UNIVERSITÀ DEGLI STUDI DI TRENTO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Specialistica in Informatica
within European Master in Informatics

Final Thesis

Performance analysis of stochastic biochemical systems simulations

Relatore/1st Reader:

Stephen Gilmore
University of Edinburgh

Corelatore/ 2nd Reader:

Prof. Paola Quaglia
Università degli Studi di Trento

Laureando/Student:

Luciano Marcon

Anno Accademico 2006 - 2007

Abstract

This dissertation presents a survey of the simulation techniques used to study the dynamics in time of biological systems. The attention in this work is focused on the simulation performance of the algorithms and on the optimizations that have been adopted to overcome the high computational load of exact stochastic simulations. Since even for small biological systems the number of reactions that are generated can be in the millions, an explicit modeling of each reaction as in the Gillespie Stochastic Simulation Algorithm (SSA) leads to a high computational load. For this reason it is important to have an efficient realization of the algorithm. The first part of this project consisted of the extension of Dizzy, an existing chemical simulator, with new formulations of the SSA called the Logarithmic Direct Method (LDM), the Optimized Direct Method (ODM) and the Sorting Direct Method (SDM). The existing software was also extended with new features for performance analysis and visualization of the various algorithms over different biological systems. Three biological examples were selected and a comparison between the new algorithms and the existing formulations of the SSA was made investigating scalability with respect to the number of reactions.

Starting from the assumption that for many practical applications even optimisations of the SSA are still simply too slow to run without the use of parallel computing, the second part of the project focused on approximate methods. These sacrifice the exactness of the SSA in favor of simulation performance. Following the good performance results obtained by other two simulators, a hybrid deterministic and stochastic simulator was implemented in Dizzy. Hybrid methods are based on the assumption that the stochasticity of fast reactions involving species with a large population becomes negligible with respect to the dynamics of the system allowing us to approximate the discrete event simulation of the SSA with a more efficient deterministic simulation. Newly developed features of Dizzy were extended and adapted in order to measure, by using an histogram distance, the accuracy of the newly developed hybrid method. The final goal of the project was to compare the performance of hibrid simulators with the performance of efficient formulations of the Gillespie stochastic simulation algorithm. A series of models was used for this analysis and results were evaluated and discussed.

Acknowledgements

Many thanks to my supervisor, Stephen Gilmore, for the helpful suggestions on this project and for reviewing this dissertation. I would like also to thank Andrea Degasperi and Luca Cacchiani, members of the research group that is working on Dizzy, for their valuable contribution on the practical aspects of the implementation.

Thanks go also to the staff of the European Master in Informatics of the University of Trento and of the University of Edinburgh for their help during my MSc and and for having given me the possibility to study in Edinburgh. Finally I would like to thank my girlfriend for her support.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Luciano Marcon*)

≪ Reasonable people adapt themselves to the world.
Unreasonable people attempt to adapt the world to themselves.
All progress, therefore, depends on unreasonable people ≫

George Bernard Shaw

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last decade, developments in high-throughput experimental techniques and the increase of genomic data have supported the system-based approach studying biological systems. Understanding how biological systems work is a valuable resource for better comprehension of the biology of disease and for model-driven drug discovery.

It is widely accepted that biological systems can be considered as complex systems where the interaction of a number of simple but specific functional elements lead to emergent properties and behaviours. Different approaches have been proposed for modelling and simulation of such complex systems. Starting from the position that, even if complex, biological systems are deterministic, mathematical models involving Ordinary Differential Equations (ODE) [18, 14] have been historically proposed for the simulation of chemical reaction systems. This analysis is applicable only for chemical systems where the population of reactants is large and where dynamics can be approximated as a continuous system. Indeed, on a small-scale population system these models fail to capture the randomness of the collisions between molecules that are the basis of all chemical reactions.

A better abstraction that takes into account the stochasticity and the discreteness of the system is given by the stochastic simulation algorithm (SSA) introduced by Gillespie in [9, 10]. In contrast with the ODE based approach that models the evolution in time of the concentrations as differentiable functions, the SSA takes into account the number of molecules of each species and using a discrete-event simulation, based on the theory of Markov processes, estimates the evolution in time of the system. More precisely, using Monte Carlo techniques the time of the next reaction event is estimated and, considering the likelihood of occurrence of each reaction, the next reaction to occur is selected. A simple algorithm is then used to update the state of the system according to the state-change vector of the reaction. This approach is more precise than ODE-based models in simulating the evolution of a biological system, because it is rigorously based on the Chemical Master Equation (CME)[11, 7]. However, a significant disadvantage is the computational cost of the procedures involved. Since even for small biological systems the number of reaction events that are generated can be in the millions, explicit modeling of each reaction as in SSA leads to a high computational load. For this reason it is important to have an efficient realization of the algorithm. A number of mathematically equivalent formulations of SSA have been proposed in order to reduce the computational cost of simulations. All these formulations focus on reducing the time needed to locate the next reaction to fire. This operation is in fact the most expensive step of SSA and to reduce its computational cost means that we would considerably increase the number of reactions that are simulated per time unit.

However, for many practical applications even optimisations of the SSA are still simply too slow to be run without the use of parallel computing. For this reason some approximate methods, which sacrifice the exactness of the SSA in favour of simulation performance, were introduced. Gillespie in [12] proposed the tau-leaping method which, by using a Poisson approximation, takes time steps larger than the firing time of a single event and can "leap" over many fast reactions and approximate the stochastic behaviour of the biological system. Subsequently this method has been improved in [24] in order to overcome the stability problem of the tau-leaping when simulating stochastic systems with vastly different timescale reactions. The multiscale time nature of many biological systems, known in deterministic simulations as *stiffness* [21], has also motivated the formulation of hybrid methods that combine the traditional deterministic approach with the SSA. In conclusion the MSSA [37], which relies on the use of the stochastic partial equilibrium assumption, has been recently formulated and proposed as an improvement of the hybrid methods.

1.1 Motivations

Computational performance is a significant limiting factor when models with a high degree of complexity are simulated. For example, an efficient whole cell simulation is still a difficult task to address in a reasonable time with the use of a typical desktop computer. The exact stochastic simulation of a cell cycle of an E.Coli, which has been estimated in [6] to involve between 10^{14} and 10^{16} events, could require even years in order to be executed on a single processor computer. For this reason studying optimisations of the stochastic simulation algorithms with the aim of reducing the computational load of exact stochastic simulation, is a very active field.

However, a theoretical performance analysis of the different formulations of SSA is a difficult task to address. The number of operations required by each formulation of SSA to find the next firing reaction is in fact strictly related to the biological system that is being simulated. Thus a general analysis of the algorithm should take into account a test-set of different representative biological systems. Moreover, even if the average number of operations required to locate the next reaction is minimized, particular attention must be paid to the computational cost required to maintain the possibly optimized data structures used to store reactions. Recent results [34] highlight that the optimised direct method (ODM) is, in most cases, the best formulation of SSA. This in contrast with the common belief that the Next Reaction Method (NRM) of Gibson and Bruck[25] is the fastest implementation of SSA because it comes with the best data structures to keep track of the reactions. This analysis becomes even more difficult when approximate methods like Hybrid Methods [1, 17] or tau-leaping[12] procedures are considered. Indeed, in this case the trade-off between the approximation introduced and the gain in performance must be taken into account. In the light of these results the development of a software with optimized simulation algorithms and new tools for the performance analysis of the simulations can be of immense help as an aid to those attempting to understand stochastic simulation algorithms or invent better ones.

1.2 Objectives

The practical aim of this work was to extend Dizzy, an existing chemical simulator, with novel formulations of the SSA called the Logarithmic Direct Method (LDM)[16], the Optimized Direct Method (ODM)[34] and the Sorting Direct Method (SDM)[19]. In order to evaluate the efficiency of these algorithms Dizzy was extended to track the performance of the algorithms during the simulation. This part of the project resulted in the implementation in Dizzy of new features for the visualization of candle stick graphs expressing the reactions per second of the different formulations of the SSA over various biological systems. Our objective was to run these three different SSA formulations on a set of biological models with different reaction number magnitudes in order to test the scalability of their performance with respect to the number of reactions.

Successively, following the good performance results obtained by other two simulation software, BioNets [5] and COPASI [29], we decided to implement in Dizzy a hybrid deterministic and stochastic simulator. Our final goal was to have a simulation framework with a wide range of simulators in order to begin an initial performance analysis. To compare the performance of the exact and the approximate methods of Dizzy newly developed features of the software were extended and adapted in order to measure, by using an histogram distance [36], the accuracy of the approximated methods.

We believe that this extension can contribute to further studies on the ongoing analysis of stochastic simulations of biological systems. This project attempts to create a software framework for performance analysis of different simulation algorithms that will help in identifying the most appropriate simulation algorithm for different biological scenarios.

1.3 Related work

Several software tools are currently available to simulate the dynamic behaviour of chemical reaction systems. However, to the best of our knowedge none of them offer a framework for a quantitative analysis of the performance of the simulation algorithm. As mentioned in the previous chapter, due to the high computational load of SSA, performance plays an important role when practical examples are simulated. Various performance comparisons between formulations of SSA can be found in the literature. For example, the performance of the Logarithmic Direct Method is compared in [16] in terms of reactions per second with the performance of the standard direct method and of the optimized direct method. However, these analyses are usually carried out by implementing ad-hoc code and are evaluated over a small test-set of biological examples. Since performance is strongly dependent on the nature of the biological system and on the realization of the SSA, the implementation of a tool capable of comparing the performance of different SSA formulations for a number of different biological systems could be very useful for further performance analysis.

In this research the extension of various different simulation tools like Stochkit [35], E-Cell [15], BioNets [5], COPASI [29] and Dizzy[31, 30] was considered for this purpose. Overall Dizzy was chosen due to some of its characteristics as mentioned below.

First of all Dizzy comes with a wide range of different simulation algorithms sharing a common model definition which does not require further modification in order to be run over the different algorithms. Secondly the modular design of the tool in which each simulator is an independent part of software that shares with the others a well-defined interface, facilitates the implementation of both the new simulation algorithms and the performance analyser. In conclusion, the possibility offered by Dizzy of creating reusable templates enables us to test the performance of the different SSAs on many biological systems.

1.4 Structure of the dissertation

Including this first introductory chapter which also surveys related works, the thesis consists of six chapters. The next chapter proceeds to review the background material needed to understand this work. Chapter 3 then presents Dizzy, the software which was extended, and outlines the main step of the implementation. Chapter 4 introduces the biological models that have been used to test and evaluate the software. Chapter 5 includes a performance analysis of the simulation algorithms over the biochemical model presented in the previous chapter and discusses the results obtained. Finally in Chapter 6 conclusions are drawn and future directions of the work are discussed.

Chapter 2

Background

This chapter reviews the theory underlying the three different approaches, deterministic, stochastic and hybrid, that have been used to simulate the dynamics of biochemical reacting systems. Finally a technique that can be used to measure differences between the results of distinct simulation algorithms results is presented.

2.1 Deterministic approach

Taking the assumption that a chemically reacting system is deterministic, such as that given an initial state always leads to same dynamics, the evolution in time of a well-mixed biological system has been traditionally studied using a mathematical formalism in which continuous variables evolve deterministically. Those mathematical models usually involve a set of coupled ordinary differential equations (ODEs) that drive the evolution in time of X_i continuous variables whose values correspond to the population of $S_i(i = 1,...,N)$ chemical species. The set of coupled ordinary differential equation is known as the reaction rate equations (RRE) and is given by the formula:

$$\frac{dX_i}{dt} = f_i(X_1,...,X_N) \qquad (i = 1,...,N) \tag{2.1}$$

where the f_i terms are functions derived from the reactions that depend on the change of concentration over time of the reactants. This model has been shown to work quite well when the population of the various species are quite large and for this reason is usually expressed with Z_i concentration variables that represent the X_i population in terms of moles per volume. Note that when the population of the variables is given by the X_i variables expressed in number of molecules, the real values that they assume are considered to be an acceptable approximation as long as the number of molecules is large. In this way the relative error can be neglected. When smaller systems with concentration larger relative to one are considered the stochastic nature of chemical reactions cannot be ignored and the deterministic approach must be substituted with other methods able to capture stochastic fluctuations.

2.2 Exact stochastic simulation methods

This section briefly reviews the Stochastic Simulation Algorithm, formally presented as the Direct Method of Gillespie [9, 10], and its main differences with other SSA formulations such as the Next Reaction Method (NRM) of Gibson and Bruck[25]. The Stochastic Simulation Algorithm is an exact procedure for generating a realization of the Chemical Master Equation (CME) [9] and then for computing the evolution in time of a “well-stirred” chemical reacting system. In general a chemical reacting system is defined by:

- a number of molecules of N different chemical species $\{S_1, ..., S_N\}$
- a set of M chemical reactions $\{R_1, ..., R_M\}$ through which the molecules interact
- The state of the system at a time t which is expressed by a vector $\mathbf{X}(t) \equiv (X_1(t), ..., X_N(t))$ where $X_i(t)$ denotes the number of molecules of the species S_i at time t.
- Each reaction R_j is defined by two quantities:
  - A *state-change* vector $\mathbf{v}_j \equiv (v_{1j}, ..., v_{Nj})$ where v_{ij} denotes the change in the S_i molecular population caused by one R_j reaction.
    In other words when reaction R_j occurs the system moves from the state vector $\mathbf{x}$ to state vector $\mathbf{x} + \mathbf{v}_j$.
  - A propensity function $a_j(\mathbf{x})$ that, given $\mathbf{X}(t) = \mathbf{x}$, is defined as the probability that one reaction R_j will occur in the system in the next infinitesimal time interval $[t, t + dt]$.

In the SSA the system is simplified and is assumed to have a constant volume Ω and a constant temperature.

2.2.1 Chemical Master Equation - CME

The mathematical basis for the CME is the probability $P(\mathbf{x},t|\mathbf{x}_0,t_0)$.

- $P(\mathbf{x},t|\mathbf{x}_0,t_0)$ is defined as the probability that, given $\mathbf{X}(t_0) = \mathbf{x}_0$, the system will be in the state $\mathbf{x}$ at time t or equivalently $\mathbf{X}(t) = \mathbf{x}$.

The CME gives a recursive definition of the time evolution of $P(\mathbf{x},t|\mathbf{x}_0,t_0)$ as follows:

$$\frac{\delta P(\mathbf{x},t|\mathbf{x}_0,t_0)}{\delta t} = \sum_{j=1}^{M} [a_j(\mathbf{x}-\mathbf{v}_j)P(\mathbf{x}-\mathbf{v}_j,t \mid \mathbf{x}_0,t_0) - a_j(\mathbf{x})P(\mathbf{x},t \mid \mathbf{x}_0,t_0)] \tag{2.2}$$

The average behaviour of $X(t)$ can be derived from the CME by multiplying by all the $\mathbf{x}$ and then summing over all the $\mathbf{x}$ obtaining:

$$\frac{d\langle\mathbf{X}(t)\rangle}{dt} = \sum_{j=1}^{M} \langle\mathbf{v}_j(\mathbf{X}(t))\rangle \tag{2.3}$$

If no fluctuation are assumed then $\langle\mathbf{v}_j(\mathbf{X}(t))\rangle = \mathbf{v}_j(\mathbf{X}(t))$ and it is possible to rewrite (2.3) as:

$$\frac{d\mathbf{X}(t)}{dt} = \sum_{j=1}^{M} \mathbf{v}_j(\mathbf{X}(t)) \tag{2.4}$$

It can be seen that the equation (2.4) corresponds to the RRE given in equation (2.1) with the terms f_i corresponding to the functions $\sum_j v_{ij}(\mathbf{X}(t))$. Thus, under the condition of no fluctuation a way to derive the deterministic approach model starting from (2.2) has been shown.

2.2.2 Stochastic Simulation Algorithm - SSA

Since the equation (2.2) corresponds to a set of coupled ODEs it can be solved analytically for a few simple examples. Moreover the CME computes the variation in time of the molecular species without taking into account the fluctuations of the chemical reacting system that is natively stochastic. Indeed the analytical solution of the equation (2.2), in the hypothetical case in which the system has no fluctuations, gives us a continuous description of the system.

In contrast the SSA implements a way to compute numerical realizations of (2.2) which can be used to simulate trajectories of $\mathbf{X}(t)$. This simulation procedure is said to be *exact* because it is based on a discrete stochastic simulation of every reaction event that occurs in the biological system, thus producing results that do not introduce approximations to the CME. This leads to a good degree of accuracy but it is extremely expensive in terms of computation because the operations that are computed are proportional to the number of reactions observed.

The mathematical basis used to simulate the trajectories is a new probability function $P(\tau, j \mid \mathbf{x}, t)$ which has the following definition:

- $P(\tau, j \mid \mathbf{x}, t)$= the probability that, given $\mathbf{X}(t) = \mathbf{x}$, the next reaction in the system will be R_j and will occur in the time interval $[t+\tau, t+\tau+dt)$.

The formal definition of this probability is obtained by joining the probability density function of an exponential random variable that gives the time to the next reaction (τ) and a random variable that gives the index of the next reaction. The former has mean $1/a_0(\mathbf{x})$ and the latter is a statistically independent integer random variable with probabilities $a_j(\mathbf{x})/a_0(\mathbf{x})$, where $a_0(\mathbf{x})$ is defined as $\sum_{k=1}^{M} a_k(\mathbf{x})$. The joint probability density function known as the Next Reaction Density Function [13] is:

$$P(\tau, j \mid \mathbf{x}, t) = a_0(\mathbf{x})e^{-a_0(\mathbf{x})\tau} \times \frac{a_j(\mathbf{x})}{a_0(\mathbf{x})} = a_j(\mathbf{x})e^{-a_0(\mathbf{x})\tau} \tag{2.5}$$

Starting from (2.5) several formulations of the SSA have been developed for computing samples of τ and the reaction index j through a Monte Carlo procedure. One of the first formulations is the *Direct Method* [9, 10] in which τ and j are selected using two random numbers r_1, r_2 which are generated from the uniform distribution in the unit time interval $[0, 1]$. The formulas for the generation of τ and j are the following:

$$\tau = \frac{1}{a_0(\mathbf{x})} ln(\frac{1}{r_1}) \tag{2.6}$$

$$j = the\, smallest\, number\, such\, that \sum_{j'=1}^{j} a_{j'}(\mathbf{x}) > r_2 a_0(\mathbf{x}) \tag{2.7}$$

These formulas can be used to iteratively select j and τ and make the system advance to a next state in accordance with the $\mathbf{v}_j$ state-change vector. In this way numerical realizations of $\mathbf{X}(t)$ can be generated. The pseudocode of the whole algorithm is given in Figure 2.1.

The high computational load of such algorithms can be traced to the operations of the equations (2.6) and (2.7) and to `Step 2` of the algorithm. The efficiency of

```
1. Initialize the time $t = t_0$ and the the system's state $\mathbf{x} = \mathbf{x}_0$
2. With the system in a state $\mathbf{x}$ at time $t$, evaluate all the $a_j(\mathbf{x})$
   and their sum $a_0(\mathbf{x})$
3. Generate values for $\tau$ and $j$ using (2.6) and (2.7)
4. Fire the next reaction by replacing $t \leftarrow t + \tau$ and $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{v}_j$
5. Record $(\mathbf{x}, \tau)$ and go to Step 2 or end the simulation
```

Figure 2.1: Pseudocode of the Direct Method

these operations is in fact highly dependent on the number of reactions M. Different formulation of the SSA rely on optimisations of (2.7) and of `Step 2` reducing the time needed to locate the next reaction to fire and the time needed to update the probability of each reaction. The next sections give a survey of the most significant formulations of the SSA.

2.2.3 First Reaction Method - FRM

One of the earliest elaborations of the SSA is the *First Reaction Method* (FRM) presented as an alternative to the Direct Method in [9, 10]. The FRM selects the next reaction to fire by generating M random numbers $r_1, ..., r_M$ from the uniform distribution $[0, 1]$ and computing:

$$\tau_{j'} = \frac{1}{a_{j'}(\mathbf{x})} ln \frac{1}{r_{j'}} \quad (j' = 1, ..., M)$$

τ and j are selected as follows:

$$\begin{aligned} \tau &= \text{the smallest of the } \tau_{j'} \\ j &= \text{the index of the smallest } \tau_{j'} \end{aligned}$$

This way of generating τ and j is in accord with the probability density function (2.5) however for chemical reacting systems with many reaction channels this system is less efficient than the *Direct Method*. Indeed, generating M uniform random numbers at every step of the simulation is very time consuming. This algorithm exhibits a time complexity $O(M)$ for the operations required to find the index j of the next reaction to fire and does not optimize the update process propensities after each reaction.

2.2.4 Next Reaction Method - NRM

An improvement in performance has been obtained by the *Next Reaction Method* (NRM) presented in [25] by Gibson Bruck. This algorithm is based on the idea that in general the execution of a reaction affects the propensities of only a small number of reactions. Starting from this assumption the NRM attempts to reduce the number of operations needed to perform the `Step 2` of the SSA by pre-computing, before the simulation, a reaction dependency graph. This graph can be used to update only the propensities of reactions that have been modified by the last fired reaction. This optimisation substantially reduces the number of calculations of `Step 2` in most models. In that regards the selection of τ and j in this algorithm is essentially a modification of the First Reaction Method. As in the FRM the M putative next firing times are computed at each iteration. In addition the NRM reduces the time needed to locate the minimal $\tau_{j'}$ by keeping the computed $\tau_{j'}$ in a binary indexed priority queue. This heap structure is constructed and maintained so that each parent node is always earlier than its daughter nodes. This method is faster than the DM when the number of species N and the number of reactions M are large. However, one disadvantage of the NRM is that during execution most of the computational time is used to maintain the heap structure and this becomes a limiting factor with very large and very coupled systems. The construction and the meaning of the dependency graph is worth explaining with the example in Table 2.1 as reported in [19].

Name	Reaction	Depend on	Affects	Update
R1	$A \rightarrow B$	A	A,B	R1,R2
R2	$B \rightarrow C$	B	B,C	R2,R3
R3	$C+D \rightarrow E$	C,D	C,D,E	R3,R4,R6
R4	$E \rightarrow E+F$	E	F	R5
R5	$F \rightarrow A$	F	A,F	R1,R5
R6	$E \rightarrow B$	E	B,E	R2,R4,R6

Table 2.1: Example of dependency graph

The main steps of the Next Reaction Method are summarized in Figure 2.2.

```
1. Initialization
    • Generation of the dependency graph
    • Calculation of propensities and generation of $\tau_{j'}$
      using $M$ random number.
    • Each $\tau_{j'}$ is inserted in the heap structure.
    • Get from the heap the smallest $\tau_{j'}$ and fire the related reaction.
2. Iterate
    • Using the dependency graph update the $a'_j(\mathbf{x})$ propensities
      that have been modified after the last reaction.
    • Regenerate related $\tau_{j'}$ and insert them in the heap.
    • Get from the heap the smallest $\tau_{j'}$ and fire related reaction
```

Figure 2.2: Pseudocode of the Next Reaction Method

2.2.5 Optimized Direct Method - ODM

The Optimized Direct Method (ODM) enhances the performance of the DM reducing the number of operations that are performed to find the reaction that satisfies (2.7). The ODM achieves its efficiency by pre-ordering the reactions so that the reactions that are executed more frequently, those with larger propensity functions, are moved into the first positions in the search order. The assumption underlying the pre-ordering is that most biological systems are made by a small number of frequently occuring reactions and a majority of reactions which are fired infrequently. The pre-ordering of the reactions is determined by executing a pre-simulation of an initial time less than the whole simulation time. In this way the ODM eliminates the overhead of maintaining a heap data structure while improving the speed of finding the next reaction to fire. In practical applications several pre-simulations are needed to determine an optimal order of the reactions, furthermore this approach relies on the strict assumption that the biological system that is analyzed has an initial reaction execution behaviour that is representative of the whole simulation.

2.2.6 Sorting Direct Method - SDM

The Sorting Direct method (SDM) [19] is another optimisation of the DM that has been developed in order to address the problem of the costly pre-simulation of the ODM. The main idea of this algorithm is to reduce the cost of `Step 2` by moving at each iteration the reaction that is fired towards the top of the reaction list. In this way the next time that this reaction will be fired its average search depth will decrease and the more probable is the reaction the smaller will be its average search depth. According to results presented in [19] in most models this method exhibits, in a long running simulation, performance that is substantially better than that of the ODM. This is due to the fact that as the run continues the ODM could increase the average search depth because the pre-simulation has not effectively predicted the reaction behavior. The pseudocode of the algorithm is given in Figure 2.3.

```
1. Generation of the dependency graph
2. Initialise the Search order as the given order of the
   reactions
3. Initialise the time $t = t_0$ and the system state $\mathbf{x} = \mathbf{x}_0$
4. With the system in a state $\mathbf{x}$ at time $t$, evaluate the
   $a_j(\mathbf{x})$ modified by the last fired reaction and the sum $a_0(\mathbf{x})$
5. Generate values for $\tau$ and $j$ using (2.6) and (2.7)
6. Fire the next reaction by replacing $t \leftarrow t + \tau$ and $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{v}_j$
7. Swap in the Search Order the selected reaction
   $j'$ with the reaction $j' - 1$.
8. Record $(\mathbf{x}, \tau)$ and go to Step 4 or end the simulation.
```

Figure 2.3: Pseudocode of the Sorting Direct Method

2.2.7 Logarithmic Direct Method - LDM

The ODM and SDM are based on the reduction of the average depth search of the next reaction to fire. Those algorithms are strictly based on the original description of the Direct Method [9, 10], thus they recalculate each time the summation $a_0(\mathbf{x})$ used in the equations (2.6) (2.7). Since $a_0(\mathbf{x})$ is first calculated in (2.6) and then some of the propensities $a_i(\mathbf{x})$ are successively accumulated in the summation in (2.7), some propensities are summed almost twice. The assumption behind the Logarithmic Direct Method [16] is that the whole recalculation of the term $a_0(\mathbf{x})$ can be avoided by keeping a vector with the partial summation of the terms $a_j(\mathbf{x})$. This vector can be successively updated, at each step of iteration, only from the point where the first partial summation has changed. The next reaction to fire can be efficiently located by performing a binary search, with search key equal to $r_2 a_0(\mathbf{x})$, over the vector containing the partial summation. In this way the average search depth can be reduced to $O(logM)$. The pseudocode of this method is given in Figure 2.4.

```
1. Generation of the dependency graph
2. Initialise the vector $subtotal$ with the
   partial summations of the propensities $a_j(\mathbf{x})$
3. Initialise the time $t = t_0$ and the system's state $\mathbf{x} = \mathbf{x}_0$
4. With the system in a state $\mathbf{x}$ at time $t$
   evaluate the $a_j(\mathbf{x})$ according to the dependency graph.
5. Recalculate the vector $subtotal$ from first point
   which has changed according to the dependency graph
6. Generate values for $\tau$ according to (2.6)
   using $subtotal[M]$ instead of $a_0(\mathbf{x})$.
7. Select the next reaction $j$ such that
   $subtotal[j-1] \leq r_2 subtotal[M] < subtotal[j]$
8. Fire the next reaction by replacing $t \leftarrow t + \tau$ and $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{v}_j$
9. Record $(\mathbf{x}, \tau)$ and go to Step 4 or end the simulation.
```

Figure 2.4: Pseudocode of the Logarithmic Direct Method

2.2.8 Accuracy of SSA formulations

Different formulations of the SSA exhibit different performance as well as a different degree of accuracy. Indeed even if the ODM, the SDM and the LDM are all derived from the Direct Method of Gillespie, which is an exact stochastic simulation method, the way in which they order the list of reaction propensities can affect the accuracy of equation (2.7) in a different way. As discussed in more detail in [13], the ODM and the SDM arrange the reactions in order of decreasing size of propensities and this can affect the accuracy of the accumulator used in the search in (2.7). Gillespie in [13] proposed the following example to explain this phenomena: consider that we were carrying k decimals in the accumulator in left term of (2.7) and suppose that the propensity function with the highest search order index is k orders of magnitude smaller than a_M, in this case the reaction R_M never fires at all because of the numerical truncation on the accumulator. In contrast, the LDM is not affected by this accuracy problem because it allows an arbitrary order of the propensities before the computation of the vector with the partial summation and successively during the binary search does not compute any sum.

2.3 Approximated stochastic methods

Approximations of the SSA have been developed in order to overcome the high computational load of an exact discrete event stochastic simulation. This section presents three promising techniques, the tau leaping, the Hybrid Method and the MSSA that have shown to considerably increase the performance of the simulation, yet giving satisfactory results. To conclude, a simple technique that can be used to measure the approximation that is introduced by these methods is reviewed.

2.3.1 Tau-leaping method

This method is based on a Poisson approximation that can be made on the occurrence probability of a reaction j when, given a system in a state $\mathbf{x}$ at a time t, there exists a time $\tau > 0$ such that $a_j(\mathbf{x})$ does not change its value by a significant amount in the time interval $[t, t+\tau)$. In this case, considering that $a_j(\mathbf{x})$ remains approximately constant over the time $[t, t+\tau)$, the numbers of firings of the reaction R_j during the time interval $[t, t+\tau)$ may be expressed by a Poisson random variable with mean and variance $a_j(\mathbf{x})\tau$. When this condition is satisfied by all the reactions it is possible to write $\mathbf{X}(t+\tau)$ in the form of the basic tau-leaping formula:

$$\mathbf{X}(t+\tau) = \mathbf{x} + \sum_{j=1}^{M} P_j(a_j(\mathbf{x})\tau)\mathbf{v}_j \tag{2.8}$$

where $\mathbf{x} = \mathbf{X}(t)$ and the term $P_j(a(\mathbf{x})\tau)$ is a statistically independent Poisson random variable with mean and variance $a_j(\mathbf{x})\tau$. As presented in [13], starting from (2.8) the strategy in Figure 2.5 can be used to generate a stochastic simulation. `Step 1` is the most difficult part of this process. Estimating an appropriate largest leap time τ, which does not produce a change in the propensity function greater than user-specified parameter ε, is in fact a difficult task to address. To discuss the details of an optimal estimation of τ is beyond the scope of this thesis, for more information refer to [38]. The tau leaping technique gives very good results when all the reactions have similar time scales. Therefore, in this case the time τ that satisfies the leap condition can

```
1. With the system in a state $\mathbf{x}$ choose a value $\tau$
   that satisfies the leap conditions.
2. for each $j$+ a sample $k_j$+ of the Poisson random variable $a(x)\tau$.
3. update the state $\mathbf{x}$ with $\mathbf{x}+\sum_j k_j \mathbf{v}_j$
```

Figure 2.5: Tau-leaping algorithm

be large enough to jump over many fast reactions. However, with many biological systems which are characterized by having multiple scales in time behaviour of the reactions, the original or "explicit" tau-leaping method becomes very slow. Indeed in this case, the time τ must be restricted to the smallest timescale in order to satisfy the leap condition. The "implicit" tau-leaping method relieves this problem by replacing the equation (2.8) with the equation:

$$\mathbf{X}(t+\tau) = \mathbf{x} + \sum_{j=1}^{M} [P_j(a_j(\mathbf{x})\tau) - a_j(\mathbf{x})\tau + a_j(\mathbf{X}(t+\tau))\tau]\mathbf{v}_j \tag{2.9}$$

The advantage of this formula is that it can be numerically solved implicitly for the state $\mathbf{x}$ at time $t+\tau$ by using Newton's method. This enhances the stability of the tau-leaping method allowing the use of larger values of τ. The equation (2.9), however, overdamps the fluctuations of the population of the fast species, thus those populations must be readjusted by using a technique known as "down shifting". Moreover both the tau-leaping methods canot guarantee that all reactant values remain positive. Indeed computing more than one reaction at each step of iteration can lead to negative values of molecular numbers when one or more reaction consume the same reactant. Negative numbers of molecules do not have any biological meaning and in order to address this problem it is possible to mark some reactions as "critical" and forbid them from firing more than once per time. This clearly results in a loss of performance.

2.3.2 Hybrid deterministic and stochastic models

Because the computational load of the Stochastic Simulation Algorithm is proportional to the number of reactions performed, when many molecules and fast reactions are involved they become quite inefficient. Indeed, even if some high frequency events do not make a considerable contribution to the dynamics of the system, the SSA simulates every reaction event spending most of the time in simulating fast reactions.

Several biological systems show a multi-scale behaviour in which fast reactions quickly reach a stable state and slow reactions drive the dynamics of the system. This becomes a limiting factor for many practical applications. Moreover, when reactions involve species with a large population, the stochasticity of those reactions becomes negligible with respect to the dynamics of the system and the discrete event simulation of the SSA can be replaced by a more efficient deterministic model simulation. The idea behind Hybrid Methods is to partition the reactions of the system in two groups, one corresponding to the reactions that are more efficiently simulated with deterministic continuous models and another one corresponding to reactions that must be simulated with a discrete event driven stochastic simulation. Finally the simulation of the whole system can be obtained by combining the result coming from the two approaches.

After a brief review of the approaches that have been used to partition the reactions, this section introduces two methods that have been proposed in order to combine deterministic and sochastic simulations: the hybrid method [1, 32] and the MSSA [37].

2.3.2.1 Reactions partitions

Determining which fast reactions should be simulated with a deterministic model is still an open problem. Three different criteria have been proposed in order to address this:

1. Make use of biological insights coming from experiments or experience.
2. Run a full SSA and by analysing the number of times that a reaction is fired choose as fast reactions those with the highest average propensities.
3. Choose during the simulation how to model a reaction based on the number of molecules of the reactants and on the reaction propensity function.

The first criteria can be based on ad-hoc laboratory experiments or driven by simple assumptions like modelling the gene regulation parts stochastically and the metabolic reactions with a continuous deterministic model.

The second criteria is a general purpose approach based on the dynamics of the system during the simulation time of interest. Its main drawback however is the computational load required to run one or a few times a full SSA before the simulation, but this is a tolerable condition if one considers that most of the computational cost in the SSA is determined by the execution of many different runs.

The third criteria consists of adaptively choosing during the simulation how to model the reactions based on the current population and on the current reactions propensities. Since the way in which reactions must be simulated could change in time, a dynamic partition is in theory more flexible and more accurate than a static partition. However, a formal and standard threshold for such partitioning has not yet been defined. Two possible measures, one based on the reactants population, another on the reactions rate, are given in [26]. The main problem is that a reliable measure that expresses how the stochastic fluctuations of the reaction affect the dynamics of the system is not yet available. Nonetheless this partitioning technique is widely used in many hybrid simulators, like Bionets [5] or COPASI [29], and has shown to give good results with many biological models as long as the user-defined parameters that define the threshold are correctly tuned. Recently in [33], some evidence that can be used to set these parameters has been identified in the divergence factor (computed as the sum of the Laypunov exponents) of the system that is considered. A possible drawback of the dynamic partition method is the computational overhead introduced by the partitioning process that can possibly result in simulations that are slower than those that can be obtained with pure stochastic methods. This partitioning criteria is also not compatible with the MSSA method that requires the definition of the reaction partition *a propri* in order to compute the stochastic partial equilibrium of the fast reactions.

2.3.2.2 Hybrid method

Following [1] the mathematical justification for the hybrid methods can be understood by defining a function $N_j(t)$ that counts the number of times that a reaction R_j occurs in the time interval $[t, t_0)$. This function can be defined using the following time transformation:

$$g_j(s|t) = \int_t^s a_j(\mathbf{X}(\tau))d\tau \tag{2.10}$$

where $a_j(\mathbf{X}(t))dt$ is the probability introduced in Section 2.2 that one R_j reaction will occur in the infinitesimal time interval $[t, t+dt)$ or equivalently, given $T_j(t)$ as the first time in which R_j occurs after t, that $T_j(t) \in [t, t+dt)$. Since the reactions are assumed to be locally independent it is also true that $P(T_j(t), T_k(t) \in [t, t+dt) =$

$a_j(\mathbf{X}(t))a_k(\mathbf{X}(t))(dt)^2$. Using $g_j(s|t)$ and considering the following definitions:

1. ξ_{jk} as a series of exponential random variables with parameter 1
   thus having $P(\xi \in [x, x+dx]) = e^{-x}$ for all $x \geq 0$, with $j = 1, \ldots, M$ and $k \in N$.
2. $S_j(n) = \sum_{k=1}^{n} \xi_{jk}$

the Poisson process $N_j(t)$ can be defined as follows:

$$N_j(t) = \sum_{n=1}^{\infty} 1_{\{S_j(n) \leq g_j(t|t_0)\}} \tag{2.11}$$

It is then easy to show that for all the reactions j the probability that exactly one reaction event R_j occurs in the infinitesimal time interval dt is equal to:

$$P[N_j(t+dt) - N_j(t) = 1|\mathbf{X}(t)] = a_j((X)(t))dt \tag{2.12}$$

Thus $N_j(t)$ and $T_j(t)$ have the same probability law and this implies, according to the definition of $N_j(t)$, that

$$\text{the random variable } T_j(t) \text{ has the same law of } g_j^{-1}(Exp(1)|t) \tag{2.13}$$

Furthermore it also implies that equation (2.4) that models the evolution in time of the number of molecules can be reformulated as:

$$\mathbf{X}(t) = \sum_{j=1}^{M} \mathbf{v}_j dN_j(t) \tag{2.14}$$

The mean of the Poisson process N_j is therefore derived as

$$\mathbf{E}[N_j(t)] = \mathbf{E}[(N_j(t) - g_j(t|t_0))^2] = \int_0^t \mathbf{E}[a_j(X(s))]ds$$

and the relative fluctuation between $N_j(t)$ and $g_j(t|t_0)$ can be calculated as

$$\frac{\sqrt{\mathbf{E}[(N_j(t) - g_j(t|t_0))^2]}}{\mathbf{E}[N_j(t)]} = \frac{1}{\sqrt{\mathbf{E}[N_j(t)]}} \tag{2.15}$$

According to the equation (2.15), when the propensity a_j is large and the population of the reactants involved in the reaction R_j are not too small, it is possible to approximate the dynamics of the stochastic part with the term $g_j(t|t_0)$ and then to model this reaction deterministically. In this way, assuming that the system has been partitioned in reactions that have to be modelled deterministically, with indexes $j \in D$, and in reaction that must be modelled stochastically, with indexes $j \in S$, the time evolution equation (2.4) can be reformulated as:

$$d\mathbf{X}(t) = \sum_{j \in D} \mathbf{v}_j a(\mathbf{X}(t))dt + \sum_{j \in S} \mathbf{v}_j dN_j(t) \tag{2.16}$$

The equation (2.16) is the basis of the hybrid method. A simplified version of the algorithm that can be used to generate numerical realizations of (2.16), by using one deterministic simulator and one stochastic simulator, is given in figure 2.6. This algorithm is a simplification of the underlying theory presented in this section. Indeed according to the mathematical theory of the hybrid method during the course of the deterministic evolution of the hybrid system the value of $g_j(\tau|t)$ of the slow reactions changes according to the differential equation:

$$\frac{d}{d\tau} g_j(\tau|t) = a_j(\mathbf{X}(\tau), \tau) \tag{2.17}$$

The algorithm given in figure 2.6 instead approximates this behaviour by executing at each step of iteration the deterministic simulation for a time taken as the minimum time between the integration time step and the expected time to the next reaction event. The stochastic simulation is performed only when the next reaction event time is smaller than the integration time step.

2.3.2.3 Multiscale Stochastic Simulation Algorithm - MSSA

The Multiscale Stochastic Simulation Algorithm (MSSA) uses a new formulation of the stochastic partial equilibrium assumption (SPEA), in order to implement an SSA in which the propensities of the slow reaction are approximated and the simulation of the fast reaction is avoided. The assumption behind this method is that fast reactions, those that can be simulated deterministically, quickly reach a stochastic partial equilibrium that can be used to generate the approximate propensities of slow reactions and to easily perform a SSA.

2.3.2.4 Stochastic partial equilibrium assumption

According to [37] the stochastic partial equilibrium assumption holds when the fast reactions of a system quickly reach a state in which they are in equilibrium. In other words the distributions of the “fast species”, species whose populations are changed

```
1. Partition the reaction into $D$ deterministic reaction
   and $S$ stochastic reactions

2. Initialise $t = t_0$ and $x_i =$ initial values

3. Compute reaction probabilities at the time $t$

4. Compute the next scheduled time $t_1 = t + \delta t$ of the deterministic
   simulator with $\delta t$ the step of the deterministic simulator

5. From the propensities $x_i$ compute the next candidate event time $dt$
   and set $t_2 = t + dt$

6. if($t_1 < t_2$)
     generate continuous predicted populations $x_{D,i}$ on the time $t_1$
     set $t = t_1$
   else
     generate continuous predicted populations $x_{D,i}$ on the time $t_2$
     fire reaction predicted by the stochastic simulation
     updating $x_{S,i}$ set $t = t_2$

7. Record $(\mathbf{x}, t)$ and go to Step 3 or end the simulation.
```

Figure 2.6: Pseudocode of the Hybrid Method

by the fast reactions, are temporarily steady. Assuming a partition of the $R = (R^s, R^f)$ reactions of the system in $R_f = R_1^f, ..., R_{Mf}^f$ fast reactions and $R_s = R_1^s, ..., R_{Ms}^s$ slow reactions, where the population are changing upon the propensities and state change vectors:

$$a_j^f(x) = a_j^f(x^f, x^s) \quad v_j^f = (v_{1j}^{ff}, ..., v_{N_f j}^{ff}) \quad j = 1, .., M_f \tag{2.18}$$

$$a_j^s(x) = a_j^s(x^f, x^s) \quad v_j^s = (v_{1j}^{fs}, ..., v_{N_f j}^{fs}, v_{1j}^{ss}, ..., v_{N_s j}^{ss}) \quad j = 1, .., M_s \tag{2.19}$$

where the $v_{ij}^{sf} = 0$ and omitted as the slow reactions cannot modify the fast species.

We consider a virtual system V modified only by the fast reactions R^f where populations are expressed in time by a vector $\widehat{X}^f(t)$. This system evolves very quickly and reaches an equilibrium in which the distribution of $\widehat{X}^f(t)$ becomes unchanged by the fast reactions and can vary only by the occurrence of the slow reactions. Indeed, when a slow reaction is executed the state of $\widehat{X}^f(t)$ is disturbed and a new equilibrium state is quickly reached by the virtual system. Moreover under the stochastic partial equilibrium we assumed that the time required to reach an equilibrium state, known as relaxation period τ_{relax}, becomes negligible if compared with the time scale of the slow reactions. Thus the virtual system can be considered to remain always at the same equilibrium, so for all the times t the distribution of $\widehat{X}^f(t)$ is equal to the distribution of $\widehat{X}^f(\infty)$.

2.3.2.5 MSSA algorithm

Using the stochastic partial equilibrium assumption specially modified propensities of the slow reactions $\overline{a}_j^s$ can be calculated according to the value of the fast species $\widehat{X}^f(\infty)$. Successively the same procedure of the SSA can be used to compute trajectories of the original system according to the new propensities. The mathematical foundation of this method is the occurrence probability p' of a slow reaction R_j^s in the time interval $[t+\tau, t+\tau+dt)$:

$$p'(\tau, j|x^f, x^s, t) = E(a_j^s(X^f(\tau), x^s)|x^f, x^s)e^{-\int_t^{t+\tau} E(a_0^s(X^f(\mu), x^s)|x^f, x^s)d\mu} \tag{2.20}$$

that can be rewritten using the stochastic equilibrium assumption on the terms $X^f(t)$ as:

$$p'(\tau, j|x^f, x^s, t) = E(a_j^s(X^f(\infty), x^s)|x^f, x^s)e^{-\tau E(a_0^s(X^f(\infty), x^s)|x^f, x^s)} \tag{2.21}$$

The two difficult steps in this procedure are the resolution of the partial equilibrium $\widehat{X}^f(\infty)$ and the estimation of the modified slow reaction propensity functions $\widehat{X}^f(\infty)$. Both these processes can be simplified if we assume that the mean values of the distribution $\widehat{X}^f(\infty)$ are a good approximation in order to compute the modified propensities $\overline{a}_j^s$, that in this way can be easily calculated according to the five following cases:

1. $\overline{a}_j^s(x) = a_j^s(x) \quad$ if $a_j^s(x)$ is independent of x^f
2. $\overline{a}_j^s(x) = c_j^s\langle\widehat{X}_i^f\rangle \quad$ if $a_j^s(x) = c_j^s x_i^f$
3. $\overline{a}_j^s(x) = c_j^s x_{i'}^s\langle\widehat{X}_i^f\rangle \quad$ if $a_j^s(x) = c_j^s x_{i'}^s x_i^f$
4. $\overline{a}_j^s(x) = \frac{c_j^s}{2}\langle\widehat{X}_i^f(\widehat{X}_i^f - 1)\rangle \approx \frac{c_j^s}{2}\langle[\widehat{X}_i^f]^2\rangle \quad$ if $a_j^s(x) = \frac{c_j^s}{2}x_i^f(x_i^f - 1)$
5. $\overline{a}_j^s(x) = c_j^s\langle\widehat{X}_i^f\widehat{X}_{i'}^f\rangle \quad$ if $a_j^s(x) = c_j^s x_i^f x_{i'}^f$

So all the modified propensities of the slow reaction can be expressed by only using an estimation of the average $\langle\widehat{X}^f\rangle$. This mean can be computed through the resolution of the algebraic equations of the equilibrium law and of the conservation law applied to the reactions of the system. The system of non linear equations can be obtained using a symbolic computation over the reactions and can be resolved by using for example

Netwon's method. For more details about this matter refer to [37]. The general pseudocode of the MSSA that made use of the modified propensities $\overline{a}_j^s$ is given in Figure 2.7.

```
1. Compute the partial equilibrium for the fast virtual system.
   Update the fast variables $X^f = \widehat{X}^f(\infty)$

2. Calcuate the modified slow propensities $\overline{a}_j^s$ and their sum $\overline{a}_0^s$

3. Generate two random numbers $r_1$ and $r_2$
   from the uniform distribution in the unit time interval $[0,1]$

4. Generate values for $\tau$ and $j$ as in (2.6) and (2.7)
   using instead the modified propensities $\overline{a}_j^s$

5. Fire the next reaction by replacing $t \leftarrow t+\tau$ and $\mathbf{x} \leftarrow \mathbf{x}+\mathbf{v}_j$

6. Record $(\mathbf{x},\tau)$ and go to Step 2 or end the simulation
```

Figure 2.7: Pseudocode of the MSSA

2.4 Approximation measurement

When approximate methods are used, the estimation of errors becomes of interest. A good error measure that is representative even when bistable systems are considered is the probability density functions distance presented in [36] as the density distance area. Usually in a distribution distance error estimation an analytical solution of the distributions is not available, for this reason the distributions are estimated from the histograms of a large number of samples where as the number of samples increases the distribution estimation becomes more accurate. To further increase the accuracy of the difference estimation the self distance coming from one of the Monte Carlo methods can be used in comparison with the distance between the distribution of the two different methods. The self distance expresses the statistical fluctuations of the Monte Carlo simulation, thus if the difference between the distributions of the two methods is smaller than their self distance it not possible to assume that the distribution of two samples are different. A drawback of this method is the computational cost that exponentially increases when the number of samples is increased. Since for stochastic simulations a sample corresponds to the result of a single run at a selected time point this estimation procedure requires a high computational load to perform several runs and reach a good estimation of the distributions.

2.4.1 Probability density function estimation

The probability density function of a discrete random variable $\mathbf{X}$:

$$p_{\mathbf{X}}(x) = \sum_{x} P(\mathbf{X} = x)\delta(\mathbf{X} - x) \tag{2.22}$$

can rarely be analytically solved in practical examples. However, if a large number of samples is available it can be approximated by an histogram function $h_{\mathbf{X}}$ constructed as follows. Consider the interval $I = (x_{min}, x_{max})$ in which all the sample values are bounded and divide it in I_i intervals, with $i = 1, \ldots, K$ defined as follows:

$$I_i = [x_{min} + \frac{(i-1)(x_{max} - x_{min})}{K}, x_{min} + \frac{(i)(x_{max} - x_{min})}{K}]$$

and consider the function $\chi(x, I_i)$ defined as:

$$\chi(x, I_i) = \begin{cases} 1 & \text{if } x \in I_i; \\ 0 & \text{otherwise;} \end{cases}$$

the histogram functions $h_{\mathbf{X}}(I_i)$ that approximate $p_{\mathbf{X}}(x)$ can be defined as:

$$h_{\mathbf{X}}(I_i) = \frac{K}{NL} \sum_{j=1}^{N} \chi(x_j, I_i) \tag{2.23}$$

Each function $h_{\mathbf{X}}(I_i)$ measures the average density function of $\mathbf{X}$ in the interval I_i. The summation term $\sum_{j=1}^{N} \chi(x_j, I_i)$ counts the sample points which belong to the interval I_i. Dividing this by N, the number of $x_1, \ldots, x_N$ samples, and the term L/K an approximation of the probability density function in the interval I_i is obtained. So as the number of intervals K increases $h_{\mathbf{X}}$ tends to $p_{\mathbf{X}}$.

2.4.2 Histogram Density Distance

Once the probability density functions of two sets of samples $\mathbf{X}$ and $\mathbf{Y}$ are estimated, using the same number of intervals K, their distance can be measured by the simple formula:

$$D_k(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{K} \frac{|h_{\mathbf{X}}(I_i) - h_{\mathbf{Y}}(I_i)|L}{K} \tag{2.24}$$

that with an explicit expression of the term $h_{\mathbf{X}}$ and $h_{\mathbf{Y}}$ can be rewritten as:

$$D_k(\mathbf{X},\mathbf{Y}) = \sum_{i=1}^{K} \left| \frac{\sum_{j=1}^{N} \chi(x_j, I_i)}{N} - \frac{\sum_{j=1}^{M} \chi(y_j, I_i)}{M} \right| \tag{2.25}$$

Equation (2.25) shows that the distance value is strongly affected by the number of intervals K, and no difference will be measured if $K = 1$ while as K, N, M grow the distance will tend to the real density area distance of the two distributions. Therefore the density area distance increases as the number of intervals increases and this allows us to introduce a degree of tolerance in the distance measure by tuning the parameter K.

2.4.2.1 Self Distance

As the self distance of a random variable expresses its stochastic fluctuation it must be considered when the distribution distance of two random variables is measured. According to [36] the Self Distance is defined as the distribution distance of two set of samples $\mathbf{X}_N$ and $\mathbf{Y}_M$ that independently follow the same distribution. As introduced in this section this distance can be measured using for example the histogram distance between two sets of samples of the same distribution. However in order to have a good self distance estimation the mean and the variance of differences between many pairs of samples set should be considered. Thus, if the number of samples is large in the case of stochastic simulation algorithms a very expensive computation is required to have a good approximation of the self distance. In order to avoid this problem an estimated bound for the average self distance can be considered. In the case of the histogram distance a good estimation is:

$$\sqrt{\frac{2K}{\pi}\left(\frac{1}{N} + \frac{1}{M}\right)} \tag{2.26}$$

where N and M are the number of samples of $\mathbf{X}_N$ and $\mathbf{Y}_M$.

This estimation was presented in [36] and has been proved to be a good approximation as long as the number of samples N and M are large. In conclusion when the histogram distance between two samples obtained from two different Monte Carlo algorithms is less than the self distance bound no conclusion can be made on the difference between the two distributions. In other words using the self distance bound in comparison with the histogram difference between the two set of samples it is possible to ensure that the difference that is measured is due to the Monte Carlo methods and not to stochastic fluctuations.

Chapter 3

Dizzy extension

This project involved the extension of Dizzy [31], an existing open source chemical stochastic simulator, with three new exact stochastic simulation algorithms and an approximate hybrid deterministic and stochastic algorithm. Dizzy was chosen among other existing software thanks to its object oriented modular architecture which made the implementation process easier. This chapter introduces Dizzy and outlines the main steps of the implementation process. Particular attention is paid here to the efficiency of the simulation algorithms. Indeed the whole project is focused on performance optimizations and analysis of the simulation algorithms. The chapter is organized as follows: the first section is a brief presentation of the existing software and its capabilities, the second section discusses the implementation details of the newly developed exact SSA formulations, the third section is focused on the implementation of the hybrid method and the fourth and last section presents the implementation of new tools for performance analysis.

3.1 Introduction to Dizzy

Dizzy is a chemical kinetics simulation software package implemented in Java that was originally developed by Stephen Ramsey at the Institute of System Biology. This project extends a modified version of Dizzy that is being developed by Laboratory for Foundations of Computer Science at the University of Edinburgh. Dizzy allows the user to simulate, using different simulation methods, the reaction kinetics of interacting species. The software is able to write and read models in a subset of the SBML Level 1 format and in the Chemical Model Definition Language (CMDL), a very intuitive language in which models can be directly specified using Dizzy. The modified version of Dizzy currently supports the following simulation algorithms:

Deterministic

- a deterministic (ODE-based) algorithm for simulating chemical reaction kinetics based on the Runge-Kutta method for the resolution of the system of ODEs using both adaptive and fixed steps.
- a deterministic algorithm that uses the library ODEtoJava for the resolution of the system of ODEs using both adaptive and fixed steps.
- an algorithm that solves a stochastic differential equation model using the Euler-Maruyama method.

Stochastic

- The Gillespie stochastic algorithm
- The Gibson-Bruck stochastic algorithm
- The explicit tau leaping algorithm
- The implicit tau leaping algorithm

Different parameters can be specified on the basis of the type of simulation algorithm that is used. When deterministic simulation algorithms are used the user can specify the timestep of the integrator that solves the system of ODEs and the maximum error allowed for the integration. If during the simulation the given time step makes the integrator exceed the specified maximum error a message is displayed and the simulation is cancelled. When stochastic simulation algorithms are used Dizzy allows the user to specify the stochastic ensenble size, which expresses the number of runs, and a confidence interval related to the results of the simulations. The following graphical output can be visualized as result of a simulation:

- A table with the values of the requested concentrations for each time point
- A plot with the average concentrations of the observed species in time, where the average is calculated as the average of the concentrations obtained from all the runs.
- A plot with concentrations in time for each single run.

- An histogram visualizing the average number of firings of the reactions with the average calculated as the average of the total number of firings.

For further details refer to [30].

3.1.1 Reaction Rates

The way in which Dizzy computes the reaction rates from the parameters that are specified in the model is particularly relevant for the implementation of our extensions. A reaction rate in Dizzy can be defined in two ways: using a built-in method or by specifying a custom expression.

The first is the default mode and consists of the simple specification of the rate as a numeric parameter. This is interpreted either as the stochastic rate constant, which expresses the numeric reaction probability density per unit time (when using SSA), or as the kinetic constant for the reactions (when solving ODEs). In the first case Dizzy automatically calculates the reaction rate by multiplying the stochastic constant rate with the number of distinct combinations of reactant molecules, which depends on the kind of reaction that is considered and on the population of the various species. For example if we are considering a bimolecular reaction like $S_1 + S_2 \rightarrow$ the reaction rate, or reaction propensity, is calculated as presented in [9] by computing $c_j S_1^n S_2^n$ where c_j is the stochastic constant and S_i^n represents the number of molecules of the species S_i. In the second case instead the reaction rate is calculated as the kinetic constant parameter multiplied by the concentrations with exponentiation for the stoichiometry. For example for the reaction $2S_1 + S_2 \rightarrow$ the reaction rate is calculated as $k_j[S_1]^2[S_2]$ where k_j is the kinetic rate and $[S_i]$ is the concentration of the species S_i.

Using “custom expression” mode an expression that will be used in order to calculate the reaction rate can be specified by the user. This feature allows the user to create custom reaction rate expressions involving symbols, arithmetic operators, and simple built-in mathematical functions that define how the reaction rate varies.

With regard to our implementation it is important to underline that irrespective of how Dizzy deals with rate calculations the same constant rate that is used for the reaction rate calculation of the stochastic simulation is used for the reaction rate of the deterministic simulation with ODEs. The way in which rates are calculated is decided upon the type of simulator that is used, stochastic or deterministic, and by the default all the populations are considered as expressed in number of molecules. Furthermore in order to increase the performance, when stochastic simulation are performed and species populations are large some approximations are made to calculate the stochastic rate. For example the reaction $3S_1 \xrightarrow{c_j} P$ whose rate should be calculated as $c_j \frac{S_1^n * (S_1^n - 1) * (S_1^n - 2)}{6}$ is calculated as $c_j \frac{(S_1^n)^3}{6}$. From the computational point of view this could seem a minor optimizations but considering that even within a small model the function that calculates the rates can be called million of times during the simulation time is considerably reduced within a tolerable approximation if populations are large.

In conclusion when the species populations are expressed in molecules the dynamics of the population coming from the ODE will represent the dynamics of the expectation value of the molecules in a stochastic simulation across a large number of runs. It must be noticed that in ODEs the number of molecules could assume real values that do not have biological meaning but again if large populations are used this can be considered a good approximation.

3.1.2 Software architecture

The first part of the project was focused on an initial analysis of the software architecture. The analysis was carried out by putting together information coming from the analysis of the source code and from the javadoc documentation of the classes. This analysis allowed us to construct the object map of Dizzy in order to ensure that our extension would have not affected the existing functionalities and would ensure the extension conforms to the existing software architecture. A simplified version of the Object map of Dizzy is reported in figure 3.1. The simplified Class Diagram is divided in two parts, the upper part whose classes are mainly contained in the `org.systemsbiology.chem` package is the hierarchical object map of simulators, the lower part whose classes, mainly contained in the `org.systemsbiology.chem.app`, represent the GUI objects and the classes that are used to process and visualize results coming from simulations. The two parts communicate through the interface ISimulator which provides an abstraction for the current simulator allowing simply calling the method `simulate`.

The implementation of the three new exact stochastic simulation algorithms consisted of modifications to the package `org.systemsbiology.chem` which was adapted with some minor modification and extended with three new classes. The hybrid method implementation and the development of tools for performance analysis focuses on both the packages `org.systemsbiology.chem` and `org.systemsbiology.chem.app`. Further detail about the implementation is given in the following section.

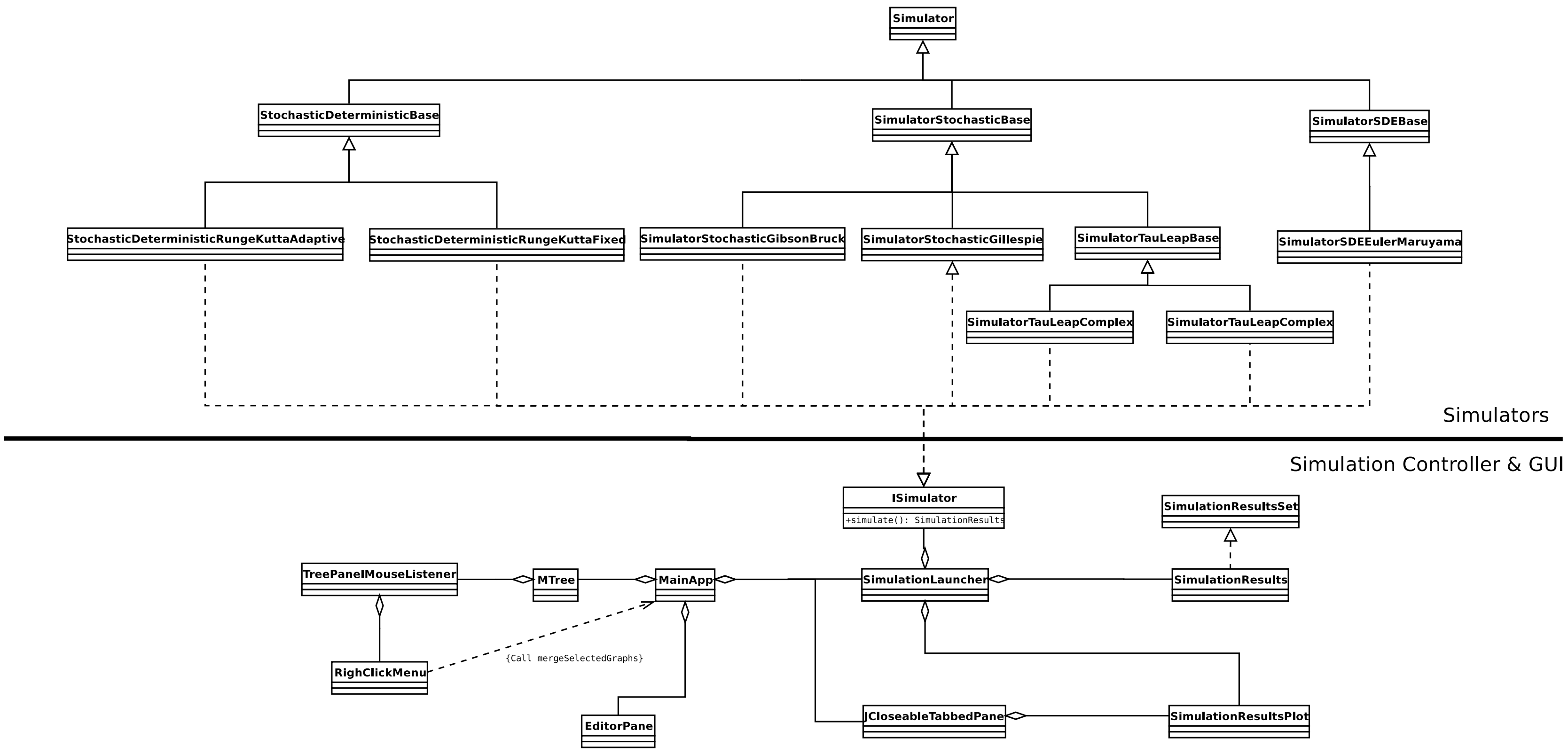


Figure 3.1: Dizzy Class Diagram

3.2 SSA formulations implementation

This section presents the three new classes which have been implemented in order to develop the optimized SSA formulations based on the Direct Method of Gillespie. The main methods of the classes are discussed and a class diagram that shows where the classes have been placed inside the existing software architecture is presented. The next subsection describes some preliminary modifications of the existing sotware structure which were done in order to facilitate the implementation of the three new algorithms.

3.2.1 Preliminary modifications

Currently Dizzy comes with two exact stochastic simulation algorithms: the Direct Method of Gillespie and the Next Reaction method of Gibson Bruck. Our extension was focused on optimizations based on the former method, however in order to increase the performance of the new algorithms some useful features already present in the Next Reaction Method were used. Indeed, as presented in section 2.2.4 a considerable improvement of the performance simulation can be obtained by recalculating at each step of iteration only the propensities of the reaction that have been modified by the last executed reaction. The Dizzy Direct Method recalculates at each iteration all the reaction propensities wasting time in order to refresh propensities that have not been modified because the last executed event did not alter their reactant populations. The Next Reaction Method implemented in Dizzy avoids this problem by computing before the simulation a reactions dependency graph that puts in correspondence each reaction with the reactancts that are altered by its execution. During the simulation the Next Reaction Method uses the information coming from the dependency graph in order to recalculate only the propensities that have changed. We created a class named `DependencyGraphCreator` that using the code present in the Next Reaction Method computes the reaction dependency of a model given as input. The existing Next Reaction Method was modified to use this class and similar features were developed for the three new algorithms. The code of the dependency graph creator was extended in order to compute the index of the first reaction (with respect to the reaction array order) which is modified by the execution of another reaction. This functionality was successively used in order to implement the Logarithmic Direct Method class. The class diagram related to this modification is shown in Figure 3.3, where the new classes are highlighted in red and the classes that have been modified in blue.

3.2.2 Optimized Direct Method - ODM

The class that implements the ODM extends `SimulatorStochasticBase` and its two main methods are `initialize` and `iterate`. The first method is the most important of the class since it implements the functionalities regarding the pre-ordering of reactions based on a pre-simulation. In more detail, when this method is called one or more stochastic pre-simulations are run using the original reaction order and during the pre-simulation the number of times that a reaction is fired is recorded. Based on the average number of times that a reaction is fired a vector of ordered reaction indexes, called `orderSearch`, is constructed by using a simple quick sort algorithm performed by the class `QuickSort` in the `org.systemsbiology.util`. This vector is used at each step of iteration in the method `iterate` in order to find the reaction that satisfies the equation (2.7). The search time is then reduced because starting from reactions with the higher propensities the probability to find the reaction that satisfies (2.7) in a short time is increased. An example of the pre-ordering that is inferred by the optimized direct method for the Galactose model [23] is given in figure 3.2 . The user can specify the number of pre-simulations and the percentage of simulation time, with respect to the total time, which are used to infer the optimal order of reactions. The Class Diagram of the class is shown in figure 3.3.

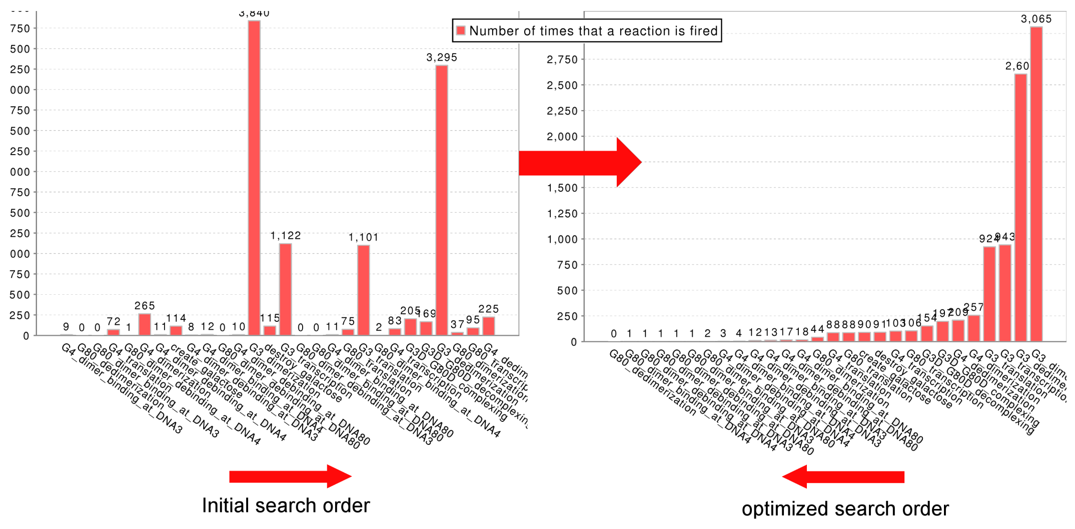


Figure 3.2: Optimized pre-ordering of reactions in the ODM

3.2.3 Sorting Direct Method - SDM

The implementation of the SDM class follows the pseudocode in figure 2.3. Similarly to the Optimized Direct Method class a vector called `orderSearch` containing the search order is used to find the reaction that satisfies (2.7). In this case the order is adjusted at each step of iteration without running pre-simulations. Each time that a reaction is fired the adjustment is performed by shifting its index by one position towards the search direction. The Class Diagram of the class is shown in figure 3.3.

3.2.4 Logarithmic Direct Method - LDM

The implementation of the SDM class follows the pseudocode in figure 2.4. As with the other algorithms the class contains the two methods `initialize` and `iterate`. Additionally here a method called `chooseIndexofNextReaction` has been implemented which performs the optimized search of the reaction. During the initialization the vectors `FirstReactionModified` and `ArrayAggregateProb` are initialized. The first vector is filled with the information coming from the dependency graph creator and contains for each reaction the index of the first reaction that is modified by its execution. The second vector is initialized with the partial summation of the reaction propensities such that:

$$\texttt{ArrayAggregateProb[i]=}\sum_{j=0}^{i}\texttt{ArrayAggregateProb[j]}$$

During the simulation at each step of iteration the index of the next reaction satisfying (2.7) is efficiently located calling the method `chooseIndexofNextReaction` to perform a binary search over the vector `ArrayAggregateProb[i]`. Once the index of the next reaction j is found the reaction is fired and the array `ArrayAggregateProb` containing the partial summation of the reaction propensities is refreshed starting from the index `FirstReactionModified[`j`]`. The Class Diagram of the class is shown in figure 3.3.

3.3 Hybrid Method implementation

A series of different implementations of the hybrid method have been proposed in the literature [32] [17] [1]. Those algorithms mainly differ in the way with which the synchronization between the deterministic simulator and stochastic simulator is performed. Hybrid methods are based on an alternation of iterations between the stochastic part and the deterministic part. While, according to the SSA algorithm, no stochastic reaction events occur, the hybrid algorithm keeps simulating the fast reactions by running the deterministic simulator which uses an integration algorithm. When a stochastic event is registered the integration used by the ODE solver must be stopped and the slow stochastic reaction has to be fired. Successively after an appropriate refresh of the populations the hybrid simulation can proceed with the integration and start again the whole process. The best way to synchronize the two parts was probably proposed in [1] where using the time transformation introduced in the equation (2.10) a hybrid simulation was implemented whose integrator stops according to the value of the term $g_j(\tau|t)$. However few currently existing ODE solvers are able to stop within a user specified control event based on $g_j(\tau|t)$. For this reason we decided to start a first implementation by combining the deterministic simulation and the stochastic simulation following the general framework proposed in [20]. These methods perform the hybrid simulation by simply taking at each step of iteration the minimum time step between the ODE solver and the stochastic next reaction event. If the deterministic time step is the minimum time only the deterministic simulator is used otherwise both the simulators are used according to the stochastic next reaction event time. The hybrid method of this project was implemented following the algorithm presented in [32]. This algorithm simulates the dynamics of a model using one deterministic simulator and one stochastic simulator and iterates according to the pseudocode presented in Figure 2.6. A description of the implementation follows.

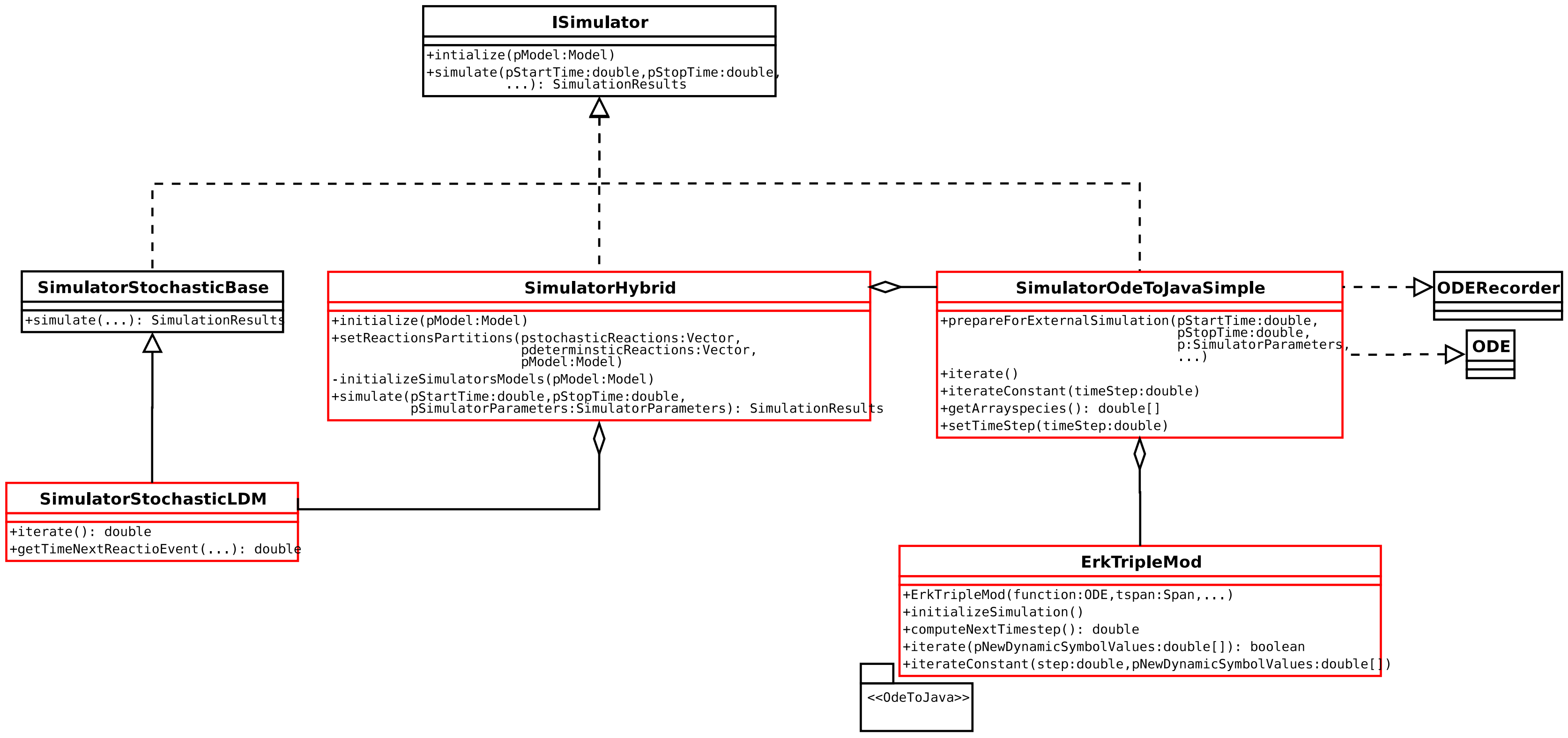

ISimulator
+intialize(pModel:Model)
+simulate(pStartTime:double,pStopTime:double, ...): SimulationResults
SimulatorStochasticBase
+simulate(...): SimulationResults
SimulatorHybrid
+initialize(pModel:Model)
+setReactionsPartitions(pstochasticReactions:Vector, pdeterminsticReactions:Vector, pModel:Model)
-initializeSimulatorsModels(pModel:Model)
+simulate(pStartTime:double,pStopTime:double, pSimulatorParameters:SimulatorParameters): SimulationResults
SimulatorOdeToJavaSimple
+prepareForExternalSimulation(pStartTime:double, pStopTime:double, p:SimulatorParameters, ...)
+iterate()
+iterateConstant(timeStep:double)
+getArrayspecies(): double[]
+setTimeStep(timeStep:double)
ODERecorder
ODE
SimulatorStochasticLDM
+iterate(): double
+getTimeNextReactioEvent(...): double
ErkTripleMod
+ErkTripleMod(function:ODE,tspan:Span,...)
+initializeSimulation()
+computeNextTimestep(): double
+iterate(pNewDynamicSymbolValues:double[]): boolean
+iterateConstant(step:double,pNewDynamicSymbolValues:double[])
<<OdeToJava>>

Figure 3.3: New SSA formulations Class Diagram

3.3.1 Implementation steps

The implementation of the hybrid method consisted of three parts:

1. Definition of the general architecture of the simulator taking into account different stochastic and deterministic simulators

2. Implementation of the reactions partition

3. Implementation of the main class of the simulator divided into:

   - adaptation of the existing simulators
   - Implementation of the simulator initialization
   - Synchronization between the stochastic and the deterministic algorithm

3.3.1.1 General architecture

Following the architecture presented in [20] an arbitrary number of different stochastic and deterministic simulators can be used together in order to implement a hybrid simulation algorithm. Our design choice was to use, as presented in [32], only one deterministic simulator and one stochastic simulator. Based on initial promising results of the newly developed Logarithmic Direct Method we decided to use this algorithm in order to compute the deterministic regime of the hybrid method. The LDM showed good performance which is independent of the ordering of reactions, moreover it is not affected by the accuracy problem presented in section 2.2.8. For the choice of the deterministic simulator Dizzy offered either the use of algorithms based on Runge Kutta implemented as native classes of Dizzy or the use of the more powerful external library ODEtoJava. Our first choice was to use the deterministic simulator natively included in Dizzy because this was an easier way to implement the synchronization between the stochastic and the deterministic simulator. Similar to all the other Dizzy simulators, this ODE-solver had a method called `iterate` that was suitable for the implementation of the hybrid simulation. However first results showed that this ODE-solver was too slow and prone to accuracy and stability numerical problems. For this reason the library ODEtoJava was successively selected and adapted to integrate with the hybrid simulator. An ODEtoJava integrator based on the Runge and Kutta algorithm [3] with adaptive step was used. This integration algorithm is a method of the Runge-Kutta family with a more convenient error estimation. Figure 3.4 summarizes our design choices.

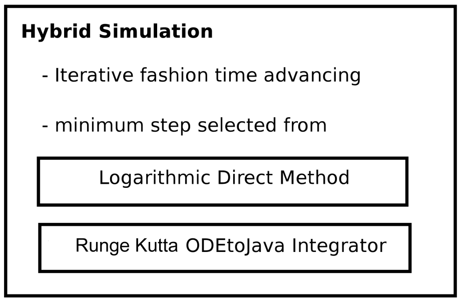
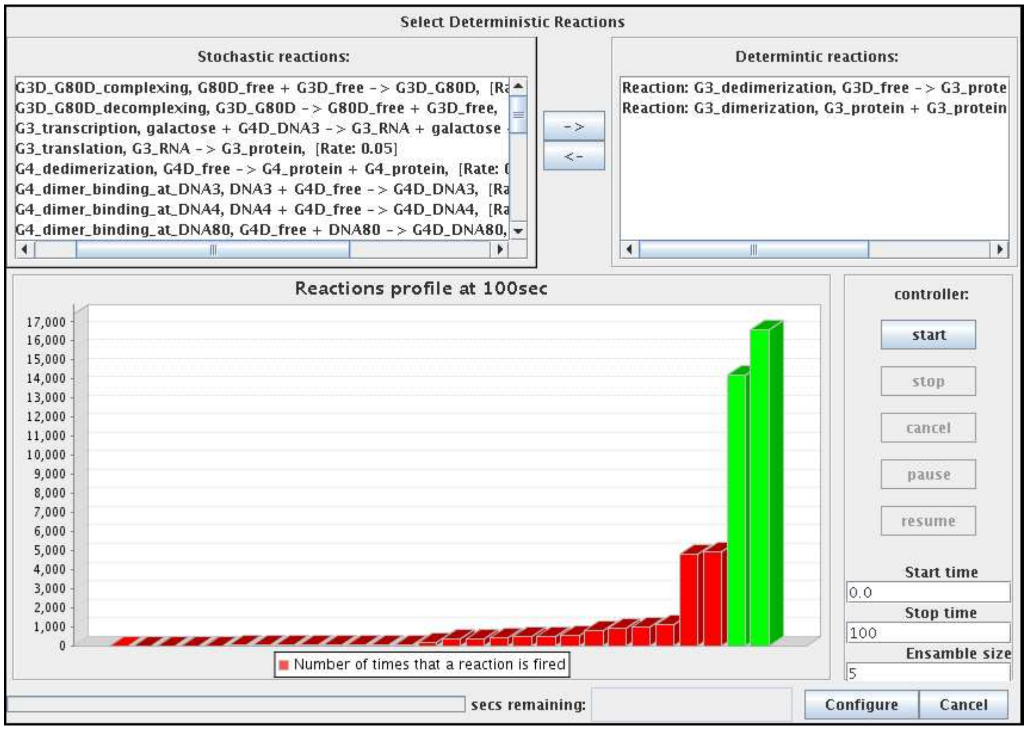

Hybrid Simulation
- Iterative fashion time advancing
- minimum step selected from
Logarithmic Direct Method
Runge Kutta ODEtoJava Integrator

Figure 3.4: Hybrid Method design choices

3.3.1.2 Reactions partition

As introduced in section 3.3.1.2 to decide an optimal partition of the reactions it is still an open problem. Typically, the partition can be made before the simulation in a static way and according to biological insights coming from experiment or empirical observation on pre-simulations, or in a dynamic way during the simulation process by observing reaction rates and reactants population. Since a consolidated theory for the dynamic partitioning has not yet been defined, we decided to implement a graphical framework to help the user during the static partitioning. This new software framework allows the user to execute a number of full stochastic pre-simulations and to observe, by using histograms, the average number of times the reactions are fired. Indeed, as discussed in [37], a good empirical way that can be used to partition the system is to select as fast reactions those that are most frequently fired. To decide how many reactions are to be considered as fast reactions is left to the user. By clicking on the histogram corresponding to one reaction the user can visualize its rate and reactants and can add it to the deterministic regime. A screenshot of the graphical tool that was developed to perform the partition is shown in figure 3.5. The class `CategoryPlot` of the Java library JFreeChart with a modified version of the `BarRenderer` class was used to implement the histogram selection. This graphic interface allows the user to choose the deterministic reactions either by using the selector located in the upper part of the window or by clicking on the corresponding bar in the histograms. Pre-simulations can be done for a simulation time and a number of times specified by the user and can be stopped at any time still allowing visualizing the partial results. Histograms expresses the average number of times that a reaction is fired during the pre-simulation.



Stochastic reactions:

```
G3D_G80D_complexing, G80D_free + G3D_free -> G3D_G80D, [Ra
G3D_G80D_decomplexing, G3D_G80D -> G80D_free + G3D_free,
G3_transcription, galactose + G4D_DNA3 -> G3_RNA + galactose
G3_translation, G3_RNA -> G3_protein, [Rate: 0.05]
G4_dedimerization, G4D_free -> G4_protein + G4_protein, [Rate: (
G4_dimer_binding_at_DNA3, DNA3 + G4D_free -> G4D_DNA3, [Ra
G4_dimer_binding_at_DNA4, DNA4 + G4D_free -> G4D_DNA4, [Ra
G4_dimer_binding_at_DNA80, G4D_free + DNA80 -> G4D_DNA80,
```

Determintic reactions:

```
Reaction: G3_dedimerization, G3D_free -> G3_prote
Reaction: G3_dimerization, G3_protein + G3_protein
```

Start time: 0.0
Stop time: 100
Ensamble size: 5

Figure 3.5: Screenshot of the Hybrid Method partitioner

3.3.1.3 Conformation of the existing classes

The class `SimulatorStochasticLDM` was not changed while several modifications were made to the library ODEtoJava. In particular the class implementing the Runge-Kutta method, was partially rewritten for iterative simulation. In its original form this class was in fact providing a main simulation method called `routine` that implemented the whole simulation process. This structure was not suitable for the iterative simulation of the Hybrid Method, therefore we decided to modify the class splitting the method `routine` in four different methods: `initializeSimulation`, `computeNextTimestep`, `iterate`,`iterateConstant`.

The method `initializeSimulation` provides the initializations of the main data structures that are used during the simulation. The `computeNextTimestep` made use of event location features already present in the ODEtoJava classes in order to return the next putative time step of the integration algorithm. In more detail a control to check the presence of discontinuities, or events, during the current time step is performed. As the discontinuities represent special regions of the ODE where behaviours can change, if an event is found, the event time is returned as next time step, otherwise the current adaptive time step is returned. This allows us to have a more reliable simulation. The method `iterate` as the name suggests implements one iteration of the integration algorithm and evaluates, based on an error estimation, the validity of the obtained results. If one iteration does not produce reliable results the time step is reduced and the method is called again until valid results are obtained. The method `iterateConstant` executes a number of iterations until a time in the given in input when performing constant step integration.

This modification resulted in a new OdeToJava class called `ErkTripleMod`. In order to use this newly developed class a new class called `SimulatorOdeToJavaSimple` that extends the Dizzy class `Simulator` was implemented. This class follows the implementation of the existing Dizzy classes that use the OdeToJava external library. The main methods of the class are `prepareForExternalSimulation`, which is used to initialize the class `ErkTripleMod` with a biological model, and four methods the methods of the `ErkTripleMod` class. The class diagram of the new OdeToJava deterministic simulator with an iterative simulation is shown in figure 3.6.

3.3.1.4 Implementation of the simulator main class

The next step of the Hybrid Method implementation was the development of the main class associated to the simulator. The class called `SimulatorHybrid` extends the class `Simulator` and conforms to the general Dizzy structure of the simulators, therefore implementing the two methods `intialize` and `simulate` that can be called through the interface `ISimulator`. The new class contains one instance of the Logarithmic Direct Method simulator and one instance of the modified deterministic simulator. A description of the main methods of this new class follows.

The method `setReactionsPartitions` is called upon reaction partitioning with the graphical tool presented in section 3.3.1.2. This method receives as input two vectors of `Reaction` objects that respectively represent reactions that must be simulated stochastically and reactions that must be simulated deterministically. These vectors are used to construct two models containing all the species of the original model but having only the reactions specified by the vectors. These two sub-models are used to initialize the stochastic simulator and the deterministic simulator that form the hybrid simulator. When the deterministic simulator is initialized an array called `deterministicSpecies` is constructed. This array contains the index of all the species that are modified during the deterministic simulation and is used to refresh, after one deterministic iteration, only the stochastic simulator species that have changed and that are relevant for the dynamics of the stochastic simulation.

The method `simulate` is the core part of the hybrid simulation. During the simulation the hybrid simulator adaptively calls and synchronizes the stochastic and the deterministic regime. The implementation of the simulator follows the pseudocode in figure 2.6, where at each step of iteration the next time step of the OdeToJava integrator and the next reaction event of the Logarithmic Direct Method are computed. If the former is smaller than the latter an iteration with the deterministic simulator is performed, otherwise an iteration with the stochastic algorithm is performed. After each iteration appropriate refresh of the species concentrations is performed in both the regimes. In particular when a deterministic iteration is chosen the species of the stochastic simulator, whose indexes are contained in the vector `deterministicSpecies`, are refreshed using a method called `updateStochasticPopulation`. In contrast, if a stochastic iteration is chosen, an initial deterministic simulation up to the next stochastic event time is performed and the stochastic species are refreshed using the method `updateStochasticPopulation`. Successively the putative next reaction event is fired and both deterministic and stochastic species are refreshed according to the reaction that was fired. At each step of the iteration concentration values and the actual time are stored in order to plot the results. The class diagram of the new class is presented in figure 3.6.

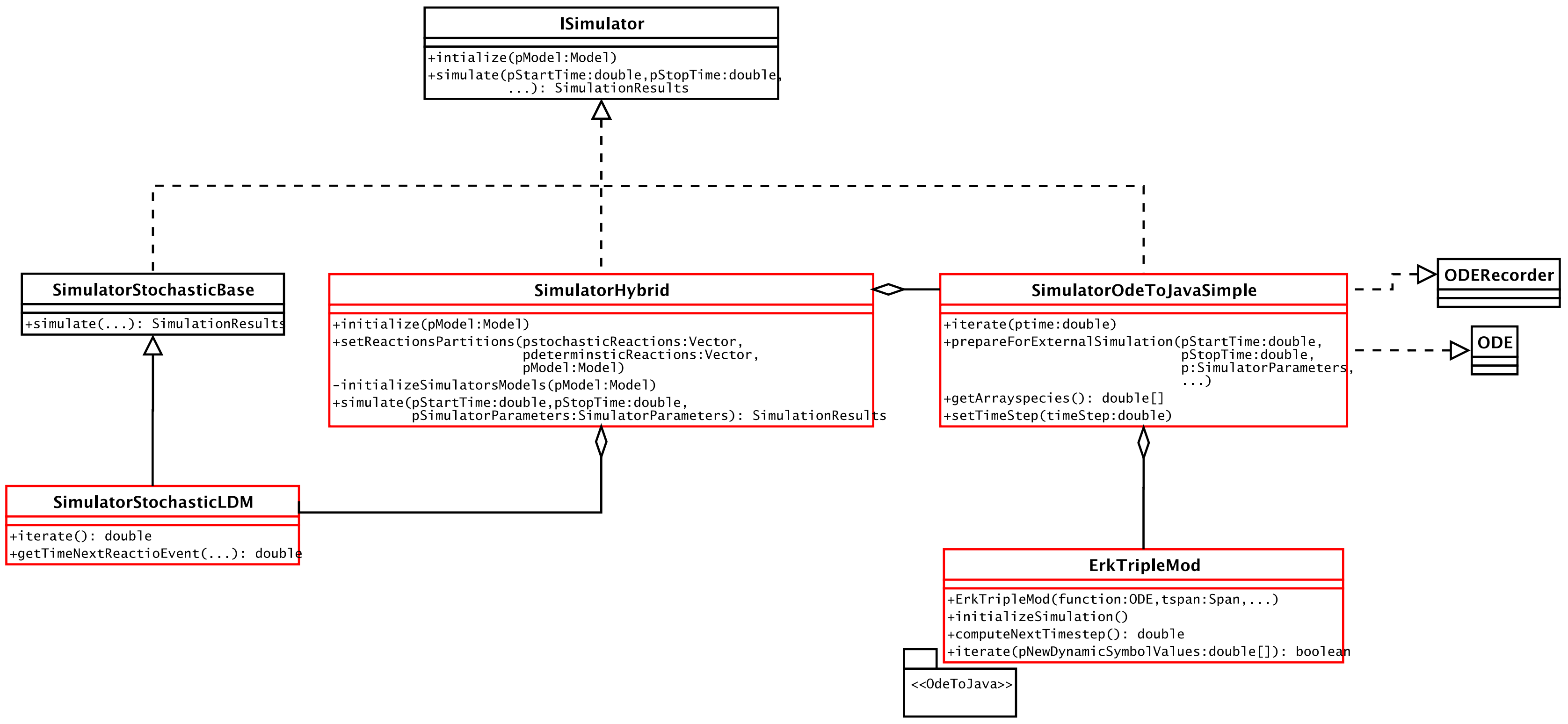

ISimulator
+intialize(pModel:Model)
+simulate(pStartTime:double,pStopTime:double, ...): SimulationResults
SimulatorStochasticBase
+simulate(...): SimulationResults
SimulatorHybrid
+initialize(pModel:Model)
+setReactionsPartitions(pstochasticReactions:Vector, pdeterminsticReactions:Vector, pModel:Model)
-initializeSimulatorsModels(pModel:Model)
+simulate(pStartTime:double,pStopTime:double, pSimulatorParameters:SimulatorParameters): SimulationResults
SimulatorOdeToJavaSimple
+iterate(ptime:double)
+prepareForExternalSimulation(pStartTime:double, pStopTime:double, p:SimulatorParameters, ...)
+getArrayspecies(): double[]
+setTimeStep(timeStep:double)
ODERecorder
ODE
SimulatorStochasticLDM
+iterate(): double
+getTimeNextReactioEvent(...): double
ErkTripleMod
+ErkTripleMod(function:ODE,tspan:Span,...)
+initializeSimulation()
+computeNextTimestep(): double
+iterate(pNewDynamicSymbolValues:double[]): boolean
<<OdeToJava>>

Figure 3.6: Class Diagram of the Hybrid Method

3.4 Performance analysis tools

In order to evaluate performance of the newly developed simulators some useful features for comparison of the simulators over different biological examples were implemented. The developed tools can be divided in two categories: tools for the analysis of exact simulation methods and tools for the comparison between approximate methods and exact methods.

3.4.1 Performance analysis of exact simulations

When comparisons between exact simulators are made the representative performance measures are the average number of fired reactions per second and the average search depth length required for reaction locations. Simple graphical visualization of these measures was implemented as result of one simulation. The new graphs can be visualized by ticking checkboxes in the simulation launcher window. After this an intuitive way to merge and compare these graphs was implemented.

The reaction per second measurement was implemented computing, for each simulation run, the time required for the simulation and the number of fired reactions. At the end of the whole simulation an average of all the ratios "number of reactions/time" is computed. The obtained value is the average number of reactions per second that are fired and is visualized in a histogram implemented using the class `CategoryPlot` of the free Java library JFreeChart [28]. The histogram shows the reactions per second, the name of the simulated model and the name of the simulator that has been used. Histograms resulting from different simulations can be merged using the right-click contextual menu as shown in figure 3.7. Results are merged together and grouped with respect to the models. In this way it is easy to compare the performance of different simulators over different biological models.

Considering that finding the next reaction is one of the most expensive steps of the SSA, the average search depth required to locate the next reaction to fire is a good measure that can help in evaluating the performance of different SSAs. Indeed, the majority of the SSAs formulations are based on improvements that aim to reduce the time required to locate the next putative reaction. Having the possibility to measure the average search depth can help in understanding how the SSAs behave with a particular biological model. This measure however does not take into account other steps of

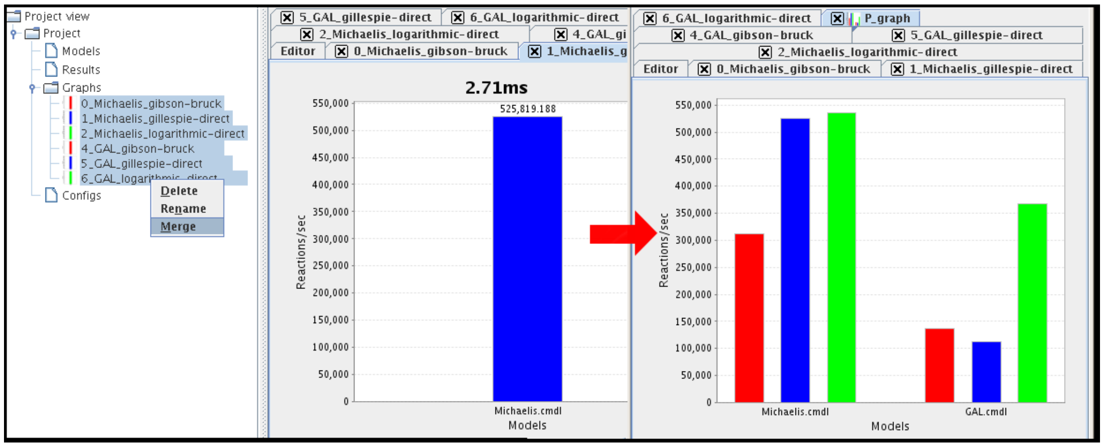


Figure 3.7: Screenshot of the reactions per second histograms

the simulation, like random number and next time generation, therefore it is not itself representative of the whole performance simulation. For each SSA available in Dizzy code used to measure the average search depth was implemented. The way in which this measurement is made is highly dependent on the simulation algorithm. In more detail for the ODM, the SDM and the DM the average search depth is measured as the number of sums that are performed within the operations of equation (2.7), while for the LDM the measure corresponds to the search depth of the binary search performed on the vector of aggregate propensities functions as presented in figure 2.4. The average search depth was not measured for the NRM of Gibson Bruck because for this method the number of operations required to locate the next reaction is virtually equal to one. Most of the time in the NRM is in fact spent in maintaining the heap-like structure that allows us to obtain the next reaction to fire with an unitary cost. The average search depth is recorded every time that a reaction is fired during the simulation, the behaviour of the average search depth during the simulation is finally visualized in a chart developed using the class `XYPlot` of the free Java library JFreeChart [28]. These graphs can be merged using the the right-click contextual menu as presented in figure 3.8.

3.4.2 Performance analysis of approximate simulations

An estimation of the accuracy of the results obtained using different stochastic simulation methods should consider the distance between estimations of the population distributions. Distributions can be estimated from the histograms of a large number of samples and distances can be measured, as presented in section 2.4, as the density

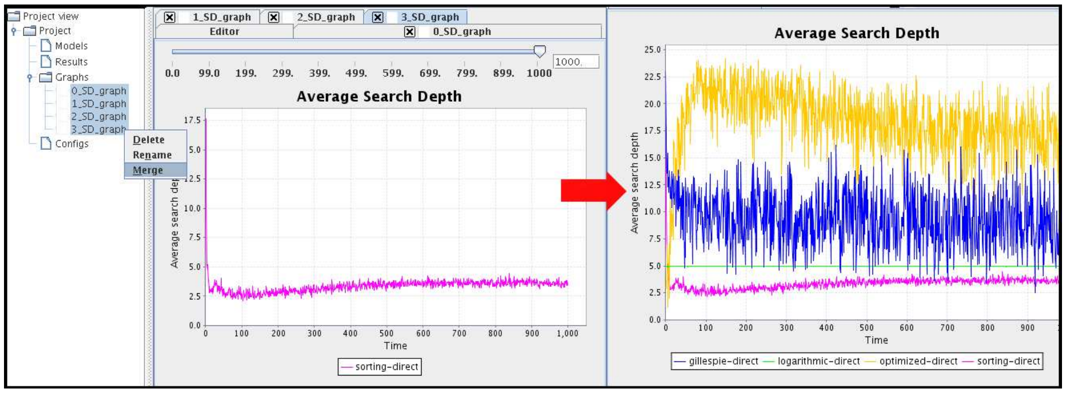
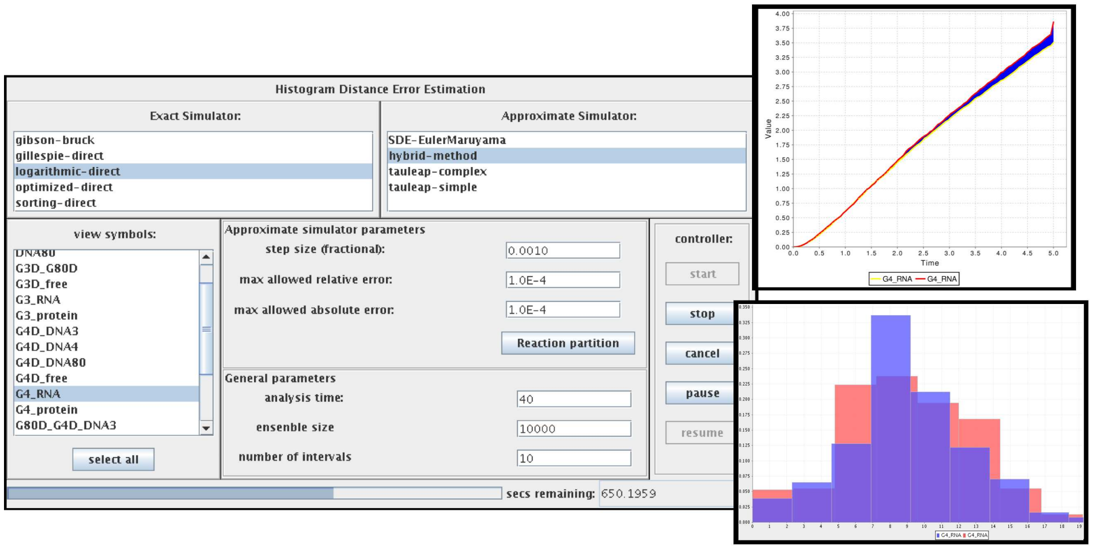


Figure 3.8: Screenshot of average search depth graphs

difference area in comparison with an estimation of the self distance. The self distance expresses the stochastic fluctuations of a single Monte Carlo method and can be calculated as the density difference area between distributions of samples. A reliable distribution estimation is usually based on a large number of realizations, which require a high computational load to be generated. For this reason a formula that computes a bound for the average self distance can be used to avoid computational complexity. This formula is a distribution indipendent common bound which depends only from the number of samples and the number of intervals used to estimate the probability density functions. A new graphical interface that can be used to compute density distance areas between approximate and exact simulations was developed. This new tool can be used to configure and run approximate and exact simulators and to compare possible errors introduced by approximate methods. In practice when a density area distance estimation is performed, two stochastic simulators are configured to simulate for the same number of runs. The user can select a time point in which the histogram distance is computed. Within this time point, results coming from all the simulation runs are stored and succesively used to estimate population distributions. Distribution estimations are visualized using histograms. The software calculates the numerical values of the distances between the approximate method distribution and the exact method distribution. When this difference is larger than the estimated self-distance bound an error is measured, otherwise errors cannot be derived because it is impossible to separate the difference from stochastic fluctations. In this case, a more accurate estimation is required and can be performed either by using a large number of samples or by increasing the number of intervals. The whole estimation process is summarized in figure 3.10.

Our tool allows the user to perform analysis specifing different simulators and parameters and to consider particular species of the biological model. An example of a typical output and a screenshot of the main window associated with the tool are shown in figure 3.9. The upper part of the window is used to select the approximate and the exact simulator, a panel on the left allows the user to select the species that are considered for the density distance area estimation. The central panel is used to define the parameters of the simulation and the number of intervals that are used to construct the histograms. Histograms express the likelyhood of occurence of population values within the intervals at a specified time point, while plots display differences between the average behaviour of the populations over the whole simulation time.


Histogram Distance Error Estimation

Exact Simulator:
gibson-bruck
gillespie-direct
logarithmic-direct
optimized-direct
sorting-direct

Approximate Simulator:
SDE-EulerMaruyama
hybrid-method
tauleap-complex
tauleap-simple

view symbols:
DNA80
G3D_G80D
G3D_free
G3_RNA
G3_protein
G4D_DNA3
G4D_DNA4
G4D_DNA80
G4D_free
G4_RNA
G4_protein
G80D_G4D_DNA3

Approximate simulator parameters
step size (fractional): 0.0010
max allowed relative error: 1.0E-4
max allowed absolute error: 1.0E-4

General parameters
analysis time: 40
ensenble size 10000
number of intervals 10

secs remaining: 650.1959

Figure 3.9: Screenshot of the error estimator tool and its typichal output

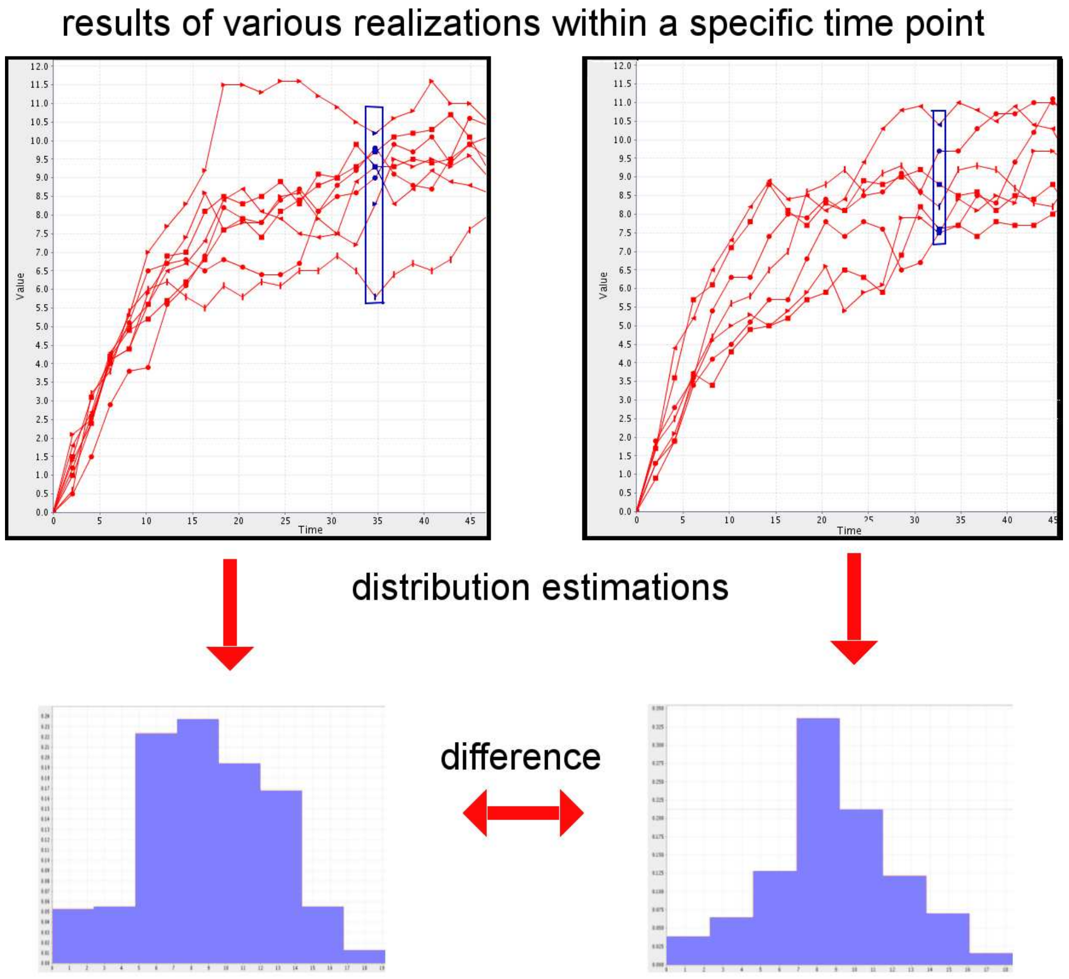

results of various realizations within a specific time point
Value
Time
distribution estimations
difference

Figure 3.10: Density difference distance area estimation

Chapter 4

Biological models

This chapter presents three biological models which were used to evaluate the performance and correctness of the new simulation algorithms. These three models were used to evaluate the performance of exact stochastic simulation algorithms. Having a different scale of reactions such models are in fact particularly suitable to study the scalability of the various algorithms as the number of reactions increases. The second model with modified parameters was also used for a first evaluation of the newly developed hybrid method. For each model a brief explanation of the dynamics and description of expected results are given.

4.1 Michaelis Menten

The first and simplest model is a model of the enzymatic reaction considered by Michaelis and Menten for their famous kinetic model. The model is the abstraction of a simple enzymatic reaction formed by three reactions in which a substrate S has to bind with the enzyme E in order to react and produce the final product P. In our model the binding between the enzyme and the substrate is reversible while the transformation of substrate in product is considered to be definitive. This very simple model is made by only three reactions:

1. $S+E \longrightarrow ES$
2. $ES \longrightarrow E+S$
3. $ES \longrightarrow P+E$

with initial number of molecules $E=100$, $S=100$, $ES=0$ and $P=0$.

1. This reaction represents the formation of the compound between substrate and enzyme. In our model this reaction was set to have a stochastic rate of 1.
2. This reaction expresses the unbinding of the compound enzyme-substrate ES into its two components substrate S and enzyme E. In our model this reaction has a stochastic rate of 0.1
3. This is the reaction that produces the final product P by consuming the substrate S and releasing the enzyme in its free form E. In our model this reaction has a stochastic rate of 0.01.

The expected dynamics of this system, with rates and concentrations specified above, follows the graph in figure 4.1. It can be seen that as the simulation proceeds the concentration of the enzyme-substrate compound ES quickly reaches in less than 1 seconds a threshold that represents the fact that all the free enzyme E is bound with the substrate S. Indeed, at the same time the concentration of the enzyme E and the substrate S decrease almost to zero. Successively S keeps decreasing while the concentration of E starts to increase representing the fact that as the P is produced the enzyme E is released in its free form. It should be seen that the concentration of P and E in fact increases following the same law. In the meanwhile the concentration of ES decreases as it is consumed by the reactions 2 and 3. The whole system reaches equilibrium at around 500 seconds because all the substrate S has been converted in product P.

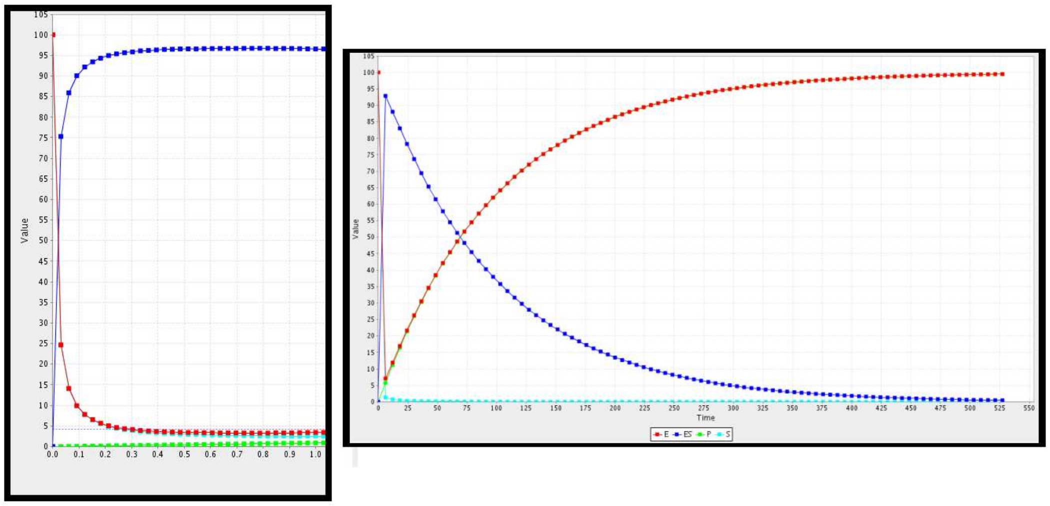


Figure 4.1: Expected dynamics for the Michaelis Menten model

4.2 GAL4 system of Yeast - GAL

This model, included in the original release of Dizzy and consisting of **28** reactions, abstracts the regulation of the production of various structural control metabolic genes in yeast as response to the presence of external galactose. The concentration of external galactose is a parameter of the model and can be either fixed or can vary in time according to a specified rate. As reported in [8] this biological system senses the the presence of external galactose and switches on a series of proteins which control the galactose metabolic pathway. As the concentration of galactose increases the concentration of activated $GAL4$ proteins increases while for lower concentration of galactose more $GAL80$ are activated. Former proteins promote the expression of GAL genes while latter proteins act as a repressor for GAL genes binding with $GAL4$ proteins and deactivating them. When the concentration of galactose is zero, activated $GAL80$ proteins are enough in order to keep the expression of GAL genes at a low level by deactivating most of the $GAL4$ proteins. The system regulates the presence of metabolic genes also upon the presence of internal galactose. In particular $GAL3$ proteins, which act as a repressor for $GAL80$ proteins, are activated when the concentration of internal galactose increases. In this way when internal galactose is produced the system will repress the repressor $GAL80$ and the expression of GAL genes will increase again. All the feedbacks do not act using single proteins of the proteins mentioned above rather they act on dimers, like $G3D$ or $G80D$, with a specific rate of creation and destruction. Details of this metabolic pathway are beyond the scope of this project, for more information refer to the original ODE model presented in [27] . Overall this pathway was particularly suitable for our analysis for two reasons: first of all it is a good example of a native Dizzy model with a medium number of reactions, secondly the different time scale of the feedbacks makes the probability of reactions change drastically during the simulation and this was a perfect example to test possible prediction problems for methods like the ODM that are based on prediction over an initial pre-simulation time. Morover, increasing the presence of external galactose this model was also appropriate for a hybrid deterministic and stochastic simulation, more details about this matter are given in section 5.2.

The main reactions of this model, including the first two reactions that expresses the time varying concentration of external galactose, are the following:

1. $\longrightarrow galactose\ [10e^{-20\frac{t}{T}}]$ where t is the current time and T the total time

2. *galactose* $\longrightarrow$ [0.1]

3. *G4D-DNA3* +*galactose* $\longrightarrow$ *G3-RNA* + *G4D-DNA3*+*galactose*

4. *G3-protein* + *G3-protein* $\longrightarrow$ *G3D-free*

5. *G3D-free* $\longrightarrow$ *G3-protein* + *G3-protein*

6. *G3-RNA* $\longrightarrow$ *G3-protein*

7. *G3D-free* + *G80D-free* $\longrightarrow$ *G3D-G80D*

8. *G80D-free*+ *G4D-DNA3* $\longrightarrow$ *G80D-G4D-DNA3*

The expected dynamics of this system follows the graphs in figure. It can be seen that when the *galactose* increases $GAL4$ proteins quickly promote the production of $GAL3$ as expressed by the fast increase in concentration of $G4D-DNA3$. When the concentration of *galactose* starts to decrease more $GAL80$ proteins inhibit the creation of $GAL3$ and this is represented by the competitive binding of $G80$ with $G4D-DNA3$ as the complex $G80D-G4D-DNA3$. Successively the $GAL3$ proteins that were produced at the beginning inhibits the $G80$ proteins forming the complex $G3D-G80D$. From the graph in figure 4.2 it can be seen that these three phases happen within a two different time scale.

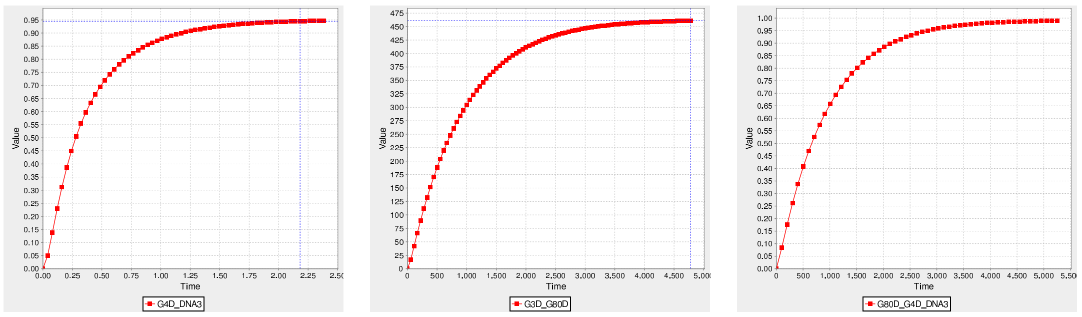


Figure 4.2: From left to right the dynamics of *G4D-DNA3*, *G3D-G80D* and *G80D-G4D-DNA3*. The population of *G4D-DNA3* reaches the maximum value approximately in two seconds while the populations of *G3D-G80D* and *G80D-G4D-DNA3* reach it at around 5000 seconds

This model was fistly simulated with an initial amount of one molecule of *galactose* varying in time according to reaction 1 and reaction 2. Succesively in order to test the hybrid method the same model was simulated with a constant coentration of 100000 molecules of *galactose*. This modification makes the reactions 3, 4, 5 and 6 fire very frequently. Therfore these reactions are more efficentely simulated with a deterministic simulation.

4.3 EGF receptors signal pathways - Shoeberl

The third model is a very detailed computational model presented in [22] which is based on the epidermal growth factor (EGF) receptor signal pathway model introduced in [4]. This model involves **218** reactions and **93** species that form the signaling pathway that is activated when an extracellular signal is received by the cell. When the signal is received the activation of the MAP kinase cascade through the kinases Raf, MEK and ERK-1/2 follows. This example was used to evaluate the performance of the novel developed algorithms when the number of reactions is very large. For this reasons we will not focus on the details of this model which complexity is prohibitive, for more details refer to [22] [4].

Chapter 5

Results

In order to evaluate the usability and the correctness of both the new simulators and the performance analysis tools, different simulations and analysis of the models presented in the previous chapter were performed. This chapter is divided in two sections, the first focuses on simulations made using SSAs formulations over the Michaelis Menten, the GAL and the EGF Shoeberl model, while the second focuses on an initial analysis of the Hybrid simulator using the GAL model with modified parameters. Each section starts with a survey of the techniques that were used to evaluate the obtained results.

5.1 Exact simulations results

The LDM, the SDM and the ODM were run in comparison with the existing stochastic simulation algorithms of Gillespie and Gibson Bruck. Simulation were performed considering three model with different scale number of reactions:

- The Michaelis Menten model - 3 reactions
- The GAL model - 28 reactions
- The EGF Shoeberl model - 218 reactions

For each model two different analysis were performed:

- Comparison of the computational performance of the algorithms by using reactions per second histograms.
- Critical evaluation of the performance comparing average search depth graphs.

Our simulations were parametrized with respect to the simulation time and the number of runs and when the ODM was used also with respect to the pre-simulation time and the number of pre-simulations. For each example we provide a table that puts in correspondence parameters with average times required to perform one run. We also provide a predicted simulation time for a number of 15.000 runs, in this way we hope to highlight how small performance differences within a single simulation become statistically relevant when larger number of stochastic simulation are considered. Experiments were run on an Intel Centrino with two 1.83Ghz cpu cores and 2GB of Ram. This section proceeds presenting the results that were obtained from the three models.

5.1.1 Michaelis Menten

The Michaelis Menten enzymatic model was simulated for 100 seconds for a number of 1000 run. All the existing exact simulators were used and the ODM was configured in order to perform one pre-simulation of 10 seconds. The average simulation times are given in table 5.1.1, while the average search depth beahviour for each SSA formulation is shown in figure 5.1. The average search depth of the Gibson Bruck NRM was ignored because it is always unitary.

Method	Run time (ms)	Time for 15000 run (seconds)
DM	5.035ms	75.52s
NRM	5.174ms	77.61s
LDM	4.781ms	71.71s
ODM	5.038ms	75.57s
SDM	4.773ms	71.59s

Table 5.1: Execution times of the SSAs on the Michaelis Menten model

The results in table show similar executions for all the methods. Indeed as expected no big differences are measured when small examples like the Michaelis Menten model are considered. However, within these small differences, execution times are in accord with the expected results. In particular as we expected the NRM of Gibson Bruck is the slowest method because the computational overhead that it introduces to create and maintain the heap structure is larger than the gain in performance coming from its unitary search depth. Nonetheless this small differences could be influenced more from the current computer state rather than from algorithms. A more reliable performance measure is the average search depth presented in figure 5.1. This graph shows that all the simulators give very similar results, but here we can also notice a general reduction of the average search depth, and then an increasing of the performance, as the simulation continues. This reduction can be traced to the fact that as the simulation proceeds the reaction that is used to create the product P, that is the first reaction in the search order, becomes more and more probable. Thus in all the methods the average search depth decrease almost to zero representing that no operations are required in order to locate the next reaction to fire. Within this general positive trend the Direct Method shows to have slightly lower average search depth values because the original reaction order is the best order therefore, if no operations are performed in order to arrange the reactions, the best performance are obtained.

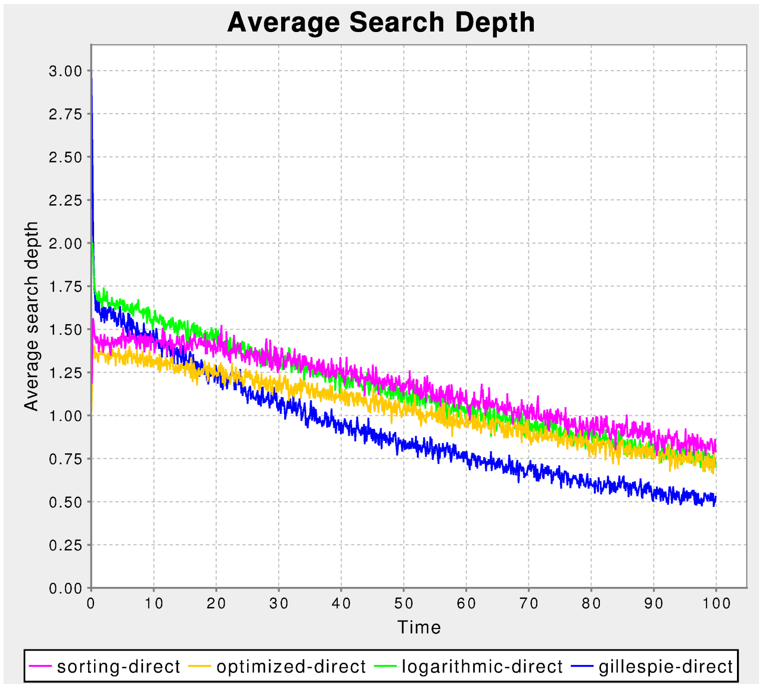


Figure 5.1: Average search depths for the Michaelis Menten model

5.1.2 GAL

The GAL4 Yeast system model was simulated for 1000 seconds for a number of 1000 run. The ODM pre-simulation was run for the 10% of the whole time. The average simulation run times are given in table 5.2 and the average search depth behaviour for each SSA formulation is shown in figure 5.2.

In contrast with the Michealis Menten simulation, times reported in table 5.2 show a big difference between the methods. In particular the difference between the run time

Method	Run time (ms)	Time for 15000 run (hours)
DM	2027.52ms	9h 12min
NRM	797.43ms	3h 19min
LDM	562.79ms	2h 21min
ODM	562.27ms	2h 20min
SDM	562.50ms	2h 21min

Table 5.2: Execution times of the SSAs on the GAL model

of the Direct Method and the run time of other methods is more significative. The ODM, SDM, and LDM are more than 4 times faster than the DM because as the number of reactions increases the performance scalability of these methods becomes more effective. Differences are also registered between the execution time of the NRM and the other methods. In a medium size example like the GAL model the computational overhead of the NRM becomes in fact more evident decreasing the whole performance of this method. Performance of the LDM, the ODM and the SDM are very similar but if the attention is focused on the average search depth noticeable differences are registered. The reason for this discrepancy is trade off between the reduction of the average search depth and the introduction of additional computational load for an optimized next reaction location, that in a medium size example plays an important role. The graph in figure 5.2 shows in fact that these three algorithms behave in a very different way. In more detail the LDM exhibits a slow average search depth that remains almost constant around $log_2(28) = 4.8$ because of the binary search that is used to locate the next reaction to fire. Therefore as expected, when bigger models are considered the LDM shows reliable performance that is independent on the number of reactions. The SDM is, with this particular example, the method that has the smallest average serach depth values. As discussed in section 4.2, GAL model rection probabilities considerably vary during the simulation thus the progressive adjustment of the order of reactions in SDM leads to a good degree of efficiency. For the same reason the ODM, which was configured in order to sort the reactions with respect of the reaction propensities of the first 10 seconds of simulation, shows an average serach depth that drastically increases, even surpassing that of the DM, as the simulation proceeds. A surprising result is that, even if the average search depth of the ODM is larger than the average search depth of the DM for more than one half of the simulation, the best overall performance is reached by the ODM. The reason for this odd behaviour is the trade-off between average search depth optimizations and the computational overhead introduced that within this particular example advantages more a method like the ODM that predicts a good reaction ordering for the first half of the simulation time without introducing overhead during the simulation. Morover even when the average search depth of the ODM is worse than that of the DM the optimized direct method has better performance thanks to the optimal way that it uses to refresh reactants population according to the dependency graph.

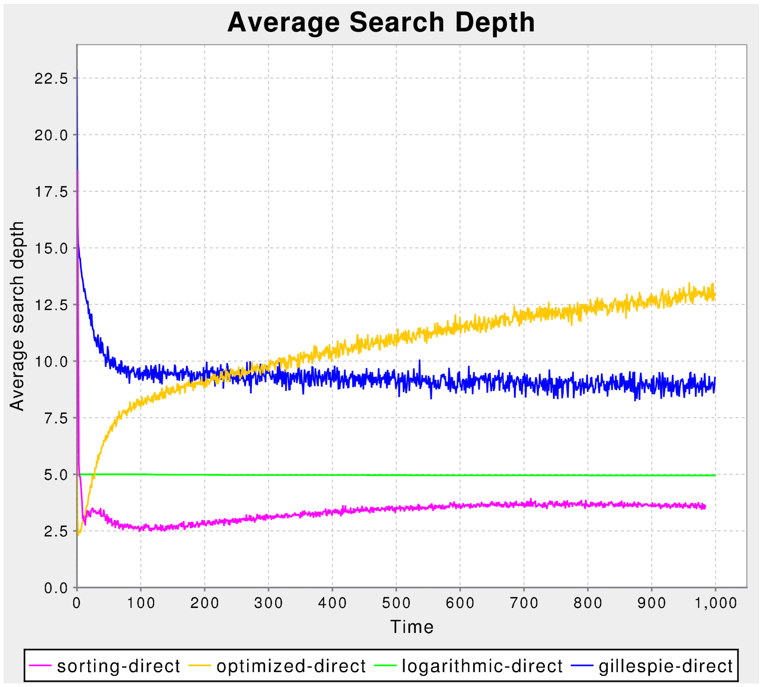


Figure 5.2: Average search depths for the GAL model

5.1.3 EGF Shoeberl

This huge model, because of its high computational load, was simulated for 1 second for a number of only 100 run. The ODM was firstly configured to perform one pre-simulation on the 10% of the whole time and successively to perform two pre-simulations of the whole simulation time. The average simulation run times are given in table 5.1.3 and the average search depth behaviours for each SSA formulation is shown in figure 5.5. A more detailed comparison between the average search depth of the ODM, the LDM and the SDM is presented in figure 5.4.

First of all looking at the results in the table it can be seen that in contrast with the results of the GAL model the NRM of Gibson Bruck has an execution run time that is comparable with those of the LDM, the ODM, and the LDM. Indeed, with this

Method	Run time (ms)	Time for 15000 run (hours)
DM	13254.44ms	55h 13min
NRM	585.61ms	2h 43min
LDM	525.27ms	2h 18min
ODM	514.93ms	2h 9min
ODM*	511.26ms	2h 8min
SDM	521.28ms	2h 10min

Table 5.3: Execution times of the SSAs on the EGF Shoeberl model

huge model the overhead needed to maintain the heap like structure becomes negligible with respect to the gain in performance reached with the unitary average search depth. However as expected, without taking in consideration the Direct Method, this algorithm is the slowest SSAs. The execution times in the table show also that the direct method is twenty times slower than other algorithms, so as expected when the number of reactions increases the gap between optimized SSAs and the DM becomes larger. Performance of the LDM, the SDM and the ODM are slightly different but in general remain of the same order. As usual a more detailed comparison can be done by looking at the average search depths in figure 5.5. As in the execution times a big difference is shown between the DM and the other methods. Similarly to the GAL example the LDM has performance that is independent of the number of reactions and is almost constant around $log_2(218) = 7.8$ because of the binary search. The SDM rapidly converges to an optimal reaction order and has the smallest average search depth values. The ODM, when run configured according to a pre-simulation of 10% of the whole time, has performance similar to that registered in the GAL example. The average search depth increases as the simulation proceeds because the reaction order that was predicted with the pre-simulation becomes less effective. However, with this model the average search depth of the ODM remains always lower than the average search depth of the DM and remains around values similar to those of the SDM and the LDM. In this example, in contrast with the GAL example, the dynamics of the initial 10% of the simulation are representative of the dynamics of the whole simulation. For this reason on the EGF Shoeberl model the ODM is the method with the best performance. Thus from our analysis results that is better to do a pre-simulation and to pre-order the reactions avoiding the introduction of computational overhead during the simulation, used to order the reactions (SDM) or to find the reaction faster (LDM).

In the light of this result we run the ODM using two pre-simulations of the whole

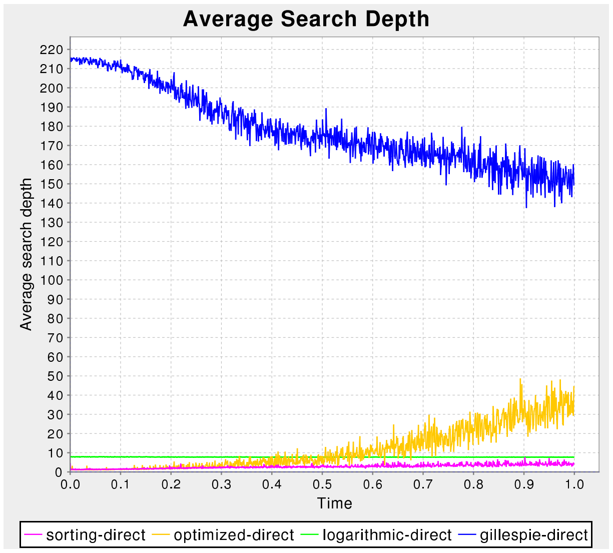


Figure 5.3: Average search depths for the EGF Shoeberl model

simulation time in order to see how much would have been the performance gain. As expected the average search depth is noticably reduced staying around values similar to those of the SDM as shown in figure , however the number of reactions per second was similar to the number registered using a pre-simulation of the 10% of the simulation as shown in . This highlights that when a model has reaction probabilities that do not vary too much in time a simple ODM configured with a pre-simulation of the 10% of the whole simulation is the best choice.

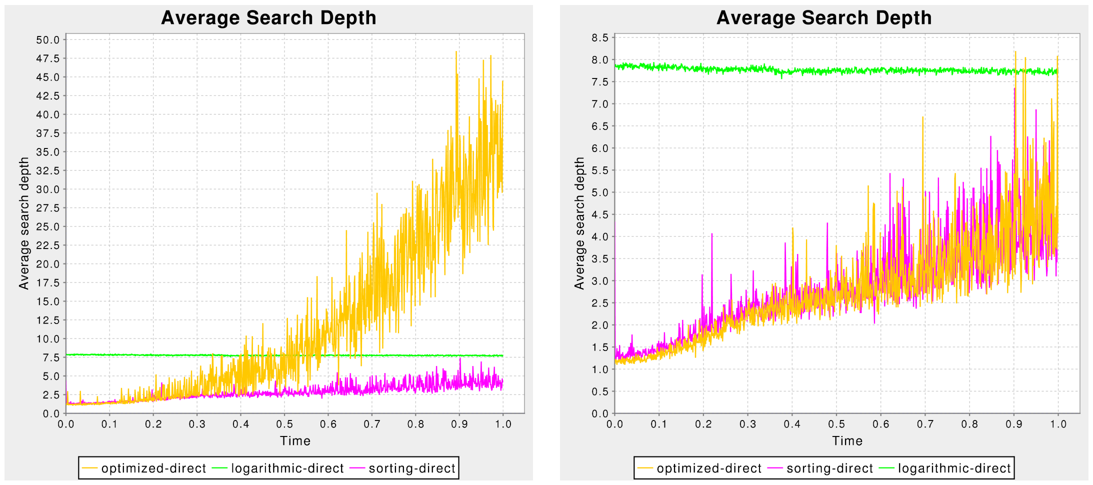


Figure 5.4: On the left average search depths of the ODM, the LDM, the SDM, for the EGF Shoeberl model, on the right the same graph with the ODM configured for a 100% pre-simulation time.

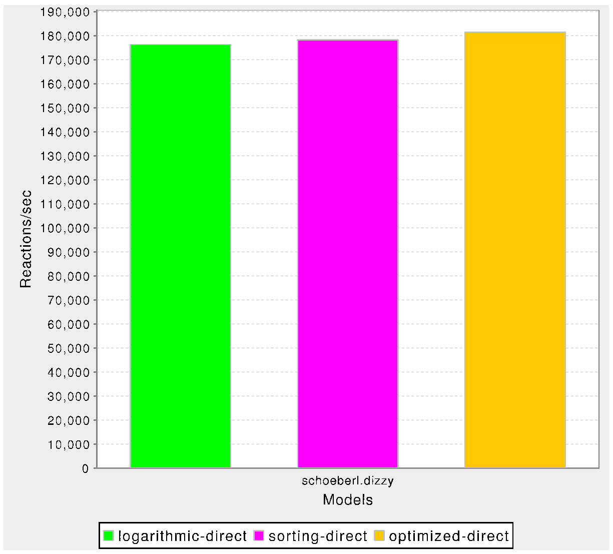


Figure 5.5: Reactions per second of the ODM with pre-simulation of 100% of the simulation time

5.1.4 General considerations

Results obtained by simulating the Michaelis Menten, the GAL and the Shoeberl model show that the LDM, the SDM and the ODM are, in general, more efficient than the DM and the NRM. Performance was studied considering two measures, the average number of reactions that are fired per second and the average search depth. Analysis based on the former highlights that within biological systems with a small number of reactions, like the Michaelis Menten model, all the algorithms exhibit similar performance. As the number of reactions increases, the DM becomes quite slow while the other methods show good performance scalability. The NRM is, in general, slower than the LDM, the SDM and the ODM, in particular when middle sized examples are considered, where the computational overhead introduced to maintain the heap-like structure affects the performance more. Our analysis also confirmed the results presented in [34] that show that the ODM is in general one of the best formulation of the SSA. Analysis based on the average search depth showed that even when the average search depth of the ODM increases during the simulation, due to wrong predictions of the optimal order of reactions, the number of reactions computed per second by the ODM is greater than that of other methods. This occurs because the ODM does not introduce computational overhead to the SSA. Taking into account only the average search depth, the best results are reached by the SDM and the LDM that have stable and reliable good performance during the entire simulation time. Figure 5.6 summarises the number of reactions per second that were obtained using the various simulation algorithms over the three biological models.

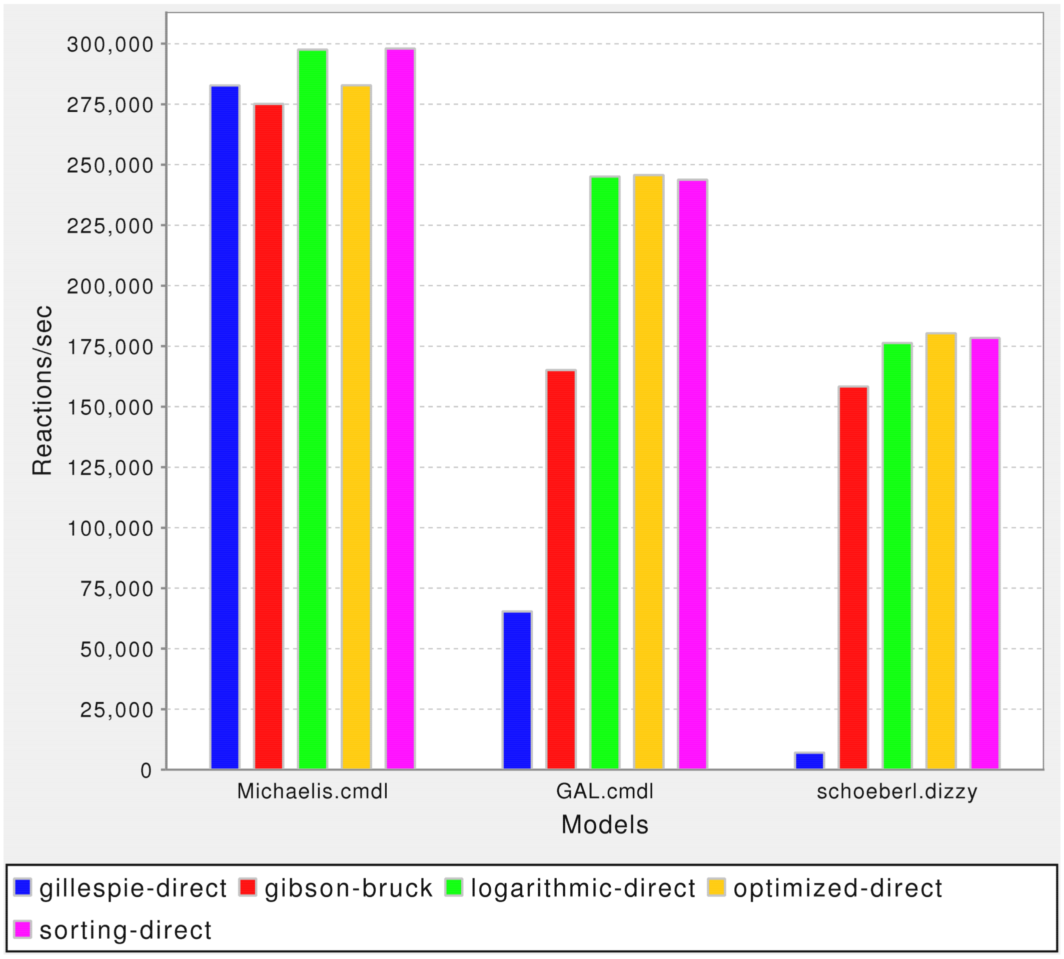


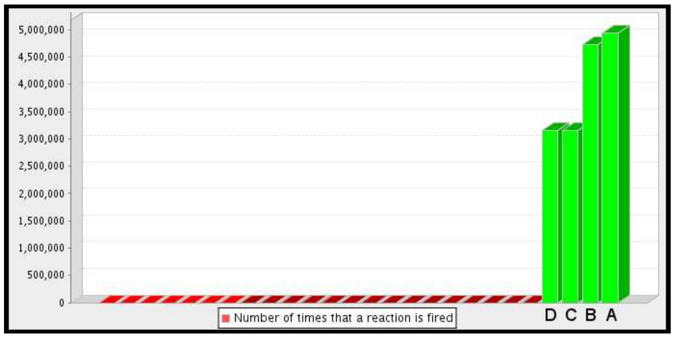
Figure 5.6: From left to right: reactions per second in the Michaelis Menten model, the GAL model and the Shoeberl model.

5.2 Approximate simulations results

In order to evaluate the performance and the accuracy of the hybrid simulator two measures were considered in comparison with an exact simulator:

- the average run time
- the density area distance

The first was the only measure considered to compare the computational performance of the hybrid method with other exact simulators. Measures like the number of firing per second or the average search depth are in fact not representative in the context of reactions modelled deterministically. The second measure allowed us to investigate the approximation that is introduced by the hybrid method when some of the reactions are modelled deterministically. The GAL example with a high constant number of galactose was used for the simulation. This example is suitable for our purpose because it exhibits a clear partition of the reactions. As shown in figure 5.7 there are four reactions that are fired more frequently.



A *G4D-DNA3 +galactose* $\longrightarrow$ *G3-RNA* + *G4D-DNA3+galactose*

B *G3-protein* + *G3-protein* $\longrightarrow$ *G3D-free*

C *G3D-free* $\longrightarrow$ *G3-protein* + *G3-protein*

D *G3-RNA* $\longrightarrow$ *G3-protein*

Figure 5.7: Fast reactions in the GAL model

The hybrid method was parametrised by changing the number of deterministically simulated reactions and by changing the integration step used as both the initial integration step and the step used for the constant step integration of the ODE-solver. Simulations highlighted the trade-off between the gain in computational performance and the loss of exactness. All the simulations were performed using the Logarithmic Direct Method as a reference for exact simulations. This simulator was chosen among all the other exact simulators for two reasons. First of all, it has stable computational performance that allowed us to concentrate more on the performance of the hybrid method. Secondly, since the hybrid method uses the Logarithmic Direct Method as internal stochastic simulator, this was a natural choice to compare the execution time of the two methods. This section proceeds presenting the results that were obtained.

5.2.1 Simulating four deterministic reactions

In our first experiment, we configured the hybrid method to simulate all the four reactions, introduced in figure 5.7, in a deterministic way. Because of the high computational complexity, the modified GAL example was simulated for only five seconds. The fifth second was selected as time point for the difference estimation. One thousand realisations were used for distribution estimations and histograms were constructed considering ten intervals. According to equation 2.26, a theoretical bound for the average self-distance was calculated:

$$\sqrt{\frac{20}{\pi}\left(\frac{1}{1000}+\frac{1}{1000}\right)} = 0.11284 \tag{5.1}$$

Several species were selected for the estimation of the density difference area between the results obtained from a full stochastic simulation with the LDM and those obtained using the Hybrid Method. Figure 5.8 shows the histograms corresponding to the distribution estimations for the species *G4-RNA* and a plot highlighting the difference between the average populations of *G4-RNA*.

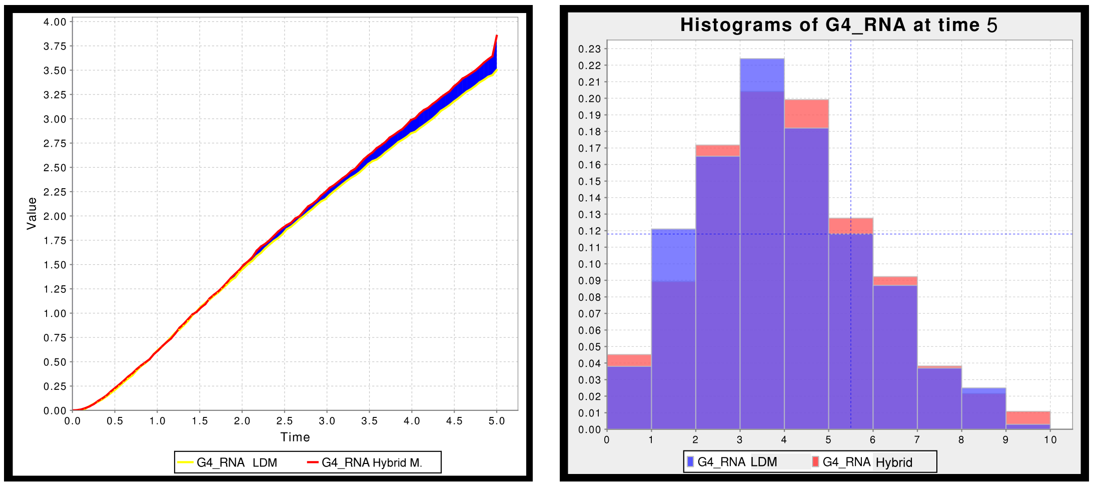


Figure 5.8: Differences between populations of *G4-RNA* measured at 5 seconds in the GAL model (Hybrid Method configured with four deterministic reactions)

The estimated density distance area for the species *G4-RNA* and the execution times are reported in the table 5.4. For all the species, we obtained a density distance area similar to the one obtained for *G4-RNA*. Thus, considering that all the distances are

Method	Run time (ms)	1000 runs time	Density distance area
LDM	1253.155ms	20min	0.10606
Hybrid Method	11.342ms	11 sec	

Table 5.4: Execution times and the estimated density distance area for *G4-RNA* in the GAL model (Hybrid Method configured with two deterministic reactions)

smaller than the average self distance bound in equation (5.1) no approximations were registered with this degree of accuracy. In conclusion, the hybrid method achieved comparable with those obtained with an exact stochastic simulation but almost one hundred times faster.

5.2.2 Simulating two deterministic reactions

In the previous experiment, because of the high computational load of the simulations, only one thousand realisations were considered to estimate the distributions of the species. However, using this number of realisations the self distance bound used to evaluate the promising performance results of the hybrid method was very large. For this reason, we repeated a similar experiment configuring the hybrid method to simulate only the two fastest reactions **A** and **B** (presented in figure 5.7) in a deterministic way. With this experiment we measured a smaller density distance area suggesting that the real self distance may be smaller. Indeed, as the number of deterministic reaction decreases, we expect the approximation introduced by the hybrid method to reduce and the simulation time to increase. The time point of observation, the number of intervals and the number of realisations were kept unchanged. Therefore, the estimated self distance was still the one presented in equation (5.1). Graphical outputs relative to the *G4-RNA* are reported in figure 5.9 while the density distance area and the execution times are presented in the table 5.5.

Method	Run time (ms)	1000 runs time	Density distance area
LDM	1271.608	21min	0.044998
Hybrid Method	174.09	3min	

Table 5.5: Execution times and the estimated density distance area for *G4-RNA* in the GAL model (Hybrid Method configured with two deterministic reactions)

For all the species, results were in line with the expected performance. Density difference area values still lie inside the estimated self distance and are reduced by

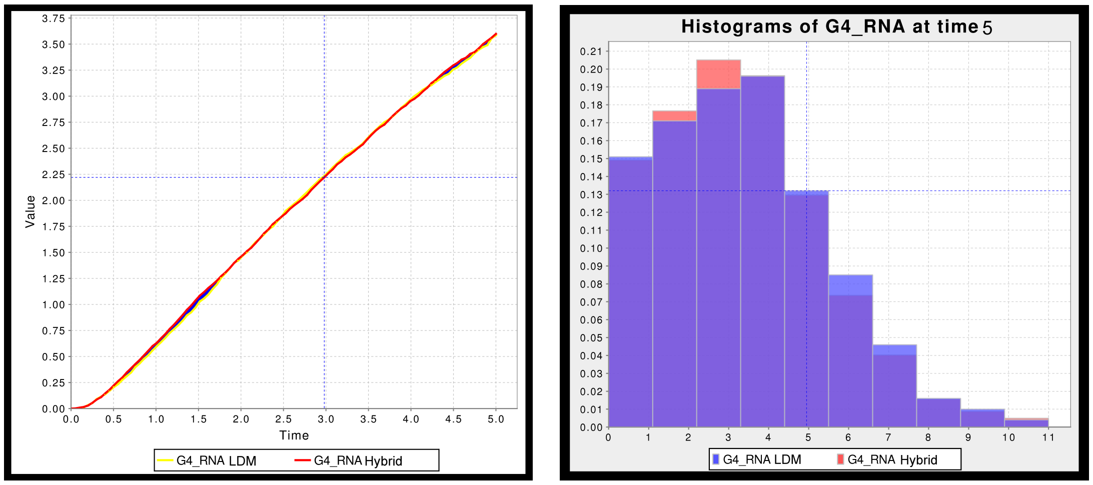


Figure 5.9: Differences between populations of *G4-RNA* measured at 5 seconds in the GAL model (Hybrid Method configured with two deterministic reactions)

almost one order of magnitude. The execution time of the hybrid method increases by a factor of ten but the method remains more than seven times faster than a full stochastic exact simulation. In conclusion, the hybrid method has shown, in both the experiments a very good performance conforming with precedent results [1, 32].

Chapter 6

Conclusions and future work

This project focused on the extension of Dizzy, an existing chemical stochastic simulator, with three new efficient formulations of the Gillespie stochastic simulation algorithm and a hybrid deterministic and stochastic simulator. Features for the performance analysis of simulators were developed in order to compare the computational performance and the efficiency of the algorithms. The aim of the project was to develop a software framework that would be useful to investigate and compare different simulators over various biological models. To the best of our knowledge, none of the existing simulation software includes a wide range of simulators and tools for the computational performance analysis and error measurement of approximate and exact stochastic simulators. The need for a tool to compare the performance of different simulation algorithms is motivated by the high computational load of stochastic simulations which made the simulation of many biological systems prohibitive without the use of parallel computing. Stochastic simulation algorithms are currently the only way to obtain exact simulations of biological models. For this reason one is usually concerned about finding the best trade-off between performance and simulation exactness with a particular model.

In the first part of this project, we showed how the new features of Dizzy can help in evaluating the scalability of different exact stochastic simulation algorithms as the number of reactions increases. In particular, we compared the performance of the different formulations of the Stochastic Simulation Algorithm over three biological models with a different scale of reactions. The comparison was made by considering the average number of reactions fired per second and the average search depth to find the next reaction. Our results highlight that the newly developed LDM, SDM and

ODM have generally a better performance than the NRM and the DM. Our analysis shows that in most cases the ODM is the best formulation of the SSA while the NRM, not considering the DM, is the simulator with the poorest performance. The average search depth in comparison with number of reactions per second shows that the computational overhead that is introduced to improve the next reaction location can play an important role on the whole simulation performance. Indeed, even if the NRM of Gibson Buck has the best data structure to find the next reaction to fire, the time that it spends to maintain this data structure noticeably increases the execution time, in particular when middle sized examples, like the GAL model, are considered. For the same reason the ODM which does not introduce any additional computational load during the simulation often gives the best results. Using our analysis we also confirmed the general good performance of the SDM, as presented in [34], and the reliability of the LDM as introduced in [16].

However, we found that with big models, like the Shoeberl model, or with models of very *stiff* systems, where a small subset of reactions are fired very frequently, even efficient formulations of the SSA were too slow to be run in a reasonable time using a typical desktop computer. In this case, approximate methods that sacrifice the exactness of stochastic simulation algorithm in favor of performance should be used. We developed an approximate hybrid deterministic and stochastic simulator that accelerates the simulation by modelling some of the fast reactions in a deterministic manner. In order to evaluate this newly developed algorithm we implemented a tool that can be used to estimate the error introduced by an approximate method with respect to results obtained from an exact stochastic simulation. This new part of the software, based on comparison between estimated probability density functions of the species populations, offers both a graphical and a numerical way to compare performance of approximate simulators with performance of exact simulators. These new features were used to test the hybrid method on the GAL model with an high initial concentration of galactose that makes four of the reactions fire very frequently. The hybrid method was firstly configured to model all the four reactions deterministically and subsequently to model only the two fastest reactions in a deterministic way. In both cases simulations gave very good results that follow the population distributions computed by exact simulation algorithms and that can be obtained saving, in the best case, more than the 90% of the execution time.

6.0.3 Hybrid Method numerical stability problems

Our implementation of the hybrid method relies on the use of existing stochastic and deterministic simulators currently available in Dizzy. We encountered a number of difficulties during the development of the hybrid method due to problems in implementing the synchronisation between the stochastic and the deterministic regime while keeping the deterministic ODE-solver numerically stable. Indeed, when a large number of concentrations and large reaction rates are considered, as in very stiff systems, ODE-solvers suffer from numerical stability problems in the phase of error estimation used to adapt the integration step. Our first implemention of the hybrid method was using an ODE-solver originally included in Dizzy with an adaptive step integration algorithm based on the Runge Kutta method. A first evaluation of the hybrid simulator showed numerous numerical accuracy problems and general poor performance, not in line with the good results presented in [1, 32], that were depending on the not optimal implementation of the integration algorithm included in Dizzy. For this reason, we opted for the more reliable and powerful external ODEtoJava library that provides several integration algorithms. The second version of the hybrid method used an adaptive step Runge Kutta integration algorithm offered by the external ODEtoJava library. This modification allowed us to avoid most of the numerical stability problems affecting the previous version but was not increasing the overall computational performance because the adaptive step was set to be too small during simulation. We resolved this problem by modifying the ODEtoJava integrator to integrate both with an adaptive and a constant integration step. More details about this modification are presented in appendix B.

6.0.4 Future work

The most difficult part in the hybrid method implementation was the synchronisation between the deterministic simulator and the stochastic simulator. Accuracy and simulation performance strongly depend on this operation and to improve it means to considerably improve the reliability of the hybrid simulator. A natural evolution of the current project is to develop a hybrid method, as presented in [1], that synchronises the two regimes considering the evolution of the propensities of the slow reactions during the deterministic simulation. This method requires the use of an integration algorithm able to stop upon a control event on the slow reaction propensities, a feature that is usually not supported in already existing ODE-solvers. So the development of this new Hybrid Algorithm would require a further modification of the ODEtoJava integration algorithm.

Owing to time restriction, only a relatively small number of performance analyses were made in this project. Using the newly developed framework for the performance comparison of different simulators, several other analysis can be made in the future. Within this context, however, a big limitation is the incompatibility of Dizzy with the SBML level 2 standard [2] which is currently used by most models available on the Internet. Therefore, performing analysis on a wide range of biological models will require the extension of Dizzy in order to read SBML level 2 files. This will allow us to investigate more about the performance of the newly developed algorithms.

6.0.5 Final observations

The extended version of Dizzy is a first example of software framework for the performance comparison of different stochastic simulation algorithms. It allows the user to test the appropriateness and the efficiency of the simulators over different biological systems by using different performance metrics. In conclusion, we believe that this software can give a valuable contribution to the ongoing analysis on computational performance and optimisations of stochastic simulation algorithms.

Appendix A

Glossary

CME - Chemical Master Equation

DM - Direct Method

FRM - First Reaction Method

LDM - Logarithmic Direct Method

MSSA - Multiscale Stochastic Simulation Algorithm

NRM - Next Reaction Method

ODM - Optimized Direct Method

ODE - Ordinary differential equation

SDM - Sorting Direct Method

RRE - Reaction Rate equation

SSA - Stochastic Simulation Algorithm

Appendix B

ODEtoJava modifications

The Hybrid Method development required the modification of the class `ErkTriple` provided by the ODEtoJava library. This class implements a Runge Kutta integration algorithm with a dynamic adaptation of the integration step. Our modification was focused on two aspects:

- Conformation of the software to an iterative-based integration
- Implementation of a method for a constant step integration

The first modification was required in order to allow the hybrid algorithm to adaptively perform an iteration with the deterministic simulator or with the stochastic simulator. The second modification was motivated by an excessive reduction of the integration step registered when the algorithm was used to integrate up to the next stochastic event time. Since this operation is performed only when the time of the next stochastic event is less than the next predicted optimal integration time, this problem was solved, introducing a tolerable error, by integrating with a constant step up to the next stochastic event time. The constant step used corresponds to the initial integration step specified by the user.

The modification resulted in a new class called `ErkTripleMod`. This class can iterate both by using a constant integration step and by choosing an optimal integration step according to an error estimation.

Bibliography

[1] Alfonsi A., Cances E., Turinici G., Di Ventura B., and W. Huisinga. Exact simulation of hybrid stochastic and deterministic models for biochemical systems. *INRIA Research report N 5435*, 2004.

[2] Finney A. and Hucka M. Systems biology markup language: Level 2 and beyond. *Biochem. Soc. Trans. Volume 31: Pages 1472-1473*, 2003.

[3] Ralston A. and Rabinowitz P. *A First Course in Numerical Analysis 2d ed.* New York: McGraw-Hill, P. 1978.

[4] Schoeberl B., Eichler-Jonsson C., Gilles E.D., and Müller G. Computational modeling of the dynamics of the MAP kinase cascade activated by surface and internalized EGF receptors. *Nat Biotechnol*, 20(4):370–375, April 2002.

[5] Adalsteinsson D., McMillen D., and Elston T.C. Biochemical Network Stochastic Simulator (BioNetS): software for stochastic modeling of biochemical networks. *BMC Bioinformatics*, 5:24, 2004.

[6] Endy D. and Brent R. Modelling cellular behavior. *Nature Volume 409(1), pp 391-395*, 2001.

[7] McQuarrie D. Stochastic approach to chemical kinetics. *Appl. Probab. 4:413-78*, 1967.

[8] Orrell D. and Bolouri H. Control of internal and external noise in genetic regulatory networks. *Journal of Theoretical Biology*, 230 No.3:301–312, 2004.

[9] Gillespie DT. A general method for numerical simulating the stochastic time evolution of coupled chemical reactions. *Comp. Phys., 22:403-434*, 1976.

[10] Gillespie DT. Exact stochastic simulation of coupled chemical reactions. *Phys. Chem., 81:2340-2361*, 1977.

[11] Gillespie DT. A rigorous derivation of the chemical master equation. *Physica A 188:404-25*, 1992.

[12] Gillespie DT. Approximate accelerated stochastic simulation of chemically reacting systems. *Phys. Chem. Volume 115, Issue 4, pp. 1716-1733*, 2001.

[13] Gillespie DT. Stochastic simulation of chemical kinetics. *Annu. Rev. Phys. Chem. A 58:35-55*, 2007.

[14] Oxtoby D.W. and Nachtrieb N.H. *Principles of Modern Chemistry 2nd Edition*. Saunders College Publishing, 1990.

[15] M. Tomita et al. E-cell: software environment for whole-cell simulation. *Bioinf. vol 15, pp 72-84*, 1999.

[16] Li H. and Petzold L. Logarithmic direct method for discrete stochastic simulation of chemically reacting systems. *Journal of Chemical Physics*, 2006.

[17] Salis H. and Kaznessis Y. Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. *J. Chem. Phys. Volume 122, 054103*, 2005.

[18] Segel I. H. *Biochemical Calculations: How to Solve Mathematical Problems in General Biochemistry, 2nd Edition*. John Wiley and Sons, 1971.

[19] McCollum J.M., Peterson G.D., Cox C.D., Simpson M.L., and Samatova N.F. The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *Computational Biology and Chem. Volume 30, Pages 39-49*, 2001.

[20] Takahashi K., Kaizu K., Hu B., and Tomita M. A multi-algorithm, multi-timescale method for cell simulation. *Bioinformatics*, 20(4):538–546, March 2004.

[21] Brenan K.E., Campbell S.L., and Petzold L. *Numerical Solution of Initial-Value Problems in Different-Algebra Equations second ed.* SIAM, Philadelphia, PA, 1996.

[22] Calder M., Duguid A., Gilmore S., and J. Hillston. Stronger computational modelling of signalling pathways using both continuous and discrete-state methods.

CMSB (Priami, C. Ed.), volume 4210 of Lecture Notes in Computer Science, 2006.

[23] Johnston M. A model fungal gene regulatory mechanism: the gal genes of saccharomyces cerevisiae. *Microbiol. Rev. 51 458-476*, 1987.

[24] Rathinam M., Petzold L., Cao Y., and Gillespie DT. Stiffness in stochastic chemically reacting systems: the implicit tau-leaping method. *Phys. Chem. Volume 119, pp. 12784-12794*, 2003.

[25] Gilbson MA. and Bruck J. Exact stochastic simulation of chemical systems with many species and many channels. *Phys. Chem. A 105:1876-89*, 2000.

[26] Neogi N.A. Dynamic partitioning of large discrete event biological systems for hybrid simulation and analysis. *LNCS Volume 2993, pp 463-476*, 2004.

[27] Atauri P, Orrell D., Ramsey S., and Bolouri H. Evolution of design principles in biochemical networks. *IEE Journal of Systems Biology*, 1:28–40, 2004.

[28] The JFreeChart project. http://www.jfree.org/jfreechart/index.html.

[29] Hoops S., Sahle S., Gauges R., Lee C., Pahle J., Simus N., Singhal M., Xu L., Mendes P., and Kummer U. COPASI—a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074, 2006.

[30] Ramsey S. Dizzy user manual. *CompBio Group, Institute for SystemsBiology*, 2006. http://magnet.systemsbiology.net/software/Dizzy/.

[31] Ramsey S., Orrell D., and Bolouri H. Dizzy: stochastic simulation of large-scale genetic regulatory networks. *Bioinform Comput Biol. Apr;3(2):415-36.*, 2005.

[32] Kiehl T.R., Mattheyses R.M., and Simmons M.K. Hybrid simulation of cellular behavior. *Bioinformatics*, 20(3):316–322, 2004.

[33] Kummer U., Krajnc B., Pahle J., Green A.K., Dixon C.J., and M. Marhl. Transition from stochastic to deterministic behavior in calcium oscillations. *Biophysical Journal Volume 89 pp 1603-1611*, 2005.

[34] Cao Y., Li H., and Petzold L. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *Journal of Chemical Physics. 121 (9), pp. 4059-4067*, 2004.

[35] Cao Y., Li H. Hall A., Lampoudi S., and Petzold L. User's guide for stochkit. *Department of Computer Science, University of California*, 2007. http://www.engineering.ucsb.edu/ cse/StochKit/.

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

[37] Cao Y., Petzold L., and Gillespie DT. Multiscale simulation algorithm with stochastic partial equilibrium assumption for chemically reacting systems. *Phys. Chem. Volume 206, pp. 395-411,* 2005.

[38] Cao Y., Petzold L., and Gillespie DT. Efficient stepsize selection for the tau-leaping simulation method. *Phys. Chem. Volume 124, 044109,* 2006.