Network Modeling and Simulation

Ilaria Cherchi
Telegram: @ilariacherchi
Giacomo Fantoni
telegram: @GiacomoFantoni
Elisa Pettinà
Telegram: @elispettina
Github: https://github.com/giacThePhantom/network-modelling-and-simulation

November 9, 2022

Contents

Chapter 1

Introduction

1.1 General introduction

To cope with the inherent multi-physics and multi-scale natures of biochemical reactions, different levels of simulation detail have been adopted to investigate their dynamical behavior:

- **molecular dynamics** (MD): microscopic level - keeps track of the structures, positions, velocities as well as possible collisions of all molecules in the system. The MD simulation requires a very detailed knowledge of molecules in both time and space and a lot of computational power
- **Brownian dynamics** (BD): focuses on the dynamics of each individual species, but skips the molecular structure information. The movement of a species is described as a random walk (or Brownian walk) among point-like structures. The time scale of BD simulation is greatly improved over MD, but it is still limited when dealing with large models.
- **deterministic simulation**: highest coarse-grained approach which focuses on the macroscopic behavior of biochemical reactions. Molecular species in the deterministic simulation approach are represented by their concentrations. The rate of change in the concentration of each species due to a reaction is directly proportional to the concentrations of species involved in the reaction. The time evolution of a biochemical reaction network is described by a set of ordinary differential equations (ODEs). The deterministic simulation is fast; however, its underlying assumption inherently oversimplifies biological reactions in which populations of molecular species are continuous variables and their changes due to single reaction firings are assumed to be negligible. The correctness of deterministic simulation is severely affected when stochasticity plays an important role in the dynamical behavior of biochemical reactions.
- **stochastic simulation:** a mesoscopic approach to provide a probabilistic description of the time evolution of biochemical reactions. It keeps track of a discrete count for the population, but abstracts all the detailed position and velocity information, of each species. Each reaction in the network is assigned a non-negative chance to fire and to drive the system to a new state. The probability that a reaction occurs in a time interval is derived from the reaction kinetics. Each stochastic simulation step will select a reaction to fire according to its probability.

Although the stochastic simulation is faster than the MD/BD approach, it is often computationally demanding for simulating large biological systems. First, biochemical reactions, due to their multiscale nature, are separated by different time scales in which some fast reactions will occur at rates greater than other reactions. The fast reactions occur frequently and drive the system very fast into a stable state. The dynamical behavior of biochemical reactions, after the short fluctuation

time at the beginning, will be determined by the dynamics of the slow reactions; however, most of the time the simulation samples the fast reactions to realize the dynamics which is not the expected behavior. Second, the population of some species involved in reactions may be larger than others by many orders of magnitude. The fluctuations of these species, when involving reactions fire, are less significant. Keeping track of large population species is obviously less efficient since a coarse-grained simulation method can be applied without loss of total simulation accuracy. Because of the inherent dynamics in biochemical reactions, a model can combine and mix all of these aspects in a very complicated manner. Third, due to the stochastic behavior in a single simulation, many simulation runs must be performed to ensure a statistical accuracy and this requires a high computational effort. These issues raise a computational challenge for developing and implementing efficient stochastic simulation methods.

1.2 Network modeling

1.2.0.1 Logic models

Recap from previous theory lecture:

What is the main issues in using logic modeling with multiple levels? The update formulae need to be defined for each level, tricky extension procedure.

1.2.0.2 Petri nets

Petri nets are specific networks introduced in 1960s, with the idea to describe *communication processes* (computer science field). We have two kinds of nodes:

- ***places*** : container of entities
  - *tokens* : entities
- ***transitions*** : possibility to move one or more tokens to other places

In order to model a chemical reaction, we can associate places to variables, transitions to chemical transformations and tokens to molecules.

PetriNets_SimaoEtAl.pdf

In the example we have numbers inserted in places. The network will evolve according to the transitions applied. A transition can be *enabled* or not to fire. E.g. there is one token in p_1, so we know that t_1 and t_2 are enabled. Instead, t_4 cannot be enabled, as 3 tokens are required but only one is present. t_2 is taking the token from p_1 and creating 2 tokens in p_2 and p_3; this allows to fire t_4, since now p_3 has the required number of tokens.

(a) shows the initial marking before firing the enable transition t; (b) shows the marking after transition labeled reaction 1 fires. Places: hydrogen, oxygen and water. We can represent the stoichiometry of the reaction through the numbers on the edges and the numbers to the tokens.

Review_ModelingComplexBiologicalSystems.pdf

Pros: we have no constraint on the data type, not strictly boolean values. They allow to extend the number of items which can be associated to a model.

Cons: there is no fixed rule for applying transitions. Furthermore, we are not encoding the reaction's complexity (all transitions are equally probable, but it is possible to weight the edges).

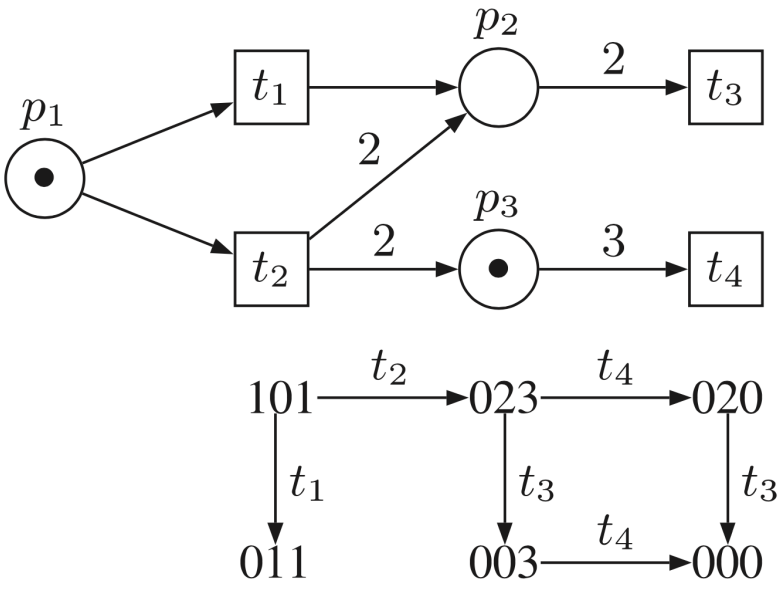


Figure 1.1: PetriNets_SimaoEtAl.pdf

Having a dynamics based on the integers can be quite useful, as if we consider single chemical events we are working with discrete data. The exact stochastic algorithm works with integers. In differential equations instead we need real numbers. Also the discretization of the time step might not be a limitation, since time can be discretized in reality. A huge approximation of the system is performed on time → we do not have any clue of how much time is passing from one step to the other. We are assuming that all the reactions take the same (unknown) amount of time.

The main limitation of network models is the **approximation of time**, we have no clue on the time required for the reaction.

Basic elements of a hybrid Petri net

By upgrading the notation we can achieve more accurate representations:

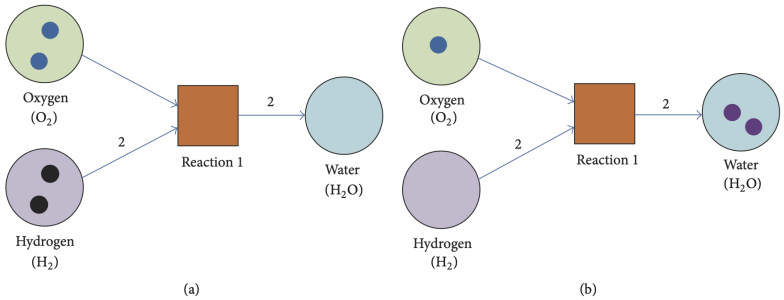


Figure 1.2: Review_ModelingComplexBiologicalSystems.pdf

- add inhibitory and test transitions
- differentiate between discrete and continuous transitions

1.2.1 Rewriting systems

1.2.1.1 P systems

Popular rewriting systems are ***P systems***, which are also called membrane systems. They are computational environment inspired to the structure and membranes. In particular, they define a hierarchy of membranes partitioning the space in different areas - similarly to a cell. In each regions we can allocate entities and apply transformation rules. The rewriting rules change the value of each letter. The pedix $_{in4}$ gives more details on the reaction.

This kind of systems tend to use *non-determinism*, they try to explore the full set of possibilities.

1.2.1.2 MP systems

The difference with standard P system is the association of functions to each reaction. In this way we can model the complexity of the reaction, since in the model we apply all possible reaction, which will produce an amount given by the function.

1.2.2 Equation-based approach

1.2.2.1 ODE systems

Example: mass-action model

$A + E \underset{k_2}{\overset{k_1}{\rightleftharpoons}} A \mid E \xrightarrow{k_3} B + E$

1.2.3 Simulation algorithms

For simulating we need a specification of stoichiometric matrix, a vector of integers (initial values) and stochastic rate. We will arrive to a formula with which we can compute the probability. ! reaction probability function is required for defining probability.

Exact simulation algorithms are computationally intensive, but provide the most accurate solution. Stepwise, we will try to define faster strategies with the aim of compromising accurate dynamics and feasible solutions. It is also possible to rely on a mixture of technologies to focus on different results.

If the well-mixed assumption is not fulfilled:

- partition the compartment in sub-compartments → approximation
- use more sophisticated algorithms

The *stoichiometric matrix* tells us how the system evolves if one of the two functions is applied, but it is not enough for computing a simulation. There can be many reactions that arrive to the same definition of stoichiometric matrix.

If we want to compute a dynamics we need to develop a series of states; at each step we require two ingredients:

- τ: tells how much later the system will evolve to another state
- μ: choose the reaction by considering the probability of execution of each reaction at the step

Reaction propensity: function needed for the derivation of probability. The higher the propensity, the higher will be the strength of the reaction. Naturally, we will have a higher probability when a higher propensity is observed, but the two quantities are something different.

The propensity is a property of the reaction, probability is a property of a reaction in the system (we need to take into account also other reactions)

Instead of performing an in depth analysis of probability, we will choose a stochastic approach. It is only necessary to compute one evolution of the system per time.

Chapter 2

Stochastic Chemical Kinetics

2.1 Rewriting biochemical reactions

Biochemical reactions are the building blocks to model biological systems. They provide a unifying notation with sufficient level of details to represent complex biological processes. Biochemical reactions decorated with reaction kinetics can be simulated by a simulation algorithm to generate a realization of their dynamics.

Chemical species in a biological system move around and gain kinetic energy. Upon collisions with other species, they undergo reactions to modify and transform into different species. In order to make this concrete, consider the transformation of a molecule of species A to a molecule of species B. It is written schematically as $A \rightarrow B$. This reaction converts one A molecule on the left side of the arrow to a B molecule on the right side. Such a transforming reaction is called a *unimolecular reaction*.

We are dealing with a structure of compartments, in which we have symbols representing chemicals and *rewriting rules* specifying the direction.

Membrane example: AABC A→B , A→C

All these computational systems are more oriented on representation, while if we are more interested in the function we can apply **equation-based methods**, for instance ODE systems. In ODE we specify the derivative in time of any of the variables in terms of a function of the state.

The stochastic simulation approach can be approximated by a deterministic one.

The main steps to follow for writing a formal mathematical description of a biological system are**:**

1. Choose representation: arrow notation
2. State of the system: integer
3. Specify the likelihood of execution for each reaction

A reaction will be faster when we have a huge amount of reactants. Why do we state so? If we think in terms of probability, an example reaction A + B → C will be faster when a there is a high quantity of A and B are present, since it is more likely for them to interact. Such statement is based on the assumption that there are no gradients in the system, i.e. well-mixed chemical system, therefore the distribution of the molecules is uniform.

Well-mixed reaction volume: reaction volume in which all the molecular species are homogeneously distributed and spatially indistinguishable.

Chemical species under the well-mixed assumption at a thermal equilibrium are uniformly distributed in the volume V and their velocities are thermally randomized according to the Maxwell-Boltzmann distribution.

In order to approximate a complex system, we can partition the main compartment in smaller sets where we can apply this assumption - *discretization* procedure.

In terms of stochastic representation we require integers e.g. probability that two molecules will interact. Therefore, Petri nets are a suitable representation for this kind of systems.

Formal representation:

The state of a spatially homogeneous biological system is determined by the population of each species, while the position and velocity of each individual molecule are ignored. Let $X_i(t)$ be the population of species S_i at a particular time t. The N- vector $X(t) = (X_1(t), ..., X_N(t))$, which determines the population of each species, constitutes the system state at the time t.

A general reaction Rj has a general scheme:

$v_{j1}^- S_1 + ... + v_{jN}^- S_N \rightarrow v_{j1}^+ S_1 + ... + v_{jN}^- S_N$

In which a species on the left side of the arrow is called a *reactant*, while a species on the right side is called a *product*. The non-negative integers v_{ji}^- and v_{ji}^+ are the stoichiometric coefficients which denote the number of molecules of a reactant that are consumed and the number of molecules of a product that are produced.

For each reaction R_j, the net change in the population of species S_i involved in the reaction is equal to $(v_{ji}^+ - v_{ji}^-)$, which can be positive, negative or zero. The net changes by all reactions are described by a stoichiometry matrix $\mathbf{v}$ with size MN . The jth row $\mathbf{v}_j$ of the stoichiometry matrix expresses the changes caused by reaction R_j and it is called the state change vector.

We can have multiple systems leading to the same stoichiometric matrix.

Stoichiometric matrix:

We require two matrices: V^+ provides the products, V^- provides the reactants.

$$V^- = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, V^+ = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 2 \\ 2 & 0 & 0 \end{bmatrix}, V = \begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{bmatrix} \quad V = V^+ + V^-$$

Suppose that at a time t the state (number of molecules in that given moment) is $X(t)$. it further assumes that the next reaction scheduled to fire at the next time $t+\tau$ is R_μ, which moves the system accordingly to a new state $X(t+\tau)$. $\mathbf{x}$ is a simple notation to represent $X(t)$.

$$X(t+\tau) = X(t) + v_\mu$$

Summary:

If we want to specify in computational terms the biological system that we are interested into, we end up with the structure of at least one compartment where we assume to have a chemical volume in which there are some entities that interact with each other. We impose a preliminary assumption, the well-mixed volume, that means that all these actors are available with equal availability in all the parts of the volume. For each compartment, we need to specify the entities (and how many molecules are available), and the reactions that provide the rules for transforming the chemicals in others along the time. All these structures of reactions can be defined in mathematical terms using matrices: the number of columns is equal to the number of variables; the number of rows is equal to the number of reactions; the stoichiometric coefficient of reactant or products.

2.2 Reaction propensity

Each reaction in the stochastic chemical kinetics is considered as a stochastic process where each of its occurrences is a random event with an assigned probability distribution. It is impossible to predict the progress of reactions deterministically, but only stochastically with a probability. The propensity of a reaction is a formula that is computed on a state of a system. It is the way that we will use to hint the probability of execution of the reaction.

The propensity a_j of a reaction R_j is defined such that $a_j(x)dt$ = probability that a reaction R_j fires in the next infinitesimal time interval $[t,t+dt)$, given the state $X(t) = \mathbf{x}$ at time t.

In a chemical setting, the probability of execution of one reaction will be proportional to the viability of the reactant. The mass action propensity is the expression of the direct proportionality: multiply the rate of the reaction by the counts of the number of distinct combinations of reactants.

The propensity of a reaction is an intrinsic property of the reaction, is linked to the phenomenon that the reaction is going to represent.

The probability will depend on the propensity of the reaction and the other reactions that are competing for the same reactant. The propensity is not affected by the products.

$a_1(0) = c_1 h_1(x(0))$

$h_2(x(0)) = \binom{100}{1}\binom{50}{1}\binom{30}{0}$ $x(0+\tau) = x(0) + V_1 = \begin{bmatrix}99 & 51 & 30\end{bmatrix}$

$x(0) = \begin{bmatrix}100 & 50 & 30\end{bmatrix}$

Here we are computing the combination, therefore we will not take into account a^2 as in the canonical law of mass action.

Reaction propensity for reactions R_j with mass action kinetics

- Synthesis reaction ($\emptyset \rightarrow$ products): the number of combinations $h_j = 1$ and propensity $a_j = c_j$
- Unimolecular reaction ($S_i \rightarrow$ products): the number of combinations $h\_j = X\_ i$ and propensity $a_j = c_j X_i$.
- Bimolecular reaction ($S_i + S_k \rightarrow$ products): the number of combinations $h_j = X_i X_k$ and propensity $a_j = c_j X_i X_k$.
- Dimerization reaction ($2S_i \rightarrow$ products): the number of combinations $h_j = \frac{1}{2} X_i(X_i - 1)$ and propensity $a_j = \frac{1}{2} c_j X_i(X_i - 1)$.
- Polymerization reaction ($3S_i \rightarrow$ products): the number of combinations $h_j = 16X_i(X_i - 1)(X_i - 2)$ and propensity $a_j = 16 c_j X_i(X_i - 1)(X_i - 2)$.
- Termolecular reaction ($2S_i + S_k \rightarrow$ products): the number of combinations $h_j = \frac{1}{2} X_i(X_i - 1)X_k$ and propensity $a_j = \frac{1}{2} c_j X_i(X_i - 1)X_k$.

For simulating we need a specification of stoichiometric matrix, a vector of integers (initial values) and stochastic rate. We will arrive to a formula with which we can compute the probability. The reaction probability function is required for defining probability.

sectionChemical Master Equation

The CME is the theoretical approach allowing to derive the complete set of probability of all possible states of the system

2.2.1 Grand probability function

$\mathbb{P}\{\mathbf{x}, t|\mathbf{x}_0, t_o\}$ = probability that the system state is $X(t) = \mathbf{x}$ at time t, given the initial state $X(t_0) = \mathbf{x}_0$ at time $t0$.

By applying the chemical master equation, we end up with a set of differential equations for this probability, which theoretically provide the complete set of any of the state of the system.

Algorithm 1 Stochastic Simulation Algorithm (SSA) - General Sketch

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1,\ldots,M$, is accompanied with the state change vector $\mathbf{v}_j$ and the propensity a_j, the initial state $\mathbf{x}_0$ at time 0 and the simulation ending time T_{max}

Output: a trajectory of the biochemical reaction network which is a collection of states $X(t)$ for time $0 \leq t \leq T_{max}$.

1: initialize time $t = 0$ and state $X = \mathbf{x}_0$
2: **while** $(t < T_{max})$ **do**
3: set $a_0 = 0$
4: **for all** (reaction R_j) **do**
5: compute a_j
6: update $a_0 = a_0 + a_j$
7: **end for**
8: sample reaction R_μ and firing time τ from pdf $p(\tau,\mu|\mathbf{x},t)$ in Eq. (2.16)
9: update state $X = X + \mathbf{v}_\mu$
10: set $t = t + \tau$
11: **end while**

Figure 2.1: SSA

2.2.2 CME

$$\frac{\mathbb{P}\{\mathbf{x},t|\mathbf{x}_0,t_o\}}{dt} = \sum_{j=1}^{M}(a_j(\mathbf{x}-\mathbf{v}_j)\mathbb{P}\{\mathbf{x},t|\mathbf{x}_0,t_o\}) - \mathbb{P}\{\mathbf{x},t|\mathbf{x}_0,t_o\}\sum_{j=1}^{M}a_j(\mathbf{x})$$

However, for computing all these probabilities in a complex system, the number of equations will increase and be practically useless. The idea is to avoid deriving everything with the chemical master equation but trying to apply another approach that is the one provided by the stochastic simulation.

2.3 Stochastic simulation

When you apply CME you explore all the possible state of the system, with SS you compute only one possible trajectory of the system (for any possibility you must run several time SS, too complex again). The stochastic simulation works because you only need an idea of the most probable conditions of the system. SS is faster than a real experiment and you can run it with a computer (you can also run thousands of simulations).

2.3.1 Probability density function

The mathematical basis of stochastic simulation is the reaction probability density function (pdf). $p(\tau,\mu|\mathbf{x},t)d\tau$ = probability that a reaction R_μ fires in the next infinitesimal time interval $[t+\tau, t+\tau+d\tau)$, given the state $X(t) = \mathbf{x}$ at time t. These probabilities can be computed by considering the idea of propensity. The propensity is a formula on the state of the system (depends on some of the variables that are in the system) that allows to provide a quantitative information that can be used to derive the probabilities. The propensity is not a probability: propensity is not in a range from 0 to 1 and it is a property of a reaction while the probability is a property of the system.

One of the crucial points that we must focus on is a way of being able of sampling from this pdf giving the fact that the computer has few ways of generating something that is stochastic. In particular, the easiest way is the random number generator. The following pseudocode implements this procedure:

The result of a SSA run is a trajectory, which shows the evolution of the biological system over time. The trajectory is a collection of states $X(t)$ that denotes the state of the system at any time $0 \leq t \geq Tmax$. It should be emphasized that because SSA is a discrete event simulation algorithm, the state changes only at discrete time instants when reactions fire. The state between two reaction firings is a constant.

Chapter 3

Implementation of the Stochastic Simulation Algorithms

3.1 Non-deterministic vs stochastic

Assumption: we are using the same model and parameters.

- Deterministic system: no randomness, we always obtain the same result.
- Non-deterministic system: there is some degree of uncertainty on different runs.
  - Exact stochastic simulation: satisfying some hypotheses, the system will behave just like a biological system. We are in a probabilistic setting. The fact that we are able to compute the probability function does not make this method deterministic, we have uncertainty. Theoretically, we could have no idea about reaction execution in a stochastic setting; in the case of exact stochastic simulation we reach a high level of accuracy thanks to the probability function.

Why do many computational environments employ a non-deterministic approach? Quite often we require to find the compromise between time and complexity.

Non-deterministic polynomial time algorithms: we do not have an efficient solution, but it seems possible to find it.

A non-deterministic setting allows us to understand whether an algorithm can be solved in polynomial time (stepwise guessing). Alan Turing realized that it was required to categorize algorithms with specific rules in order to compare them → Turing machine.

3.2 Exact stochastic simulation algorithms

The direct method of Gillespie defined a couple of formulae able to understand how the system will execute in terms of time (τ) and reactions (μ). Since we are reasoning in infinitesimal time, each reaction occurs and ends exactly at time τ, hence we cannot have multiple reactions firing simultaneously.

Fig 3.1 Marchetti's book

a_0 is the sum of all propensities in the system.

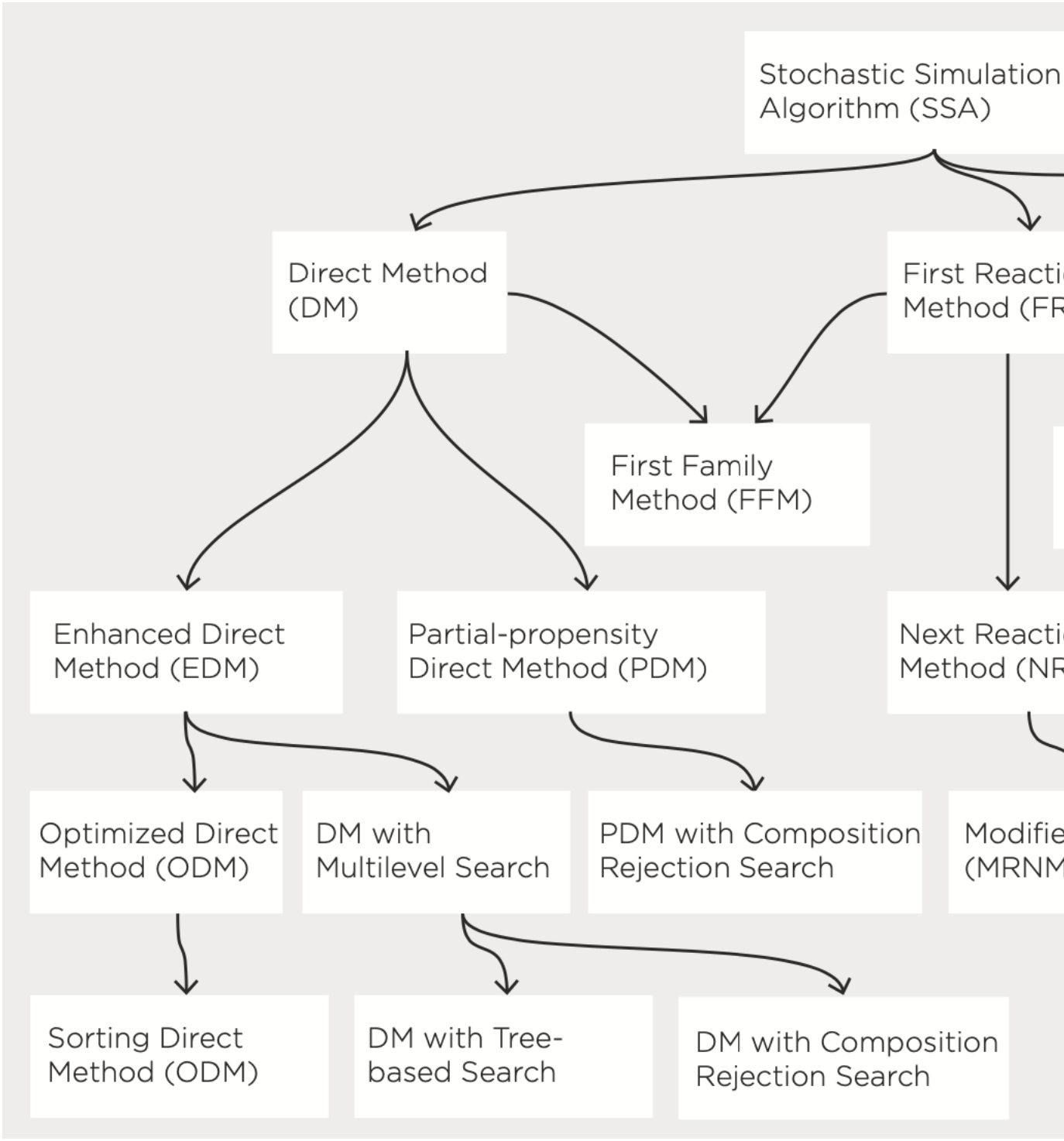


Figure 3.1: Fig 3.1 Marchetti's book

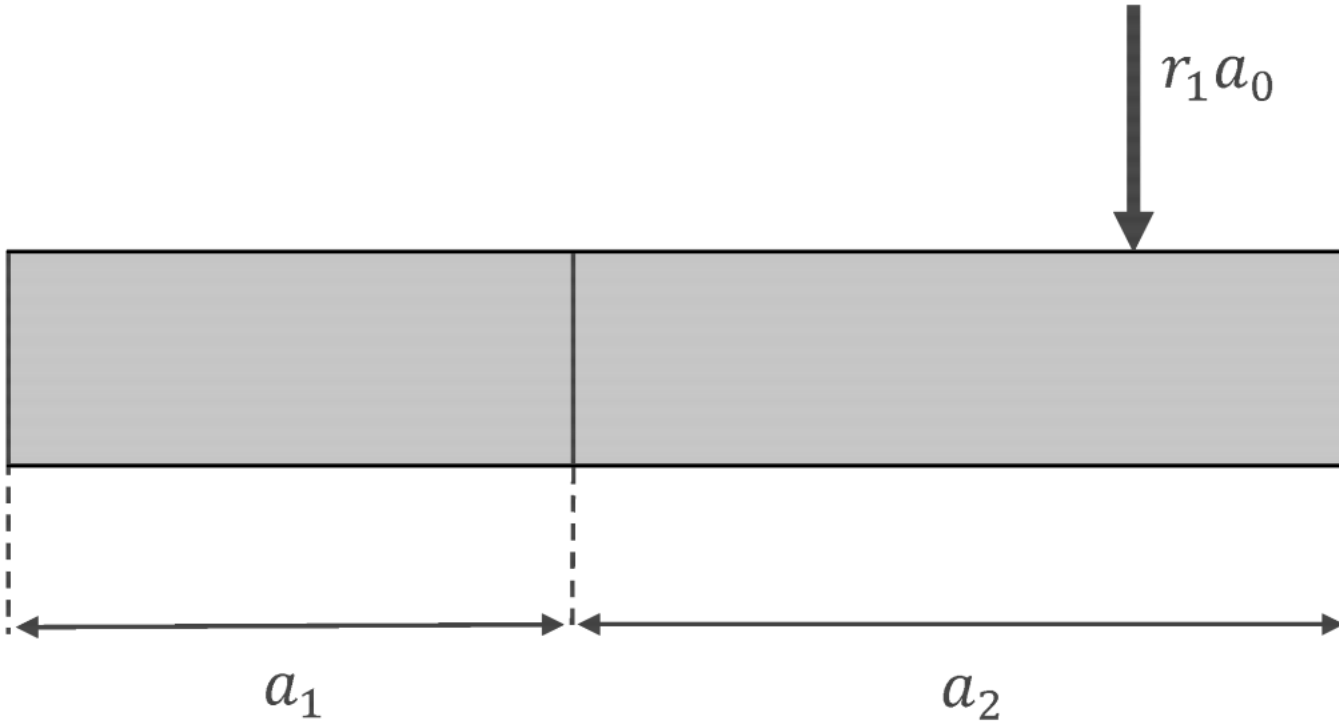


Figure 3.2: Fig 3.2

1. We sample one random number from the distribution $a_0 = \sum_{j=1}^{M} a_j \rightarrow V_1 = U(0,1)$
2. Scale it to the maximum $V_1 \cdot a_0 = U(0, a_0)$
3. See where this number will point over the different propensities
   Fig 3.2
4. Generate another random number $V_2 = U(0,1)$
5. $\tau \sim Exp(a_0)$
6. $\tau = \frac{1}{a_0} ln(\frac{1}{V_2})$

Algorithm 2 Direct Method (DM)

Input: a biochemical reaction network of M reac
accompanied with the state change vector $\mathbf{v}_j$
0 and the simulation ending time T_{max}

Output: a trajectory $X(t)$, $0 \leq t \leq T_{max}$, of the bi

1: initialize time $t = 0$ and state $X = \mathbf{x}_0$
2: **while** $(t < T_{max})$ **do**
3: set $a_0 = 0$
4: **for all** (reaction R_j) **do**
5: compute a_j
6: update $a_0 = a_0 + a_j$
7: **end for**
8: generate two random numbers $r_1, r_2 \sim \mathrm{U}($
9: select R_μ with the smallest index μ such t
10: compute $\tau = 1/a_0 \ln(1/r_2)$
11: update state $X = X + \mathbf{v}_\mu$
12: set $t = t + \tau$
13: **end while**

This algorithm can be improved by avoiding to recompute all the propensities, taking into account

the fact that many reaction will keep the same probabilities over time. We can work on the complexity of the for loop through a dependency graph.

3.2.1 Enhanced Direct Method

Our aim is to group reactions modifying at the same time in order to update propensities e.g. all reactions involving the same components. Let's specify this concept in a formal way:

For each reaction $*R_j*$ with $j = 1, ..., M$, define $Reactants(R_j) = S_i|S_i$ is a reactant of R_j, and

$Products(R_j) = S_i|S_i$ is a product of R_j.

A **reaction dependency graph** ...

We tend to have an improvement in time complexity when the graph is not highly connected, while the space complexity is higher (since we need to generate a data structure).

Improvements for direct method:

Sort: find the smallest index for which we can satisfy the condition for μ. On average, it is more efficient to keep reactions with highest probability at the beginning, such that we can easily match indexes to requirement. How can we compute the number of firings? We can set an oder beforehand based on previous knowledge or run a pre-simulation with a non-optimized algorithm (we still have a speed up since we are working with a smaller setting).

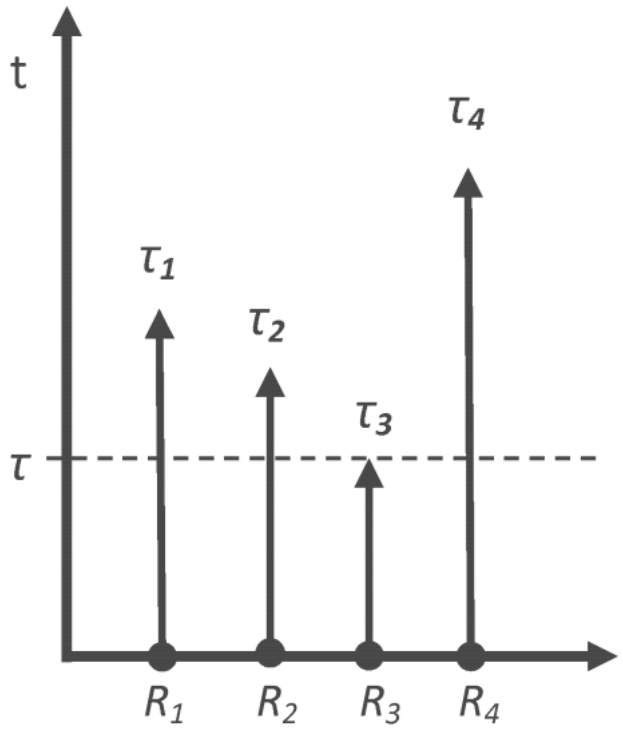
We cannot directly compute the propensity right away, as it heavily depends on the dynamics of the system, the state can change. It is required to compute some heuristics to sort the reaction order $\rightarrow$ standard Gillespie.

It is also possible to apply a ***multi-level search***. We group the reactions and select again a reaction in the group. As the dependency graph, here the issue is time complexity.

3.3 First Reaction Method (FRM)

Instead of computing one τ, compute a τ for each reaction. Example: $\tau_1 = Exp(a_1) = 1/a_1 ln(1/V_1)$. We are assuming that no other reactions are firing in the middle. We can generate M random numbers and end up with M τ. Then we choose the reaction with minimum τ, which will be the first selected one.

$\mu = R_\mu$ st $\tau_\mu = \min_j \tau_j$



Pro: sometimes the search is quicker, simpler to parallelize

Cons: we generate a lot of random numbers with respect to the direct method.

After the first step, we need to recompute. Even here, there is the possibility to improve the algorithm e.g. computation of the propensity.

3.4 First Family Method

Tries to combine the good features of the direct method with first reaction method. The idea is to reach an implementation which can be partially parallelizable: divide the reaction in "families" or groups. The idea is to divide the reaction in n groups $r_1, \dots r_n$ and families e.g. 3 families. We should associate a theoretical propensity to each group, i.e. the sum of the propensity in the family $a^1 = \sum_{j \in f_1} a_j$.

$FRM = \tau_1 = \frac{1}{e^1} ln(\frac{1}{r_1})$, we do not know which one of the reactions will be applied. In order to decide, we apply the direct reaction method formula for selecting the index. Given that we are working with a subset, we will scale over the sum, the a_0 of the group. It is true that we are generating random numbers, but over the number of the families → reduction. This time we have n families + 1 random number to select the family. Danger: if we have big disequilibrium in propensity, we might end up selecting always one of the families. We can parallelize by linking families to CPU.

Last time, we were introducing Next Reaction Method, which can be considered as an evolution, trying to apply the same reasoning as FRM in a more efficient way (by applying an efficient handling of random numbers). If we are able to do so we will need to compute less random numbers.

DM → $2 \cdot n_{steps}$

FRM → $M \cdot n_{steps}$

FFM $\rightarrow (n_{families} + 1)n_{steps}$
NRM $\rightarrow M + n_{steps}$

If we take a look at the difference among methods, it is not very clear which is the winner - even though NRM seems to be one of the most optimized, it is usually the most efficient in requiring less random numbers. Of course everything has a price, we need to add computations.

NRM is the most efficient in random number generation, while FRM is the one consuming the most.

3.5 Next Reaction Method

Why do we need to recompute time points? Two issues:

1. multiple reactions can be executed at the same time $\rightarrow$ the reaction propensities might change
2. we are computing τ over propensities, but the propensities were computed on initial reaction settings. E.g. $R_2 : A \rightarrow B$, $R_1 : A \rightarrow B$, the two reactions depend on each other.

τ is modelling the instant in which the reaction is assumed to fire, it is just an event. We can subtract the time for passing to following reactions, but we must also update the propensity.

$a_1 \rightarrow a_1^{new}$, we have three possibilities:

1. $a_1 = a_1^{new} R_2$ is not affecting the propensity, i.e. reactant is not modified by R_2, so R_1 remains unchanged.
2. $a_2^{new} > a_1$. R_2 has been applied, something has changed and the firing prob of R_1 is increasing. Therefore, τ_1 needs to be updated, we expect it to be smaller $\rightarrow$ rescaling through the ratio of the propensities e.g. $(\tau_1 - \tau_2) \cdot \frac{a_1}{a_1^{new}}$
3. $a_1^{new} < a_2$ higher τ_1

We should not think of another event, we are just updating the same event through the formula!

The algorithm was developed in the 2000s, there are a lot of improvements related to technology advancements. In addition to rescaling formula, there are other improvements to keep the complexity of the algorithm as low as possible:

- we avoid generating too many random numbers
- reaction dependency graph (introduced for Direct Method): recompute the propensity of the reaction only if it is needed
- searching the reaction: minimum is linear over the number of reactions. **Binary heap**: decrease to logarithmic scale, select winner in constant time.

Gibson 2000: original paper
Thanh 2014: latest algorithm

3.6 RSSA

The Rejection-based Simulation Algorithm seems to be an approximation, but everything is exact. It tries to reduce computation complexity, but instead of limiting the amount of random numbers, it focuses on another bottleneck: the computation of the reaction propensity. The computational problem becomes more complex if we do not rely on mass action.

The RSSA Algorithm therefore tries to reduce the number of times in which we compute the reaction propensity. We will need a placeholder and at the same time to keep the computation exact.

Instead of using the state of the system, we use a ***fluctuation interval*** (a bound). From the state of the system X we generate a lower and upper bound; to do so, we just use a parameter providing a percentage, which is called δ ($0 < \delta < 1$). Therefore, the bounds will be computed as:

- lower $X = X - X \cdot \delta$
- upper $X = X + X \cdot \delta$

Given that the events of the reaction are modifying a few molecules per time, we can assume that for a certain amount of steps we will remain inside the same range of variability. We are trying to obtain a propensity computed over bounds; by doing so, we can apply the same computation for a higher number of steps, as we only need to recompute the propensity outside of the fluctuation interval. We can also compute the upper bound for the sum of the propensities $\bar{a}_0$. a_1 will be in the middle, but we do not want to compute it.

It seems like working with these bounds does not lead to an exact simulation: we need to avoid as much as possible the exact computation of the propensities. Indeed, in the vast majority of cases we will have the possibility to choose the reaction without the real propensity.

We will perform some computations and, given the presence of bounds, we will do some mistakes; the algorithm will evaluate the results and eventually reject them (rejection-based algorithm). For doing so, the algorithm generates the events with the upper bound of the propensity $\bar{a}_0$. By using an upper bound τ will be lower, so we will generate more events with respect to the direct method. We apply a sort of Gillespie, we replace the standard propensity with the upper bound and choose the reaction as in the direct method.

How can we understand whether an event is real or fake? In the case in which $\bar{a}_0 = a_0$ we are in the standard Gillespie. We should apply a similar reasoning as NRM for scaling for checking the validity of candidate reactions. Let's imagine that we are selecting R_2: we should generate another random number, scale it over the bound and understand if we are in the exceeding (fake) or inside (real) area.

- If the random number is lower than the lower bound, we can accept it without computing the real propensity.
- If the random number is between the bounds we require the exact propensity for discriminating:
  - $\underline{a}_2 < u < a_2$ accept
  - $a_2 < u < \bar{a}_2$ reject

The higher the fluctuation interval, the higher the uncertainty but lower the complexity $\rightarrow$ find the right compromise with some heuristics on δ value.

Chapter 4

Approximation algorithms

Can we assume that in any condition the stochastic simulation bottleneck is a problem? It depends. If we are in a condition in which all species are present in low number and it takes a lot to cure, the stochastic simulation is not really a bottleneck. We might have an issue if we wish to simulate for a long time, multiplicative factors. At the moment, it is not feasible to compute exact a stochastic simulations to all the models we need to observe, in particular for huge number of species and reaction, therefore there is a huge research in developing alternative strategies that allow to compute the system in an *accurate* way (lose correctness) with lower computational cost and time.

Why should we decide to use approximations in simulations?

1. it might be the only possible and feasible solution to solve a problem.
2. reality is affected by error, so even when we are using exact stochastic simulation algorithms we should consider a small degree of approximation. A certain deal of error could aid in retrieving a more realistic result.

In chapter 3 all the algorithms were belonging to exact stochastic simulation setting, so the result is the same. Instead in this case, each algorithm approximates in a different way, assume different approximation $\rightarrow$ not a consistent set.

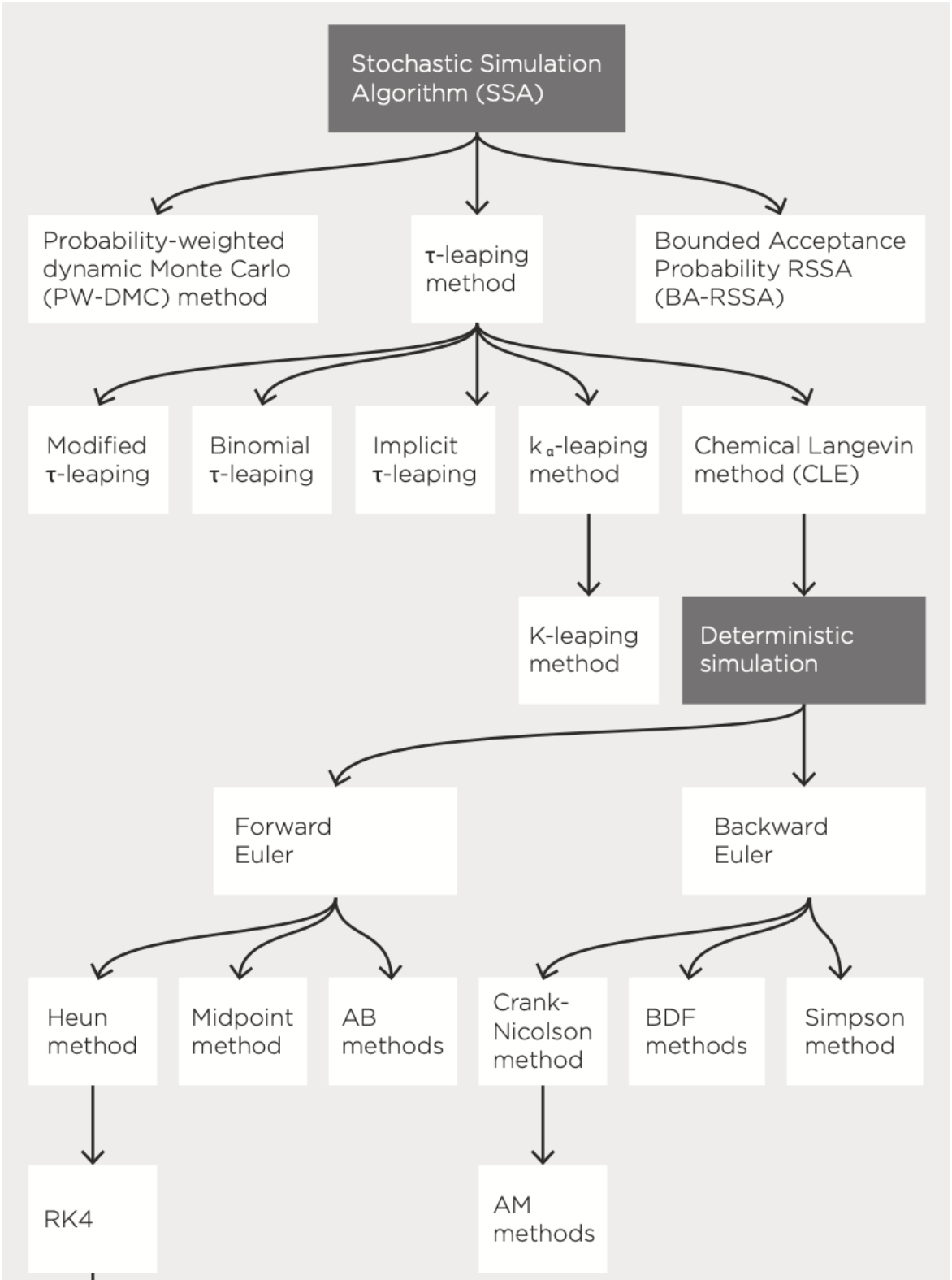
We should observe that of the three main computational strategies presented here, there is one which is much more popular with respect to other. The τ leaping method is well known and has been further developed, so we will be focusing on it.

Before introducing the algorithm, let's try to think about the possible bottlenecks which could prevent us from having a fast algorithm. We could improve:

- work on the number of reaction events, group them in such a way to reduce the number of reactions in the system. The τ leaping methodology is working in this direction
- improve the computation of the propensity. For instance, the probability-weighted Monte Carlo is particularly promising in situations in which we have huge differences in propensities among reactions.

4.1 Probability-Weighted Dynamic Monte Carlo Method

The probability-weighted dynamic Monte Carlo method (PW-DMC) is an approximation approach for improving computational efficiency of stochastic simulation of reaction networks where some


Stochastic Simulation Algorithm (SSA)
Probability-weighted dynamic Monte Carlo (PW-DMC) method
τ-leaping method
Bounded Acceptance Probability RSSA (BA-RSSA)
Modified τ-leaping
Binomial τ-leaping
Implicit τ-leaping
k α-leaping method
Chemical Langevin method (CLE)
K-leaping method
Deterministic simulation
Forward Euler
Backward Euler
Heun method
Midpoint method
AB methods
Crank-Nicolson method
BDF methods
Simpson method
RK4
AM methods
RK45

reactions have propensities significantly larger than other reactions. The principle of this algorithm is a sort of modification of the probability distribution NRM through *weighted sampling*. The idea is to introduce a bias weight value on the propensity, in order to take into the account the number of times in which the reaction is executed. We will have a lower amount of iterations at the price of applying reactions a different number of times according to the probability.

The weight is a multiplicative factor of the stoichiometric coefficient. μ is the smallest reaction index μ such that:

$$\sum_{j=1}^{\mu} a_j^w \geq r_1 a_0^w$$

Second, PW-DMC has to correct the firing time τ of reaction R_μ because it is selected through the weighted sampling. The generation of τ is adapted from SSA in which it is generated from an exponential distribution with rate a_0^w. PW-DMC then updates the state at time $t + \tau$ by assuming that there are w_μ consecutive firings of R_μ in the time interval $[t, t + \tau)$.

This methodology started to provide some hints for the approximation setting. In the complete algorithm we have the concept of effective propensity, allowing to reach the speed up.

4.2 Bounded Acceptance Probability RSSA

If we compare RSSA and DM algorithm, the main difference is that the RSSA algorithm is more complex than DM at each iteration. The bottleneck in RSSA is that we use the third random numbers for the rejection test, so this algorithm tries to speed up the process and reduce the complexity. If we decide to accept everything, we will end up with a strong bias. In order to avoid the high computational costs, we can only apply rejection to some cases $\rightarrow$ set up bounds, reaching bounded acceptance probability RSSA. If the probability of the candidate reaction is greater than certain amount, we accept it without the rejection test.

Algorithm: we are still executing one reaction at a time. Comparing in terms of accuracy, the approximation of the previous method is not applied. Here we are not working with the real propensity. In this version we are totally avoiding the rejection-based part, the approximation in this case increases the error. It could be possible to apply a strategy in which we apply rejection based only in certain conditions (more complex).

By looking at the literature, it is interesting to rank strategies in terms of accuracy and time.

4.3 τ-Leaping Method

The τ-leaping method is a stochastic approximate algorithm for improving performance of stochastic simulation. Its aim is to discretize the time axis into time intervals and to approximate the number of reactions firing in each time interval. The simulation then leaps from one time interval to the next interval with many reaction firings performed simultaneously.

What is remarkable here, is that we are doing an approximation over the number of iterations which is much more relevant than previous approaches. We are grouping a mixture of events, which can be defined as a *macrostep*. If τ is big, the number of iterations will be low and the complexity will decrease. We should constraint the error in such a way that we are aware of its value as a constant.

Which could be a high-level property of tau to apply macrostep without a crude error in the simulation? We should impose certain logics on τ.

[lo schermo non va]

"vabbè, let me see. Ragazzi non facciamo più niente, nel senso, cosa devo fare?Sono arrabbiatino!... [funziona] La forza? It's possible that? Ahahah"

We want to limit the bias on the propensity. This translates in tau selection satisfying the so-called "**leap condition**":

There exists a leap $\tau > 0$ such that the change in propensity a_j of each reaction R_j with $j = 1, ..., M$ during the time interval $[t, t + \tau)$ is negligibly small.

Negligibly small means lower than some threshold. By applying a proper reasoning we reach the following formula, in which we derive the update of the state. Once we have selected τ, we will evolve the system by adding to he current state a certain amount of firing for each one of the reactions in the system [for each reaction j from 1 to m we apply a certain amount of time].

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

k_j will be generated through a Poisson distribution, which will depend over the specific propensity multiplied by the size of the step. The algorithm is applying any of the reactions per a specific amount of time.

Pros: we are allowed to work with higher τ with respect to exact stochastic simulations.

Cons: if we only have rare events, the algorithm will not be the best solution (as it will try to apply all reactions at all the times). Given however that in any system there are at least some reactions with high propensity, at the very end we still gain a huge speed-up.

4.3.1 Choosing tau - Leap selection section

4.3.1.1 Postleap τ Selection

Choose a τ from a Poisson distribution and see if it is fine for the simulation. We start from a predefined arbitrary τ (small), then we have a trial procedure. The algorithm will compare the difference in the propensity before and after, to assess whether τ is of the right size. E.g. if the change in propensity is small, we can use a greater τ.

The complexity of the τ-leaping is moved to a sub-problem, which is the choice of τ . We could also perform a preleap selection or choose other strategies.

The Poisson distribution could provide more events than the ones that are possible in the system; in this case we end up with a negative population. While performing τ-leaping, we will need to make sure that a negative scenario is never reached through a test. The real amount of time executing k should be computed as the mean from k generated from Poisson distribution and the availability of the reactants [non chiaro].

We can bridge τ-leaping with exact stochastic simulation when we have only rare events

4.4 Chemical Langevin Method

We can use the Normal distribution instead of Poisson, and scale it with mean and variance with another Normal formula based on a (0,1) distribution, obtaining a simplified method. We will not have main events, but main *drivers* in the system. Stochasticity is strongly approximated, we are moving to an idea of average, real numbers. This will be the starting point for the deterministic simulation.

COPASI is a well-known simulator. It exploits Gillespie's algorithm and τ-leaping algorithm. IN general, any simulator offers many algorithms, so we should be able to choose the best computational approach $\rightarrow$ calibrating the model. The stochastic rate provides us the speed, we require to derive it with some methods; all simulations algorithm compute deterministically or stochastically a speed for the reaction, computed over parameters of the model e.g. the rates. The rates, as the initial values, are not free-lunch, they can be derived through experiments.

Often times, we work with models with not fully known parameters; in the second module we will explore solutions for *estimating the parameters* (reverse engineering approach).

During the last lecture, we focused on τ-leaping. ***Adaptive algorithms*** select a delta and change it adaptively during the simulation in order to reach the best results, for instance post leap τ selection.

Chemical Langevin Method allows to modify the evolution of the system; the update formula depends on the propensity, but allows to distinguish a totally deterministic part from the stochastic one. This algorithm also has an important difference with previously seen strategies: we do not have integers, we will be provided with a real number adding some noise. This dynamics is over continuous numbers, tends to an average behaviour.

Chapter 5

Deterministic simulations

A *deterministic* way of calculating the dynamics of a system, when the last (noise) term of CLM equation becomes negligibly small compared with the second one, can be applied. This happens in the limiting case $*a_j(\mathbf{x})\tau ß \inf *$, $*j = 1, ..., M*$, and the deterministic simulation produces an average behaviour of the system that is very close to the one that results by averaging an infinite number of stochastic simulations of the system starting from the same initial state. If we work in terms of *moles*, we are moving Avogrado numbers; it is not so distant from the reaction size, the assumption is correct enough. The deterministic simulations should give as an output the same result as an average stochastic simulation.

ODEs can be safely used to simulate a biochemical system that satisfies the spatial homogeneity and continuum hypothesis.

Continuum hypothesis: "A biochemical system satisfies the continuum hypothesis if the number of molecules for each species is large enough to safely approximate molecular abundances by concentrations that vary continuously (as opposed to integer- valued molecule counts)."

Sometimes, models simply use the ODE formalism, regardless of the hypothesis (since the ODE is the only computational framework allowing to reach the end of the simulation).

We are required to apply conversions and derive the ODE according to the following table:

Table 4.4 Marchetti's book

Here we are not considering the combination of the reactants. c = stochastic rate, k = deterministic rate.

Table 4.5 Marchetti's book

It is possible to go back to the stochastic setting from the deterministic one.

CLM equation: if we assume that the product is crazily high i.e. close to infinite, we can assume that the two parts of the equations have different orders; in this specific condition, the noise part becomes negligible.

5.1 Deterministic approximation

If the propensity is approaching infinite, we can assume that the deterministic setting is motivated. We are required to introduce the law of mass action and employ tables [last lecture] to derive a set of ODEs. In this context, the law of mass action is a bit different from what we have observed at the beginning of the course. We need to remember that the constant of proportionality, i.e. rate, is not the same as the stochastic setting.

$$

Reaction type	Reaction
Zero-order reaction	$\emptyset \xrightarrow{k} A$
First-order reaction	$A \xrightarrow{k} B$
Second-order reaction	$A + B \xrightarrow{k} C$
Second-order reaction (same reactant)	$A + A \xrightarrow{k} B$
Third-order reaction	$A + B + C \xrightarrow{k} D$

Figure 5.1: Table 4.4 Marchetti's book

Reaction order	Reaction
Zero-order reaction	$\emptyset \xrightarrow{c} A$
First-order reaction	$A \xrightarrow{c} B$
Second-order reaction	$A + B \xrightarrow{c} C$
Second-order reaction (same reactant)	$A + A \xrightarrow{c} B$
Third-order reaction	$A + B + C \xrightarrow{c} D$

Figure 5.2: Table 4.5 Marchetti's book

```
\\frac{d[S_i]}{dt} = \\sum^M_{j=1}(                    k_j\\mathbf{v}_{ji} \\prod^{N
```

$$

The equation $\frac{d[S_i]}{dt}$ should represent a molar concentration over time. We have an equation for each of the species $i = 1, \ldots, N$. It should take into consideration the effect of all substances in the system $\rightarrow$ sum over M, considering all reactions, which will be multiplied by the stoichiometric index. The system is computed as k (constant of proportionality), stoichiometric index and the product of all the reactants. If a species is not a reactant, $S_i = 0$, we end up with 1, only product of the reactants.

$$\frac{d[A]}{dt} = -\frac{d[B]}{dt} = -V_{MAX} \cdot \frac{[A]}{K_M + [A]}$$

We can just rely on a simplified set of equations to avid mass action correspondence. V_{MAX} = maximum velocity of the enzymatic reactions, K_m, called *Michaelis constant*, indicates the concentration of the substrate at which the reaction rate is half of its maximum value.

5.2 Numerical solution of ODEs

Often times in computational biology it is unlikely to have enough power for deriving an analytical solution of the system, reality is different and the complexity prevents this kind of approach. Therefore we rely on numerical methods to compute the simulation, adding an additional layer of approximation. The algorithm uses the idea of derivative to understand the next value. For a value of t>0, the level of approximation could be remarkable and depends on the dynamics of the system.

Remember that when we simulate in time we have the issue that the computation of the next state depends on the previous state: we are dealing with an iterative formula, so potentially we can have a huge error explosion. At the basis of such method is the concept of derivative and geometrical rule to identify next step; Euler's method is the simplest one.

5.2.1 Euler's method

The first examples of explicit/implicit numerical methods are the *forward Euler method/backward Euler method* for updating the system state:

- Forward Euler: $[X_{n+1}] = [X_n] + hF(t_n, [X_n])$
- Backward Euler: $[X_{n+1}] = [X_n] + hF(t_{n+1}, [X_{n+1}])$

Not surprisingly, for increasing the order of the algorithm we will require a more accurate approx of the derivative, translating in additional evaluation of the ODE.

5.2.2 RUNGE-KUTTA Method

RK4 is a 4th order method, good compromise in accuracy and computational requirements. This algorithm uses a sort of average over Ks, which are computed by solving the set of ODEs as following:

$*K1 = F(t_n, [X_n])$

$K2 = F(t_n + 2h, [X_n] + h2K_1)*$

$*K3 = F(t_n + h2, [X_n] + h2K_2)$

$K4 = F(t_{n+1}, [X_n] + hK_3)*$

Is it possible to increase accuracy without increasing complexity?

"[entra una ragazza in classe, dice "ups", esce imbarazzata perché ha sbagliato aula] M: ahaha. SERIO,DETERMINISTICO"

5.2.3 Midpoint method

2nd order method, we compute the new state at line 5 by two elements: x the current state and x old (the previous one). In this case, the multi-step method links the increased order to the number of step previously considered to compute the slope.

Drawback: what happens in the first step? We require an initial state and and additional time series!

Limitation of multi-step algorithms: they all require a discretization step, it rules the movement. The time should not be computed during the iteration, chosen from the beginning; if we think about this, it can be a limitation of the application of the methodologies, which ask the user an information which might be unknown. Which is the right time for simulating? Of course we can try to keep the discretization step as small as possible, but the right question that should be asked is: which is the error value? Keeping the error below a certain threshold is not easy, since the dynamics evolve with the system. In order to achieve error control, we should apply more requirements to discretization, leading us to seek an *adaptive solution.*

In order to derive a rough estimate of the error we should compare the state computed by two algorithms of different order: one should be more accurate than the other, to have an idea of the degree of approximation.

5.2.4 Adaptive methods - Runge-Kutta-Fehlberg

"Strapopular algorithm: Runge-Kutta-Fehlberg, RK45 for friends".

RK45: two methods, 4th and 5th order to apply error evaluation [4th for dynamics, 5th for estimating the error]. The algorithm will scale the step (h), in such a way that we keep the deviation below an error threshold. A formula will allow to play with h according to the deviation (increase if deviation is little).

Recall that for the 5th order we require 6 derivations. The two methods are similar, they share the same Ks, therefore at the price of computing a fifth order method we will have the possibility to obtain two evaluations → quite efficient method. ODE45 is this algorithm!

Error computation: $\Delta_{n+1} = \frac{|[\tilde{X}*n+1]-[X*n+1]|}{h}$

The error estimate is then compared to the error threshold ϵ_t provided by the user. If $\Delta_{n+1} \leq \epsilon_t$, the local truncation error is assumed to be smaller than the threshold, the state $[X_{n+1}]$ is accepted and the algorithm moves one step forward. In the other case, the new state is not accepted and the next state is evaluated again using a different (smaller) value of h. In both cases, the value of h is updated as

$$h_{n+1} = h_n \sigma\sigma = (\frac{\epsilon_t}{2\Delta_{n+1}})^{1/4}$$

Issue: it's possible that for specific dynamics we will need an infinitesimal h to satisfy the error threshold. We can avoid this by updating the strategy, for instance impose an additional threshold to h or insert an heuristic.

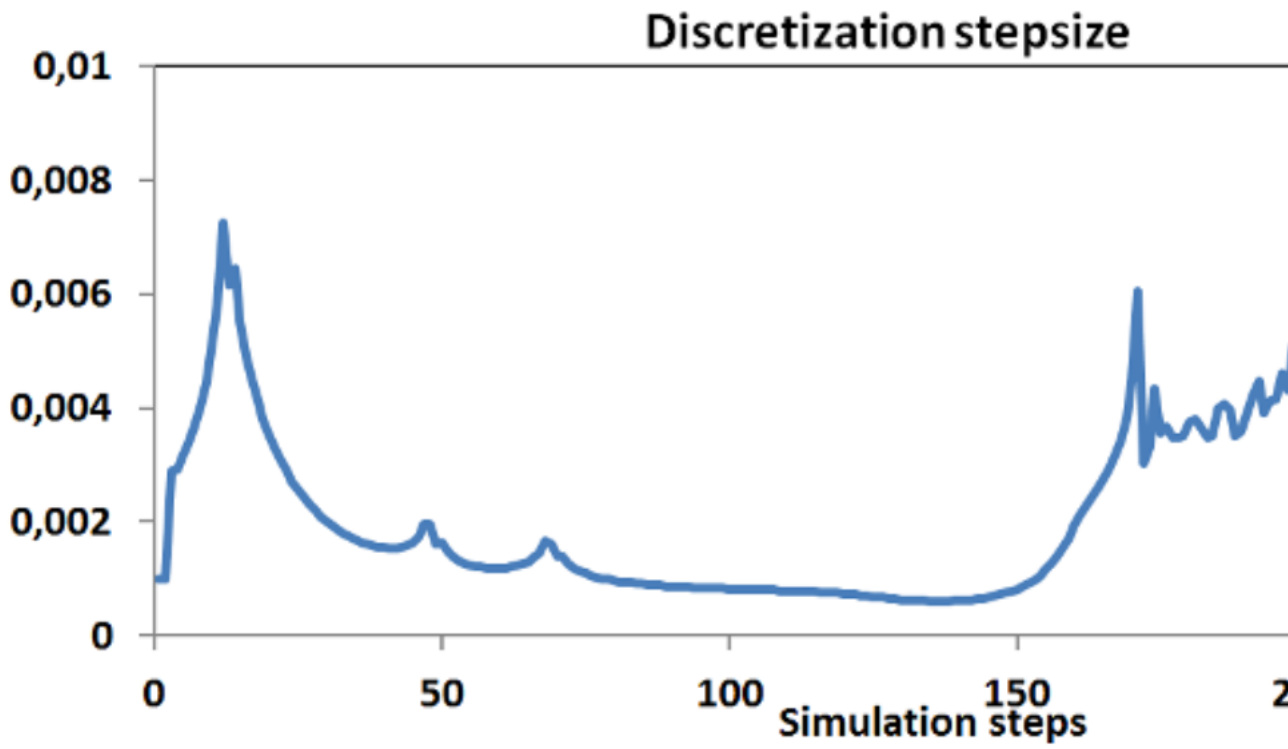


Figure 4.9. Value of h used at each step

We can end up in situations in which the system changes a lot → *stiffness condition*, if we work with an adaptive method the property of the dynamics leads to instability in deriving the discretization step. Stiffness is a couple property of ODEs and the numerical scheme used to solve the system. This means that the same system of ODEs may exhibit stiffness only when it is simulated with some of the numerical schemes introduced in this chapter. In MATLAB we can use ODE15S in case of stiffness (happens quite often in biology simulations).