, which has been set to 0.05.

S_{56} for $t = 25$); this occurs for species far away from the distribution mean where only a few states are visited and stochastic fluctuations may have a larger impact. It could be that the number of replications — 1000 per default — is not large enough for these cases; furthermore, it is known that the KS-Test is not the best choice for comparing discrete distributions — it seems to be useful if the distribution is nearly continuous (S_6 to S_{12}), but fails if the domain covers only a few states and discontinuities become apparent (S_{15}).

In contrast, the test rejects the null hypothesis for the majority of the observed species when comparing τ-leaping ($\epsilon = 0.03$) with ODM samples. The reason for this can be seen in Figure 6.8, which displays the normalized state frequencies for the species S_6, S_9, S_{12}, and S_{15} at $t = 5$ as histograms. The red curve represents the normal distribution with mean and standard deviation taken from the ODM samples. While the standard deviation of the τ-leaping samples seem to fit the reference, the mean is slightly shifted towards either a higher (S_6) or a lower (S_9 and S_{12}) population count; a similar picture is drawn for $t = 25$. However, the results look much better for a smaller ϵ value (0.003); for this setup the accuracy of τ-leaping gets very close to an exact algorithm, but this comes at the cost of longer execution times (see Figure 6.6). Summing up, when it comes to execution speed, the approximative algorithm is the best choice for several models. But looking at the accuracy analysis of the LCS

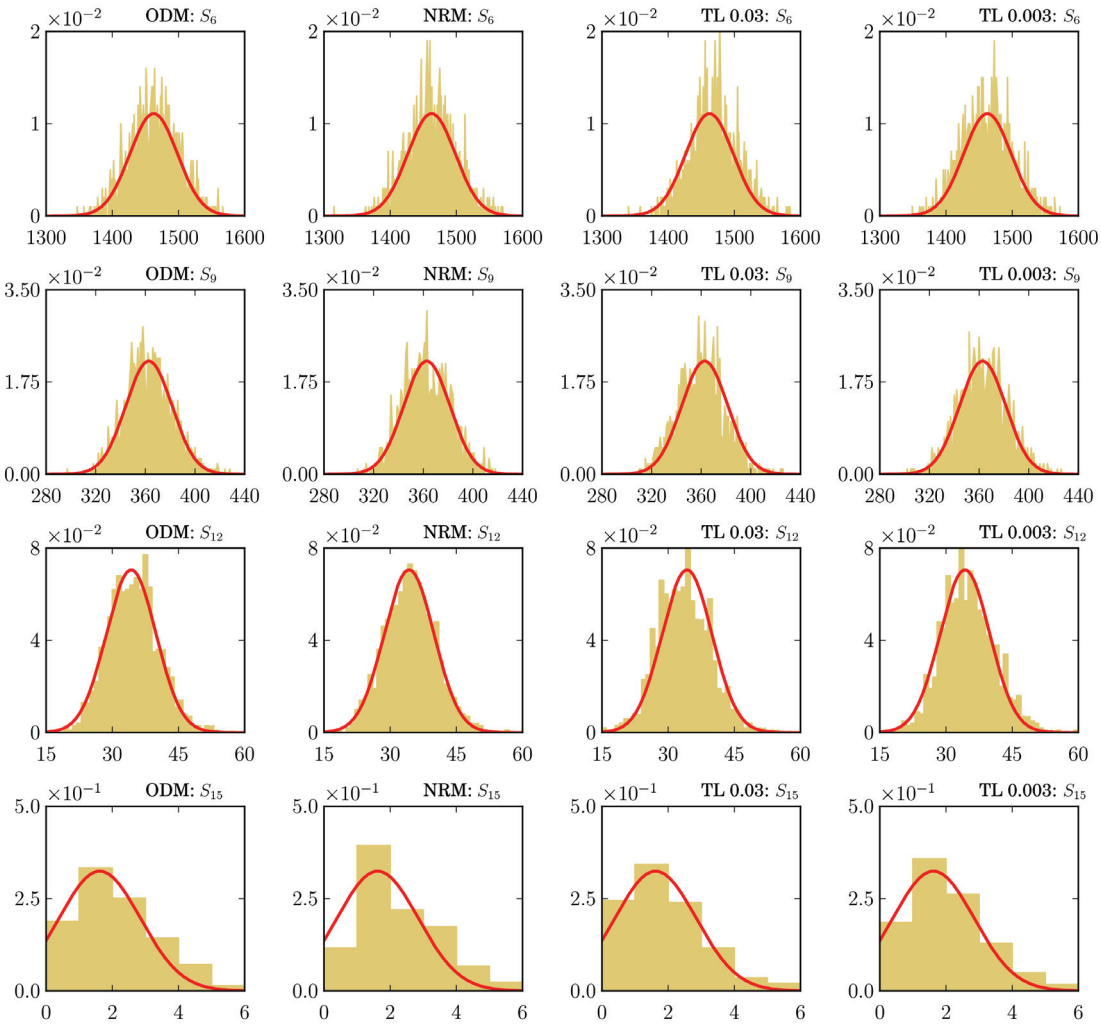


Figure 6.8: Species state distributions for the LCS model at $t = 5$. The mean and standard deviation of the samples generated by the ODM — taken as the reference distribution — is plotted as a red line. Notice how the histograms for the τ-leaping ($\epsilon = 0.03$) algorithm differ for S_6, S_9, and S_{12}; its mean is slightly shifted towards either a larger or smaller value. The last row shows the state frequencies for S_{15}, a species near the "edge" of the distribution (near the end of the wave shown in Figure 6.3). With only six states visited after 1000 replications, the distribution is very sensitive to stochastic fluctuations — as a result, both the NRM and τ-leaping ($\epsilon = 0.03$) get rejected — but not τ-leaping with $\epsilon = 0.003$.

model, the results show a difference in the state distributions — for this model the τ-leaping trajectories for the majority of the species are sample paths of a *different* stochastic process, in comparison with exact variants. At the end a choice has to be made whether to sacrifice accuracy for a smaller run time or not.

6.2 Performance Analysis for Spatial Algorithms

The step from non-spatial methods to spatial methods is generally done by connecting well-stirred sub-volumes and letting particles diffuse from one site to its neighbors. Three algorithms supporting this type of models, the Next Sub-volume Method (NSM), Gillespie's Multi-particle Method (GMPM), and the spatial τ-leaping algorithm introduced in Chapter 4, have been studied for this part of the evaluation. But before continuing with the benchmark models it is necessary to discuss some general aspects of the study.

Apparently, being stochastic algorithms each one requires a source of random numbers, just like in the non-spatial case. Apart from the size and structure of the reaction network, i.e., the number of species and reactions, the rate constants and the dependency between reactions, two additional factors now also characterize a model: the number of sub-volumes and the initial distribution of the particles. Adding more sub-volumes to a model should also increase the run time for each of the algorithms under study. Taking the NSM as an example, with additional sites the respective diffusion and reaction rates get added to the overall rate sum and thus the average time step until the next event will occur is decreased. Similarly, both the GMPM and Sτ need to iterate over more sub-volumes, so the number of operations performed during one algorithm iteration is increased.

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 ($1/\sum_{l=1}^{L}\sum_{i=1}^{N} s_0^l(\mathbf{X})$) 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 $1/\sum_{l=1}^{L}\sum_{i=1}^{N} a_0^l(\mathbf{X})$. However, spatial τ-leaping is a different subject. As it was shown with the theoretical analysis in Section 4.3, the closer a system is to a homogeneous distribution, the better Sτ should perform compared to an NSM execution.

Furthermore, it shall be noted that only the diffusion but not the reaction constants have been varied; for all experiments the latter have been set to fixed values. 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. Similar to the non-spatial evaluation, to limit the number of independent variables it has been decided to concentrate on the *ratio* between diffusion and reaction constants: multiplying the former by a factor of, e.g., 100 and keeping the latter constant shifts the simulation towards more diffusion events.

6.2.1 Benchmark Models

The test models for reaction-diffusion systems are specifically designed to represent different conditions of spatial inhomogeneities in species distribution. Four model types are used, each one with particular characteristics: two variants of a *radial model*, the *molecule generator model* from Section 5.3, and the *protein phosphorylation model*. Common to all models 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 — initially all particles are located in the center sub-volume — to a nearly uniform molecule distribution.

The second variant includes two additional species and a single reaction rule:

$$R_1 : A + B \xrightarrow{c_p} C \tag{6.1}$$

Similar to the first model, species A is located within the center sub-volume only. In contrast, species B is ubiquitously present, i.e., it is distributed homogeneously throughout the volume. Both A and B can diffuse freely. Though both variants of the Radial Model start with A inside a single sub-volume, the first variant 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.

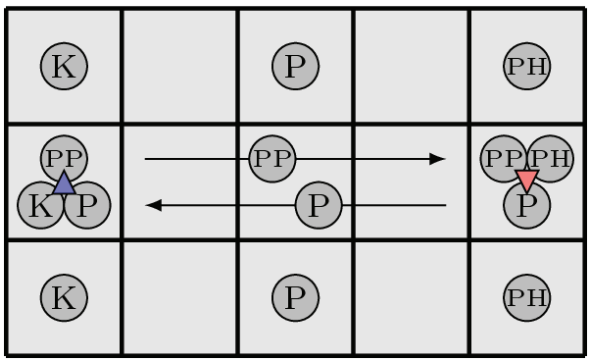


Figure 6.9: The 2D phosphorylation model. Initially, the protein molecules P are located in the center sub-volumes, while kinase K (left) and phosphatase PH (right) are only present at the model boundaries. During reaction R_1, the kinase adds a phosphate molecule to the protein (triangle up), while R_2 removes it via the phosphatase (triangle down).

Protein Phosphorylation Model (Phospho)

$$\begin{aligned} R_1 &: P + K \xrightarrow{c_1} PP + K \\ R_2 &: PP + PH \xrightarrow{c_2} P + PH. \end{aligned} \tag{6.2}$$

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 [BK99]: a protein P is phosphorylated by a membrane-bound kinase K and dephosphorylated by a cytosolic phosphatase PH. Figure 6.9 shows the basic setup: starting in the center sub-volumes, the P molecules diffuse into all directions. Some of them will react with the kinase located at one side of the model grid, resulting in the phosphorylated protein form PP. Eventually, PP will diffuse to the other end of the volume, where a phosphatase PH removes the phosphate again, returning P back to its original state.

6.2.2 Results

Overall Execution Speed Evaluating algorithm performance for reaction-diffusion systems is much harder than assessing algorithm performance for well-stirred systems: the degree of inhomogeneity within the model has to be considered, the level of detail with which the volume is discretized, as well as the extent of diffusion between those sub-volumes. All these characteristics are explored by testing the algorithms on the

benchmark models discussed in the previous section. Each model was subject to factorial experiments that shall illuminate the impact of their parameters on algorithm runtime performance.

More than 150 model setups have been studied, in contrast to the 7 setups that have been used for well-stirred systems (TIS, LCS, and some CCS setups with varying k. cf. Section 6.1.2). This required some precautions regarding the overall experiment design, in order to make the whole study feasible. In particular, three mechanisms of the *JAMES II* experimental layer [HEU08] have been applied:

- The execution times often vary by several orders of magnitude, even on the same model setup. To avoid overly long execution times, all executions are *censored* (i.e., aborted) that exceeded a predetermined timespan. If not stated otherwise, the maximal admissible duration for a single run was set to 1200 seconds, i.e., 20 minutes. Censored runs will be marked by an asterisk (*) in the graphs.

- Execution times do not only vary strongly between simulation configurations, but also between different parameterizations of the same model. To avoid a time-consuming trial-and-error searches for acceptable simulation end times, a simple [EU09] calibration approach is employed that automatically determines a simulation end time in the interval of $[10^{-2}, 10^{2}]$ seconds, so that NSM approximately runs for 300 seconds. If no simulation end time is given, this mechanism has been used to determine a good end time for each model setup individually.

- A custom technique for sequential experiment design [Rob52] has been used to adaptively replicate faster algorithms more often than others [ELU09]. While the policy initially replicates each simulation configuration four times (i.e., even the execution of a previously censored configuration gets repeated that often), several remaining replications are then allocated to the faster algorithms by a biased random selection. This scheme allows to invest additional computing time to efficiently determine which of the algorithms is likely the best choice. However, as can be seen below there is little doubt about the algorithm with generally superior runtime performance.

Only four replicated runs for each setup seems very little, so the standard deviation of all replicated run times was also checked, but found to be in reasonable limits: for all runs longer than a couple of seconds, it was much lower than 10% of the execution time. For the sake of clarity, the standard deviation is omitted and only averaged results are shown.

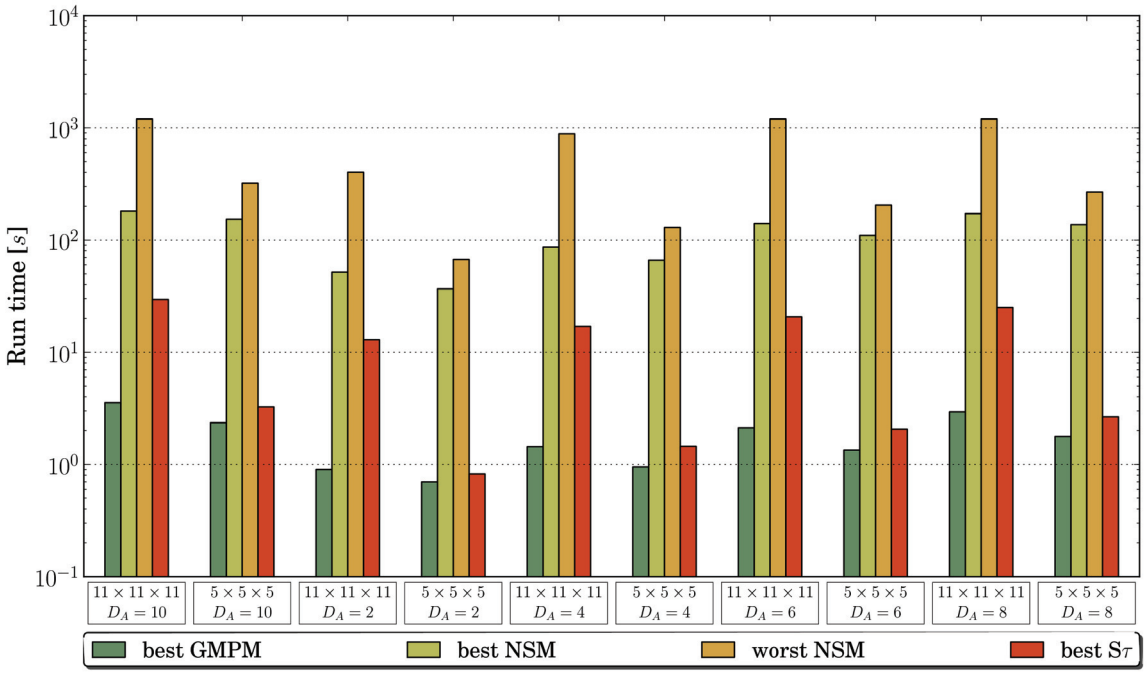


Figure 6.10: Execution times for the Molecule Generator model with 10^6 C particles in centre sub-volume. The diffusion rate D_A has been varied from 2 to 10 and the model size is either $5 \times 5 \times 5$ (125 sub-volumes) or $11 \times 11 \times 11$ (1331 sub-volumes). All replications ran for 3.0 s.

While both GMPM and NSM were combined with the same event queue implementations used for the SSA evaluation on well-stirred systems, $7 \times 6 \times 5 = 210$ parameter setups for the Sτ algorithm were defined: all combinations of $n_c \in \{2, 3, 4, 5, 7, 9, 10\}$, $\epsilon \in \{0.03, 0.034, 0.038, 0.042, 0.046, 0.05\}$, and $N_{SSA} \in \{10, 20, 30, 40, 50\}$. Several publications mention what parameter sets turned out to be suitable during their study. The values here have been chosen to represent a common range of configurations, as suggested in various literature, e.g., [CGP06, San09b].

Figure 6.10 presents some rather encouraging performance results for Sτ, which consistently outperforms NSM but in turn is outperformed by GMPM. It was expected that GMPM will be faster than Sτ because its strong approximations cut much computational load. Another interesting aspect of the results in Figure 6.10 is the impact of the event queue implementation on NSM performance. If the best Sτ configuration is compared to the best NSM configuration, it outperforms it usually by about one order of magnitude. If, however, the *worst* NSM configuration is taken, the comparison gets even more impressive. This clearly shows two important points: a suitable event

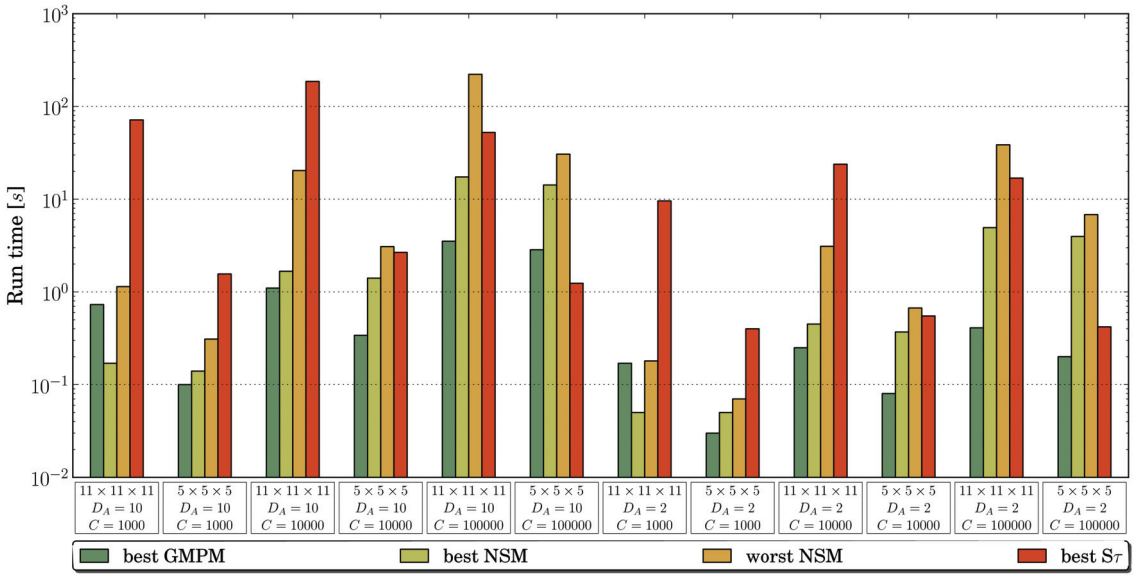


Figure 6.11: Execution times for the Molecule Generator model setup from Figure 6.10 ($D_A \in \{2, 10\}$), but now with 10^3 to 10^5 particles in the centre sub-volume.

queue is essential for NSM 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 setups given above (and almost all others) the `MListRe` event queue from *JAMES II* was found to yield the best results.

Finally, Figure 6.10 also shows that the performance of Sτ strongly depends on the model size: the Sτ run times between otherwise equal 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 (here, simulation end time was fixed at 3.0 seconds for all setups).

After now having gained some confidence in the favorable run times of the new Sτ algorithm, it may seem safe to move forward and check its accuracy with respect to NSM. But things are more intricate: the whole 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 problems where it performs *worse* than its competitors.

Figure 6.11 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 Figure 6.10). The results for $D_A \in \{4, 6, 8\}$ are similar. Here, it becomes apparent that Sτ

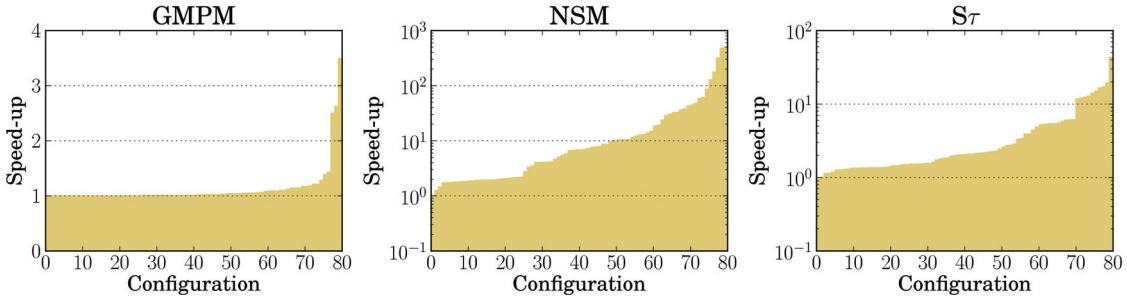


Figure 6.12: Sorted speed-ups attainable per algorithm for 80 Molecule Generator and Radial model set-ups. Note the different scales of the logarithmic y-axes.

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 ($> 10^5$). The differing diffusion rate 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 the size of the models, i.e., the difference between best and worst NSM configuration increases. It should also be noticed that for some model setups ($X_C(0) = 1000$, size $= 11 \times 11 \times 11$) NSM is even *faster* than GMPM, even though it does not use any approximations. This rather counter-intuitive finding again illustrates the importance of model features. Here, the effect of D_A is again negligible. Figure 6.11 also backs up what Figure 6.10 already suggested: Sτ is faster than the worst NSM configuration in six out of twelve setups, but in four of those setups Sτ is slower than the best NSM configuration. Experiments neglecting event queue implementation specifics for NSM would not yield any meaningful result in these cases. Of course, the design space of simulation configurations must not only be explored for NSM, but also for GMPM and Sτ. To do so, the potential speed-up per reaction-diffusion algorithm has been analyzed, i.e., the execution time ratio between its fastest and its slowest configuration. This was done for 79 setups of the Molecule Generator and Radial models; the results are shown in Figure 6.12. 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 — the performance difference between the best and the worst simulation configuration with GMPM was usually less than 10% of the fastest execution time, i.e., the speed-up achievable by choosing a different GMPM setup was usually below 1.1. However, this observation only holds for the models that have been analyzed in this present study; GMPM's event queue implementation may still be relevant for much larger problems. For the problems considered so far, 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. Figure 6.10 and Figure 6.11). For several models, a speed-up over 50 was observed, i.e., 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 optimal in all other cases. The results also showed that the simple event queue implementation is by far the worst choice. Therefore, the design space explored in the following experiments was restricted to the NSM and GMPM configurations that use the MListRe queue; it performs well on all model setups and within both NSM and GMPM.

For Sτ, Figure 6.12 shows that there are some cases in which its 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. It was further analyzed how the individual parameters n_c, ϵ, and N_{SSA} influence the execution speed: of the 70 model setups for which the best Sτ configuration needed more than 1 second to simulate, the optimal configuration 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, it has been observed that the choice of ϵ has *little* impact on runtime performance: e.g., although ca. 84% of the best configurations had ϵ set to 0.03, this is also true for the worst 70 % percent. This is somehow surprising, as the value of ϵ had a considerable influence on the execution speed of the non-spatial variant. One answer can be found in the theoretical analysis in Section 4.3, which showed how the leap size depends on the state of the model. Furthermore, it should be remembered that ϵ is only used to determine τ for the non-critical reactions. In case for systems with a high level of heterogeneity in the particle distribution, there are many sub-volumes that have critical events, even if no reactions can take place — because diffusion events are also classified into critical and non-critical.

The run times for a single setup usually cluster around the different n_c values only. As neither any impact from N_{SSA} could be measured, the number of simulation configurations for Sτ has been reduced to four: a default one with $n_c = 10$, $\epsilon = 0.03$, and $N_{SSA} = 10.0$, and one adjusted configuration for each parameter, i.e., one with $n_c = 2$, one with $\epsilon = 0.05$, and one with $N_{SSA} = 50.0$. Still, none of the following experiments gave results that indicate a strong impact of ϵ or N_{SSA}. All in all, the design space has been reduced from over 200 simulation configurations to just six: NSM + MListRe, GMPM + MListRe, and the four Sτ configurations mentioned above. This greatly

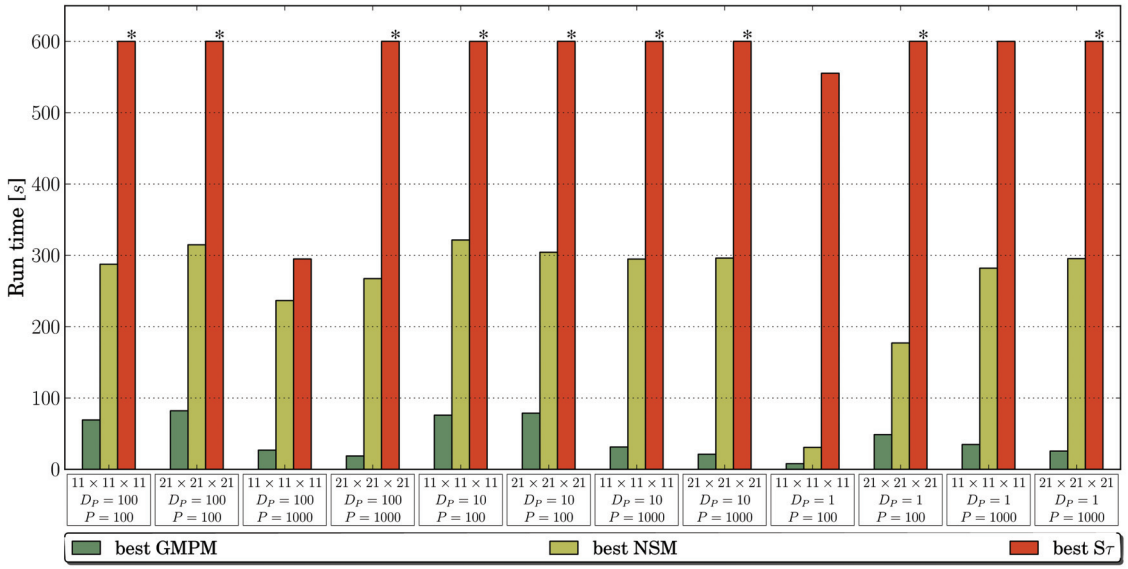


Figure 6.13: Algorithm performance on different set-ups of the phosphorylation model. Similar results were obtained for $X_K(0) = X_{PH}(0) = 100$ instead of 1000. Simulation times vary across model set-ups, as their complexity differs strongly (they have been calibrated automatically to let NSM run for $\approx 300s$, configurations with $D_P = 1, X_P(0) = 100$ where so easy that even simulating until time 100 did not impose greater load for NSM). Unfinished executions were aborted after 10 minutes.

facilitated the exploration of the problem space.

Figure 6.13 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 algorithm by far. Note that only for two problems Sτ could finish in time, both of which are of moderate size ($11 \times 11 \times 11$). Again, it is only comparable to NSM performance if the number of particles is sufficiently high and there are many diffusion events.

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 such scenarios for *both* NSM and Sτ. As Figure 6.14 shows, both algorithms are censored at least once, and in both cases the respective other method was faster by roughly one order of magnitude. Again, Sτ could gain significant speed-up (over 60 in two cases) 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

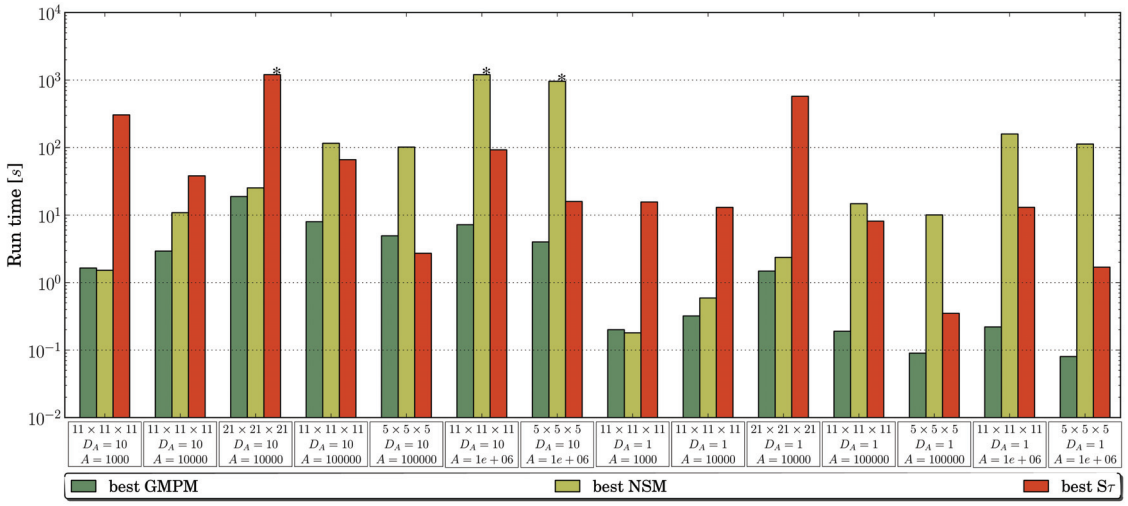


Figure 6.14: Algorithm performance on different set-ups of the Radial model. Sτ outperforms NSM on small to medium models with sufficiently many particles. The simulation end time was set to 5 s.

the results is depicted in Figure 6.15. Compared to NSM, Sτ could achieve a speed-up of up to 6.5 (40 model setups have been used for the comparison). As for the Protein Phosphorylation model, the simulation end time was calibrated to let NSM run for approximately 5 minutes (300 s) — and again, Sτ fares well as long as the model is not too big. Moreover, GMPM is always the fastest algorithm in these scenarios; this performance pattern is preserved through *all* combinations of D_A and D_B shown in figure Figure 6.15.

Accuracy One conclusion from the lasts paragraphs is: the GMPM is almost always the fastest algorithm, leaving its competitors often far behind. The NSM and Sτ algorithm shine for some model setups; they still get beaten by GMPM, but at least for some configurations they can be considered as being on par with it. But the fastest algorithm is worth nothing if it fails to produce accurate results.

Table 6.2 puts the results from the execution speed analysis into perspective. Using the test procedure introduced in section Section 5.2.2, the accuracy experiments show that, at least for the analyzed models, GMPM is far from capturing the spatial particle distribution accurately. The null hypothesis, stating that both NRM and GMPM sample from the same underlying distribution, is rejected for a significance level $\alpha = 0.05$ in each experiment. Figure 6.16 shows a simple visualization of the radial model's state at $t = 1.0$. The size and the color of a box represent the mean and standard

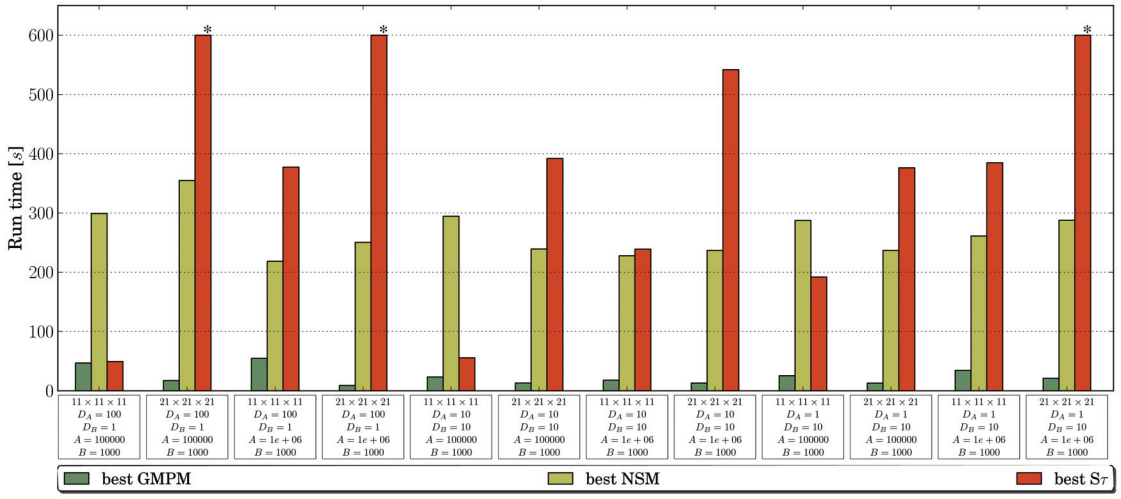


Figure 6.15: Algorithm performance on different set-ups of the Radial 2 model. The results are mixed, Sτ is quite fast for some setups but is beaten by both the GMPM and NSM for the majority of models.

			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, $D_A = 10$,	0.1	1782.16	3129.69	0	1768.64	0.58
$D_B = 1$, $X_A(0) = 10^5$,	0.2	1875.26	3183.37	0	1866.51	0.33
$X_B(0) = 10^3$	0.3	1887.16	3127.39	0	1898.22	0.02
Phospho, $D_P = D_{PP} = 10$,	0.1	738.18	1099.98	0	732.85	0.19
$X_P(0) = 1.21 \cdot 10^5$,	0.5	1112.22	1233.13	0	1099.12	0.34
$X_K(0) = X_{PH}(0) = 100$	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 6.2: 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 $\alpha = 0.05$. As can be seen, GMPM fails to capture the spatial distribution for every test model, while the spatial τ-leaping algorithm shows good performance, except for the molecule generator model.

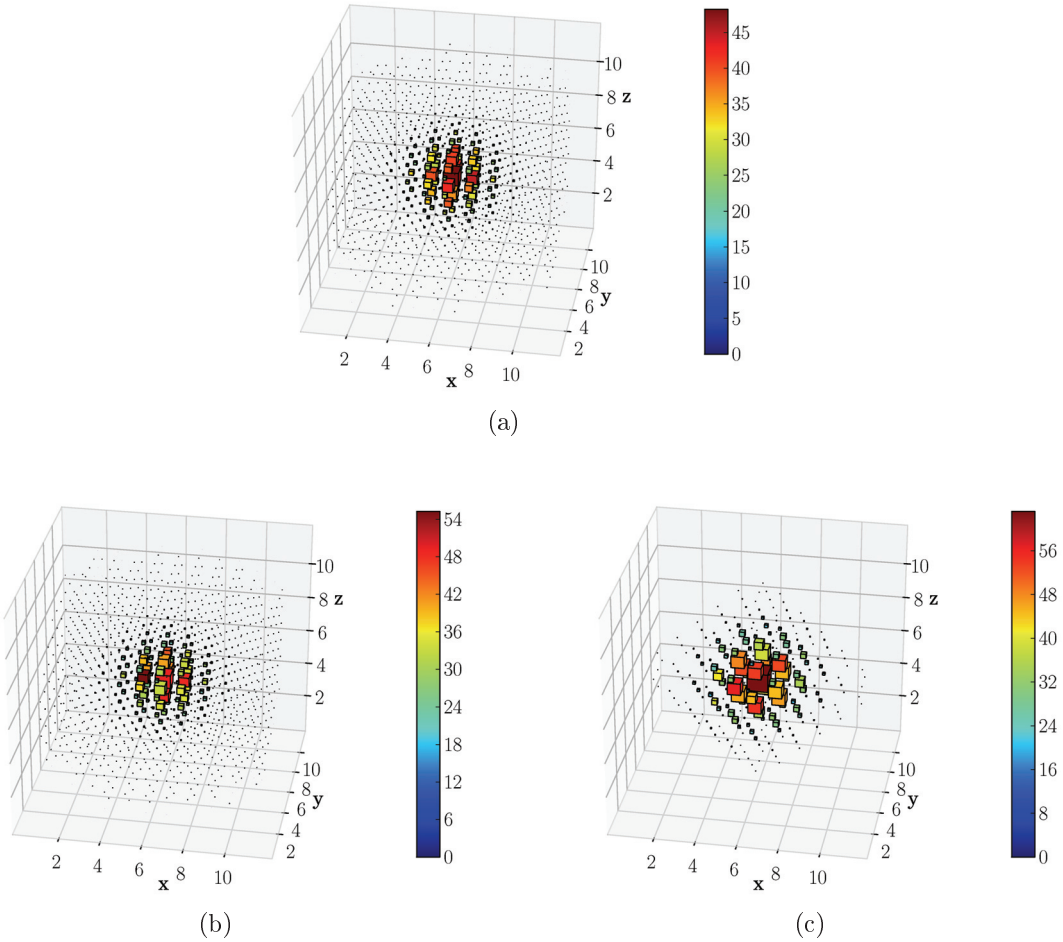


Figure 6.16: Snapshot of the radial model state at $t = 1.0$ for the NSM (a), Sτ (b) and GMPM (c). Parameters are set to $D_A = 1$, $X_A(0) = 10^5$. The size and the color of a box encode the mean and standard deviation of the A particle population in a sub-volume; mean values have been rescaled to fit the interval $[0, 1]$. NSM and Sτ show a good overall agreement, while the result for GMPM looks quite different.

deviation, respectively, of the A particle population inside the sub-volume, calculated for a sample of 50 replications. The results for NSM (Figure 6.16a) and Sτ (Figure 6.16b) look similar, though there are some differences in the standard deviations. Most of the particles are still concentrated around the center, but some managed to diffuse into the outer sub-volumes — after 50 replications nearly all sub-volumes contained at least one A particle

The outcome for the GMPM (Figure 6.16c) differs significantly. Particles are distributed in a three-dimensional "checkerboard" pattern: during each iteration *all* molecules diffuse from a sub-volume into its neighbors, leaving the former empty. Additionally, the particles did not diffuse as far as observed for the other methods; in fact, for this model it is fairly easy to calculate the number L_i of sub-volumes after i iterations that will contain at least a single molecule:

$$L_0 = 1 \tag{6.3}$$
$$L_i = L_{i-1} + 4i + 2\sum_{j=1}^{i-1} 4j + 2$$

With $D_A = 1$, the interval between two diffusion phases is $t_A = 1/6$, so during $[0, 1]$ the GMPM performs 6 diffusions. Inserting this into equation 6.3 gives 377 sub-volumes that are not empty at the end of the interval, roughly one third.

Both the GMPM and Sτ fail for the molecule generator model; the null hypothesis is rejected for every time point. At first glance the state plot for $t = 0.5$ (Figure 6.17) actually shows a good approximation by Sτ compared to the NSM distribution. A more detailed view at the test results revealed that the variance in the number of molecules is high for most of the sub-volumes, so a reason could be the limited replication count. This number has been increased to include 100 samples per experiment, which reduced the variance, but the test still rejects both Sτ and GMPM. To sum it up: despite being slower than the GMPM, the particle distribution simulated by Sτ matches the NSM outcome more closely than its competitor for the two radial variants and the phosphorylation model. Though a visualization shows a good agreement also for the molecule generator, the statistical test concludes that the distributions differ significantly.

As a reminder: it was mentioned in Section 3.3.2 that GMPM does not have any algorithm parameters to control the accuracy; this can only be done by decreasing the interval between diffusion events, which in turn depends on the diffusion constants of

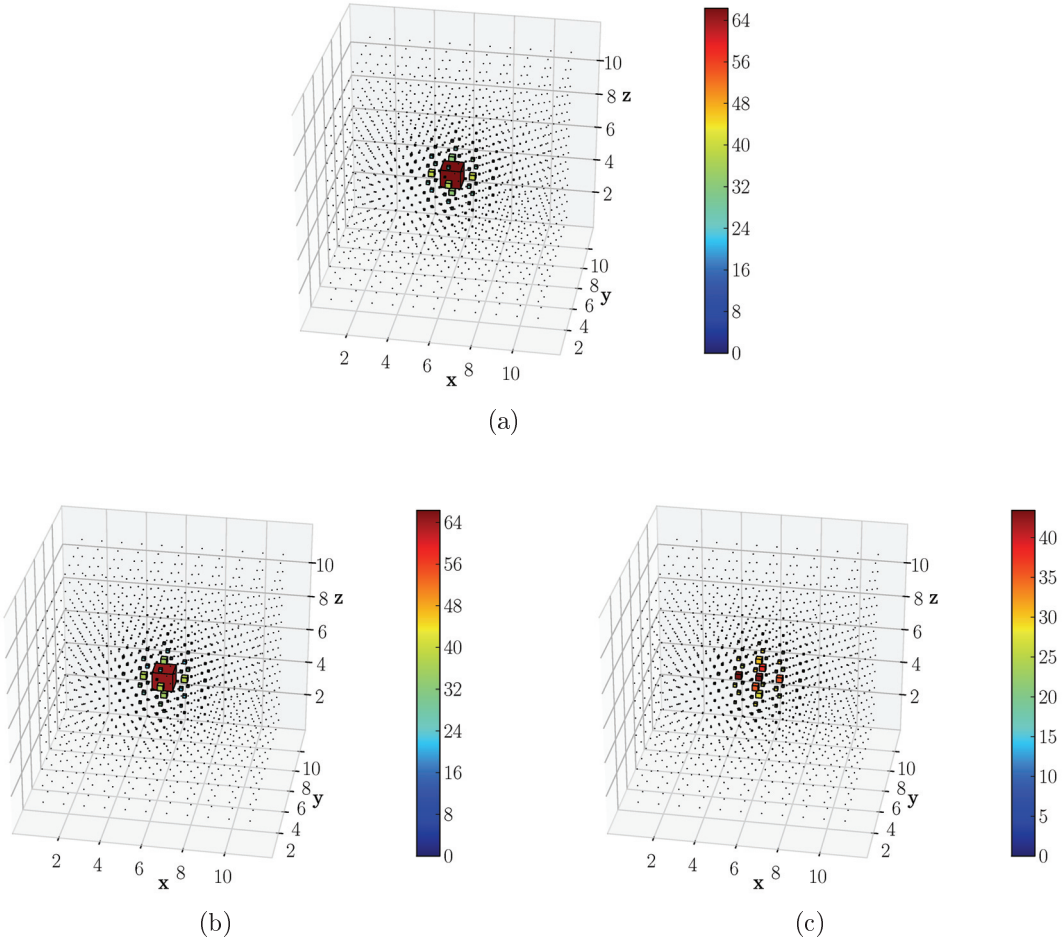


Figure 6.17: Snapshot of the MolGen model state at $t = 0.1$ for the NSM (a), Sτ (b) and GMPM (c). Parameters are set to $D_A = 10$ and $X_C(0) = 10^6$. The size and the color of a box encode the mean and standard deviation of the A particle population in a sub-volume; mean values have been rescaled to fit the interval $[0, 1]$.

Symbol	Feature Description
L	Number of sub-volumes (cf. Section 2.5).
P	Overall number of particles in initial state: $\sum_{l=1}^{L}\sum_{i=1}^{N} x_{l,i}$
$\bar{P}$	Average initial number of particles: $\frac{P}{L}$
$\hat{P}$	Maximum initial number of particles per SV: $\max_{l\in[1,L]}\{\sum_{i=1}^{N} x_{l,i}\}$
D	Sum of diffusion rates: $\sum_{i=1}^{N} D_i$
$\bar{D}$	Average diffusion rate: $\frac{D}{N}$
$\hat{D}$	Maximum diffusion rate: $\max_{i\in[1,N]} D_i$
I	Initial diffusion propensity sum — the factor $\frac{2n}{\lambda_{sv}}$ (cf. Section 2.5) is omitted: $\sum_{l=1}^{L}\sum_{i=1}^{N} D_i \cdot x_{l,i}$.
$\bar{I}$	Average initial diffusion propensity: $\frac{I}{L\cdot N}$
$\hat{I}$	Maximum initial diffusion propensity: $\max_{l\in[1,L],i\in[1,N]}(D_i \cdot x_{l,i})$.
I_σ	Standard deviation initial diffusion propensity: $\sqrt{\frac{1}{L\cdot N}\cdot\sum_{l=1}^{L}\sum_{i=1}^{N}(x_{l,i}-\bar{I})^2}$

Table 6.3: Investigated model features.

the species and the grid spacing λ_{sv}. This dissertation takes the position that changing the latter corresponds to an entire new model, hence doing this only for GMPM makes it impossible to compare the results with NSM and Sτ. That is why the same models have been used for all methods.

Further Analysis of Runtime Performance The results discussed above motivate to refocus on the performance of NSM and Sτ, as GMPM may be not accurate enough for many models and the questions associated with them. Furthermore, the runtime performance analysis suggested that model properties, e.g., the overall number of particles or the number of sub-volumes, have a considerable impact on algorithm performance — not just their overall execution time, but their relative *order*. To find out which model properties determine the relative order of the algorithms, some general features from the 155 model setups that were evaluated w.r.t. execution speed have been extracted.

The features are summarized in table Table 6.3. Some of these are fixed throughout the simulation (e.g., L and D), others refer to the initial state (e.g., P and I). It is reasonable to include features of the initial state into the analysis, as it can be easily inspected before the simulation starts. Furthermore, if a feature of the initial state is decisive for algorithm performance, this could motivate the development of adaptive approaches that observe this feature's fluctuation during simulation execution and change the simulation algorithm accordingly. To account for inhomogeneity within the sub-volumes of a model, the analysis did not just consider sums or averaged features

(e.g., $\bar{P}$ and $\bar{D}$), but also maximum values (e.g., $\hat{P}$ and $\hat{D}$) and a standard deviation (I_σ).

The features defined in Table 6.3 were extracted from all model setups and, together with the results from the performance analysis, fed into the WEKA [WF99] machine learning toolkit to find relationships between model features and the relative performance of the algorithms. Although learning to classify a problem according to the algorithm that is most likely to succeed is not an easy task and requires much more performance data (from more models), such analyses have already been conducted successfully (e.g., [LBNS09]); they may help to discover new aspects of the given algorithms. In this case WEKA's implementation of Quinlan's C4.5 decision tree learning algorithm [WF99, p. 159 et sqq.] has been used. C4.5 it is a supervised machine learning algorithm for classification, i.e., it considers a *training set* of examples and categorizes them into a finite number of classes. Furthermore, C4.5 is able to cope with numerical attributes, so that it can be provided with the model features from Table 6.3. The decision trees generated by C4.5, depicted in Figure 6.18, show which model features are the most relevant for deciding which algorithm ought to perform best on an untested model with given features. A decision tree is read top-down: tree B, for instance, decides to use NSM instead of Sτ if the maximal amount of particles in a sub-volume, $\hat{P}$, is $\leq 10^4$. If $\hat{P} > 10^4$, however, it would check if the number of sub-volumes L is $\leq 11^3$, and so on. Decision trees order attributes by their relevance; for the above example, $\hat{P}$ is more relevant than L, so it appears at a higher node in the tree.

The analysis was focused on two questions: is it possible to predict, given only a model's features (Table 6.3),

a) which algorithm will be the fastest?

b) whether Sτ will outperform NSM?

For both questions, it is assumed that all three algorithms are configured optimally, i.e., only their best-performing simulation configurations are considered. The trees A and B from Figure 6.18 answer both questions, respectively; they were checked with 10-fold cross-validation [WF99, p. 125].

While I, $\bar{I}$, $\hat{I}$, and I_σ showed little correlation with the runtime performance, most other features had some impact on the algorithms' execution speed. The most important features refer to the number of particles, i.e., $P/\hat{P}/\bar{P}$: NSM may outperform GMPM only on 'sparse' models with more sub-volumes than particles (tree A, Figure

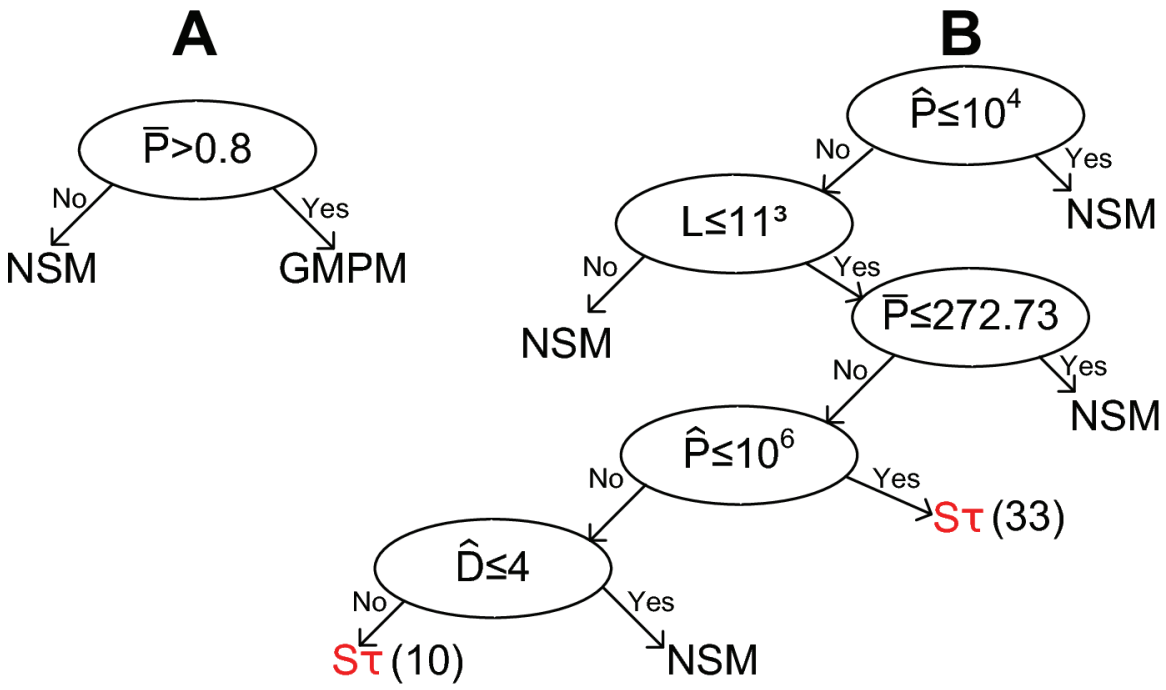


Figure 6.18: J48 decision trees to decide which of all algorithms is the fastest (A) and whether Sτ will perform better than NSM (B). Tree A misclassifies only 3 of 155 models (in two of those cases Sτ is the fastest algorithm). Tree B, however, misclassifies 12 of the 155 models (ca. 8%) — so with the given features it is rather unclear whether NSM or Sτ shall be applied to a model. In B, the cases where Sτ is classified to be better are marked with red and the number of correctly classified examples is given in parentheses. This indicates how often Sτ outperformed NSM in which region of the problem space — in total, this happened 46 out of 155 times (the other cases are misclassified).

6.18), i.e., those rare cases where GMPM's diffusion approximation is not beneficial because in most iterations only a single particle will diffuse from one sub-volume to the other.

If more accuracy is desired, however, GMPM may not be a good option. Tree B decides whether to use Sτ or NSM for a given model. In general, it can be seen that Sτ is only beneficial if there is a sufficient amount of particles ($\hat{P} > 10^4$, $\bar{P} > 272.73$) and not too many sub-volumes ($L \leq 11^3$). If this is the case, NSM only outperforms Sτ if many particles are initially concentrated in a single sub-volume ($\hat{P} > 10^6$) and they are diffusing rather slowly ($\hat{D} \leq 4$) — but this was only the case for two model setups. All in all, tree B agrees with our prior analysis that Sτ is inferior to NSM for large numbers of sub-volumes and small numbers of particles.

Note that the coefficients given in Figure 6.18, e.g., 272.73, are not very meaningful as such — only *factorial* experiments on the benchmark models have been conducted to explore the overall performance space. The node $L \leq 11^3$ in tree B, for instance, is unlikely to demarcate a true crossing point (which could be $12^3, \ldots, 20^3$ instead) — only models with 5^3, 11^3, and 21^3 sub-volumes were included in the study, so the border has to lie on one of those values. Similar arguments apply for all other features — the trees just provide a quantitative summary of the presented results and help to localize the regions in the problem space where one algorithm performs better than the other. There was also an attempt to predict absolute run times and the speed-up between different algorithmic variants, but the resulting trees had very large error rates — which also suggests that much more data is required for a good prediction.

Figure 6.19 is another approach to visualize some results from the evaluation study. It displays optimal and non-optimal models, i.e., those which are executed much faster (blue) or slower (red) with Sτ compared to the NSM. For example, according to this figure, Sτ favors models from the Molecule Generator type that are small or medium sized and have initially a large number of particles. If the size exceeds a certain value, then the algorithm should be replaced with the NSM.

Parallel Performance Section 4.4 discussed a parallel extension of the Sτ algorithm, which will be referred to as pSτ in the following. Instead of having one processor do all the work, i.e., finding a leap candidate and calculating the state update, the set of sub-volumes is split up and distributed among the available computational resources, hoping that each task can be processed faster if one processor is responsible for only a fraction of the sub-volumes. JAVA itself already offers classes for a concurrent execution,

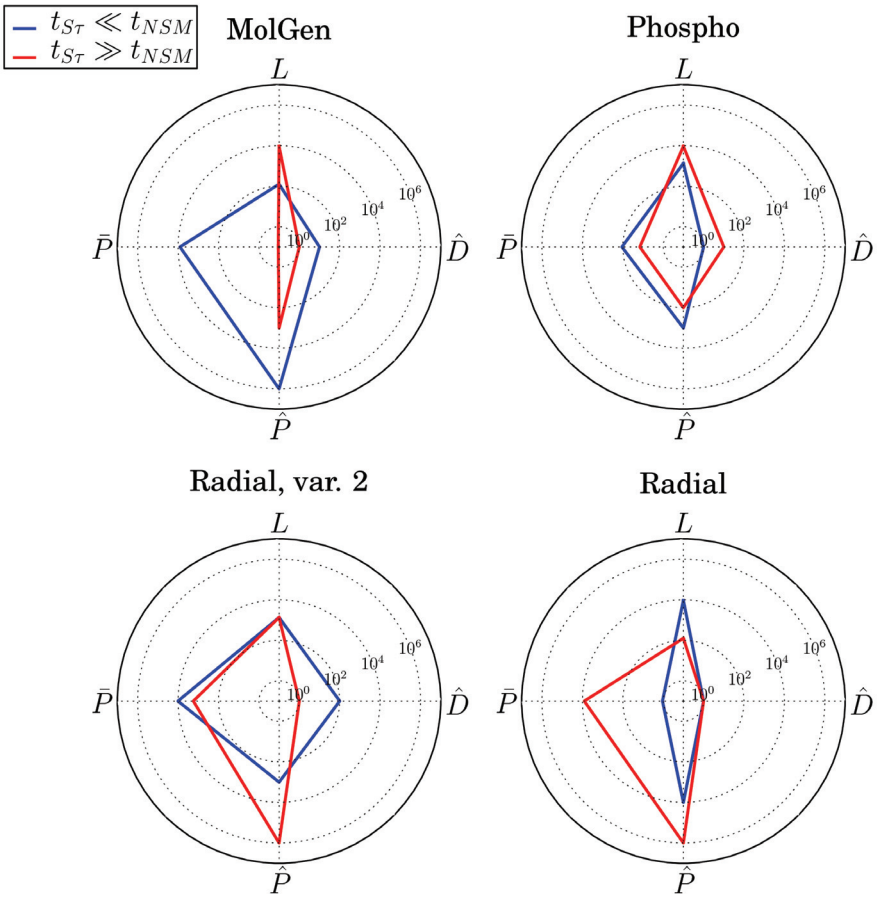


Figure 6.19: Spider charts visualizing the features of models for which Sτ is significantly faster (blue) and slower (red) than NSM. As an example, the plots for the phosphorylation model look quite similar. Indeed, most of the setups were simulated faster with the NSM, so there seems to be no feature that would advise to use Sτ instead.

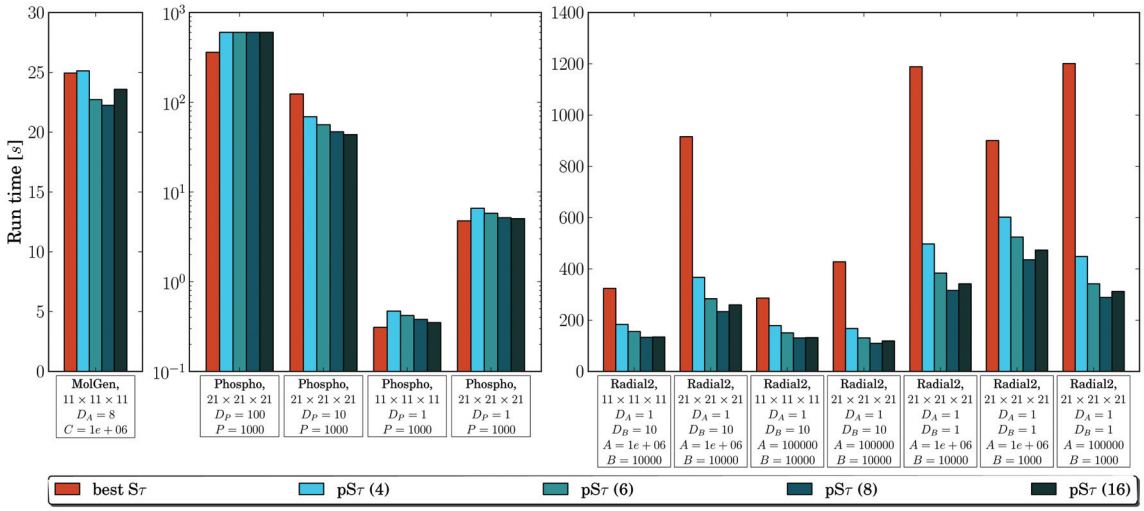


Figure 6.20: Algorithm performance for a parallel execution of different model types. The number of threads used is given in brackets in the legend. Both the Molecule Generator and Phosphorylation model show only marginally lower execution times, in some cases it event takes longer to finish the simulation. However, the results for the second Radial model variant looks much better: here speed-ups factors 2 to 4 can be achieved.

so the original Sτ algorithm has been modified to utilize those by letting *threads* handle the two main tasks during each iteration. Threads are created within processes and execute a certain piece of code concurrently to their parents and other threads. On a single-core machine the operating system (OS) has to switch between threads; each one runs for some amount of time, is then put into a wait state and the OS decides what thread should be resumed next. As a result, only one thread is "active", i.e., it solely uses the CPU, at any time. However, this restriction is lifted if more than one CPU is available. With n cores present, n threads can run in parallel; it is the operating system's responsibility to schedule the tasks on each CPU independently, i.e., to decide what thread to execute next on which processor.

Figure 6.20 shows performance results for a selection of model setups. Note that the machine used for the study has eight cores, yet 16 was the maximum number of threads used; this is one reason why for some setups a configuration with 16 threads is actually slower than one with eight. At best, the run time for both the Molecule Generator and the Phosphorylation model types was only marginally lower compared to a sequential execution — in fact, for the majority of the tested model setups a single run needed *longer* to finish the task. The reason for this can be found by analyzing how long it

takes for the sequential variant to find a τ candidate and update the state matrix in each iteration and comparing these results with the individual times for each of the $n \in \{4, 6, 8.16\}$ processors in a parallel run. It turns out that at least for the Molecule Generator and Phosphorylation models the time needed for these calculations is simply too short to make a distribution of the work load profitable. This is reasonable, as in both models a lot of sub-volumes remain empty during a simulation run and the algorithm simply skips those where nothing can happen during the next interval.

However, the case is different for the second Radial model variant; Figure 6.20 suggests that this model type is a better choice for a parallel execution. A simulation is at least twice as fast as a sequential run, for larger models this factor even increases to four. Best results are achieved for setups using eight threads on the same number of cores; adding more threads does not further lower the execution time due to the overhead generated by switching threads. Again, looking at the wall-clock times required for the two main tasks shows that here the time intervals for both the τ calculation and the state update in a sequential run are large enough to allow a speed-up when parallelizing the operations. In contrast to the other two models, every sub-volume contains at least B particles, so the algorithm cannot skip any sites during an iteration; in this case distributing the work among several processors proves to be beneficial.

As a conclusion: yielding a speed-up from a parallel execution of Sτ depends on how long the sequential variant needs to find the leap value and execute the state update; if this requires a considerable amount of time, e.g., the algorithm needs to process every sub-volume, then the execution times can be reduced in a multi-processor environment. Nevertheless, it should be considered that while a speed-up of four is possible, this required eight processors to achieve — which could also be utilized to execute eight sequential runs concurrently, i.e., to perform a "parallelization across simulations", as was discussed in Section 3.1.3. In other words: making a single run four times faster on eight processors is still not as good as running eight sequential variants on each computational units. However, if only a single run is needed at all, e.g., to do a more detailed analysis of an interesting trajectory, then the parallel execution should be preferred over a sequential run.

6.3 Summary

In this chapter the aspects of a performance study discussed in the previous chapter have been applied to evaluate the execution speed and accuracy of both non-spatial

and spatial simulation algorithms. Each section started by introducing the benchmark models used in the subsequent analysis. For non-spatial algorithms, emphasis was laid on the number of species and reactions, the initial amount of particles, and the dependency between reactions. With respect to execution speed, the τ-leaping algorithm proved to be significantly faster than its competitors (DM, FRM, ODM, NRM with different event queue implementations) for two benchmark models, the Totally Independent System (TIS) and the Cyclic Chain System (CCS). However, this was not the case for the Linear Chain System, where the particle distribution evolved as a "wave" through the state space and species near the end of this wave turned out to be problematic for τ-leaping: the calculated leap values were close to the threshold that would initiate a switch to an exact variant. As a result, the exact NRM and ODM variants require less time than the approximative τ-leaping algorithm.

Another interesting observation regards the dependence of NRM to the selected event queue implementation. The speed-up can be impressive if a more advanced queue variant is used instead of a simple ordered list; however, it has also been shown that this is not generally the case: for the CCS model, the setup NRM/MListRe, which was best for the other two models, was actually much slower than the combination NRM/Simple — in fact, it was the worst setup over all algorithms for a certain model parameterization.

In the further progress, τ-leaping was also subject to a more in-depth analysis. It is the only algorithm that depends on both a sub-algorithm (a random number generator) and specific parameters, such as the error control parameter ϵ and the threshold N_{SSA}. The results confirmed what has been expected: decreasing ϵ or n_c (which is used for testing whether a reaction is considered as being critical or not) both lead to longer run times; no noticeable influence could have been observed for the remaining parameters N_{SSA} and γ. While being fast, the trajectories generated by τ-leaping with default parameters are significantly different to exact results; making the error control parameter smaller resolved this issue, but this comes at the cost of a lower execution speed.

The spatial simulation algorithms considered in the second study all have their strengths and weaknesses. GMPM's execution speed, for instance, was excellent for almost all of the test model setups 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 the spatial τ-leaping is somewhere located between these two; it is generally more accurate than GMPM, but also takes longer to finish a single run. On the

other hand, it clearly outperforms NSM on various model setups, while still producing comparable results. Using the machine learning toolkit WEKA, example decision trees that suggest an algorithm depending on certain model characteristics have been presented in the last part of the study; the information from such data mining approaches could be used to offer algorithm selection capabilities in simulation tools, e.g., *JAMES II*.

A separate section has been dedicated to the study of pSτ, the parallel spatial τ-leaping variant. The results have shown that the run time of a single replication can be decreased for certain models, which could be useful for an in-depth analysis of interesting trajectories. However, if an ensemble of replications is required, then a "parallelization across the simulation" that concurrently executes sequential runs on the processors may be the better option.

7 Multi-algorithm Simulation

Early on during the PhD project, while learning about different ways how to simulate reactions between and diffusions of particles, there was the idea to also *explore* if it is possible to let those algorithms cooperatively accomplish a simulation task. Some authors presented methods that already realize this, e.g., in [TKHT04], where discrete stochastic and continuous deterministic simulation schemes are successfully combined as part of a new coordinator algorithm. However, being focused on the spatial simulation, the question for this dissertation was: could something similar be also done for methods in this area?

After motivating this research direction, the following parts of this chapter will introduce a general way how to describe a discrete event simulation in the context of SSA. Based on this, it is then discussed how to extend this interpretation — first for simulations considering individual entities and eventually also for a combination of the latter with a population-based stochastic simulation.

7.1 Problem Statement

The sections presented so far focused on the simulation of *populations*: entities are not considered as individuals, but rather belong to some class, e.g., a species, and are represented as either discrete or continuous amounts. All interactions are located at the class level: reactions are defined between species, not particles, an abstraction backed up by the assumption of a well-stirred system in which all particles can potentially interact with each other between two events. With a strong physical basis, stochastic simulation algorithms evolved over the last thirty years into an essential tool for the study of biochemical reaction systems; more efficient variants are published regularly to keep up with models getting increasingly more complex. Compared to alternative approaches, e.g., molecular or Brownian dynamics simulators at individual level or numerical solvers for ordinary differential equations, they take an in-between position: stochastic effects influence the path a model follows through the state space and discrete

particle numbers reflect that there are indeed individuals at the lowest level, though it may not be required to keep track of their states.

But some scenarios may ask for a finer control over the particles and how they can interact. For example, the spatial τ-leaping algorithm introduced in the previous section operates on a rectangular grid of sub-volumes, each of them being a well-stirred environment just like in the non-spatial case. Particles are assumed to be propagated solely by diffusion, i.e., an apparently random displacement caused by collisions between the object and solvent particles. As a consequence, the only way a modeler can influence the movement of particles is to assign different diffusion constants, which in fact only alters the average time interval between diffusion events of a species. In contrast, nature is quite inventive when it comes to the task of transporting something from place A to B. Just one example: entities could either move randomly according to Brownian motion or, once attached to some cable-like structure, follow specific paths through space towards some destination. Actually, this has been observed for organelles in algae cells; they can bind to certain filaments and, if the "energy carrier" ATP is present, start moving unidirectional along them [Kac85]. The authors also report that organelles can "switch lanes", i.e., change their transport cable.

While being attached to a filament, an organelle could also collide with others or some obstacle. An encounter like this often resulted in a dissociation from the cable and the particle returned to Brownian motion. So the physical presence of some other object had an influence on the organelle's motion; the space in front if it was simply blocked by something and it was not able to continue its movement as intended. Now such a scenario is not specific to the system it was observed in; any "solid" object (from a macroscopic perspective) makes it impossible for others to occupy the same space at the same time. This *excluded volume* effect is known to have a serious impact on the dynamics of reaction-diffusion systems [CKL04, Ell01, EM06, Sch04]. Before any reaction can take place, the reactants first have to move into close proximity, then collide (with a sufficient energy and correct orientation), and finally undergo the necessary changes to form the product. As contradictory as it may sound at first, under excluded volume conditions this process can be both sped up or slowed down. If the reactants are separated by many obstacles, e.g., other free moving molecules, static structures such as filaments, and so on, then it may be very difficult if not impossible for them to meet; they are blocked from each other and constantly bump into their surrounding neighbors, which additionally slows them down. On the other hand, in case they are already located close together, then those crowding agents now may

avoid that the reactants once more drift apart and therefore promote the reaction. But excluded volume is not considered in stochastic simulation algorithms; the well-stirred premise, as mentioned above, allows a particle to potentially interact with any other during the interval between two events. Thus, from the perspective of this particle, there are no obstacles blocking its way and it can freely diffuse through the volume. One idea to still model the excluded volume effect could be to reflect the presence of other objects by lowering the speed at which molecules travel. Reaction equations actually combine the two steps from above (move into proximity, then react) and can be rewritten to separate between both processes:

$$R_1 : A + B \xrightarrow{k} C \Longrightarrow A + B \underset{k_r}{\overset{k_D}{\rightleftharpoons}} AB \xrightarrow{k_f} C, \tag{7.1}$$

with $k_D = 4\pi DR = 4\pi(D_A + D_B)(r_A + r_B)/2$ [VS17] as rate for the formation of AB, a state with A and B being close enough to react, r_A and r_B as radii of A and B particles, k_r determining how fast both reactants, once being in state AB, drift away, and k_f as intrinsic reaction rate. Assuming $dx_{AB}/dt = 0$, i.e., a steady-state for AB, the kinetic constant k can be expressed in terms of k_D, k_r, and k_f. As a first step, considering that the number x_{AB} of AB complexes does not change over time, it is possible to write:

$$\frac{dx_{AB}}{dt} = k_D x_A x_B - x_{AB}(k_r + k_f) = 0 \tag{7.2}$$

$$x_{AB} = \frac{k_D x_A x_B}{k_r + k_f} \tag{7.3}$$

Now the change of C can be given in dependence of k and k_f, which, together with the expression for AB above, leads to:

$$\frac{dx_C}{dt} = k x_A x_B = k_f x_{AB} = \frac{k_f k_D x_A x_B}{k_r + k_f} \tag{7.4}$$

$$k = \frac{k_f k_D}{k_r + k_f} = \frac{4\pi(D_A + D_B)(r_A + r_B)k_f}{2(k_r + k_f)} \tag{7.5}$$

Lowering the diffusion constants D_A and D_B decreases the kinetic constant k, thus on average it takes longer until a pair of A and B particles form a product C. But this adjustment takes place at the species level; all particles are affected, independent of their location. Additionally, regardless how many other objects are present, the propensity of R_1 still depends only on the number of A and B particles; it does not

matter whether there are, e.g., ten or ten thousand C particles also floating around in the volume.

On some occasions it may be desirable to define a more complex behavior for a single species than it is possible with reaction equations and the underlying assumption of mass action kinetics. For example, thorough observations provided additional information about the internal behavior of species A from the example above. It was known that there is some process going on inside particles of this species and this has been captured so far by the first order reaction $A \xrightarrow{k_j} C$. This abstraction might have been sufficient for previous studies, but it turned out that the reduction to a single constant is too simple in light of the new insights. So how can those internals be represented? Why not introduce new species and reactions into the existing reaction network, e.g.:

$$R_j : A \xrightarrow{k_j} C \Longrightarrow \begin{cases} R_{j,1} : A_1 + A_2 \xrightarrow{k_{j,1}} A_3 \\ R_{j,2} : A_1 + A_3 \xrightarrow{k_{j,2}} A_4 \\ R_{j,4} : A_3 + A_4 \xrightarrow{k_{j,3}} C. \end{cases} \tag{7.6}$$

Species A is no longer present, but has been replaced by A_1 to A_4. Though possible, this modification puts the intermediates on the *same level* as C, something that may not be intended by the modeler; for instance, they would all be present in the same volume, despite A_1 to A_4 being more like *internal species* of an A particle, reacting according to a separate network (and very likely: inside a separate volume, i.e., they are enclosed by an A particle). Furthermore, all alternatives considered still assume mass action kinetics and exponential event times, but a C particle could be actually produced out of an A particle in, e.g., fixed intervals.

If the conclusion is that population-based approaches (CME or ODEs) may not be sufficient for certain model requirements, what are possible alternatives? Some of the above, e.g., molecular crowding, can be realized with molecular or Brownian dynamics (MD/BD) simulators, so why not use one of those if additional detail is required? Both techniques consider *only* individuals, the former one even solvent molecules (e.g., water), those small particles causing the random movement of larger objects dissolved in them. Though the performance of MD and BD simulators increased over the last years [ZTW05, AB04], propagating individuals and checking for reactive collisions can still be very cost-intensive; doing this for thousands or millions of particles for a larger time interval may take a long time.

The idea proposed in the following sections blends population and individual-based

simulation methods together; some species, such as A from the example, are represented with individual particles having an own state, while others, maybe of lesser interest, are present as a population.

7.2 Introduction — Discrete Event Simulation with SSA

In reaction networks defined so far the particles of a species were represented only as numbers. The state vector $\mathbf{x}$ or matrix $\mathbf{X}$ stored how many of them are present at any given time point; this was possible because particles had no distinct features, e.g., a position, shape, or size, and it was not necessary to identify a particular molecule. The only components having a state were the species and it got modified with each reaction execution: according to the stoichiometry, reactant particles get removed from the system and the products added.

Central to this section shall be an alternative perspective on the modeled system which puts a stronger focus on the particles themselves. Starting from the discussion made in the problem statement, species now fall into two main groups. Members of the first one have their state represented just like it was in the previous sections — as an amount of particles. However, species belonging to the second group are handled quite differently. Their particles are present as individual entities with own state variables; rules now apply to those and therefore modify the species state only indirectly. But before making this step and introducing particles as individuals, the notion of states and rules in the context of stochastic simulation algorithms shall be revisited; the sections following up will then show that the proposed extension, although maybe uncommon at first, still shows some similarities with the algorithms discussed up to this point.

Going back to the introduction of this dissertation, a discrete event simulation iteratively performs the following three steps:

1. Get the interval until the next event will occur.
2. Get the type of the next event.
3. Modify the state by executing the event from step 2 and update the global time with the interval from step 1.

Each simulation algorithm discussed so far basically implements those steps. Point one

from the enumeration can be formalized into an *interval selection function*

$$t : \mathbb{N}^{L \times N} \to \mathbb{R}_0, \tag{7.7}$$

which maps the current state (a vector in the non-spatial case $L = 1$ and a $L \times N$ matrix for spatial models) to the time interval until the next event. But before continuing with the remaining steps, some words about events and reactions. Reaction equations are the rules describing how to update the state once the reaction takes place. Though for exact algorithms the analogy "event = reaction" can be made, this is no longer the case for approximative variants because during a single leap more than one reaction is allowed to occur. To reflect this, the state update for non-spatial algorithms has been generalized to

$$\mathbf{X}(t + \tau) = \mathbf{x} + \sum_{j \in [1,M]} k_j \mathbf{v}_j \tag{7.8}$$

in the algorithm template from Algorithm 2.1. This very line actually represents the execution of the *next event*, i.e., it applies each rule R_j k_j times to the current state, with a single rule invocation defined as

$$\begin{aligned} R_j : \mathbb{N}^N \times \mathbb{N} &\to \mathbb{N}^N \\ R_j(\mathbf{x}, k) &= \mathbf{x} + k\mathbf{v}_j; \end{aligned} \tag{7.9}$$

exact algorithms with the firing vector $\mathbf{k}$ defined as $\mathbf{1}_\mu^N$, i.e., only a single non-zero entry at k_μ, can then be seen as special cases. But what are the event types in a stochastic simulation, if not the reactions? In fact, there is only a single type: for exact algorithms it is "the next reaction takes place", for approximative variants "the leap condition is violated" (note that the latter is very general; if critical reactions are handled as well, then an additional event type could be called "the next critical reaction takes place").

That said, the second and third point from the enumeration could be summarized in the context of stochastic simulation as follows: determine the (reaction) rules the imminent event is composed of and apply them to the state. Or, in terms of a *rule selection function*:

$$r : \mathbb{N}^N \times \mathbb{R}_0 \to \mathbb{N}^N, \tag{7.10}$$

update the state vector $\mathbf{x}$ based on the current state and the interval $\tau \equiv t(\mathbf{x})$ using elements from the rule set R. Keeping t, R_j, and r in mind, the next section shall introduce a general algorithm for the discrete event simulation of individual particles,

built upon the same functions basically used throughout the dissertation.

7.3 Individual-based Discrete Event Simulation

The first step towards an algorithm for individuals is actually a step backwards, from very specific function definitions for t, r, and R_j to more general versions. With the particles being now the smallest stateful components in the system, species no longer have an explicit state but are themselves dependent on the variables defined for the particles, which are not limited to merely integer valued numbers. Instead, they can be chosen arbitrarily; for example, an essential feature of an object is its position in space, usually expressed as a coordinate tuple (x, y, z) with $x, y, z \in \mathbb{R}$. However, the precise nature of the state variables does not actually matter for this introduction as it is likely to change between different models. Therefore, the k-th individual from species S_i shall be simply represented by the feature set $F_{i,k} = \{f_{i,k,1}, f_{i,k,2}, \ldots\}$, with, e.g., $f_{i,k,1}$ being the position (x, y, z) of the entity, $f_{i,k,2}$ its current shape, $f_{i,k,3}$ the electrical charge, and so on. This means adding another layer to the state hierarchy used so far; the status of the entire model is defined via the species states, i.e., $X = \{X_1, \ldots X_N\}$, but now $X_i = \{F_{i,1}, \ldots, F_{i,O}\}$. Over the course of a simulation run, the model state is changed at different points in time with each event execution. A single *trajectory* through the state space $\tilde{X}$ (the set of all states a model can be in during any run[1]) shall be defined as the collection of states visited by the model during the simulation interval $[t_{start}, t_{end}]$, i.e., $Y = \{X_t : X_t \in \tilde{X} \wedge t \in [t_{start}, t_{end}]\}$.

Similar to the last section, a discrete event simulation of a model that is built upon individual entities can be expressed in terms of the three functions t, r, and R_j. The only differences are the domains they are defined on: the input is not an integer-valued vector or matrix, but the feature sets of all individuals present inside the volume. Using only the *current state* in t and r in the last section is an additional abstraction backed by the assumption that the underlying stochastic process exhibits the Markov property, i.e., it is memoryless. So more generally, $t : \tilde{X}^{\leq t} \to \mathbb{R}_0$ and $r : \tilde{X}^{\leq t} \times \mathbb{R}_0 \to \tilde{X}$, with $\tilde{X}^{\leq t}$ as state history $\{X_{t^*} : X_{t^*} \in \tilde{X} \wedge t^* \in [t_{start}, t_{end}] \wedge t^* \leq t\}$ (if not explicitly stated otherwise, only the current state is used in the forthcoming sections and the index t therefore will be omitted).

[1] For simplicity it is assumed that the number N of species does not change during a simulation run. However, each set $X_i, i \in [1, N]$ can contain an arbitrary number of feature sets (individuals). An empty set for any X_i shall denote that there are no particles present of this species.

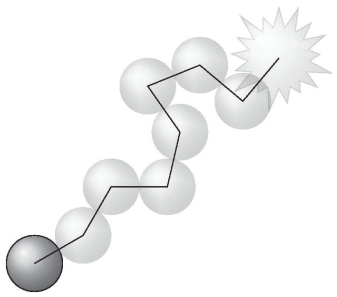


Figure 7.1: A simple model for a single particle which moves according to Brownian motion and will eventually degrade.

Rules selected from the set R, as usual, take the current state X and modify it, i.e., the most basic rule definition is $R_j : \tilde{X} \to \tilde{X}$. But what if the order in which the rules are applied actually matters? In previous algorithms there was either a single reaction or multiple ones firing during the interval, but even for the latter it was not necessary to ensure that some reaction has to be executed before another one. In contrast, applying rules to several individual features at the same time can lead to entirely different outcomes. For example, what rule should be executed first: the position update of an object or its decomposition into products? The difference between both alternatives is the final placement of the products; the latter inserts new particles at the origin of the object, the former somewhere around its target position. Avoiding this non-determinism is the task of the rule selection function; a preference order could be defined, e.g., by choosing a rule at random or using rule composition, i.e., $R_1(R_2(X))$, which first applies R_2 to the current state and then executes R_1 on the result.

Example 7.3.1 This example presents a very primitive individual simulator managing only a single entity that moves according to Brownian motion and can dissolve during an infinitesimal time interval dt with probability λdt (Figure 7.1). A particle propagated solely by diffusion over a time Δt is displaced in any direction by an average distance of $\sqrt{2D\Delta t}$, with D being its diffusion constant [Cha43]. Admittedly, instead of performing a simulation one could simply sample the time until the particle dissolves and then use this value to determine where it happens — repeating this several times will eventually provide a good picture of the distances the particle can travel until it degrades. However, despite the fact that no one would actually implement such a simulator, it still serves the purpose of demonstrating the basic principles sketched above.

The only state variable required is a vector $\mathbf{p} = (x, y, z)$ to keep track of the particle's position in space, so the state of the model can be written as $X = \{\mathbf{p}\} = \{(x, y, z)\}$[2]. Two rules are defined; the first one represents the degradation of the particle, which is reflected by removing its feature set from the model state[3]:

$$R_1 : \tilde{X} \to \tilde{X} \tag{7.11}$$

$$R_1(X) = \emptyset \tag{7.12}$$

The second rule describes how to update the position of an object if its movement follows Brownian motion:

$$\begin{aligned} R_2 &: \tilde{X} \times \mathbb{R}_0 \to \tilde{X} \\ R_2(\{(x, y, z)\}, \tau) &= \{(x + \mathcal{N}(0, \sqrt{2D\tau}), \\ &\qquad y + \mathcal{N}(0, \sqrt{2D\tau}), \\ &\qquad z + \mathcal{N}(0, \sqrt{2D\tau}))\}. \end{aligned} \tag{7.13}$$

In contrast to simple reactions, R_2 additionally depends on the elapsed time interval τ to calculate the displacement. Both remaining functions t and r are quite simple. The interval between position updates is fixed at Δt and the time until the particle dissolves can be sampled from $\mathrm{Exp}(\lambda)$. So with $\tau_1 = -\ln U(0, 1)/\lambda$ and $\tau_2 = \Delta t$, the time and type of the next event are given by:

$$\begin{aligned} t &: \tilde{X} \to \mathbb{R}_0 \\ \tau &\equiv t(X) = \min\{\tau_1, \tau_2\} \end{aligned} \tag{7.14}$$

and

$$\begin{aligned} r &: \tilde{X} \times \mathbb{R}_0 \to \tilde{X} \\ r(X, \tau) &= \begin{cases} R_1(R_2(X, \tau)) & \text{if } \tau_1 = \tau \\ R_2(X, \tau) & \text{else.} \end{cases} \end{aligned} \tag{7.15}$$

Note how rule R_2 will always be executed, regardless of the time to the next event. If the particle does not dissolve during Δt, then it is simply moved by R_2 to the new

[2] A condensed notation is used here. The state is actually $X = \{X_A\}$, with $X_A = \{F_{A,1}\}$ and $F_{A,1} = \{\mathbf{p}\}$. If it is clear from the context, then a shortened notation will be used for the rest of this chapter.

[3] Similar to footnote 2: the new state is $X = \{X_A\}$, with $X_A = \{\} = \emptyset$.

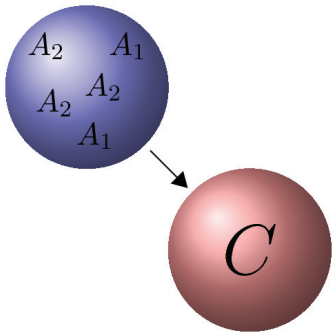


Figure 7.2: Particle conversion example. A reaction network is defined inside an individual whose final reaction will trigger the conversion into another species.

position. Otherwise, i.e., $\tau_1 < \Delta t$, rule R_1 will be applied *after* R_2 (now with τ_2 as parameter, instead of Δt) because the particle is still subject to Brownian motion and thus will be "pushed around" by the solvent molecules for a time τ_2 until it finally degrades.

In Example 7.3.1 the rule selection function uses *conditions* to select the appropriate event (which consists of one or more rules) that should be applied to the state. The most basic condition for an event is that the time interval until it will occur next is equal to the minimum over all events as calculated by t (e.g., $\tau_1 = \tau$; later during this chapter more complex conditions will also be defined.

State variables and the rules modifying them can be arbitrarily complex. In the problem statement discussed above the question was raised how to model an n-step conversion from one species to another, whereas each of those steps is not necessarily a first order reaction itself. Simply adding the intermediate (or inner) species to the model did not do the trick: the information that some species are only present inside others got lost, with the consequence that all are treated at the same level. But now state variables are no longer restricted to numerical values, but can be basically anything — even the state vector of a reaction network. The following example shows how to define a discrete event individual simulator that represents the conversion process as intended (Figure 7.2).

Example 7.3.2 Again, only a single particle A is considered, which is transformed

into another particle C according to an internal reaction network defined by:

$$\begin{aligned} R_1 &: A_1 + A_2 \xrightarrow{k_1} A_3 \\ R_2 &: A_1 + A_3 \xrightarrow{k_2} A_4 \\ R_3 &: A_3 + A_4 \xrightarrow{k_3} \text{create } C \text{ particle} \end{aligned} \tag{7.16}$$

The state variables of A are given with the feature set $F_A = \{\mathbf{p}, \mathbf{x}, V\}$; here, $\mathbf{p}$ is the position of A, the second vector $\mathbf{x} = (x_1, x_2, x_3, x_4)$ holds the number of $A_i, i \in \{1, 2, 3, 4\}$ particles and V denotes the current volume of the particle. It is necessary to translate the reactions from Equation 7.52 into rules that operate on the model state $X = \{F_A\}$. Adapting the first and the second one is trivial, so only R_1 will be shown here:

$$\begin{aligned} &R_1 : \tilde{X} \to \tilde{X} \\ &R_1(X) = \{\{\mathbf{p}, (x_1 - 1, x_2 - 1, x_3 + 1, x_4), V\}\}; \end{aligned} \tag{7.17}$$

the new state vector of the internal reaction network is a result of adding the state change vector $\mathbf{v}_1 = (-1, -1, 1, 0)$ of R_1 to the current state $\mathbf{x}$, i.e., $\mathbf{x}_{new} = \mathbf{x} + \mathbf{v}_1$.

R_3 simply removes the feature set of the A individual from the state and adds C's state variables F_C[4]:

$$\begin{aligned} &R_3 : \tilde{X} \to \tilde{X} \\ &R_3(X) = \{F_C\}. \end{aligned} \tag{7.18}$$

Similar to the previous example, the interval selection function determines the minimum elapsed time span until the first event will occur — which is in this case the time until the next rule invocation:

$$\begin{aligned} t &: \tilde{X} \to \mathbb{R}_0 \\ \tau &\equiv t(X) = \min\{\tau_1, \tau_2, \tau_3\}, \end{aligned} \tag{7.19}$$

with

$$\tau_1 = \frac{-\ln U(0,1)}{V k_1 x_1 x_2}, \quad \tau_2 = \frac{-\ln U(0,1)}{V k_2 x_1 x_3}, \quad \tau_3 = \frac{-\ln U(0,1)}{V k_3 x_3 x_4}. \tag{7.20}$$

For this example, the relation "rule = event" actually holds, so it should be no surprise

[4]The expanded notation is $X = \{X_A, X_C\}$, with $X_A = \{\}$ and $X_C = \{F_C\}$

that only a single rule is executed during any iteration:

$$r : \tilde{X} \times \mathbb{R}_0 \to \tilde{X}$$
$$r(X, \tau) = \begin{cases} R_1(X) & \text{if } \tau_1 = \tau \\ R_2(X) & \text{if } \tau_2 = \tau \\ R_3(X) & \text{else.} \end{cases} \tag{7.21}$$

Representing independent, individual entities whose behavior is partially controlled by reaction networks located inside of them is nothing new. It can be compared to stochastic simulations performed within compartments, i.e., encapsulated volumes separated by membranes (or any comparable border structure); see, for example, the two modeling languages Sπ@ calculus [VB07] and Beta Binders [PQ05]. The key difference is the perspective taken to look at the modeling and simulation task: either it is more language-centric, as in the given references, or focused on executing the algorithms and performing simulations, the main field of this dissertation.

Having identified the basic ingredients for discrete event simulations with species or individuals as interacting components, the next step is combining them into a multi-algorithm method to benefit from their advantages: the former offers a fast execution, even with many particles, while the latter attracts with an increased detail in the entity representation (size, position). But doing so requires that entities from both levels are aware of each other. Individuals occupy space that is not accessible by population particles, which therefore collide and potentially react with those obstacles. The next sections will discuss these mutual influences in more detail and present several alternatives how to capture them into a model.

7.4 Populations and Individuals

There are two questions two answer: how to represent individuals in the grid of sub-volumes and how does this has an effect on the dynamics of a spatial stochastic simulation algorithm. The most simple way to show that some areas inside the grid are blocked off by some obstacles is to remove sub-volumes from the model. Though the volume might be discretized into, e.g., a $5 \times 5 \times 5$ grid, not every sub-volume needs to contain particles or be a potential diffusion target for others. If the initial population of some site l is zero and l does not appear in any neighbor index vector

$\mathbf{C}_k, k \in [1, L], k \neq l$, then it will never receive any particles and its next event time is always infinity — it is practically only present as an obstacle inside the grid, blocking the way for any particle trying to diffuse into this non-existent sub-volume.

Now it is not difficult to fill these empty regions with a life of their own; in fact, Example 7.3.2 already illustrated one way how to do this. All that is needed are new sets of state variables specific to sub-volumes not managed by the spatial SSA; those sites are now treated as individuals in the same sense as introduced in the last section. As a result, *two* algorithms are used side-by-side to simulate this compound model, the first one taking care of the interactions within and between the well-stirred sub-volumes while the other processes events issued by the individual sites — which essentially summarizes the basis for this part of the dissertation. Starting from the very simple case of immobile individuals filling out an entire sub-volume, the subsequent steps will then gradually increase the modeling detail and consider moving particles as well as entity shapes other than a cube.

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & \cdots & x_{1,N} \\ x_{2,1} & \cdots & x_{2,N} \\ \cdots & \cdots & \cdots \\ x_{L,1} & \cdots & x_{L,N} \end{pmatrix}$$

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25

$X = \{X_1\}$
$X_1 = \{F_{1,7}, F_{1,14}, F_{1,22}\}$

Figure 7.3: 5×5 grid with different state representations for populations and individuals.

If both simulations do not interfere with each other, i.e., one algorithm is not allowed to modify the other's state, then they can actually be run in parallel. Particles in well-stirred sub-volumes simply bump off of the individuals without causing any state changes inside them; conversely, individuals do not remove or add any particles from or to the surrounding sub-volumes. Easy as this may be, it is probably not what a user has in mind. Instead, combining both approaches makes only sense if there is some way to express *interactions* between them. If a particle of the correct type hits an individual, than it is expected that the latter somehow reacts to this event by changing its state. So in addition to the already existing *intra-rules*, which change the state of either the population or individual model, a third type, the *inter-rule*, is necessary. As the name suggests, inter-rules are defined on both state representations: the $L \times N$ matrix $\mathbf{X}$ holding the number of particles for each species in the sub-volumes and the set of individuals X from the previous section (see also Figure 7.3); thus, their basic form, i.e., without any additional parameters such as

sub-volume or individual indices, is:

$$\mathcal{R}_j : \mathbb{N}^{L\times N} \times \tilde{X} \to \mathbb{N}^{L\times N} \times \tilde{X}. \tag{7.22}$$

Example 7.4.1 Going back to Example 7.3.2, it is now supposed that a particle of species A_1 acts as the trigger for the transformation process from A to C (the internal reaction network from 7.52 is taken over without changes). Initially, A only contains particles of A_2 while A_1 is abundant in the sub-volumes around the individual. Therefore, an A_1 molecule first has to collide with A and enter it before initiating the state change. Putting all this together, the state at the beginning of a simulation run is given by $\{\mathbf{X}, X\}$, with

$$\mathbf{X} = \begin{pmatrix} x_{1,A_1} \\ x_{2,A_1} \\ \vdots \\ x_{L,A_1} \end{pmatrix} \qquad X = \{\{\mathbf{p}, (0, x_2, 0, 0), V\}\}. \tag{7.23}$$

One way to write down a rule modeling the just described interaction is:[5]

$$\begin{aligned} \mathcal{R}_1 &: \mathbb{N}^{L\times N} \times \tilde{X} \times \mathbb{N} \to \mathbb{N}^{L\times N} \times \tilde{X} \\ &\mathcal{R}_1(\{\mathbf{X}, X\}, l) = \{\mathbf{X} - \mathbf{1}^{L\times N}_{l,A_1}, \{\{\mathbf{p}, (x_1 + 1, x_2, x_3, x_4), V\}\}\}. \end{aligned} \tag{7.24}$$

The rule has an additional input l, which is used to specify the index of the sub-volume that looses the particle; it does not pose any constraints on the selection of l, the rule simply removes one particle of A_1 from it and adds it to the state of the individual. Determining the appropriate sub-volume l is done in the rule selection function r that will be discussed below.

After the first A_1 particle is "transfered" into the individual, the reaction network defined in Equation 7.52 can start the transformation process. Note that more than one A_1 particle is required for the entire conversion because both R_1 and R_2 require them as reactants.

As it was shown, inter-rules are the connections between algorithms; they allow a simultaneous update of more than one state representation within a single event execution. However, they will not be invoked unless they are selected by r — which neither of the functions defined so far will do as they only take either the state matrix

[5] The symbol $\mathbf{1}^{L\times N}_{l,i}$ shall denote an $L \times N$ matrix with a single 1 at position (l, i).

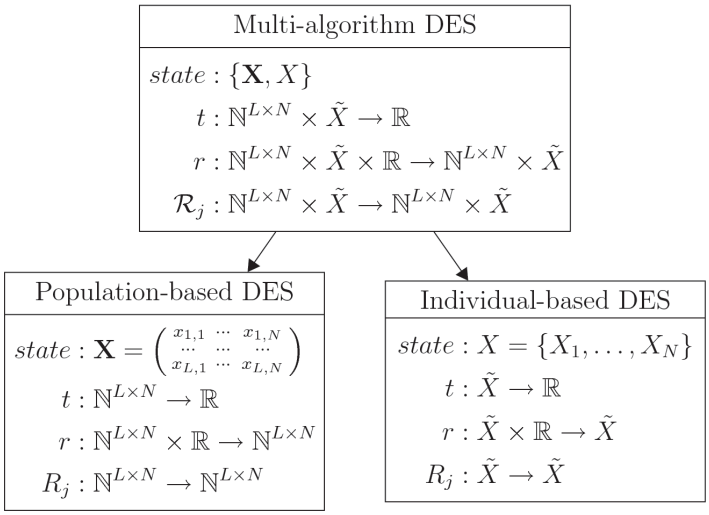


Figure 7.4: A multi-algorithm discrete event simulation makes use of the t, r, and R functions defined for lower level algorithms (here: algorithms for a population-based and individual-based DES). It additionally defines inter-rules, which can act on the states of all subsidiary DES.

or the state set as an input, but not both. The answer is to implement the selection and processing of inter-rules at a higher level that *operates on multiple states*, abstractly depicted as a hierarchy shown in Figure 7.4. Existing functions r, t, and R_j, established for either a population or individual-based algorithm, can very likely be reused in the overlaying level, though it may be necessary to adapt them. It is possible that some decisions whether to apply a rule or not can no longer be made at the lower levels if they take part in a multi-algorithm DES. In that case the responsibility of selecting and executing those rules has to be relocated to the next higher node in the hierarchy.

Example 7.4.2 (Continuation of Example 7.4.1) Just writing down the inter-rule $\mathcal{R}_1$ is not enough, what is still missing are expressions for t and r. Before introducing those, some more details about the population-based DES without the presence of any individual A are required. First of all, A_1 particles diffuse between sub-volumes, which can be generalized with the following rule:

$$R_D : \mathbb{N}^{L\times N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}^{L\times N} \tag{7.25}$$

$$R_D(\mathbf{X}, l, l^*) = \mathbf{X} - \mathbf{1}^{L\times N}_{l,A_1} + \mathbf{1}^{L\times N}_{l^*,A_1}. \tag{7.26}$$

In addition to the state matrix, R_D also takes the identifiers of the source and target sub-volumes as inputs. A diffusion from a site l to l^* removes one particle of A_1 from

l and adds it to l^*'s state vector. To make this example slightly more interesting, A_1 particles can also degrade with a certain rate c_d:

$$R_d : \mathbb{N}^{L\times N} \times \mathbb{N} \to \mathbb{N}^{L\times N} \tag{7.27}$$

$$R_d(\mathbf{X}, l) = \mathbf{X} - \mathbf{1}^{L\times N}_{l,A_1}. \tag{7.28}$$

The next event that will take place is either one particle leaving a site and diffusing into a neighbor or a degradation somewhere inside a sub-volume. From the introduction and the previous sections it is already known how to write down the interval selection function. First, let:

$$t_D(\mathbf{X}, l) = \frac{-\ln U(0,1)\lambda_{sv}^2}{2nD_{A_1}x_{A_1}} \qquad t_d(\mathbf{X}, l) = \frac{-\ln U(0,1)}{c_d x_{A_1}} \tag{7.29}$$

$$\hat{t}_D(\mathbf{X}) = \min_{l\in[1,L]}\{t_D(\mathbf{X}, l)\} \qquad \hat{t}_d(\mathbf{X}) = \min_{l\in[1,L]}\{t_d(\mathbf{X}, l)\} \tag{7.30}$$

denote the time spans until the next diffusion ($\hat{t}_D$) and degradation ($\hat{t}_d$) will take place, respectively. Then t is defined as:

$$\begin{aligned} t &: \mathbb{N}^{L\times N} \to \mathbb{R}_0 \\ \tau &\equiv t(\mathbf{X}) = \min\{\hat{t}_D(\mathbf{X}), \hat{t}_d(\mathbf{X})\}. \end{aligned} \tag{7.31}$$

The next steps will make use of the following auxiliary functions:

$$p : \mathbb{N} \to \mathbb{N}^3, \quad n : \mathbb{N} \to \mathbb{N}. \tag{7.32}$$

The first one, $p(l)$, is a mapping between sub-volume identifiers and vectors (x, y, z) of coordinates representing the corresponding position in the grid. The second function $n(l)$ returns one of the $2n$ neighbor identifiers $l^* \in \mathbf{C}_l$.

With the help of p and n the rule selection function can finally be written down. Let

$$l_D = \operatorname*{argmin}_{l\in[1,L]}\{t_D(\mathbf{X}, l)\}, \quad l_d = \operatorname*{argmin}_{l\in[1,L]}\{t_d(\mathbf{X}, l)\}, \quad l^* = n(p(l_D)), \tag{7.33}$$

then

$$
\begin{aligned}
r &: \mathbb{N}^{L\times N} \times \mathbb{R}_0 \to \mathbb{N}^{L\times N} \\
r(\mathbf{X},\tau) &= \begin{cases} R_D(\mathbf{X}, l_D, l^*) & \text{if } \tau = \hat{t}_D(\mathbf{X}) \\ R_d(\mathbf{X}, l_d) & \text{if } \tau = \hat{t}_d(\mathbf{X}), \end{cases}
\end{aligned} \tag{7.34}
$$

Everything is now set up to perform a basic but not very exciting spatial stochastic simulation. Particles diffuse through the volume and they will degrade until all of them are eventually gone for good. Extending this simulation to also include an A particle somewhere inside the volume requires a new set of functions to take care of applying the inter-rule defined in Example 7.4.1 if an A_1 particle happens to "collide" with the individual. So how may those functions look like? As it turns out, defining t is simply a matter of checking whether the individual or the population-based DES will execute an event first:

$$
\begin{aligned}
t &: \mathbb{N}^{L\times N} \times \tilde{X} \to \mathbb{R}_0 \\
\tau &\equiv t(\mathbf{X}, X) = \min\{t(\mathbf{X}), t(X)\},
\end{aligned} \tag{7.35}
$$

The next event time of the entire simulation is thus the minimum of the next event times reported by all low-level algorithms. However, the rule decision gets more complex due to the presence of the inter-rule. It is unknown to the population DES that one sub-volume is different than the others. Hence whenever a particle is about to leave a site, the higher level instance has to test whether the target is a "regular" sub-volume or the individual A:

$$
\begin{aligned}
r &: \mathbb{N}^{L\times N} \times \tilde{X} \times \mathbb{R}_0 \to \mathbb{N}^{L\times N} \times \tilde{X} \\
r(\mathbf{X}, X, \tau) &= \begin{cases} r(X,\tau) & \text{(a) if } \tau = t(X) \\ r(\mathbf{X},\tau) & \text{(b) if } \tau = \hat{t}_d(\mathbf{X}) \vee (\tau = \hat{t}_D(\mathbf{X}) \wedge p(l^*) \neq \mathbf{p}) \\ \mathcal{R}_1(\mathbf{X}, X, l) & \text{(c) if } \tau = \hat{t}_D(\mathbf{X}) \wedge p(l^*) = \mathbf{p}. \end{cases}
\end{aligned} \tag{7.36}
$$

The first choice (a) applies the r function from the individual DES if its internal reaction network fires next. If instead the population DES has an event scheduled before the other simulator, then either the second (b) or third (c) choice get selected; both conditions then read: if a degradation takes place next *or* the target of a diffusion is not A, then let the r function from the population DES update the state (b), otherwise

execute the inter-rule (c).

There are two things to note here: first, the application of the second and third rule make use of the fact that the position $\mathbf{p}$ of an individual always corresponds to a sub-volume coordinate; it has been written earlier that for now an individual occupies an entire sub-volume, which is a special case and shall be generalized below. The second point is that for this example *each* collision of an A_1 particle with the individual leads to a state change. This contrasts with a real system, where it is also possible that an A_1 particle bounces off without inducing a reaction.

Now that the basics of a multi-algorithm simulation have been introduced, the next paragraphs will take this approach one step further and discuss how to represent mobile, sphere-shaped individuals.

7.5 Mobile Individuals

Section 7.3 already covered the movement of an object subject to Brownian motion. This is just one example how an individual can be propagated through the volume: getting pushed around by solvent molecules. Another, more complicated movement pattern was referred to in the problem statement. Organelles have been observed that not only diffuse, but also can attach to filament structures inside the cell and move along them. Consequently, the rules governing the position update can be arbitrary complex and an individual may be propagated differently depending on its current state.

The last section laid out the basic interaction scheme between algorithms: a higher level component has to take care of the inter-rules modifying several state representations at once. A simple model has been used to demonstrate this concept, which shall now also be taken as the starting point for the discussion on how to represent moving entities. As a reminder, an A particle is a special type of sub-volume with an own reaction network working inside of it. Whenever one of the surrounding A_1 particles tries to diffuse into the region occupied by the individual, it is consumed and can take part as a reactant in the internal reaction network. The difference to the former example is that A is no longer immobile but can move into adjacent sites in a single step. As the target may not be empty, particles currently present there will get "pushed out" of the sub-volume by the individual moving into it. Once A has reached its target position, the now occupied sub-volume is empty, thus no reactions will take place inside it and

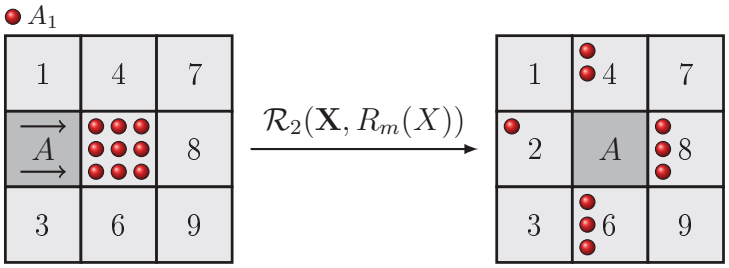


Figure 7.5: Applying the rule composition $\mathcal{R}_2(\mathbf{X}, R_m(X))$ to the state shown on the left side moves the A particle from sub-volume 2 to 5 and distributes the A_1 particles currently inside 5 among the neighbors.

no particles leave or enter. From the perspective of the population-based DES algorithm, this site has a next event time of infinity and if it gets selected as the target of a diffusion event, the higher level component will apply $\mathcal{R}_1$ instead. At the same time the sub-volume located at the old position of the individual is "re-integrated" into the lower-level DES, i.e., particles diffuse into it and reactions can take place.

Expressing this scenario in terms of a multi-algorithm DES can be done in a two step process: move the individual to its new position, then update the populations in the sub-volumes next to it. Similar to Section 7.3, an additional propagation rule takes care of the first part:

$$\begin{aligned} R_m &: \tilde{X} \to \tilde{X} \\ R_m(\{\{\mathbf{p}, \ldots\}\}) &= \{\{m(\mathbf{p}), \ldots\}\}. \end{aligned} \tag{7.37}$$

Function m selects one of the $2n$ neighbor coordinates of $\mathbf{p}$, e.g., either at random or according to a certain movement pattern (along a filament, only in increasing x direction etc.). Some variant of R_m is likely to be found in each individual-based DES; being an intra-rule, its effect is restricted to the state set X — yet it should now also trigger the distribution of the particles at the target site. The link between both levels is done via rule composition in the high-level multi-algorithm DES. Let the function

$$p^{-1} : \mathbb{N}^3 \to \mathbb{N} \tag{7.38}$$

denote the inverse of $p(l)$, i.e., given a sub-volume coordinate (x, y, z) the function returns the index of the appropriate sub-volume index, so $p^{-1}(p(l)) = l, l \in [1, L]$. The following inter-rule distributes the particles currently present inside sub-volume

$l = p^{-1}(\mathbf{p})$ among the adjacent sites:

$$\begin{aligned} &\mathcal{R}_2 : \mathbb{N}^{L\times N} \times \tilde{X} \to \mathbb{N}^{L\times N} \times \tilde{X} \\ &\mathcal{R}_2(\mathbf{X}, X) = (\mathbf{X} + \sum_{i=1}^{N}\Big(-x_{l,i}\mathbf{1}_{l,i}^{L\times N} + \sum_{k=1}^{2n} h(x_{l,i}, k)\mathbf{1}_{\mathbf{C}_{l,k},i}^{L\times N}\Big), X), \end{aligned} \tag{7.39}$$

This rule indeed looks quite complicated. The first part of the right hand side is a matrix update, setting the entries in the l-th row to zero (thus no particles are left inside sub-volume l) and adding the displaced particles to the neighbor states. Function $h(x, k)$ calculates how many particles are pushed towards each site $l^* \in \mathbf{C}_l$; if, e.g.,

$$\mathrm{P}(l^* \in \mathbf{C}_l \text{ is selected as the target for a single particle}) = \frac{1}{2n} \tag{7.40}$$

and $x_{l,i}$ being the number of particles for species S_i inside l, then $h(x_{l,i}, 1) = \mathrm{Binom}(x_{l,i}, 1/2n)$, $h(x_{l,i}, 2) = \mathrm{Binom}(x_{l,i} - h(x_{l,i}, 1), 1/(2n-1))$, and, more general[6]:

$$\begin{aligned} h &: \mathbb{N} \times \mathbb{N} \to \mathbb{N} \\ h_1 &\equiv h(x, 1) \sim \mathrm{Binom}(x, 1/2n) \\ h_k &\equiv h(x, k) \sim \mathrm{Binom}(x - h_{k-1} - \ldots - h_1, 1/(2n - k + 1)). \end{aligned} \tag{7.41}$$

The final step missing is to integrate R_m and $\mathcal{R}_2$ into the time interval and rule selection functions from Equations 7.35 and 7.36. For simplicity, movement updates are assumed to take place at fixed time intervals Δt. Then finding the next event time is simply a matter of adding Δt to the interval selection:

$$\begin{aligned} t &: \mathbb{N}^{L\times N} \times \tilde{X} \to \mathbb{R}_0 \\ \tau &\equiv t(\mathbf{X}, X) = \min\{t(\mathbf{X}), t(X), \Delta t\}, \end{aligned} \tag{7.42}$$

If R_m gets invoked before $\mathcal{R}_2$, the latter uses the updated position $\mathbf{p}$ as an input for

[6] $h(x, k)$ is actually a sample from a binomial distribution, which itself is dependent on previously generated samples.

p^{-1} and thus distributes the particles as intended:

$$r : \mathbb{N}^{L\times N} \times \tilde{X} \times \mathbb{R}_0 \to \mathbb{N}^{L\times N} \times \tilde{X}$$

$$r(\mathbf{X}, X, \tau) = \begin{cases} r(X, \tau) & \text{if } \tau = t(X) \\ r(\mathbf{X}, \tau) & \text{if } \tau = \hat{t}_d(\mathbf{X}) \vee (\tau = \hat{t}_D(\mathbf{X}) \wedge p(l^*) \neq \mathbf{p}) \\ \mathcal{R}_1(\mathbf{X}, X, l) & \text{if } \tau = t_D(\mathbf{X}) \wedge p(l^*) = \mathbf{p} \\ \mathcal{R}_2(\mathbf{X}, R_m(X)) & \text{if } \tau = \Delta t \end{cases} \tag{7.43}$$

To sum it up: r *re-uses* a rule defined at a lower level as part of a rule composition. Note that making the decision when to move the individual is not necessarily required to be performed in the higher-level component and could be left within the individual-based DES.

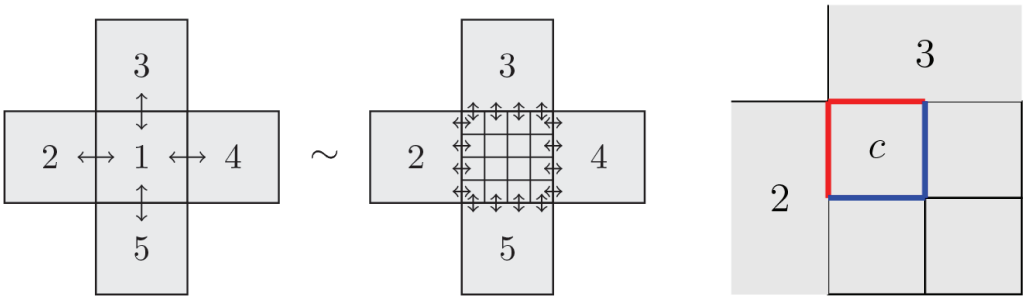
7.6 Individuals With Arbitrary Shapes

In the previous sections an individual had exactly the size and shape of a single sub-volume — nothing more, nothing less. It moved by "jumping" from one site to another and particles present at the target position are "pushed out" into neighboring sub-volumes. But biological entities assume various shapes, e.g., they are elongated like filaments, spherical or oval like cells — but very unlikely take the form of a cube. This restriction shall now be lifted and the following paragraphs will introduce one way how to allow arbitrary shapes for individuals but still represent them inside the grid of sub-volumes.

Multi-resolution Spatial Simulation Before going into detail on the multi-algorithm simulation, it is necessary to take a small "detour" and discuss a multi-resolution extension for population-based, spatial algorithms. Up to this point each sub-volume was assumed to have a fixed side length λ_{sv} and its interior was considered to be a well-stirred mix of particles. For some species A, the rate at which its particles leave a sub-volume towards a neighboring site is given by:

$$d(x_A) = \frac{2nD}{\lambda_{sv}^2} x_A, \tag{7.44}$$

with x_A being the current number of A particles. Now the algorithms are modified such that some sub-volumes can be represented with an increased spatial detail. Let



(a) (b)

Figure 7.6: Multi-resolution population-based spatial simulation. (a) With $k = 4$, the center sub-volume has been replaced with 16 sub-cells, yet the behavior of the system, i.e., the number of diffusing particles into and out of the sub-volumes, should still be similar to the original system. (b) Sub-cell c has four neighbors, two of them being sub-volumes (red, $f_c = 2$), the rest other sub-cells (blue, $g_c = 2$).

the parameter $k \in \mathbb{N}$ denote the *partition depth* for a partition of a sub-volume into a number k^n of smaller volumes (they will be referred to as *sub-cells* in the further course) with a respective length scale of $\lambda_{sc} = \lambda_{sv}/k$. The diffusion rate for a species A leaving one of those sub-cells is:

$$d_k(x_A) = \frac{2nDk^2}{\lambda_{sv}^2} \frac{x_A}{k^n}. \tag{7.45}$$

But simply replacing one large sub-volume with the smaller sub-cells, as shown in Figure 7.6a, would lead to the wrong results because the rate for particles leaving a sub-cell towards one of the neighbors 2 to 5 is based on the distance λ_{sc} — they would leave sub-volume 1 much more frequently than in the original system. To see this, remember that the rate for particles diffusing away via one of the $2n$ sides should match for both systems, so:

$$\frac{d(x_A)}{2n} = \alpha k^{n-1} \frac{d_k(x_A)}{2n} \tag{7.46}$$

Now this may need some explanations. If a sub-volume is divided into k^n sub-cells, then each of its sides actually consists of k^{n-1} sub-cell sides and the rate at which particles diffuse through one of those is $d_k(x_A)/2n$. The additional $\alpha = 1/k$ is required as a *scaling factor*, ensuring that the diffusion rates match between the original and partitioned system (see Figure 7.7). Multiplying Equation 7.45 with $\alpha/2n$ gives therefore

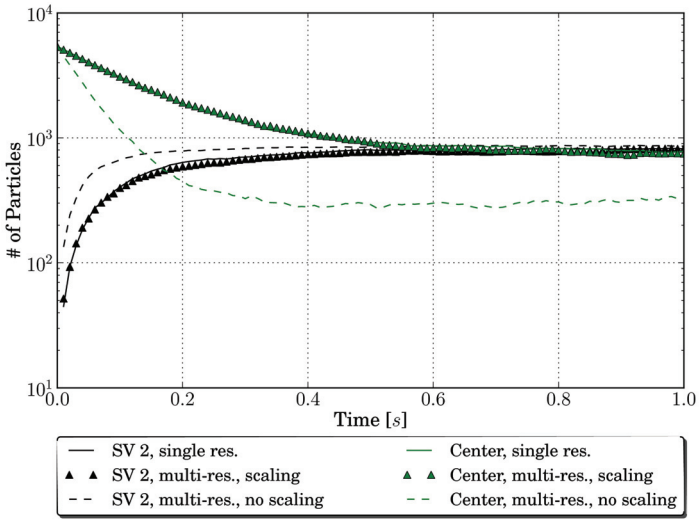


Figure 7.7: Example trajectories for a 3D version of the system shown in Figure 7.6a. There is a single species A initially present only in the center sub-volume. The plot shows how its population changes over time in the center sub-volume and one of its neighbors. Solid lines represent the non-partitioned model, triangle and dashed lines depict the results for a system with a partitioned center sub-volume (a $3 \times 3 \times 3$ grid). Without scaling (dashed line), the dynamics are quite different compared to the non-partitioned case.

the diffusion rate for a species A in case the neighbor is a non-partitioned sub-volume (sub-cell c to sub-volume 2 or 3 in Figure 7.6a).

Algorithms for a population-based spatial simulation can be easily adapted for the multi-resolution case. A sub-cell is treated just like a normal sub-volume, with its own entries in the state matrix $\mathbf{X}$ and connectivity matrix $\mathbf{C}$. The next event time for a diffusion from a sub-cell c to some neighbor is given by:

$$\tau_c = \min_{i \in [1,N]} \{t_1(x_{c,i}), t_2(x_{c,i})\}, \tag{7.47}$$

with

$$t_1(x_{c,i}) = \frac{-\ln U(0,1) 2nk}{f_c d_k(x_{c,i})}, \quad t_2(x_{c,i}) = \frac{-\ln U(0,1) 2n}{g_c d_k(x_{c,i})}. \tag{7.48}$$

Factor f_c counts how many sides of c have a non-partitioned sub-volume as a neighbor, while g_c does the same for sub-cells (ref. Figure 7.6b); if either of both happens to be zero, than the respective time is set to infinity. If a diffusion takes place next in c and $t_1 < t_2$, then a target is selected out of the set of neighboring sub-volumes; if instead $t_2 < t_1$, a sub-cell is chosen. Whenever a particle enters a partitioned sub-volume from

a non-partitioned one, it is added to the state of a randomly selected sub-cell.

Discretizing the Shape In the previous section about representing individual entities at the population level, an individual's shape exactly matched the size of a single sub-volume, regardless of its true appearance. A first step towards more complex shapes can be done by allowing several sub-volume to make up an individual's form, so that at least something that is larger than a site can be somehow represented. But this would still only provide a very rough approximation and does not solve the problem how to treat smaller entities. However, the basic idea to use more than one sub-volume goes in the right direction — but the sub-volumes an individual is composed of have to be smaller, which can be realized by *partitioning* the original sites into smaller sub-cells. This technique is nothing new and already existed for decades in the field of computer graphics under the name of *voxelization*, "the process of converting a geometric representation of a synthetic model into a set of voxels (pixels having a volume, i.e., cubes) that best represent that synthetic model within the discrete *voxel space*" [KCY93] (also known as volume synthesis, 3D scan-conversion, or cell decomposition).

Applying voxelization to spatial stochastic algorithms essentially results in a dynamic multi-resolution spatial simulation. Sub-volumes intersecting with an individual are split up into sub-cells, for which then the partition algorithm detects the sites currently occupied (ref. to Figure 7.8a). Particles present in sub-cells now being part of the individual are distributed among free sites, which can be either cells of the same sub-volume or neighbors thereof.

The interaction between individuals and particles within sub-volumes (or sub-cells) is very similar to what has been discussed earlier. Whenever a particle tries to diffuse into a region belonging to an individual, the upper level component of a multi-algorithm simulation intercepts this event and checks if a state changing rule needs to be applied or not. Conversely, each position update of an individual causes a repartitioning of all sub-volumes intersecting with it. However, this could be a performance bottleneck if it occurs very frequently, i.e., if individuals move fast and thus the time between two position updates is very short. There are different methods how to make this step more efficient. For example, instead of dividing a sub-volume into k^n cells a quadtree partition can be used [FB74] (or its 3D equivalent, the octree), which may reduce both the effort to find an individual's space occupancy and the number of sub-cells added to the multi-resolution spatial simulation algorithm (see Figure 7.8b). Furthermore, it is

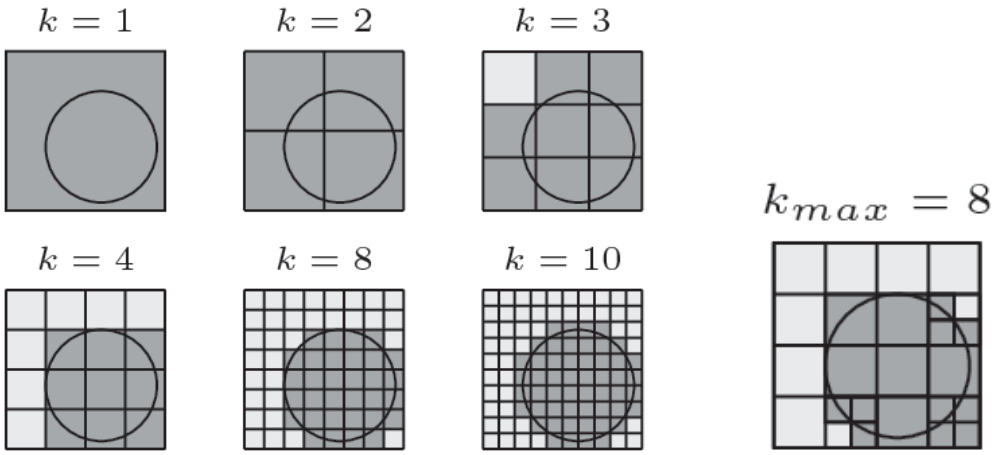


Figure 7.8: Partition of a sub-volume to represent a circular-shaped individual. The darker regions are the sub-cells currently occupied by the object (for $k = 1$, this is the entire sub-volume). (a) Increasing the partition depth k better captures the appearance of the entity. (b) A quadtree partition, with a maximum depth of eight. The number of new sub-volumes is 24, much less than the 64 required in (a) and $k = 8$.

not necessary to actually include the sub-cells in the simulation, but simply use them only to calculate the time when either a particle will leave the sub-volume or collide with the individual. After the partitioning, each one of the sub-cells not occupied has the same concentration of particles and three possible types of neighbors: another free sub-cell, a sub-cell being part of an individual or a sub-volume. Let x_A denote the total number of particles for species A in sub-volume l and $\hat{k}$ the number of free sub-cells in l, so $\hat{k} \in [1, k^n]$ (at least one sub-cell has to be free, otherwise nothing would take place inside the sub-volume). Now equations similar to 7.47 and 7.48 are used, which calculate the times when the next particle will either leave the sub-volume (t_1 and t_2) or hit the individual (t_3):

$$t_1(x_A) = \frac{-\ln U(0,1)\lambda_{sv}^2 \hat{k}}{f_l D k x_A}, \quad t_2(x_A) = \frac{-\ln U(0,1)\lambda_{sv}^2 \hat{k}}{g_l D k^2 x_A}, \quad t_3(x_A) = \frac{-\ln U(0,1)\lambda_{sv}^2 \hat{k}}{h_l D k^2 x_A}, \tag{7.49}$$

with

- f_l as the number of sides leading towards a non-partitioned sub-volume (red in Figure 7.9)
- g_l as the number of sides leading towards sub-cells of neighboring sub-volumes

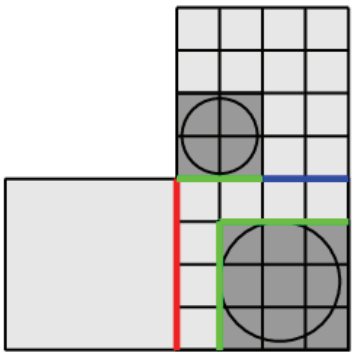


Figure 7.9: Spatial multi-resolution simulation without explicitly including the sub-cells. In the bottom right sub-volume nine sub-cells are occupied by an individual. There are two sub-cell sides leading out of it into cells of another sub-volume (blue), four sides are shared with a non-partitioned sub-volume (red), and a diffusion via eight sides would lead to a collision with an individual (green).

(blue in Figure 7.9), and

- h_l as the number of sides leading towards sub-cells occupied by an individual (green in Figure 7.9).

This modification can be interpreted as a more finer event classification: particles still leave the sub-volume, but depending on their target the diffusion event will apply different intra- and inter rules.

To sum this part of the chapter up: Individual entities occupy space which is not accessible by any other object, neither other individuals nor the well-mixed particles within sub-volumes. By discretizing their shape they can be physically represented at the level of a population-based spatial stochastic simulation: from the latter's perspective there exist some regions in space no particle can diffuse into. Any of those attempts is instead intercepted by a higher level simulation component and interpreted as a collision with an individual — which could trigger a state change in the latter. So both levels influence each other: the population level provides potential reactions partners, but does so without requiring them to be present as stateful entities; individuals, on the other hand, reduce the volume available for the particles to react and diffuse, an effect known as excluded volume. Though this effect is not represented among populations (each particle is assumed to have a negligible size compared to a sub-volume), there should be differences in the dynamics between systems with and without the presence of individuals.

7.7 Realizing the Multi-algorithm DES Concept

To provide a proof of concept for the presented multi-algorithm discrete event simulation approach, a prototype algorithm supporting all the capabilities introduced so far has been implemented. This section will first focus on the necessary extension for the Next Sub-volume Method to allow a multi-resolution grid representation. It then continues with the actual multi-algorithm simulator and discusses several details of its implementation. The section concludes with examples showing how to apply the algorithm to a selection of problems.

7.7.1 Prerequisites: Multi-resolution NSM

Representing individuals within a grid of well-stirred sub-volumes, as discussed in Section 7.6, requires a spatial population-based algorithm that supports the *splitting* and *merging* of sites. Splitting is the process of dividing a sub-volume into k^n smaller sub-cells (with k as partition depth), each having approximately the same concentration; "approximately" because working with discrete numbers of particles makes it sometimes impossible to distribute a sub-volume population equally among all sub-cells. Merging is the reverse process: the state a of sub-volume is restored from the states of the cells it has been split into.

The NSM (see Algorithm 3.5) has been chosen as the algorithm to adapt for a multi-resolution spatial simulation (mrNSM). As a first step, the state matrix $\mathbf{X}$ is modified to also include the state vectors of all sub-cells; so instead of $L \times N$, its size is now $(L+1)k^n \times N$, with rows one to L holding the state of the sub-volumes and $L+1$ to Lk^n the number of particles inside the sub-cells:

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\ \cdots & \cdots & \cdots & \cdots \\ x_{L,1} & x_{L,2} & \cdots & x_{L,N} \\ x_{L+1,1} & x_{L+1,2} & \cdots & x_{L+1,N} \\ \cdots & \cdots & \cdots & \cdots \\ x_{L+k^n,1} & x_{L+k^n,2} & \cdots & x_{L+k^n,N} \\ x_{L+k^n+1,1} & x_{L+k^n+1,2} & \cdots & x_{L+k^n+1,N} \\ \cdots & \cdots & \cdots & \cdots \\ x_{L+2k^n,1} & x_{L+2k^n,2} & \cdots & x_{L+2k^n,N} \\ \cdots & \cdots & \cdots & \cdots \\ x_{(L+1)k^n,1} & x_{(L+1)k^n,2} & \cdots & x_{(L+1)k^n,N} \end{pmatrix} \tag{7.50}$$

Algorithm 7.1: Pseudo-code description for the mrNRM's split method that distributes the current state of a sub-volume among its sub-cells.

Parameters:
$\mathbf{X}$ (current state, an $(L+1)k^n \times N$ matrix)
EQ (event queue)
k, n, L (partition depth, dimension of the system, number of sub-volumes)
Input:
$l \in [1, L]$ (sub-volume index)

1 get the indices of all sub-cells belonging to l:
2 $SC \leftarrow [L + (l-1)k^n + 1, L + lk^n]$;
3 distribute the particles from l among the cells in SC:
4 $\mathbf{X}_c \leftarrow \lfloor \mathbf{X}_l / k^n \rfloor$, $\forall c \in SC$;
5 distribute remaining ($\mathbf{X}_l \mod k^n$) particles randomly among the cells in SC;
6 remove l from EQ ;
7 calculate: $a_c, d_c, s_c, \tau_c \; \forall c \in SC$;
8 enqueue each sub-cell $c \in SC$;

Algorithm 7.1 shows a pseudo-code description of the split method. Line 4 calculates the integer part of the division $\mathbf{X}_l/k^n$, i.e., the number of particles that each sub-cell initially contains at least. The remaining particles ($\mathbf{X}_l \mod k^n$) that cannot be distributed equally are then assigned randomly to the sub-cells in the next line. By removing the split sub-volume l from the event queue in line 6, it will not be considered by the mrNSM any longer during the process of determining the next event. Algorithm 7.2 describes the reverse operation to the split method: the number of particles from each of l's sub-cells is added to retrieve the state of l. In the next step, the sub-cells get removed from the event queue and the index of the re-created sub-volume added.

Apart from adding the split and merge methods, it is also necessary to modify the steps that calculate the diffusion rate of a species and determine the target sub-volume of a diffusion event; they are now additionally dependent on the neighbors, i.e., whether the source and target are of the same type (i.e., both are either a sub-volume or a sub-cell) or not. Processing a diffusion event for a sub-volume is only slightly different from the original method: the diffusion rate is calculated as usual, however, if during the state update the selected target site is split, then the diffusing particle gets added to a random sub-cell. The case is different for a sub-cell. If all neighbors are also sub-cells, then the process is the same as for a sub-volume; but if not, then the rates for particles leaving towards a sub-volume need to be calculated with the scaling factor in mind,

Algorithm 7.2: Pseudo-code description for the mrNRM's merge method that restores the state of a split sub-volume.

Parameters:
$\mathbf{X}$ (current state, an $(L+1)k^n \times N$ matrix)
EQ (event queue)
k, n, L (partition depth, dimension of the system, number of sub-volumes)
Input:
$l \in [1, L]$ (sub-volume index)

1 get the indices of all sub-cells belonging to l:
2 $SC \leftarrow [L + (l-1)k^n + 1, L + lk^n]$;
3 update l's state:
4 $\mathbf{X}_l \leftarrow \sum_{c \in SC} \mathbf{X}_c$;
5 calculate: a_l, d_l, s_l, τ_l;
6 remove each sub-cell $c \in SC$ from EQ;
7 enqueue l;

as discussed in Section 7.6. What has been already indicated by Figure 7.7 is further underlined by Figure 7.10, which shows the particle distribution of a single species for a simple 5×5 model as simulated by the original NSM and the multi-resolution variant with (Figure 7.10 c) and without (Figure 7.10 d) considering the scaling factor; without it, diffusion events out of the sub-cells into sub-volumes occur much more frequently than between sub-volumes, leaving a smaller number of particles within the sub-cells.

It could be interesting to see how the choice of k influences the execution speed of the algorithm. A larger value allows a finer voxelization of the individual but also increases the number of sub-cells that get added when a sub-volume is split. Figure 7.11 plots the run time of the algorithm for different values of k (the model is the same as in Figure 7.10). Increasing the number of sub-cells makes the simulation slower, so a trade-off must be found between execution speed and spatial resolution.

The implemented mrNSM algorithm is a stand-alone module and can be run independently from the multi-algorithm simulator that will be introduced next.

7.7.2 The Multi-algorithm Simulator

The design of the algorithm which coordinates the execution of underlying mrNSM and individual-based simulators follows closely the abstract description given throughout Sections 7.4 to 7.5. Both subsidiary simulators implement methods representing the interval and rule selection functions; the former provides the time and the latter the

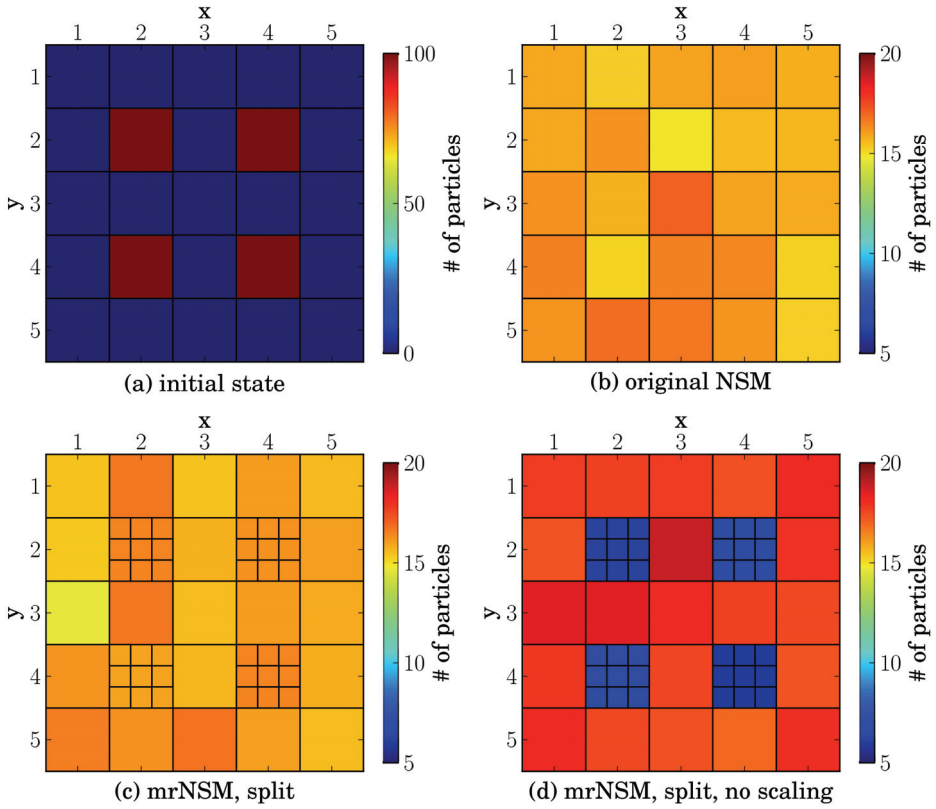


Figure 7.10: Multi-resolution NSM results for a simple 5×5 model (populations are averaged over 100 replications). (a) Initially only four sub-volumes contain 100 particles each, the remaining sites are empty. (b) After $0.5s$ the particles are distributed nearly homogeneously in the volume. (c) The four sub-volumes that initially contained the particles have been split before the simulation starts and again merged at the end. With scaling enabled, the mrNSM shows similar results to the original variant. (d) However, without scaling, the final particle distribution is considerably different.

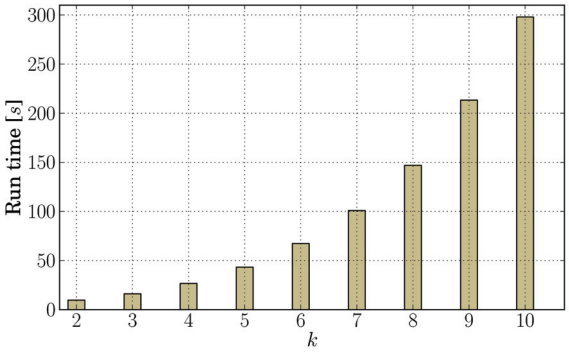


Figure 7.11: The run time of the mrNSM algorithm for different values of k for the model shown in Figure 7.10 (averaged over 50 replications). Decreasing the size of the sub-cells and thus making their total number larger has a significant impact on the execution speed of the algorithm.

information about the type of the next event. An additional methods takes care of executing the selected event. The multi-algorithm (MA) simulator can access these functions and decide which simulator should step next. By analyzing the returned event information it is able to interfere with the normal event processing. For example, after an individual has changed its position, the set of sub-volumes occupied by it has to be updated. Furthermore, these information are also required for deciding if an inter-rule can be applied. Again, as an example, before the mrNSM executes a diffusion event it is tested whether the target site is occupied by an individual. If this is the case, then the MA simulator checks the registered inter-rules for one that is defined for the pair of particle and individual species and, assuming the search is positive, applies it.

Splitting, Merging, and Particle Displacement One important aspect of the algorithm is the dynamic splitting and merging of sub-volumes as part of the voxelization of an individual. As it was mentioned, the former takes place whenever an individual changes its position or new objects are created. In the current implementation a shape (e.g., a circle) is assigned to each individual, which is used for an overlap test with the set of sub-volumes. Given a side length λ_{sv} and the coordinates (x_l, y_l) of sub-volume l, the spatial extent of the site ranges from $(\lambda_{sv}x_l, \lambda_{sv}y_l)$ to $(\lambda_{sv}x_l + \lambda_{sv}, \lambda_{sv}y_l + \lambda_{sv})$. Testing for occupancy starts with an arbitrary point $\mathbf{p}$ lying inside the individual's shape, e.g., the center of a circle. The sub-volume $\hat{l}$ containing this point is the one for

which:

$$(\lambda_{sv} x_{\hat{l}}, \lambda_{sv} y_{\hat{l}}) < \mathbf{p} \leq (\lambda_{sv} x_{\hat{l}} + \lambda_{sv}, \lambda_{sv} y_{\hat{l}} + \lambda_{sv}). \tag{7.51}$$

Knowing that $\hat{l}$ is overlapped by the individual, the test continues with its neighbors $\mathbf{C}_{\hat{l}}$ and subsequently processes and splits all sub-volumes that

- are occupied by the individual or
- adjoin a site that is (partly) occupied[7].

While the first condition does not need an explanation, the second one seems counter-intuitive as more sub-volumes get split than should be actually necessary. For each split sub-volume the resulting sub-cells are also tested for intersection with the individual; all overlapped sub-cells eventually represent the shape of the individual at the population level. The reason for the second condition is that it is unclear how to test for a collision between a particle and an individual if the former diffuses from a sub-volume into a split site where only some sub-cells belong to the individual[8].

Merging reduces the number of sub-cells and should take place whenever a sub-volume is no longer occupied by an individual. Detecting this is not as simple as it may initially appear: the occupancy check described above is done for a single individual, testing all sub-volumes it previously overlapped and merging those for which this is no longer the case does not consider sub-cells which are occupied by more than one individual. To facilitate the merge process, two additional data structures are required (see also Figure 7.12). The first one can be thought of as a mapping from a sub-volume index to a set of individuals; with it, the MA simulator keeps track about what sub-volumes are currently "controlled" by any individual. The second structure is basically the reverse mapping, i.e., from individuals to the sub-volumes they control. At the beginning of the occupancy check, the set of indices for the active individual is retrieved from the second map and used to remove for each of its entries the individual from the first map (a new set is built during the process described above and the appropriate entries made in the first map). Whenever a sub-cell executes an event, it is checked whether the sub-volume it belongs to has any controlling individuals (i.e., if there are sub-cells occupied by an individual); if not, then all sub-cells the site has

[7] Of course it should be guaranteed that the test is only performed for sites that does not have been checked before, otherwise the procedure runs indefinitely.

[8] An alternative is possible: when a particle diffuses from a sub-volume into a split site, the target sub-cell is determined randomly. If the cell is occupied by the individual, then a collision takes place — but it is unknown where the particle hit the individual in the first place if the target sub-cell is located within the individual, i.e., if it is entirely surrounded by occupied cells.

SV → {individual}
$1 \rightarrow \{I_1\}$
$2 \rightarrow \{I_1, I_2\}$
$3 \rightarrow \{\}$
$\vdots$
$L \rightarrow \{I_3, I_4\}$

individual → {SV}
$I_1 \rightarrow \{1, 2\}$
$I_2 \rightarrow \{2\}$
$I_3 \rightarrow \{12, 14\}$
$\vdots$
$I_O \rightarrow \{L, L-1\}$

Figure 7.12: Additional data structures used by the MA simulator. The left one is a mapping from sub-volumes indices to the set of individuals that currently control the site. A second structure holds the reverse mapping, i.e., from individuals to the set of sub-volume indices they control. No individual occupies any sub-cell of sub-volume 3, so it is candidate for merging.

been split into can be merged. The just described process can be seen as a delayed merging, triggered by an event execution in the mrNSM algorithm.

During the split process, the particles present inside occupied sub-cells get displaced ("pushed out") into free sites around the individual. What is simple for the case when the individual has the size of a sub-volume and only moves from one site to a neighbor (see Section 7.5) gets more complicated for complex shapes that get voxelized into a number of sub-cells; in fact, this issue, among others, has not been resolved satisfactorily so far. Currently there are two approaches. The first method collects all states of the overlapped sub-volumes and distributes the particles randomly among the free sub-cells; it is used when a new individual is introduced to the systems as there is no distinct direction into which the particles are pushed. But using this also after an individual has moved to a new position can lead to strange results. Considering all free sites, and not just the ones lying ahead of the individual, may result in a particle appearing "behind" the object, i.e., it is moved opposing to the direction the individual is translated. This scenario is avoided by the second distribution method. Only sub-volumes located in the movement direction are affected and the particles should therefore be displaced into sites next to those.

Step Size of an Individual While an individual moves through space sub-volumes get split with each position updated and merged again whenever their sub-cells are no longer occupied. The maximum displacement of an individual during a movement event should be limited to a certain size, otherwise there may be large "holes" in the spatial distribution of the particles, i.e., a large number of sub-volumes previously occupied

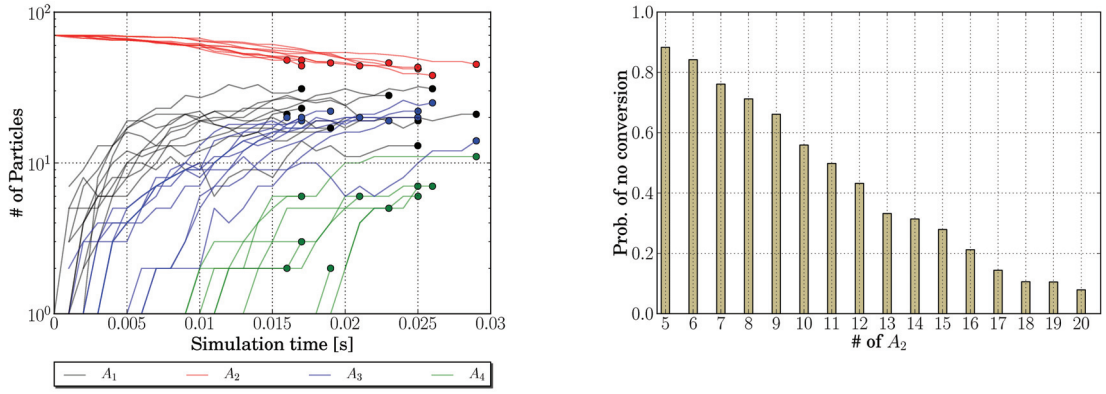
but after the position update empty until particles diffuse back into them from the surrounding sites. To attenuate these artifacts, the upper bound for the distance an individual is allowed to move at each iteration shall be given by the side length λ_{sc} of a *sub-cell*. For Brownian motion, the time between two movement events is fixed at Δt_{BD}, whose size depends on the microscopic time scale of the system $\tau = \min_i\{\tau_i\}$, with $\tau_i = (2r_i)^2/D_i$, r_i as radius of particle i, and D_i as its diffusion constant, and is usually in the range of $10^{-7}\tau$ to $10^{-4}\tau$[9]. Restricting the maximum size of a step to value smaller than λ_{sc} means setting $\Delta t = \min\{\Delta t_{BD}, \min_i\{\lambda_{sc}^2/D_i\}\}$.

A First Example To show how the multi-algorithm simulation works, Example 7.4.1 shall be re-visited:

$$\begin{aligned} R_1 &: A_1 + A_2 \xrightarrow{k_1} A_3 \\ R_2 &: A_1 + A_3 \xrightarrow{k_2} A_4 \\ R_3 &: A_3 + A_4 \xrightarrow{k_3} \text{create } C \text{ particle} \end{aligned} \tag{7.52}$$

The model is defined for a 5×5 grid, with the individual A located at coordinate $(2, 2)$, i.e., it completely fills out the center sub-volume. An equal amount of A_1 particles is initially placed inside each sub-volume; the species has a diffusion constant $D_{A_1} = 10$ and its particles are allowed to move freely inside the volume. At the start of a simulation run the individual A contains only A_2 particles, so A_1 needs to be transfered from the outside into A to initiate the conversion process; the probability for this event to occur when an A_1 particle collides with A is set to 0.5. Furthermore, all internal reactions have equal rate constants of 1.0. Note that reactions R_2 and R_3 both require A_3 as a reactant: R_2 transforms A_1 and A_3 into an A_4 particle while R_3 takes one of the latter and A_3 to trigger the creation of a type C individual. However, it is not sure that A gets transformed at all. If all A_2 particles have reacted with A_1, but the resulting A_3 are only consumed via R_2, then the conversion reaction will never fire. This makes stochasticity an important factor which cannot be neglected. Figure 7.13a provides a look inside an A individual during the simulation. It shows the trajectories for the four internal species; a dot at the end marks the time point when A is transformed into a C

[9] The time scale can be derived from the mean square displacement in one direction of a particle undergoing Brownian motion, $\langle x^2 \rangle = 2D_i\Delta t$, and setting $\langle x^2 \rangle = (2r_i)^2$. τ_i is then the time required for one particle to move a distance corresponding to its own diameter. To avoid missing collisions between individuals, the actual time step Δt_{BD} must be much smaller than the time scale of the system.



(a) (b)

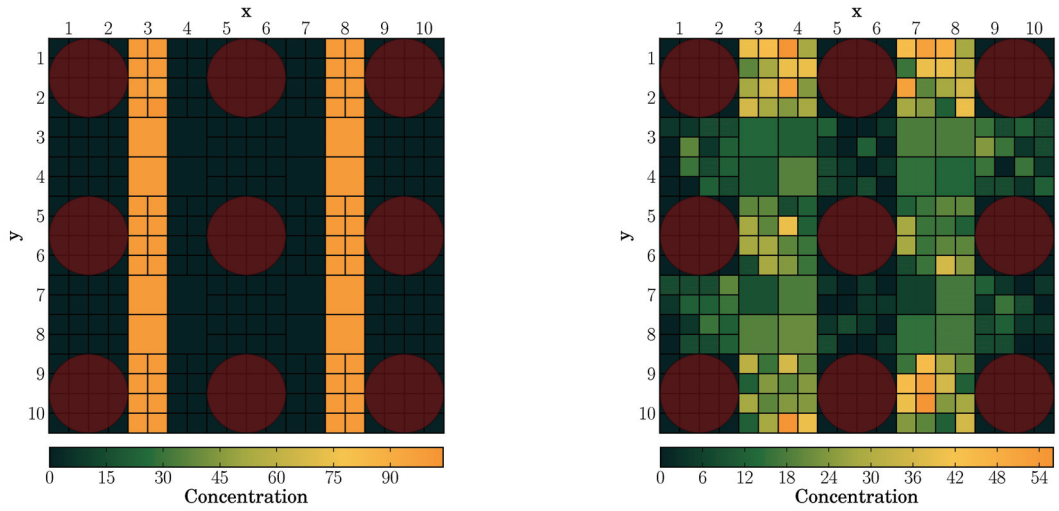
Figure 7.13: Analysis of Example 7.4.1. (a) Trajectories for the four internal species A_1 to A_4 of an A individual. The dot at the end of each line marks the time point where an A_3 particle reacted with an A_4 particle and thus initiated the conversion from an A to the C individual. (b) Probability that the conversion takes place for different initial values of A_2 (based on 1000 replications).

individual. Initially there were 50 A_1 particles per non-occupied sub-volume and 70 A_2 molecules inside A; with this amount of A_2 the conversion always took place in each of the 100 replications. In Figure 7.13b, the probability that an A particle is transformed into a C particle (or: that reaction R_3 will fire at all) is plotted against different initial values for A_2 (based on 1000 replications); it clearly shows that stochasticity plays a major role when it comes to the fate of an A particle, e.g., with 11 A_2 molecules initially present there is nearly a fifty percent chance that A will not be transformed.

7.8 Example Experiments

The chapter shall be concluded with two example studies: the first one analyzes if the physical presence of the individuals has any influence on the dynamics simulated by the population-based algorithm, the second one discusses a more complex qualitative model for the process of lipolysis.

Crowding The problem statement already introduced the excluded volume effect caused by macromolecules: each individual occupies a certain amount of space which is not accessible by others. Reaction dynamics are altered in two possible ways when crowding is considered, depending on the location of the reactants. If they are already



(a) (b)

Figure 7.14: Concentration (number of particles / area of sub-volume or sub-cell) of A and B particles for the crowding experiment. Nine individuals have been added to the system, which occupy approximately 28% of the available volume. The plots show the state of the system at $t = 0$ (a) and $t = 0.5$ (b).

close together and cannot diffuse away because the path is blocked by others, then the probability for a reactive encounter is increased; the contrary is the case for a scenario where they are still far away from each other and it is difficult for them to move into proximity and initiate the reaction.

As a first simple experiment it shall be observed if the excluded volume effect can also be observed in a multi-algorithm simulation; the focus is laid on the population level, as crowding effects are inherently present in an individual-based simulation. The initial assumption is that the presence of individuals should be noticeable simply because the available overall volume for reactions and diffusions is reduced and thus the effective concentration of the particles increased; additionally, the individuals act as obstacles for the particles, blocking their path through space. The model is defined over a 10×10 grid with a variable number of individuals that are either static, i.e., cannot move, or follow Brownian motion (with a diffusion constant set to 1.0). Figure 7.14a shows the initial setup: particles for two species A and B are separated from each other and only located inside two columns; they can both diffuse freely and annihilate each other on

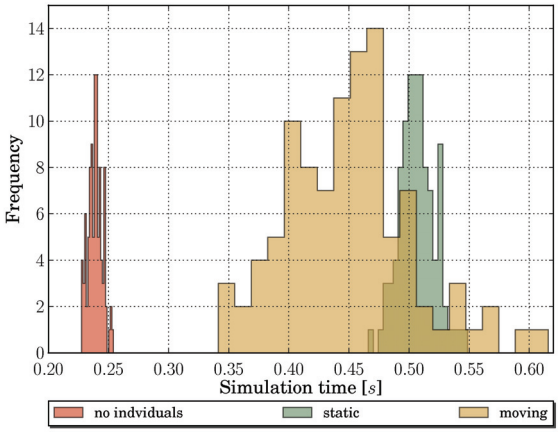


Figure 7.15: Histograms for the simulation times at which half of the population of A and B particles have degraded. While the results are closely scattered around the mean for dilute case, it takes not only longer if individuals are added to the system but the simulation times are also spread over a larger internal.

contact, represented by the following reaction:

$$R_a : A + B \xrightarrow{c_1} \emptyset, \tag{7.53}$$

with $c_1 = \infty$, so as soon as two particles of A and B are inside the same sub-volume they react. How many individuals are added to the systems depends on how pronounced the crowding effect should be; in normal cells approximately 20% to 40% of the entire volume is excluded by macromolecules [Ell01, CKL04], so the size and number of individuals is chosen to reflect these scenarios. For example, if all individuals have the same circular shape with radius $r = 1$, then the area a single object occupies is $\pi r^2 = \pi$; given the 10×10 grid and $\lambda_{sv} = 1$, then nine individuals exclude almost 28% of the volume.

In Figure 7.15 the histograms (50 replications) for the time it takes until half of the population has degraded are shown for three scenarios: without, nine static, and nine moving individuals. Comparing the respective distributions reveals that they are significantly different; if individuals are added to the systems, then the average time until 500 reactions took place between A and B particles is nearly doubled. But not only the mean, also the standard deviation in the end times is larger when individuals are present ($\sigma = 0.05$); the results are spread over an interval ranging from 0.34 s to 0.62 s, compared to the dilute scenario, where the times do not deviate that much from the mean (0.23 s to 0.25 s, with $\sigma = 0.006$). However, the difference between

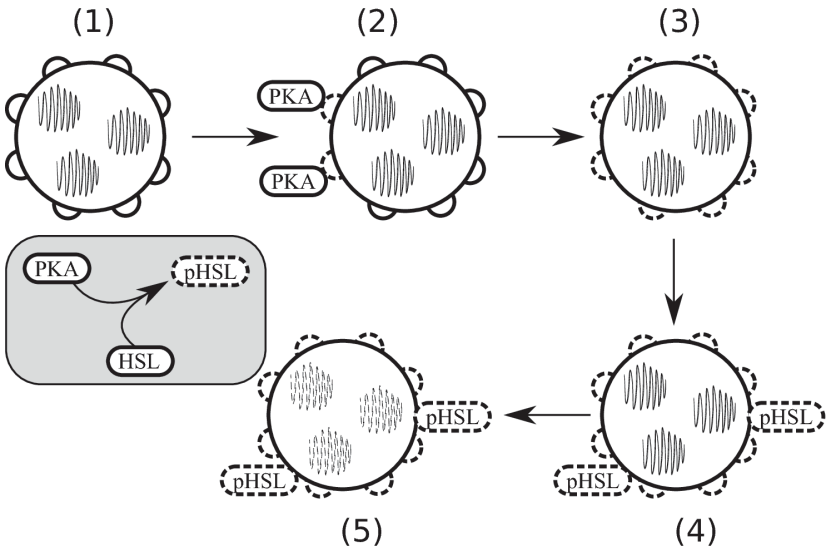


Figure 7.16: Illustration of the lipolysis model. (1) A lipid droplet with a coating of perilipin proteins, which protect the stored TAG from getting hydrolyzed by $pHSL$ (gray box). (2) Perilipin gets phosphorylated by the protein kinase A (PKA). (3) - (4) When the droplet is no longer protected by perilipin, phosphorylated HSL is able to attach to the droplet and initiate the hydrolysis of TAG (5).

crowded and dilute conditions is still small compared to the two orders of magnitude as predicted in, e.g., [Ell01]. Nevertheless, it can be shown that at least for systems where the reactants first have to travel a certain distance in space the excluded volume effect caused by the presence of larger individuals cannot be neglected.

A Model for Lipolysis One highly efficient energy source for several organisms is fat: it produces more energy per kilogram than saccharides (e.g., sugar), so basically less resources are needed to ensure the function of the body. Fat is stored as triacylglycerols (TAG) in *lipid droplets* [MP06], "containers" that have TAG at their core and are surrounded by a monolayer of phospholipids and associate proteins. Though all cells can develop droplets, they are mainly found inside special "fat cells", the adipocytes. If the organism requires energy from fat, the TAG have to be broken down into glycerol and free fatty acids (the actual *lipolysis* process, see, e.g., [KBRS+07, MSZ+06, LBS+99]), with the latter then being used in the energy production process. This conversion from TAG into fatty acids is done by a lipase; in adipocytes, this role is taken by the hormone-sensitive lipase (HSL). Figure 7.16 shows a schematic representation of the process. Before HSL can become active, it first needs to be phosphorylated by a kinase (protein kinase A, PKA) and then "dock" at the droplets to start the hydrolysis of

TAG. However, the translocation of the lipase to the droplet is prevented by perilipin, a protein present at the surface of the TAG storage; but if perilipin is phosphorylated as well (again by PKA), then the surface structure is altered and phosphorylated HSL ($pHSL$) is able to dock and induce the hydrolysis process.

This is a very basic description of lipolysis, a more detailed discussion can be found in, e.g., [LBS+99]. The system is suitable for a multi-algorithm simulation: there are larger individuals moving through space, with reactions occurring inside of them them (the lipid droplets and the hydrolysis of TAG), and smaller particles surrounding the individuals that both interact with each other (phosphorylation of HSL by PKA) and with the larger entity (phosphorylation of perilipin by PKA, translocation of phosphorylated HSL to the droplet).

For the individual DES it is assumed that the droplets are of spherical shape with a certain radius r; they have as features a position $\mathbf{p}$, the number of unphosphorylated perilipin A proteins ($perA$), and an internal species state vector holding the amount of stored TAG and bounded phosphorylated HSL enzymes; so the state of the individual DES can be defined as:

$$\begin{aligned} X &= \{X_{LD}\} \\ X_{LD} &= \{F_{LD,1}, \ldots, F_{LD,O}\} \\ F_{LD,i} &= \{\mathbf{p}_i, r, x_{i,perA}, \mathbf{x}^i\}, \end{aligned} \tag{7.54}$$

with $\mathbf{x}^i = (x_{i,pHSL}, x_{i,TAG}, x_{i,freeA}, x_{i,gly})$. The number of unphosphorylated perilipin proteins ($x_{i,perA}$) is not part of the internal state vector; perilipin does not participate in any of the reactions within the individual but its state is changed solely via an inter-rule, i.e., whenever a PKA kinase attaches a phosphate to the protein. Inside an individual $F_{LD,i}, i \in [1, O]$ a single reaction can take place: the conversion of a TAG into free acid and glycerol molecules (whose numbers are given by $x_{i,freeA}$ and $x_{i,gly}$) by $pHSL$:

$$\begin{aligned} R_l &: \tilde{X} \times \mathbb{N} \to \tilde{X} \\ R_l(X, i) &= \{\{\ldots, \{\mathbf{p}_i, r, x_{i,perA}, \mathbf{x}^i_{new}\}, \ldots\}\} \\ \mathbf{x}^i_{new} &= \mathbf{x}^i + (0, -1, 1, 1)^T; \end{aligned} \tag{7.55}$$

it is assumed that the $pHSL$ enzyme does not get consumed during this process. Moving the individual is done using the previously defined intra-rule from Equation 7.37.

The state for the population-based DES is given by an $L \times 3$ matrix $\mathbf{X}$:

$$\mathbf{X} = \begin{pmatrix} x_{1,PKA} & x_{1,HSL} & x_{1,pHSL} \\ \cdots & \cdots & \cdots \\ x_{L,PKA} & x_{L,HSL} & x_{L,pHSL} \end{pmatrix} \tag{7.56}$$

storing the number of PKA, HSL, and $pHSL$ particles currently present in the volume. The only reaction that is defined is the phosphorylation of HSL by PKA, which results in one $pHSL$ molecule:

$$R_p : PKA + HSL \rightarrow PKA + pHSL \tag{7.57}$$

All molecules are furthermore allowed to diffuse freely between the sub-volumes.

The link between the individual- and population-based DES is done via two inter-rules. Whenever a PKA molecule collides with a droplet, there is a chance that one of the unphosphorylated perilipin proteins gets phosphorylated by the kinase:

$$\begin{aligned} \mathcal{R}_{pp} &: \mathbb{N}^{L\times N} \times \tilde{X} \times \mathbb{N} \rightarrow \mathbb{N}^{L\times N} \times \tilde{X} \\ &\mathcal{R}_{pp}(\{\mathbf{X}, X\}, i) = \{\mathbf{X}, \{\{\ldots, \{\mathbf{p}_i, r, x_{i,perA} - 1, \mathbf{x}^i\}, \ldots\}\}. \end{aligned} \tag{7.58}$$

No sub-volume index is required as the PKA molecule does not get consumed by the reaction and the state of the sub-volume where the diffusion event originated is therefore left unchanged. The second inter-rule represents the docking of an HSL particle:

$$\begin{aligned} \mathcal{R}_{pH} &: \mathbb{N}^{L\times N} \times \tilde{X} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}^{L\times N} \times \tilde{X} \\ &\mathcal{R}_{pH}(\{\mathbf{X}, X\}, i, l) = \{\mathbf{X} - \mathbf{1}^{L\times N}_{l,pHSL}, \{\{\ldots, \{\mathbf{p}_i, r, x_{i,perA}, \mathbf{x}^i_{new}\}, \ldots\}\} \\ &\mathbf{x}^i_{new} = \mathbf{x}^i + (1,0,0,0)^T; \end{aligned} \tag{7.59}$$

the $pHSL$ molecule gets transfered from sub-volume l into the individual. Instead of writing down the time interval and rule selection functions, whose main components have been already introduced above, the discussion shall be restricted to a specific event condition. Depending on how the lipolysis process shall be represented, it may be that not all perilipin proteins need to be phosphorylated before a $pHSL$ enzyme is able to attach to the droplet. A condition for inter-rule $\mathcal{R}_{pH}$ therefore probably considers the current value of $x_{i,perA}$: for example, the rule shall only be applied if the amount of unphosphorylated perilipin has dropped below a certain threshold; an

alternative definition could make the probability for executing the rule dependent on this amount, with a lower value for $x_{i,perA}$ making it more likely that a $pHSL$ docks to the individual.

Figure 7.17 shows snapshots of the model state at different time points. Performing a qualitative analysis, the following parameterization has been chosen arbitrarily: the reactions constants for the events that apply the intra-rules R_l and R_p have been set to 1.0; the probability that a collision of a PKA molecule with a lipid droplet (LD) leads to a phosphorylation of one of the perilipin proteins is 0.1; after all of the latter have been phosphorylated, a $pHSL$ enzyme can dock at a droplet with probability 0.25 and start initiating the hydrolysis. At the beginning of a simulation, each LD individual consists of a layer that contains 10 perilipin proteins, which protect 70 TAG molecules from being hydrolyzed by $pHSL$ enzymes; in all non overlapped sub-volumes 50 PKA and 100 HSL particles are present. Qualitative models, similar to benchmark models, can be used to test various assumptions made regarding the dynamics, e.g., how modifying certain reaction constants change the behaviour of the system or how much longer it would take if the number of perilipin proteins is increased (see Figure 7.18b). As an example, Figure 7.18 shows the distribution of the simulation times at which half of the TAG molecules have been hydrolyzed (100 replications); for a comparison, the histogram for a modified model is given in Figure 7.18a as well, where an additional degradation reaction $pHSL \rightarrow HSL$ has been added, which directly competes with the inter-rule that attaches a $pHSL$ molecule to the lipid droplet.

7.9 Additional Remarks

There are several topics that have not been covered in the last sections. For instance, nothing has been written about collision detection between individuals. However, as this is specific to an individual-based simulation algorithm and the focus of this dissertation was more on how to *combine* it with a spatial stochastic simulation in discretized space, this was intentionally left out.

Furthermore, it is still an open issue how to determine the probabilities for the decision whether a collision between a particle and an individual should lead to the application of specific inter-rules. Looking at algorithms for an individual-based simulation, one can find several different techniques how it is ensured that the dynamics at the microscopic scale reflect those that have been observed at the macroscopic level.

Using probabilities dates back to the works of Collins and Kimball [CK49]; this

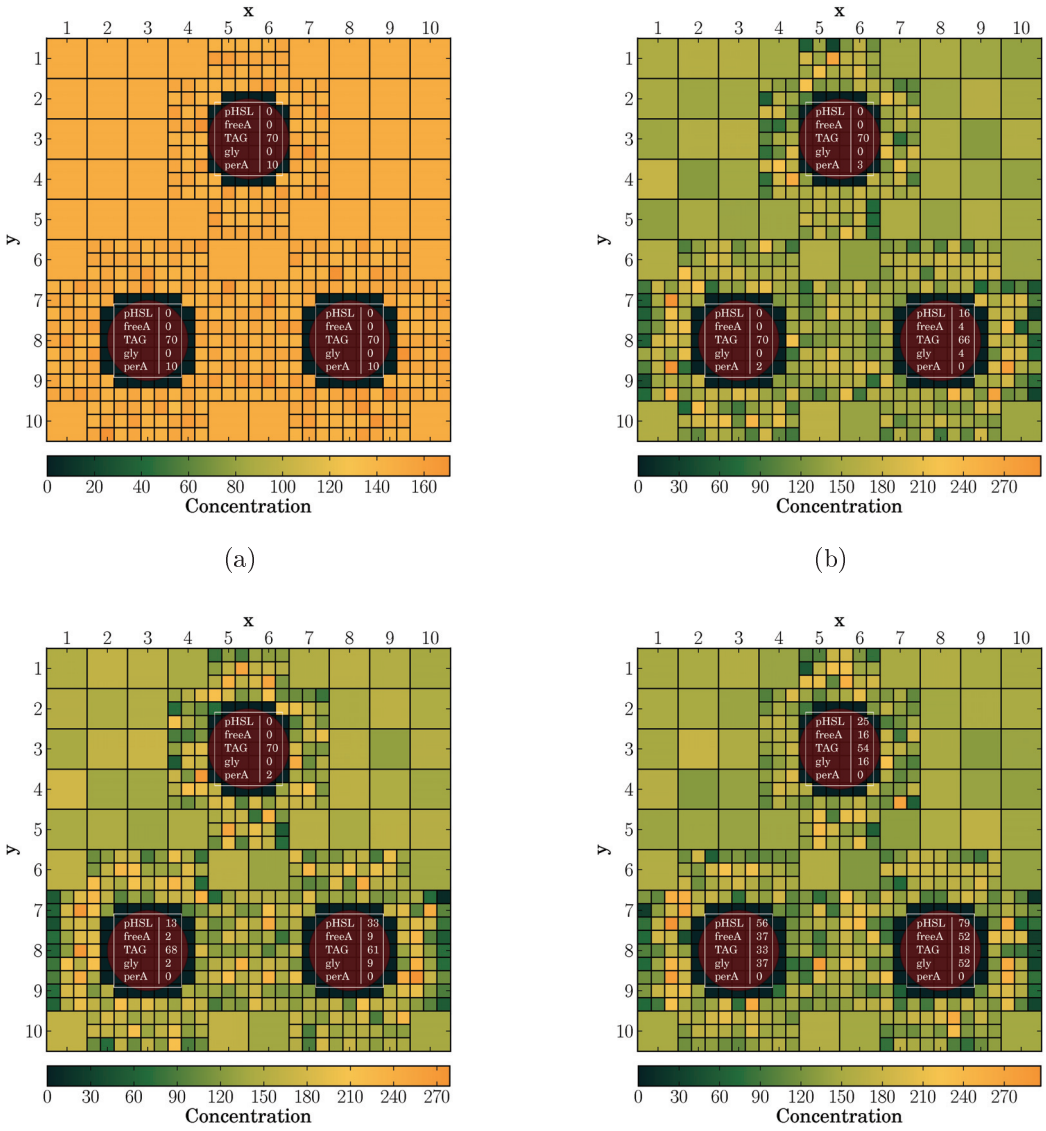
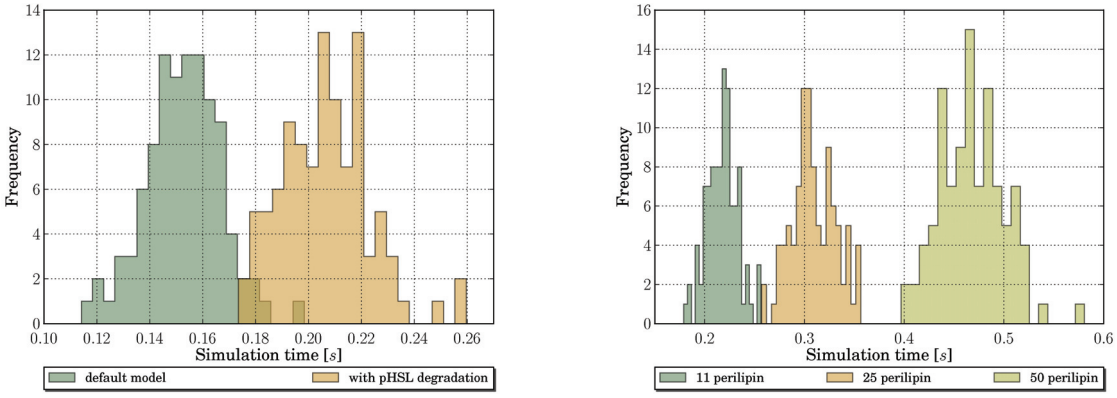


Figure 7.17: Series of states for the lipolysis model. (a), $t = 0$: The initial state of the model, three lipid droplets have been placed inside the volume. There are 50 PKA and 100 HSL molecules within each sub-volume. (b), $t = 0.1$: The lower right droplet lost its protective coating of perilipin proteins and, as a result, the first phosphorylated HSL enzymes have already docked to it and initiated the lipolysis. (c), $t = 0.12$: All perilipin proteins have been phosphorylated for the lower left droplet. (d), $t = 0.18$: The TAG conversion started in the third droplet. At this point half of the TAG population have been hydrolyzed into glycerol (gly) and free fatty acids ($freeA$).



(a) (b)

Figure 7.18: Histograms for the simulation times at which half of the TAG population has been hydrolyzed. (a) The results indicate that it takes longer for the conversion process to reach the set threshold if the phosphorylated HSL enzymes are allowed to get dephosphorylated. (b) Results for different numbers of perilipin proteins. As expected, the conversion takes considerably longer if more proteins protect the stored TAG.

method is also used, e.g., in the thesis by Arjunan [Arj09], where the individuals move between spheres arranged in a grid and two entities can react with a certain probability if one of them tries to move to the site occupied by the other. The author establishes a relation between the macroscopic rate constant of the reaction and the probability that it will occur on collision; the latter is determined *a priori* by taking the ratio between the number of reactive encounters (which is given for a specified time interval by the reaction propensity) and the expected number of collisions, calculated for a homogeneous entity distribution.

An alternative method without probabilities is the *Smoldyn* algorithm presented by Andrews and Bray [AB04]. A critical argument against probabilities is that in Brownian dynamics (BD) simulations a non-reactive collision is most likely followed by subsequent collisions between the same reactant partners, which implies that the reaction probability must very small (infinitesimal when the number of collisions is infinite, as argued by Andrews and Bray). In Smoldyn, this is avoided by letting two individuals undergo a reaction whenever their center-to-center distance is smaller than a specific binding radius, whose value is based on the macroscopic rate constant of the reaction; this radius does not have to correspond with the size of the involved reactants, so they do not need to collide at all but only move into close proximity.

When an individuals dissociates into two products, then the latter are placed at a specific distance, the unbinding radius, which must be larger than the binding radius to prevent the entities from reacting with each other again instantly. It should be noted that in the proposed multi-algorithm simulator the problem of infinite collisions between particles and individuals is not as present as in purely BD approaches because the spatial resolution is constrained to the size of the sub-cells and an exact collision detection, which would be triggered very often if the reactant's positions are very close to each other, is thus not performed.

The current implementation of the multi-algorithm simulator assumes that the probabilities for reactive encounters are known by the modeler. With more complex individuals, this probability likely depends on their state; for example, in the lipolysis model from Section 7.8, the reaction between a $pHSL$ molecule and a lipid droplet can only occur when there are a few or no unphosphorylated perilipin proteins left for protecting the TAG from getting hydrolyzed. So providing the probability *a priori* is in those cases not possible. But if there are no such dependencies, then second variant discussed above, using binding radii instead of probabilities, could be utilized for the presented MA simulator: if the reaction constants between particles and individuals are known, then the shape defined by the binding radius may be represented at the population level and used to decide whether particles are close enough to an individual to initiate a reaction. This may be one starting point for further research.

7.10 Summary

This chapter presented an exploration into the topic of multi-x methods by defining a discrete-event simulation (DES) scheme that coordinates the execution of two subordinate DES, the first one operating on populations of species in a discretized volume and the second one on individual entities. Having both populations and individuals coexisting inside the same environment makes it possible to represent phenomena such as the relocation of particles into confined mobile compartments where they can participate into complex internal reaction networks that influence the behaviour of the individual.

As a starting point, Section 7.2 established a basic tool set for the description of discrete-event simulations (DES), which has been based on concepts that have already been introduced in Chapter 2: the representation of a system as a set of state variables that are modified at discrete time instances by the application of rules. Two essential

functions that steer the execution of a DES have been defined, the time selection function t and the rule selection function r: the former calculates when the next event will take place, the latter provides the rules that should be applied when the event is executed. Each one of the algorithms discussed in the survey (Chapter 3) basically implements these functions and it has been shown in Section 7.3 how this general framework can also be applied to representations that interpret species as a set of individual entities.

A first step towards a multi-algorithm simulation was made by introducing the concept of inter-rules that act as links between different DES: in contrast to intra-rules, e.g., reactions, they have access to more than one state representation (Section 7.4). Intra-rules are still handled by the respective DES, but the decision when an inter-rule should be applied is made within a coordinator component. This hierarchical scheme is used to describe a combination of a population-based spatial stochastic DES with an individual DES; here inter-rules can be used to model, e.g., the transport of a particle into an individual when both collide.

Making both DES actually aware of each other (i.e., representing a collision between a particle and an individual) was the second major topic during the first part of this chapter (Sections 7.5 and 7.6). In the most simple case the individuals fill out a single sub-volume and can only move into adjacent sites; after the position update, particles currently present inside the target sub-volume get displaced into the neighbors. For a finer shape representation a technique called voxelization is used that approximates the form of an individual by partitioning the sub-volumes into smaller sub-cells and checking each cell if it overlaps with the shape. Allowing this "splitting" essentially resulted in a multi-resolution spatial stochastic simulation, but it was necessary to introduce a modification to the diffusion rate calculation.

The presented theoretical concept of a multi-algorithm DES has been realized as a coordinator that manages the executions of a multi-resolution Next Sub-volume Method and a simple individual-based simulation algorithm (Section 7.7). Though there are still open issues, e.g., how to determine the probability that a collision between a particle and an individual leads to the application of inter-rules, the method is already suitable to perform experiments with qualitative models, e.g., to test whether the physical presence of the individuals has an influence on the dynamics at the population level (Section 7.8).

8 Conclusion and Future Work

All of the individual research projects presented in this dissertation find their place in the larger scheme of evaluating, improving, and exploring algorithms for a stochastic simulation of biochemical reaction networks. Despite being placed in the middle, Chapter 5 has been the first milestone. A start into the field of stochastic simulation is soon accompanied by a feeling of becoming overwhelmed: there are a plethora of algorithms already available, both exact and approximative, and several more are added if spatially inhomogeneous systems and multi-x methods are considered as well. Faced with the task to evaluate these algorithms, it has been realized that some fundamental concepts are required to ensure a sound comparison. Without exception, every stochastic simulation algorithm needs at least one sub-algorithm (a source for random numbers); more sophisticated methods require additional inputs, such as further sub-algorithms or parameters. It is one key statement of this thesis that these dependencies cannot be neglected during a study, a point which is underlined by the results from Chapter 6. Looking at these also provides arguments for the working hypothesis that there is no "silver bullet", i.e., an algorithm dominating all competing variants for an arbitrary model instance, which has been formulated in the introduction. Each method has strengths and weaknesses, as evidenced by the spatial τ-leaping algorithm discussed in Chapter 4. Developed to offer an approximative algorithm that applies the leap idea to spatially discretized models, it is indeed faster than an exact variant if there are sufficient particles present in all sub-volumes; otherwise it cannot compete because the time required to find a leap candidate bears no proportion to the step size actually taken. An additional improvement in terms of distributing the work load among more than one processing unit showed better results for selected models, but also serves as a demonstration for the limits for a "parallelization inside a simulation run": if the overall work (in this case: finding the leap value and calculating the state update) is already small beforehand, then it is very unlikely to gain any benefits from distributing it.

A different research direction has been explored with Chapter 7. When it became clear that some models may require a more flexible representation of a species, an

approach was discussed that left the modeler the option to define a species as either a collection of individual entities, each one equipped with its own state, or a population of indifferent particles, distributed inside a discretized space. This basic idea lead to the general concept of a multi-algorithm discrete-event simulation: algorithms still handle intra-rules by themselves, however, their execution is synchronized by a higher level component which also takes care of the selection and application of *inter-rules* that form a link between the individual simulations. Artificial and biologically inspired models provided a glimpse at the questions that could be formulated and answered by the proposed method.

Future Work There are several routes that could be taken in future research endeavors, some of which have already been mentioned in the respective chapters. A direct continuation of the work done for evaluating simulation algorithm is the application of data mining tools to generate decision trees out of a set of model features and the data stored during the performance analysis. This has been shown only exemplarily in the dissertation, without going much into detail. The next step will be to apply these decision trees in real simulation studies in order to suggest a suitable algorithm that is expected to meet the requirements set by both the model and the user. Obviously, this selection would be based on the structure of the model and the initial state. With the latter being variable over time, a re-evaluation of the algorithm choice could be made during a simulation, which may be a step towards resolving the problem that a method performs well at the beginning but gradually becomes worse over time due varying system dynamics. Another step beyond the evaluation is an integration of the results into algorithm ontologies, such as the *Kinetic Simulation Algorithm Ontology* (KiSAO, [KLN08, KLNK09]), which offer a classification of algorithms into different categories. Efforts like KiSAO could benefit from performance studies in that the information provided with each algorithm are enriched by listing those model characteristics which proved to be (in)favorable for the method during an evaluation.

The evaluation concepts and their application in the two studies presented in this work merely scratched the surface of what is possible — and what may be necessary for algorithms to come. As one example, the GMPM requires an SSA as a sub-algorithm, which itself could have additional dependencies that have to be satisfied, so the configuration of GMPM is not flat but hierarchical; this aspect of the algorithm has not been considered in this work and could be analyzed in future evaluations. With regard to multi-algorithm methods, the problem of hierarchical configurations may become of

increasing interest and worthwhile to explore.

A clear direction for spatial τ-leaping is, from the current point of view, difficult to perceive. Though first steps have been made, it is still not entirely analyzed how strong the impact of spatial inhomogeneity is on the performance; what conditions does the particle distribution have to satisfy in order that spatial τ-leaping is considerably faster than an exact alternative? Going from an explicit variant to an implicit, as it has been done in the non-spatial case, is an arguable option: for this, the species need to a) have high diffusion constants and b) be distributed almost homogeneously. For those models the explicit algorithm would be very busy simulating the diffusion events, although their execution only marginally changes the model state. But if two sub-volumes have an almost equal amount of particles for all species, then the diffusions between them can be interpreted as reversible reactions being in partial equilibrium — and these can be ignored when calculating an implicit τ candidate [San09b]. However, without a nearly well-stirred mix of particles the final implicit leap value would be closer or even equal to the explicit one (the minimum is taken over all sub-volumes and with some of them having, e.g., empty neighbors the diffusion events between those sites would not be in partial equilibrium); in this case it is likely that the explicit variant is preferred over the implicit.

Still in a rather early state, the multi-algorithm DES offers a wide area for further exploration, both in a theoretical and practical direction. Though a prototype simulator is available, there is no formal way how to specify models for the use in a maDES. An ongoing work in the context of the research training group dIEM oSiRiS aims at developing a rule-based formalism that shall equip the user with a convenient language for expressing those models. On the practical side, the most obvious starting point is applying the method to real-world problems. The models presented in this work are biologically inspired, however, their parameterization was not based on real data and finding it, especially regarding the inter-rules, is still an open problem. An approach has been mentioned (using the technique behind Smoldyn to represent the binding radii at the population level), yet there is the question if there are alternatives that allow more complex interactions. Here a close cooperation with domain experts from various fields, e.g., biology and physics, will certainly be necessary.

To sum this work up: the task of an efficient simulation of non-spatial and spatial reaction networks has been addressed from three different directions. The first one, **efficiency via evaluation**, was aimed at comparing algorithms and identifying meth-

ods that perform better than others for specific problem instances. A second direction, **efficiency via improvement**, has been followed with the extension of leap methods into space and it was shown that for certain model characteristics the execution is faster than exact variants. Finally, **efficiency via separation of concerns** was explored in terms of a multi-algorithm DES concept where species can be either represented as populations or individuals.

Bibliography

[AB04] S. S. Andrews and D. Bray. Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. *Phys Biol*, 1(3-4):137–151, 2004. ISSN 1478-3967.

[ACK06] A. Auger, P. Chatelain, and P. Koumoutsakos. R-leaping: Accelerating the stochastic simulation algorithm by reaction leaps. *The Journal of Chemical Physics*, 125(8):084103+, 2006. doi:10.1063/1.2218339.

[And07] D. F. Anderson. A modified Next Reaction Method for simulating chemical systems with time dependent propensities and delays. 2007.

[And08] D. F. Anderson. Incorporating postleap checks in tau-leaping. *The Journal of Chemical Physics*, 128(5):054103, 2008. doi:10.1063/1.2819665.

[Arj09] S. N. V. Arjunan. *Modeling Three-Dimensional Spatial Regulation of Bacterial Cell Division*. Ph.D. thesis, Graduate School of Media and Governance, Keio University, 2009.

[BC00] M. Bienz and H. Clevers. Linking colorectal cancer to wnt signaling. *Cell*, 103(2):311–320, 2000.

[BCK08] B. Bayati, P. Chatelain, and P. Koumoutsakos. Multiresolution stochastic simulations of reaction-diffusion processes. *Phys. Chem. Chem. Phys.*, 10(39):5963–5966, 2008. doi:10.1039/b810795e.

[BHP81] J. H. Blackstone, G. L. Hogg, and D. T. Phillips. A two-list method for synchronization of event driven simulation. In *ANSS '81: Proceedings of the 14th annual symposium on Simulation*, pp. 95–101. IEEE Press, Piscataway, NJ, USA, 1981.

[BK99] G. C. Brown and B. N. Kholodenko. Spatial gradients of cellular phosphoproteins. *FEBS Letters*, 457(3):452–454, 1999. ISSN 00145793. doi:10.1016/S0014-5793(99)01058-3.

[BKL75] A. Bortz, M. Kalos, and J. Lebowitz. A new algorithm for monte carlo simulation of ising spin systems. *Journal of Computational Physics*, 17(1):10–18, 1975. ISSN 00219991. doi:10.1016/0021-9991(75)90060-1.

[BM96] F. Baras and M. M. Mansour. Reaction-diffusion master equation: A comparison with microscopic simulations. *Physical Review E*, 54(6):6139–6148, 1996. doi:10.1103/PhysRevE.54.6139.

[CCM09] A. J. Chien, W. H. Conrad, and R. T. Moon. A wnt survival guide: from flies to human disease. *The Journal of investigative dermatology*, 129(7):1614–1627, 2009. ISSN 1523-1747. doi:10.1038/jid.2008.445.

[CG91] F. E. Cellier and J. Greifeneder. *Continuous System Modeling*. Springer, 1 edition, 1991. ISBN 0387975020.

[CGP05a] Y. Cao, D. T. Gillespie, and L. R. Petzold. Accelerated stochastic simulation of the stiff enzyme-substrate reaction. *The Journal of chemical physics*, 123(14):144917, 2005. ISSN 0021-9606. doi:10.1063/1.2052596.

[CGP05b] Y. Cao, D. T. Gillespie, and L. R. Petzold. Avoiding negative populations in explicit poisson tau-leaping. *J. Chem. Phys.*, 123:4104+, 2005. doi:10.1063/1.1992473.

[CGP05c] Y. Cao, D. T. Gillespie, and L. R. Petzold. The slow-scale stochastic simulation algorithm. *The Journal of chemical physics*, 122(1):14116, 2005. ISSN 0021-9606. doi:10.1063/1.1824902.

[CGP06] 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, 2006. ISSN 0021-9606. doi:10.1063/1.2159468.

[CGP07] Y. Cao, D. T. Gillespie, and L. R. Petzold. Adaptive explicit-implicit tau-leaping method with automatic tau selection. *The Journal of Chemical Physics*, 126(22):224101+, 2007. doi:10.1063/1.2745299.

[CH09] F. Ciocchetta and J. Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theorical Computer Science*, 410(33-34):3065–3084, 2009. doi:10.1016/j.tcs.2009.02.037.

[Cha43] S. Chandrasekhar. Stochastic problems in physics and astronomy. *Reviews of Modern Physics*, 15(1):1–89, 1943. doi:10.1103/RevModPhys.15.1.

[CK49] F. Collins and G. Kimball. Diffusion-controlled reaction rates. *Journal of Colloid Science*, 4(4):425–437, 1949. Cited By (since 1996) 244.

[CK06] F. E. Cellier and E. Kofman. *Continuous System Simulation*. Springer, 1 edition, 2006. ISBN 0387261028.

[CKL04] N. A. Chebotareva, B. I. Kurganov, and N. B. Livanova. Biochemical effects of molecular crowding. *Biochemistry*, 69(11):1239–1251, 2004. ISSN 0006-2979.

[CLP04] Y. Cao, H. Li, and L. Petzold. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *J Chem Phys*, 121(9):4059–4067, 2004. ISSN 0021-9606. doi:10.1063/1.1778376.

[CP06] Y. Cao and L. Petzold. Accuracy limitations and the measurement of errors in the stochastic simulation of chemically reacting systems. *J. Comput. Phys.*, 212(1):6–24, 2006. ISSN 0021-9991. doi:10.1016/j.jcp.2005.06.012.

[CV06] A. Chatterjee and D. G. Vlachos. Temporal acceleration of spatially distributed kinetic monte carlo simulations. *J. Comput. Phys.*, 211(2):596–615, 2006. ISSN 0021-9991. doi:10.1016/j.jcp.2005.06.004.

[CVK05] A. Chatterjee, D. G. Vlachos, and M. A. Katsoulakis. Binomial distribution based tau-leap accelerated stochastic simulation. *The Journal of Chemical Physics*, 122(2):024112, 2005. doi:10.1063/1.1833357.

[CX06] X. Cai and Z. Xu. K-leap method for stochastic simulation of gene expression. In *2006 IEEE International Workshop on Genomic Signal Processing and Statistics*, pp. 81–82. IEEE, 2006. ISBN 1-4244-0384-7. doi:10.1109/GENSIPS.2006.353166.

[DC09] C. Dittamo and D. Cangelosi. Optimized parallel implementation of gillespie's first reaction method on graphics processing units. *Computer Modeling and Simulation, International Conference on*, 0:156–161, 2009. doi:10.1109/ICCMS.2009.42.

[DI00] C. Demetrescu and G. Italiano. What do we learn from experimental algorithmics? In *Mathematical Foundations of Computer Science 2000*, pp. 36–51. 2000. doi:10.1007/3-540-44612-5_3.

[DM08] L. Dematté and T. Mazza. On parallel stochastic simulation of diffusive systems. In M. Heiner and A. M. Uhrmacher (eds.), *Computational Methods in Systems Biology*, volume 5307, chapter 16, pp. 191–210. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-88561-0. doi:10.1007/978-3-540-88562-7_16.

[DNZ01] E. J. D'Amico, T. B. Neilands, and R. Zambarano. Power analysis for multivariate and repeated measures designs: A flexible approach using the SPSS MANOVA procedure. *Behavior Research Methods, Instruments, & Computers*, 33(4):479–484, 2001.

[EE04] 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. doi:10.1049/sb:20045021.

[EHU08] R. Ewald, J. Himmelspach, and A. M. Uhrmacher. An algorithm selection approach for simulation systems. In *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation (PADS) 2008*, volume 22, pp. 91–98. IEEE Computer Society, Los Alamitos, CA, USA, 2008. doi:10.1109/PADS.2008.9.

[Ell01] J. R. Ellis. Macromolecular crowding: an important but neglected aspect of the intracellular environment. *Current Opinion in Structural Biology*, 11(1):114–119, 2001. doi:10.1016/S0959-440X(00)00172-X.

[ELSS02] M. B. Elowitz, A. J. Levine, E. D. Siggia, and P. S. Swain. Stochastic Gene Expression in a Single Cell. *Science*, 297(5584):1183–1186, 2002. doi:10.1126/science.1070919.

[ELU09] R. Ewald, S. Leye, and A. M. Uhrmacher. An efficient and adaptive mechanism for parallel simulation replication. In *PADS '09: Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, pp. 104–113. IEEE Computer Society, Washington, DC, USA, 2009. ISBN 978-0-7695-3713-9. doi:10.1109/PADS.2009.11.

[EM06] R. J. Ellis and A. P. Minton. Protein aggregation in crowded environments. *Biological chemistry*, 387(5):485–497, 2006. ISSN 1431-6730. doi:10.1515/BC.2006.064.

[EU09] R. Ewald and A. M. Uhrmacher. Automating the runtime performance evaluation of simulation algorithms. In M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls (eds.), *Proceedings of the Winter Simulation Conference*, pp. 1079–1091. IEEE Computer Science, 2009.

[FB74] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974. ISSN 0001-5903. doi:10.1007/BF00288933.

[FH77] M. Feinberg and F. Horn. Chemical mechanism structure and the coincidence of the stoichiometric and kinetic subspace. In *Archive for Rational Mechanics and Analysis*, volume 66, pp. 83–97. Springer, 1977. doi:10.1007/BF00250853.

[FHL10] L. Ferm, A. Hellander, and P. Lötstedt. An adaptive algorithm for simulation of stochastic reaction-diffusion processes. *Journal of Computational Physics*, 229(2):343–360, 2010. ISSN 00219991. doi:10.1016/j.jcp.2009.09.030.

[Fic55] A. Fick. Über diffusion. *Annalen der Physik und Chemie*, 170(1):59–86, 1855. ISSN 1521-3889. doi:10.1002/andp.18551700105.

[FW95] T. Fricke and D. Wendt. The markoff-automaton - a new algorithm for simulating the time-evolution of large stochastic dynamic systems. *International Journal of Modern Physics C (IJMPC)*, 6:277–306, 1995.

[Gai79] B. Gaines. General systems research: quo vadis? In *General Systems: Yearbook of the Society for General Systems Research*, volume 24, pp. 1–9. 1979.

[GB00] 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, 2000. doi:10.1021/jp993732q.

[Gil76] D. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976. ISSN 00219991. doi:10.1016/0021-9991(76)90041-3.

[Gil77] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[Gil01] D. T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics*, 115(4):1716–1733, 2001. doi:10.1063/1.1378322.

[GLP07] D. T. Gillespie, S. Lampoudi, and L. R. Petzold. Effect of reactant size on discrete stochastic chemical kinetics. *The Journal of chemical physics*, 126(3):034302, 2007. ISSN 0021-9606. doi:10.1063/1.2424461.

[Gor77] G. Gordon. *System Simulation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1977. ISBN 0138817979.

[Gra93] P. Grassberger. On correlations in "good" random number generators. *Physics Letters A*, 181(1):43–46, 1993. doi:10.1016/0375-9601(93)91122-L.

[GT03] R. Goh and I. Thng. Mlist: An efficient pending event set structure for discrete event simulation. *International Journal of Simulation - Systems, Science & Technology*, 4(5-6):66–77, 2003.

[GT05] N. Gilbert and K. G. Troitzsch. *Simulation for the Social Scientist*. Open University Press, 2005.

[Hel98] P. Hellekalek. Good random number generators are (not so) easy to find. *Math. Comput. Simul.*, 46(5-6):485–505, 1998. ISSN 0378-4754.

[HEU08] J. Himmelspach, R. Ewald, and A. M. Uhrmacher. A flexible and scalable experimentation layer. In S. Mason, R. Hill, and O. R. L. Moench (eds.), *Proceedings of the Winter Simulation Conference*. 2008.

[HFE05] J. Hattne, D. Fange, and J. Elf. Stochastic reaction-diffusion simulation with mesord. *Bioinformatics*, 21(12):2923–2924, 2005. ISSN 1367-4803. doi:10.1093/bioinformatics/bti431.

[Him07] J. Himmelspach. *Konzeption, Realisierung und Verwendung eines allgemeinen Modellierungs-, Simulations- und Experimentiersystems - Entwicklung und Evaluation effizienter Simulationsalgorithmen*, volume 4. Sierke Verlag, Göttingen, 1 edition, 2007. ISBN 978-3-940333-73-5. Dissertation, Universität Rostock.

[HU07a] J. Himmelspach and A. M. Uhrmacher. The event queue problem and pdevs. In *Proceedings of the SpringSim '07, DEVS Integrative M&S Symposium*, pp. 257–264. SCS, 2007.

[HU07b] J. Himmelspach and A. M. Uhrmacher. Plug'n simulate. In *Proceedings of the 40th Annual Simulation Symposium*, pp. 137–143. IEEE Computer Society, 2007.

[IHC10] K. A. Iyengar, L. A. Harris, and P. Clancy. Accurate implementation of leaping in space: The spatial partitioned-leaping algorithm. *The Journal of Chemical Physics*, 132(9):094101, 2010. doi:10.1063/1.3310808.

[Jef05] P. R. Jeffries. A frame work for designing, implementing, and evaluating simulations used as teaching strategies in nursing. *Nursing Education Perspectives*, 26(2):96–103, 2005. doi:10.1043/1536-5026(2005)026<0096:AFWFDI>2.0.CO;2.

[Jen96] B. Jenkins. ISAAC, a fast cryptographic random number generator. `http://www.burtleburtle.net/bob/rand/isaacafa.html`, 1996.

[JLNU10] M. John, C. Lhoussain, J. Niehren, and A. Uhrmacher. The attributed pi calculus with priorities. *Transactions on Computational Systems Biology XII. Special Issue on Modeling Methodologies*, 5945:13–76, 2010. doi:10.1007/978-3-642-11712-1_2.

[Joh03] D. Johnson. *Data Structures, Near Neighbor Searches, and Methodology*, chapter A theoretician's guide to the experimental analysis of algorithms, pp. 215–250. American Mathematical Society, Oxford, United States, 2003. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.1550`.

[JPE+08] M. Jeschke, A. Park, R. Ewald, R. Fujimoto, and A. M. Uhrmacher. Parallel and distributed spatial simulation of chemical reactions. In *PADS '08: Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, pp. 51–59. IEEE Computer Society, Washington, DC, USA, 2008. ISBN 978-0-7695-3159-5. doi:10.1109/PADS.2008.20.

[JU10] T. Jahnke and T. Udrescu. Solving chemical master equations by adaptive wavelet compression. *J. Comput. Phys.*, 229:5724–5741, 2010. ISSN 0021-9991. doi:http://dx.doi.org/10.1016/j.jcp.2010.04.015.

[Kac85] B. Kachar. Direct visualization of organelle movement along actin filaments dissociated from characean algae. *Science*, 227(4692):1355–1357, 1985. doi:10.1126/science.4038817.

[Kal09] G. Kalantzis. Hybrid stochastic simulations of intracellular reaction-diffusion systems. *Computational biology and chemistry*, 33(3):205–215, 2009. ISSN 1476-928X. doi:10.1016/j.compbiolchem.2009.03.002.

[KBRS+07] J. Kovsan, R. Ben-Romano, S. C. Souza, A. S. Greenberg, and A. Rudich. Regulation of adipocyte lipolysis by degradation of the perilipin protein. *Journal of Biological Chemistry*, 282(30):21704–21711, 2007. doi:10.1074/jbc.M702223200.

[KCY93] A. Kaufman, D. Cohen, and R. Yagel. Volume graphics. *Computer*, 26(7):51–64, 1993. ISSN 0018-9162. doi:10.1109/MC.1993.274942. The definition can be found in the online version at http://www.cs.sunysb.edu/ vislab/projects/volume/papers.html.

[KLN08] D. Köhn and N. Le Novère. The kinetic simulation algorithm ontology (kisao) - a proposal for the classification of simulation algorithms in systems biology. Invited Talk, Super-Hackathon, Okinawa, Japan, 2008.

[KLNK09] D. Köhn, N. Le Novère, and C. Knüpfer. Beyond structure: Kisao and teddy - two ontologies addressing pragmatical and dynamical aspects of computational models in systems biology. 3rd International Biocuration Conference, 2009. doi:10.1038/npre.2009.3137.1.

[Knu76] D. E. Knuth. Big omicron and big omega and big theta. *SIGACT News*, 8(2):18–24, 1976. ISSN 0163-5700. doi:10.1145/1008328.1008329.

[Knu97] D. E. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd Edition)*. Addison-Wesley Professional, 3 edition, 1997. ISBN 0201896842.

[Kur72] T. G. Kurtz. The relationship between stochastic and deterministic models for chemical reactions. *The Journal of Chemical Physics*, 57(7):2976–2978, 1972. doi:10.1063/1.1678692.

[KW78] G. A. Korn and J. V. Wait. *Digital Continuous-system Simulation*. Prentice Hall, 1978. ISBN 013212274X.

[LBNS09] K. Leyton-Brown, E. Nudelman, and Y. Shoham. Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM*, 56(4):1–52, 2009. ISSN 0004-5411. doi:10.1145/1538902.1538906.

[LBS+99] C. Londos, D. L. Brasaemle, C. J. Schultz, D. C. adler Wailes, D. M. Levin, A. R. Kimmel, and C. M. Rondinone. On the control of lipolysis in adipocytes. *Annals of the New York Academy of Sciences*, 892(THE METABOLIC SYNDROME X: Convergence of Insulin Resistance, Glucose Intolerance, Hypertension, Obesity, and Dyslipidemias-Searching for the Underlying Defects):155–168, 1999. ISSN 1749-6632. doi:10.1111/j.1749-6632.1999.tb07794.x.

[LK99] A. Law and D. W. Kelton. *Simulation Modeling and Analysis (Industrial Engineering and Management Science Series)*. McGraw-Hill Science/Engineering/Math, 1999. ISBN 0070592926.

[LL97] A. LaMarca and R. E. Ladner. The influence of caches on the performance of sorting. In *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pp. 370–379. Society for Industrial and

Applied Mathematics, Philadelphia, PA, USA, 1997. ISBN 0898713900. doi:10.1006/jagm.1998.0985.

[LP06] H. Li and L. Petzold. Logarithmic direct method for discrete stochastic simulation of chemically reacting systems. Technical report, Department of Computer Science, University of California at Santa Barbara, Santa Barbara, USA, 2006.

[LY06] Lucien Birgé and Yves Rozenholc. How many bins should be put in a regular histogram. *ESAIM: P&S*, 10:24–45, 2006. doi:10.1051/ps:2006001.

[Mar95] G. Marsaglia. The Marsaglia random number CDROM including the Diehard battery of tests of randomness. `http://www.stat.fsu.edu/pub/diehard/`, 1995.

[Mar03] G. Marsaglia. Seeds for random number generators. *Commun. ACM*, 46(5):90–93, 2003. ISSN 0001-0782. doi:http://doi.acm.org/10.1145/769800.769827.

[MBS08] S. Macnamara, K. Burrage, and R. B. Sidje. Multiscale modeling of chemical kinetics via the master equation. *Multiscale Modeling & Simulation*, 6(4):1146–1168, 2008.

[McG96] C. C. McGeoch. Feature article - toward an experimental method for algorithm simulation. *INFORMS Journal on Computing*, 8(1):1–15, 1996.

[McL68] J. McLeod. *Simulation; the dynamic modeling of ideas and systems with computers.* McGraw-Hill, 1968. ISBN 9780070454330.

[Mil99] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999. ISBN 0521658691.

[Miy99] T. Miyaoka. Increased expression of wnt-1 in schizophrenic brains. *Schizophrenia Research*, 38(1):1–6, 1999. ISSN 09209964. doi:10.1016/S0920-9964(98)00179-0.

[MKFK04] R. T. Moon, A. D. Kohn, G. V. Ferrari, and A. Kaykas. Wnt and β-catenin signalling: diseases and therapies. *Nat Rev Genet*, 5(9):691–701, 2004. ISSN 1471-0056. doi:10.1038/nrg1427.

[MLB07] T. Márquez-Lago and K. Burrage. Binomial tau-leap spatial stochastic simulation algorithm for applications in chemical kinetics. *The Journal of Chemical Physics*, 127(10):104101, 2007. doi:10.1063/1.2771548.

[MN98] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998. ISSN 1049-3301. doi:10.1145/272991.272995.

[Mor02] B. M. E. Moret. Towards a discipline of experimental algorithmics. In M. H. Goldwasser, D. S. Johnson, and C. C. McGeoch (eds.), *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, volume 59 of *DIMACS Monographs*, pp. 197–213. AMS Press, 2002.

[MP06] S. Martin and R. G. Parton. Lipid droplets: a unified view of a dynamic organelle. *Nature Reviews Molecular Cell Biology*, 7(5):373–378, 2006. ISSN 1471-0072. doi:10.1038/nrm1912.

[MPC+06] J. M. McCollum, G. D. Peterson, C. D. Cox, M. L. Simpson, and N. F. Samatova. The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *Computational Biology and Chemistry*, 30(1):39–49, 2006. doi:10.1016/j.compbiolchem.2005.10.007.

[MSZ+06] H. Miyoshi, S. C. Souza, H.-H. Zhang, K. J. Strissel, M. A. Christoffolete, J. Kovsan, A. Rudich, F. B. Kraemer, A. C. Bianco, M. S. Obin, and A. S. Greenberg. Perilipin promotes hormone-sensitive lipase-mediated adipocyte lipolysis via phosphorylation-dependent and -independent mechanisms. *Journal of Biological Chemistry*, 281(23):15837–15844, 2006. doi:10.1074/jbc.M601097200.

[MWKA07] M. Matsumoto, I. Wada, A. Kuramoto, and H. Ashihara. Common defects in initialization of pseudorandom number generators. *ACM Trans. Model. Comput. Simul.*, 17(4), 2007. ISSN 1049-3301. doi:10.1145/1276927.1276928.

[Nus05] R. Nusse. Wnt signaling in disease and in development. *Cell research*, 15(1):28–32, 2005. ISSN 1001-0602. doi:10.1038/sj.cr.7290260.

[NVBDG77] I. Nemes, T. Vidóczy, L. Botár, and D. Dezsö Gál. A possible construction of a complex chemical reaction network. *Theoretical Chemistry Accounts: Theory, Computation, and Modeling (Theoretica Chimica Acta)*, 45(3):215–223, 1977. ISSN 1432-881X (Print) 1432-2234 (Online). doi:10.1007/BF00552683.

[NZC+07] J.-q. Niu, H.-r. Zheng, J.-s. Chen, M. Ma, and X.-f. Wang. A distributed-based stochastic simulation algorithm for large biochemical reaction networks. In *ICBBE 2007. The 1st International Conference on Bioinformatics and Biomedical Engineering, 2007*, pp. 502 –505. 2007. doi:10.1109/ICBBE.2007.132.

[Ope] OpenMP. http://openmp.org. Accessed 04/2010.

[PAYS09] K. S. Perumalla, B. G. Aaby, S. B. Yoginath, and S. K. Seal. Gpu-based real-time execution of vehicular mobility models in large-scale road network scenarios. In *PADS '09: Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, pp. 95–103. IEEE Computer Society, Washington, DC, USA, 2009. ISBN 978-0-7695-3713-9. doi:10.1109/PADS.2009.22.

[PB46] R. L. Plackett and J. P. Burman. The Design Of Optimum Multifactorial Experiments. *Biometrika*, 33(4):305–325, 1946. doi:10.1093/biomet/33.4.305.

[PL09] L. Petzold and H. Li. Efficient Parallelization of Stochastic Simulation Algorithm for Chemically Reacting Systems on the Graphics Processing Unit. *International Journal of High Performance Computing Applications*, 00:1094342009106066, 2009. doi:10.1177/1094342009106066.

[Pla83] R. L. Plackett. Karl pearson and the chi-squared test. *International Statistical Review / Revue Internationale de Statistique*, 51(1):59–72, 1983. ISSN 03067734. doi:10.2307/1402731.

[Pol00] P. Polakis. Wnt signaling and cancer. *Genes & development*, 14(15):1837–1851, 2000. ISSN 0890-9369. doi:10.1101/gad.14.15.1837.

[PQ05] C. Priami and P. Quaglia. Beta binders for biological interactions. In *Computational Methods in Systems Biology*, pp. 20–33. Springer, 2005.

[RBK08] D. Rossinelli, B. Bayati, and P. Koumoutsakos. Accelerated stochastic and hybrid methods for spatial simulations of reaction-diffusion systems. *Chemical Physics Letters*, 451(1-3):136–140, 2008. ISSN 00092614. doi:10.1016/j.cplett.2007.11.055.

[Riv72] P. Rivett. *Principles of Model Building: The Construction of Models for Decision Analysis*. John Wiley & Sons Ltd, 1972. ISBN 0471724653.

[RKDB06] 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. *Bioinformatics*, 22(15):1895–1901, 2006. doi:10.1093/bioinformatics/btl271.

[RKS98] A. Rojnuckarin, S. Kim, and S. Subramaniam. Brownian dynamics simulations of protein folding: Access to milliseconds time scale and beyond. *Proceedings of the National Academy of Sciences of the United States of America*, 95(8):4288–4292, 1998.

[Rob52] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[RPCG03] M. Rathinam, L. R. Petzold, Y. Cao, and D. T. Gillespie. Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. *The Journal of Chemical Physics*, 119(24):12784–12794, 2003. doi:10.1063/1.1627296.

[San09a] W. Sandmann. Sequential estimation for prescribed statistical accuracy in stochastic simulation of biological systems. *Mathematical Biosciences*, 221(1):43–53, 2009. ISSN 00255564. doi:10.1016/j.mbs.2009.06.006.

[San09b] W. Sandmann. Streamlined formulation of adaptive explicit-implicit tau-leaping with automatic tau selection. In *WSC'09: Proceedings of the 2009 Winter Simulation Conference, Austin, Texas*, pp. pp. 1104–1112. 2009.

[SBE09] P. Sjoberg, O. G. Berg, and J. Elf. Taking the reaction-diffusion master equation to the microscopic limit. eprint arXiv:0905.4629, 2009.

[Sch04] S. Schnell. Reaction kinetics in intracellular environments with macromolecular crowding: simulations and rate laws. *Progress in Biophysics*

and Molecular Biology, 85(2-3):235–260, 2004. ISSN 00796107. doi:10.1016/j.pbiomolbio.2004.01.012.

[Sci] J. Scimark. http://math.nist.gov/scimark2/. Accessed 02/2010.

[SES02] P. S. Swain, M. B. Elowitz, and E. D. Siggia. Intrinsic and extrinsic contributions to stochasticity in gene expression. *Proceedings of the National Academy of Sciences of the United States of America*, 99(20):12795–12800, 2002. doi:10.1073/pnas.162041399.

[SF01] P. Sanders and R. Fleischer. Asymptotic complexity from experiments? a case study for randomized algorithms. In *Algorithm Engineering: 4th International Workshop, WAE 2000, Saarbrücken, Germany, September 2000. Proceedings*, pp. 135+. Springer, 2001.

[Sha98] R. E. Shannon. Introduction to the art and science of simulation. In *WSC*, volume 2, pp. 7–14. IEEE Computer Society Press, Los Alamitos, CA, USA, 1998. ISBN 0-7803-5134-7.

[She07] D. J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 2007. ISBN 1584888148, 9781584888147.

[SJUS08] H.-J. Schulz, M. John, A. Unger, and H. Schumann. Visual analysis of bipartite biological networks. In *Proceedings of the Eurographics Workshop on Visual Computing for Biomedicine 2008*. Delft, Netherlands, 2008.

[Sol09] D. Soloveichik. Robust stochastic chemical reaction networks and bounded tau-leaping. *J. Comput. Biol.*, 16:501–522, 2009.

[Sta03] M. Stamp. Once upon a time-memory trade-off. `http://www.cs.sjsu.edu/faculty/stamp/RUA/TMTO.pdf`, 2003.

[Ste08] B. Stein. Model construction for knowledge-intensive engineering tasks. In Y. Liu, A. Sun, H. T. Loh, W. F. Lu, and E.-P. Lim (eds.), *Advances of Computational Intelligence in Industrial Systems*, volume 116 of *Studies in Computational Intelligence*, chapter 7, pp. 139–167. Springer, 2008. ISBN 978-3-540-78296-4. doi:10.1007/978-3-540-78297-1_7.

[Sto01] D. Stockburger. *Introductory Statistics: Concepts, Models, and Applications, Second Edition*. Atomic Dog Publishing, 2nd edition, 2001. ISBN 1931442460.

[STP08] A. Slepoy, A. P. Thompson, and S. J. Plimpton. A constant-time kinetic monte carlo algorithm for simulation of large biochemical reaction networks. *The Journal of Chemical Physics*, 128(20):205101, 2008. doi:10.1063/1.2919546.

[SV05] A. Samant and D. G. Vlachos. Overcoming stiffness in stochastic simulation stemming from partial equilibrium: A multiscale monte carlo algorithm. *The Journal of Chemical Physics*, 123(14):144114+, 2005. doi:10.1063/1.2046628.

[SYSY02] R. Srivastava, L. You, J. Summers, and J. Yin. Stochastic vs. deterministic modeling of intracellular viral kinetics. *Journal of Theoretical Biology*, 218(3):309–321, 2002. ISSN 00225193. doi:10.1006/jtbi.2002.3078.

[TB04] T. Tian and K. Burrage. Binomial leap methods for simulating stochastic chemical kinetics. *The Journal of Chemical Physics*, 121(21):10356–10364, 2004. doi:10.1063/1.1810475.

[TB05] T. Tian and K. Burrage. Parallel implementation of stochastic simulation for large-scale cellular processes. In *HPCASIA '05: Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region*, p. 621. IEEE Computer Society, Washington, DC, USA, 2005. ISBN 0-7695-2486-9. doi:10.1109/HPCASIA.2005.67.

[TD06] W. Trochim and J. P. Donnelly. *The Research Methods Knowledge Base*. Atomic Dog, 2006. ISBN 1592602916.

[TKHT04] K. Takahashi, K. Kaizu, B. Hu, and M. Tomita. A multi-algorithm, multi-timescale method for cell simulation. *Bioinformatics*, 20(4):538–546, 2004. ISSN 1367-4803. doi:10.1093/bioinformatics/btg442.

[TO01] M. Thattai and A. van Oudenaarden. Intrinsic noise in gene regulatory networks. *Proceedings of the National Academy of Sciences of the United States of America*, 98(15):8614–8619, 2001. doi:10.1073/pnas.151588598.

[TTO+03] Y. Terada, H. Tanaka, T. Okado, H. Shimamura, S. Inoshita, M. Kuwahara, and S. Sasaki. Expression and function of the developmental gene wnt-4 during experimental acute renal failure in rats. *J Am Soc Nephrol*, 14(5):1223–1233, 2003. ISSN 1046-6673. doi:10.1097/01.ASN.0000060577.94532.06.

[VB07] C. Versari and N. Busi. Stochastic simulation of biological systems with dynamical compartment structure. In M. Calder and S. Gilmore (eds.), *Computational Methods in Systems Biology*, volume 4695 of *Lecture Notes in Computer Science*, chapter 6, pp. 80–95. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-75139-7. doi:10.1007/978-3-540-75140-3_6.

[VDB04] R. Vuduc, J. W. Demmel, and J. A. Bilmes. Statistical models for empirical search-based performance tuning. *Int. J. High Perform. Comput. Appl.*, 18(1):65–94, 2004.

[Vla08] D. G. Vlachos. Temporal coarse-graining of microscopic-lattice kinetic monte carlo simulations via tau-leaping. *Physical Review E*, 78(4):046713+, 2008. doi:10.1103/PhysRevE.78.046713.

[VS17] M. Von Smoluchowski. Versuch einer mathematischen Theorie der Koagulationskinetik kolloidaler Lösungen. *Zeitschrift für physikalische Chemie*, 92(2):129–168, 1917.

[WF99] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[WGMH10] V. Wolf, R. Goel, M. Mateescu, and T. Henzinger. Solving the chemical master equation using sliding windows. *BMC Systems Biology*, 4(1):42+, 2010. ISSN 1752-0509. doi:10.1186/1752-0509-4-42.

[WLV07] Weinan, Liu, D., and Vanden-Eijnden, E. Nested stochastic simulation algorithms for chemical kinetic systems with multiple time scales. *Journal of Computational Physics*, 221(1):158–180, 2007. ISSN 00219991. doi:10.1016/j.jcp.2006.06.019.

[WP03] M. J. P. Wolf and B. Perron. *The Video Game Theory Reader*. Routledge, 1 edition, 2003. ISBN 0415965799.

[WSV+02] M. van de Wetering, E. Sancho, C. Verweij, W. de Lau, I. Oving, A. Hurlstone, K. van der Horn, E. Batlle, D. Coudreuse, A.-P. Haramis, M. Tjon-Pon-Fong, P. Moerer, M. van den Born, G. Soete, S. Pals, M. Eilers, R. Medema, and H. Clevers. The β-catenin/tcf-4 complex imposes a crypt progenitor phenotype on colorectal cancer cells. *Cell*, 111(2):241–250, 2002.

[Zei84] B. P. Zeigler. *Multifacetted modelling and discrete event simulation*. Academic Press Professional, Inc., San Diego, CA, USA, 1984. ISBN 0-12-778450-0.

[ZPK00] B. P. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation, Second Edition*. Academic Press, 2 edition, 2000. ISBN 0127784551.

[ZTW05] J. S. van Zon and P. R. Ten Wolde. Green's-function reaction dynamics: A particle-based approach for simulating biochemical networks in time and space. *J Chem Phys*, 123(23):234910, 2005. ISSN 0021-9606. doi:10.1063/1.2137716.

Thesis Statements

Title:	Efficient Non-Spatial and Spatial Simulation of Biochemical Reaction Networks
Name:	Matthias Jeschke

1. An ever increasing number of algorithm variants for the stochastic simulation of reaction networks makes it important to establish concepts for a thorough evaluation of these methods.

1.1 An algorithm evaluation is aggravated by various factors, e.g., the dependence of methods to algorithm parameters or sub-algorithms, differences in the hardware, or implementation issues. The influence of these factors can be alleviated by performing simulations within an environment that ensures the same conditions for each algorithm under scrutiny.

1.3 Using benchmark models instead of real-world problems for evaluation studies offers several advantages. They are parameterizable and scalable and thus capable of covering a large area of the model space; furthermore, they allow to represent both isolated model characteristics (e.g., how the particles are distributed in space) and a combination thereof.

1.4 Performance results underline the statement that there is no "silver bullet", i.e., an algorithm that dominates all other variants for arbitrary model instances. They also show that statements regarding the performance can only be made in a rather narrow scope.

2. The basic idea of leap methods can also be applied to spatially inhomogeneous systems. Doing so requires the inclusion of incoming diffusion events into the leap calculation and considering diffusions and reactions as dependent events.

2.1 Estimating or assessing the performance of spatial τ-leaping (Sτ) is much more difficult than for the non-spatial variant because the (time-varying) particle distribution in space has a considerable influence that allows only small steps in case of large inhomogeneities.

2.2 A "parallelization inside a simulation run" for Sτ should only be used for the analysis of single runs; if many trajectories are needed, a "parallelization across a simulation" technique is likely faster.

3. In a multi-algorithm DES (maDES), intra-rules are still handled inside the individual simulations, but the addition of inter-rules requires a coordinator component which has to decide when those should be applied. It also takes care of the synchronization between the subsidiary simulations.

3.1 A maDES can combine population-based and individual-based simulations, which makes it possible to create more complex models for studying, e.g., the effect of crowding or active transport processes.

3.2 The physical presence of individual entities in a maDES can influence the dynamics of diffusion dependent reactions at the population level.