Alberto Castellini

Algorithms and Software for Biological MP Modeling by Statistical and Optimization Techniques

Ph.D. Thesis

May 25, 2010

Università degli Studi di Verona
Dipartimento di Informatica

Advisor:
Prof. Manca Vincenzo

Series N°: **TD-02-10**

Università di Verona
Dipartimento di Informatica
Strada le Grazie 15, 37134 Verona
Italy

Copyright 2010 © Alberto Castellini - All rights reserved

To Silvia,
my future wife

Contents

1

Introduction

1.1 From natural computing to systems biology

Computer science is a field of research which deals with informational processes. Most of the efforts in the history of this science have been addressed to the analysis and development of artificial systems. Nowadays it is commonly accepted the belief that also living systems perform crucial informational processes, in order to keep themselves far from the thermodynamical equilibrium, to adapt to the environment, and to evolve [210].

Natural computing is the area of computer science that investigates models and computational techniques inspired by nature and, dually, attempts to understand the world around us in terms of information processing [114, 199, 200]. Significant advances in this area have been made through two main branches:

- computing *performed by natural materials*, which aims to generate novel computing paradigms employing natural materials (e.g., DNA molecules, light, etc.) to perform computations, instead of classical silicon-based processors;
- computing *inspired by nature* (also called biologically inspired computing), whose aim is to develop nature-inspired tools (i.e., algorithms) and models for efficient problem solving.

The first branch mainly involves *DNA computing* and *quantum computing*. The former attempts to perform computations by using high parallel operations on DNA molecules, i.e., hybridization, polymerase chain reaction (PCR), and so forth. This field has been initiated in 1994 by Adleman who devised a breakthrough experiment showing the possibility to compute by molecules [2]. The latter branch aims to generate new computing paradigms based on quantum physics [63, 166].

As for the branch of computing "inspired by nature", one of the most known research area is *evolutionary computing*. Evolutionary techniques make use of bio-inspired concepts, such as *evolution* and *selection*, to find suitable solutions to optimization problems usually spanning huge solution spaces. As an example, *genetic algorithms* (GAs) [97] are a searching technique which employs genetic-inspired operations, like crossover, mutation and selection, to find exact or approximate solutions to optimization and searching problems. Whereas GAs evolve solutions

(named *chromosomes*), represented by vectors of variables (named *genes*), a similar technique called *genetic programming* (GP) [123] performs an evolution of (opportunely encoded) computer programs in order to let them execute specific user-defined tasks. *Swarm intelligence* techniques, such as *particle swarm optimization* [116] and *ant colony optimization* [54], are evolutionary artificial intelligence techniques based on the study of collective behavior in decentralized and self-organized systems. *Neural networks* [24], which abstract some feature of brain and nervous system, are a paradigm of information processing successfully used in many fields, like regression and pattern recognition.

The recently-born area of *membrane computing*, also belonging to the branch of computing "inspired by nature", has been introduced by Gheorghe Păun in 1998, and in 2003 has been addressed by the American Institute for Scientific Information as "a fast emerging research front" in computer science. As we will see in more details in Chapter 3, membrane systems, or *P systems*, are distributed and parallel theoretical computing devices inspired by the functioning and structure of living cells. The aim of this formalism is to perform computational tasks by mimicking cellular behaviors [182, 184, 185]. Two main topics have been investigated in this area. The first one concerns the computational power (and the universality) of different classes of P systems [44, 185], the second is related to the usage of P systems as a bio-inspired modeling framework for biological systems [39,44,106,108,141,170,171,173,180,196,220]. In this thesis we present some new results about the second topic, and in particular about *metabolic P systems* (shortly *MP systems*) [17–22,29–36,67–69,71,72,136–149,168] a special class of P systems conceived by Vincenzo Manca to model dynamics of biological phenomena related to metabolism in the living cell. Our final aim is to provide biology and medicine with new (simulation-based) analytical tools for systems understanding and knowledge discovery. For this reason our investigations can be also related to the research area of *computational systems biology* [119], which recently stemmed from the combination of life sciences and engineering.

According to the principles of this discipline, biological systems should be investigated from a systemic and holistic perspective, since some behaviors of these complex systems cannot be understood by just explaining the behaviors of their parts. This idea come from the observation that in complexly interconnected systems components behave differently within the system than in isolation. In other words [234], "there is a tremendous difference between a living organism and a bottle containing all its chemical components. A birthday cake is more than flour, milk, eggs, sugar and candles". This difference lies in the *organization* of system components and in their *interconnections*. In many cases there exists a hierarchy in organizational levels corresponding to a hierarchy of *timescales* at which processes occur, and a hierarchy in *spatial* organization. A key challenge of systems biology concerns the development of mathematical and computational models which consider these hierarchies for a better explanation of biological processes.

1.2 Principles of modeling

The vast structure of knowledge we now call *Science* originates from humankind's discoveries about itself and the surrounding environment. The search for expla-

nations dates back to humans' first observation of the "movement" of the sun, which challenged many philosophers and scientists from Aristotle and Ptolemaeus to Brahe, Kepler, Halley, Newton, Euler and Einstein, just to mention a few major scholars. The principal aim of a scientific investigation is to achieve a deeper understanding and control of some part of the universe (which includes all *measurable* entities and processes within the physical reality), but no significant part of the universe is sufficiently simple to be grasped and controlled without abstraction. *Abstraction* consists in replacing the part of the universe under investigation by a *model* of similar but simpler structure. The active use of models is necessary because the subject of investigation is usually too complex to work with. The complexities may arise from two fronts: *i)* the very large number of interactions among the elements of the system, or *ii)* the presence in the system of a "black box", namely, an element that is inaccessible to observation. In fact, it is sometimes impossible to "observe" all the elements of a system, especially at the time desired [152].

A romantic metaphor for modeling, inspired by the Plato's *Myth of the cave*, is suggested in [17] where the modeling process "is seen as the work of an artist while reproducing reality in a paint. The artist looks at the real world and, based on all his knowledge of colors, materials and lighting effects, he makes a rough sketch of the whole picture. This initial representation is then subsequently integrated by a cycle of comparisons of the representation on the canvas with the real scene, ending up in a more detailed picture". A key point is that, by changing the abstraction level of the paint (both in space and time), different aspects of the pictured reality may pop up (see Figure 1.1). Undoubtedly, a good model is like a piece of art, in which we can find some relevant traits of the described reality. The (modeling) approach of collecting quantitative observations, storing them and eventually trying to explain the main features of the system which generated these observations, has been successfully employed in several scientific disciplines, such as, astronomy, physics, chemistry, life sciences, engineering, meteorology, sociology and economics. All these disciplines, indeed, deal with *inverse-engineering problems of understanding, predicting and controlling system behaviors from observations.*

The heart of modeling consists of identifying *variables* and *invariants* of observed behaviors. Many systems, such as, a cell, a city, a company or a river, are always different in many important aspects if observed at different instants of time, but their essence keeps unchanged during all the instants of their life. The philosopher says [193], "Panta rei" ($\pi\grave{\alpha}\nu\tau\alpha\ \rho\grave{\epsilon}\iota$), that is, "everything is changing" or also "existence is change". But, as mentioned in [143]: "when something changes according to a rule, something does not change... Existence is a mysterious mixing of variation and invariance underlying objects and events, at each level of reality". In any system evolving in time, a set of variables can be identified, such as, the amount of drug in a body, the gross domestic product of a nation or the income of a company. Since they define the behavior of a system, variables must satisfy some rules, which are the invariants of the behavior. For instance, the second Newton's law is an invariant rule describing, in terms of differential equations, the relationship between two variables of a physical system, i.e., force and acceleration.

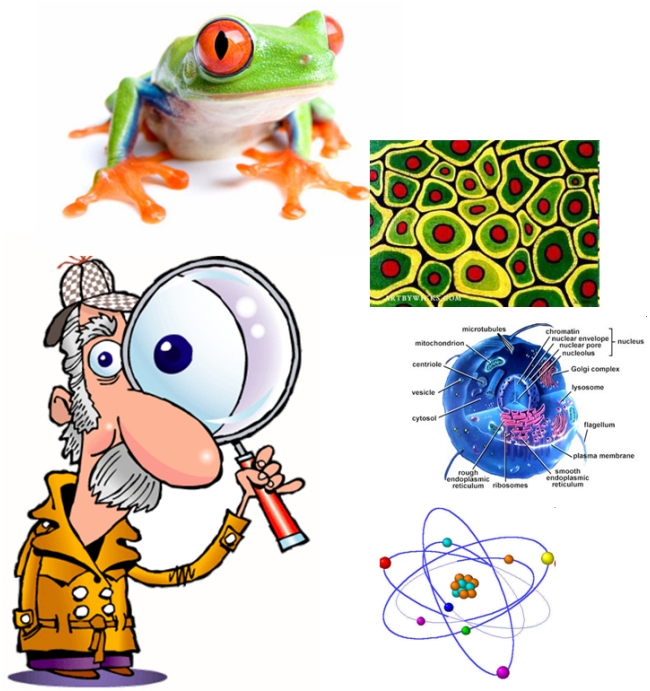


Fig. 1.1. Complex systems display different aspects when viewed from different "distances". Models should be able to adapt their spatial and temporal abstraction levels to the dimension and the "speed" of the system under investigation.

Many kinds of models have been devised for analyzing various kinds of systems from different perspectives. A coarse classification can be done between continuous [109] and discrete [130, 184] models, static [25, 45] and dynamical [48, 225] models, deterministic [109] and stochastic [79] models. As said in [152]: "The proper selection of an appropriate model for a certain system is critically important. The value and power of any model as a deductive or inductive tool depends on the speed and freedom with which the investigator can visualize relationships and concepts based on it. It is important to notice that there may be a number of different, yet equally successful, models for the same subject of investigation. Having achieved one model does not give the right to state, "this is how it works". Establishing a particular model above all others is like erecting a barrier to open thought and to any amendment that could someday further clarify the subject. A model is an invention, not a discovery. It may prove to be a valid description, but this is far from being the essential truth".

Differential equations have been the most employed modeling framework for dynamical systems since the first discoveries of planetary motion laws. Their application to the analysis of very complex systems formulated within life sciences, economy, meteorology, and many other areas, however, have recently pointed out some limitations of this framework. A new generation of discrete dynamical models are being proposed which replace the classical differential viewpoint with an algorithmic perspective. Their aim is to provide new insight on the mechanisms

involved in the generation of complex dynamics in order to open new frontiers for tackling the challenges of the new millennium in several scientific fields [143].

Just to hint the main traits of this new paradigm, let us consider a simple phenomenon from a discrete point of view. Suppose to observe a device generating words over a particular alphabet. Given a set of words observed from the device, some very natural questions arise: "which is the grammar of the language generated by this device?" and, "how the internal structure of the device realizes the generation of these words?" As pointed out in [143]: "If we consider the observations of any system as words of a language, then the search for invariant rules underlying processes can, in principle, correspond to the search of a grammar". From this perspective, the search for models can be supported by new tools imported from formal language theory [201]. These tools have rarely been employed before for modeling and they can yield innovative modeling approaches. As for metabolic phenomena, we will show in the following of this thesis that concepts and rules usually expressed in terms of differential equations can be reformulated in terms of some special classes of grammars. Moreover, these grammars are often directly related to the biochemical mechanisms generating the biochemical phenomena.

1.3 Motivations and results

In this thesis the research field of biological system modeling is introduced by an analysis of the literature about some conventional (i.e., differential equations, Gillespie's models) and unconventional (i.e., P systems and metabolic P systems) modeling frameworks. Subsequently I report the results achieved during my Ph.D. with regard to three research topics, namely:

- equivalences between MP systems and hybrid functional Petri nets,
- statistical and optimization perspectives in the generation of MP models from experimental data,
- development of the virtual laboratory MetaPlab, a Java software based on MP systems.

All these topics have a strong scientific *motivation*. The equivalence between MP systems and hybrid functional Petri nets results specially relevant in the framework of biological modeling, since it guarantees robust modeling capabilities for MP systems. The second research topic concerns the core of modeling. Indeed, generating new models of biological systems is usually the best way to gain new insight about them, which is very important in fields like medicine, biology and pharmaceutics. The last point is more applicative, and is motivated by the necessity of computational tools for assisting modelers and biologists to cope with the high complexity of biological systems, in which a very large number of elements with different functions interact selectively and nonlinearly to produce coherent behaviors.

Chapter 2 presents two classical frameworks for biological systems modeling, namely, *ordinary differential equations* (ODE) [109] and *Gillespie's stochastic models* [79]. In this chapter the problem of modeling is tackled by means of several examples, such as, the variation of a population size over time, the administration

of drugs and the irreversible isomerization. Along a discussion about ODEs, the concepts of rate of change, reaction rate constants, reaction orders, are described and main numerical procedures for solving differential equations are explained as well. At the end of this section S-systems are presented [205, 234]. On the other hand, the discussion about Gillespie's algorithm refers to the basis of molecular collision and stochastic modeling.

In Chapter 3 the modeling framework of P systems is presented and several extensions of this formalism are described [44, 185], such as, P systems with symport/antiport, P systems with active membranes, tissue-like P systems, and so forth. Some results about the computational power of P systems are reported, but the focus is subsequently diverted to three variants of P systems developed for modeling biological systems, namely, stochastic P systems [171], probabilistic P systems [172, 173] and abstract rewriting systems on multisets (ARMS) [221].

Chapter 4 copes with metabolic P systems [137, 140–142, 145], the modeling framework, introduced by Manca, to which all the original contributions of this thesis refer. After a formal definition of MP systems with flux regulation maps (MPF) and MP systems with reaction maps (MPR) we report some equivalences between the two classes [141]. Subsequently, MP graphs are introduced [144] as a graphical formalism for easily visualizing MP models. The chapter then presents two important equivalences: one between MP systems and ODE [68], and another, developed in [30,31,35], between MP systems and hybrid function Petri nets. This equivalence, reported in Section 4.4, represents the **first original result** of this thesis. The author has contributed to conceive the mapping procedures between the two formalisms, to prove two theorems about the same equivalence, and to perform in silico experiments (simulations) for the case study of the *lac* operon gene regulatory mechanism and glycolytic pathway. The chapter ends with the description of a very important theory devised by Manca, namely, the log-gain theory [140, 142], which permits a first step towards the actual generation of MP models from observed data.

Chapter 5 contains the **second original result** of this thesis, pertaining to the generation of MP models from observed data. At the beginning of this chapter the reverse-engineering problem of flux regulation function synthesis is introduced [33]. It involves two main phases: the discovery of reaction fluxes from data (by means of the log-gain theory) and the generation of functions able to fit these fluxes. Some important function forms for representing complex biochemical reaction mechanisms are reviewed [46] and two regression techniques are subsequently presented for synthesizing these functions, namely, linear regression [1] and neural networks [24]. The original contributions of the author concern three topics: *i)* the employment of neural networks (with traditional and evolutionary learning techniques) for the generation of flux regulation functions from data [33] (Sections 5.3 and 5.4), *ii)* a new approach for feature selection with neural networks [36] (Section 5.5), and *iii)* a complete pipeline for data analysis [32] which addresses the entire process of flux regulation function synthesis from data preparation to model validation (Section 5.5). The proposed techniques are tested by two case studies, namely, the mitotic oscillator in early amphibian embryos [33] and the non photochemical quenching phenomenon (NPQ) [32].

Chapter 6 presents MetaPlab [146, 241], a Java software based on MP systems and implemented to automatize modeling and analysis of biological systems. As a virtual laboratory this software is equipped with an extensible set of virtual tools, in the form of plugins, performing various processing jobs, such as, dynamics computation, flux discovery (by means of the log-gain theory), and regulation function synthesis (by linear regression and neural networks). This chapter starts with a brief overview on some prominent tools for bioinformatics and systems biology. Then it describes the main features of MetaPlab, that range from an innovative software architecture to a set of user friendly interfaces. The **third original result** of this thesis concerns the author's contributions to this software for: *i)* design and development of the plugin architecture on which the software is based [34], *ii)* implementation of four plugins (i.e., neural network regression tool, linear regression tool, flux discovery tool, and dynamic computation tool) [146, 241], some of which have been realized in collaboration with other students, *iii)* design of the MetaPlab website [241], *iv)* realization of the MetaPlab user guide and plugin tutorials [146].

In Chapter 7 we report some conclusions about the overall research project presented in this thesis and a we suggest few ideas for the development of future research lines in this field.

2

Traditional models for biological systems

In this chapter we introduce two of the main mathematical approaches for modeling biological systems and their dynamics, namely, *ordinary differential equations* (ODE) [109] and *Gillespie's stochastic models* [79]. Many other approaches have been proposed for similar purposes, such as, Petri nets [154, 189], cellular automata [78, 248], Lindenmayer's systems (L systems) [130], and so on. We mainly focus on ODE and Gillespie's models because they are the most traditional frameworks for deterministic and stochastic modeling respectively. For both the approaches we present a few theoretical foundations, some applications and we outline advantages and disadvantages of using these techniques in practice. ODE systems are explained in Section 2.1, where case studies of drug administration and biochemical system modeling are considered in Subsections 2.1.2 and 2.1.3 respectively. S-systems [205–207, 234], a class of ODE systems developed specifically for modeling biochemical systems, are introduced and theoretically motivated in Subsection 2.1.4. The second part of this chapter concerns stochastic models of chemical reactions. At the beginning of Section 2.2 we introduce some basis of molecular collisions (Subsection 2.2.1), in which the stochastic approach is rooted. Subsequently, we motivate the employment of the "stochastic simulation approach", proposed by Gillespie, in place of the "master equation approach" for computing biochemical dynamics (Subsections 2.2.2 and 2.2.3). The chapter ends with an example showing the application of the Gillespie's algorithm to the case study of irreversible isomerization (Section 2.2.4).

2.1 Differential equation models

2.1.1 Introduction

To understand complex systems often we need to know how their main components evolve over time. Measurements, apt to reveal dynamics, can be very difficult to execute and sometimes even impossible, while it is often easier to observe temporal *changes* of system variables and to describe their evolution by means of instantaneous parameters, called *rates*.

Let us analyze, for instance, the variation of a population size over time [109]. Let be $p(t)$ the number of individuals in a given area at the the time t. If at time

$t+T$ the number of individuals in the same population is $p(t+T)$, then the change in population size over the discrete interval T is $p(t+T) - p(t) = N_T$. Moreover, we can suppose that the longer is the interval T the greater is the number of added individuals, thus, writing $N_T = N \cdot T$ we have

$$p(t+T) - p(t) = N \cdot T \tag{2.1}$$

or

$$\frac{p(t+T) - p(t)}{T} = N. \tag{2.2}$$

Letting $T \to 0$, we obtain the *differential equation* (2.3) which describes the instantaneous change of population size by means of the *(instantaneous) rate of change* N:

$$\frac{dp(t)}{dt} = N. \tag{2.3}$$

Equation (2.3) represents a very simplistic mathematical *model* of the population grow. Of course, one can consider that the more individuals there are at time t the more births are likely to occur. In this case, N depends also on $p(t)$ and the model becomes

$$\frac{dp(t)}{dt} = N_0 p(t) \tag{2.4}$$

where N_0 is a constant, called *specific growth rate*, describing the growth rate of a single individual.

What we want to investigate by means of a differential model is the dynamics of one or more elements of the system. The investigation often starts from some knowledge about the rates of change of all these elements and their *initial conditions*. For instance, given an initial population of 100 individuals and a specific growth rate of 0.8 new individuals per minute per individual, the question is: "how many individuals will be present in the system after 2 minutes?" The answer comes from solving (integrating) the differential equation (2.4), whose general solution is

$$p(t) = p(0)e^{N_0 t}. \tag{2.5}$$

Setting $p(0) = 100$ and $N_0 = 0.8$ the population size after 2 time units turns out to be $p(2) = 100e^{0.8 \cdot 2} \approx 495$ individuals. Figure 2.1 shows the general trend of an exponential evolution for $p(0) = 1$ and $N_0 = 1$.

Equation (2.4) has a very simple form and it can be easily solved by analytical techniques but, in general, differential equations of complex systems present more intricate forms which may be insoluble. Only specific types of equations can be solved by means of analytical techniques [109], while for the majority of the real-world models, numerical procedures are required to compute approximated solutions. *Euler's method* can be seen as a prototype for all numerical methods that solve differential equations by a step-by-step process [7]. This technique can be applied to differential equations having a general form

$$\frac{dy(t)}{dt} = f(t, y(t)) \tag{2.6}$$

where f is a continuous function. Euler's method approximates the function $y(t)$ in the interval $(t, t+h)$ by the first two terms of the Taylor expansion of $y(t)$

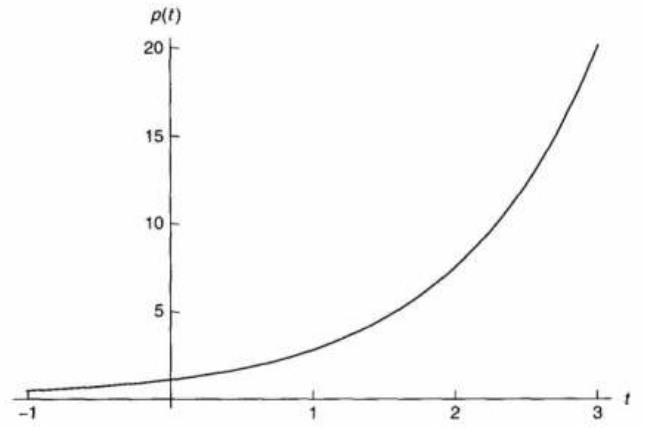


Fig. 2.1. Exponential growth generated by Equation (2.5) for $p(0) = 1$ and $N_0 = 1$ [109].

$$\hat{y}(t+h) = y(t) + h \cdot f(t, y(t)) \tag{2.7}$$

where $f(t, y(t))$ represents the slope of the $y(t)$ tangent in t and h is the length of the time interval (see Figure 2.2).

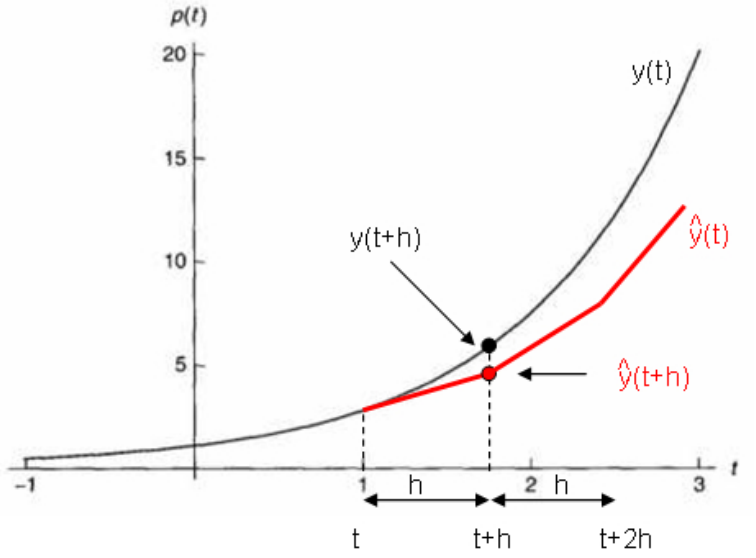


Fig. 2.2. Numerical approximation of $y(t)$ by Euler's method.

The low accuracy of Euler's method, due to its first order (linear) approximation, induced mathematicians to develop more precise techniques based on higher-order approximations, multiple steps, and many other mathematical improvements. Among them, *Runge Kutta* methods [7], developed in 1900, are an

important family of implicit and explicit approximation techniques that extend Euler's method.

The differential equations framework is quite general and it enables to model many different kinds of systems, such as biological systems [109], but also economical systems and social systems [90]. In the following we show some applications of this modeling framework.

2.1.2 A case study: administration of drugs

This example shows how a differential equation model supports the estimation of dosage levels and administration intervals of a drug in order to maintain specific concentrations of substances in a body [109].

When a drug is administered, its concentration in the body fluid rapidly grows to a maximum values and then it starts to decrease because of various degradation processes. The decreasing rate is often proportional to the drug concentration $c(t)$, thus, the following differential equation represents a model of this phenomenon:

$$\frac{dc(t)}{dt} = -\frac{c(t)}{k} \tag{2.8}$$

where constant k is related to the speed of degradation. The solution of this equation, reported below, is a function which decreases exponentially from the initial condition c_0:

$$c(t) = c_0 e^{-t/k} \tag{2.9}$$

An analysis of this solution function shows that concentration is divided by e every k hours, which is very important if we want to forecast the drug concentration after n doses administered every Δt hours. This concentration value is in fact computed by the following equation [109]

$$\begin{aligned} c_{n-1} &= c_0(1 + e^{-\Delta t/k} + e^{-2\Delta t/k} + \ldots + e^{-(n-1)\Delta t/k}) \\ &= c_0 \frac{1 - e^{-n\Delta t/k}}{1 - e^{-\Delta t/k}} \end{aligned} \tag{2.10}$$

and its trend is depicted in Figure 2.3. The residue r_n measured just before the $(n+1)$-th administration is

$$r_n = c_{n-1} e^{-\Delta t/k} = c_0 e^{-\Delta t/k} \frac{1 - e^{-n\Delta t/k}}{1 - e^{-\Delta t/k}} \tag{2.11}$$

The mathematical analysis of Equation (2.10) shows that the drug concentration never exceeds the limit value c_M, where

$$c_M = \frac{c_0}{1 - e^{-\Delta t/k}} \tag{2.12}$$

because term $e^{-n\Delta t/k}$, at the numerator of Equation (2.10), tends to 0 when n grows. c_M may be considered a good estimate of the concentration immediately after a dose, for n and Δt sufficiently large. Indeed, it can be found [109] that, if $n\Delta t > 5k$ then c_{n-1} differs from c_M by less than 1%. In other words, to reach

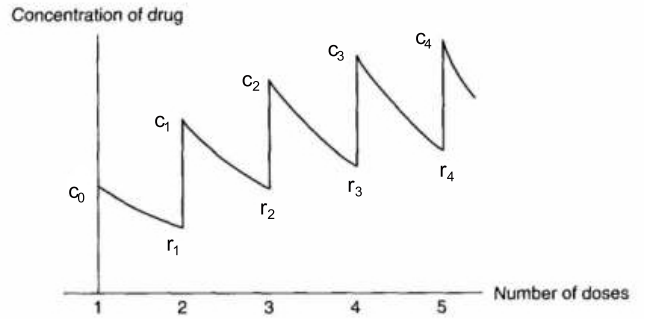


Fig. 2.3. Concentrations of drug during the first five administrations [109].

the maximum concentration in 5 doses, one should make the interval between two doses larger than k. Of course, if $\Delta t/k$ increases, then the limit c_M decreases towards c_0.

The residue just before the n-th dose, when n tends to infinite, is similarly given by

$$r = c_M e^{-\Delta t/k} = \frac{c_0}{e^{\Delta t/k} - 1}. \tag{2.13}$$

Interesting considerations may be made even by analyzing this equation. We observe that r decreases as $\Delta t/k$ increases. Moreover, being $c_M = c_0 + r$, the larger $\Delta t/k$ the larger the concentration variation between two doses. Therefore, a trade-off has to be found between keeping the residue among a certain level and reaching the maximum concentration in a few doses. Figure 2.4 displays the stable oscillatory behavior reached after administrating a sufficient number of doses.

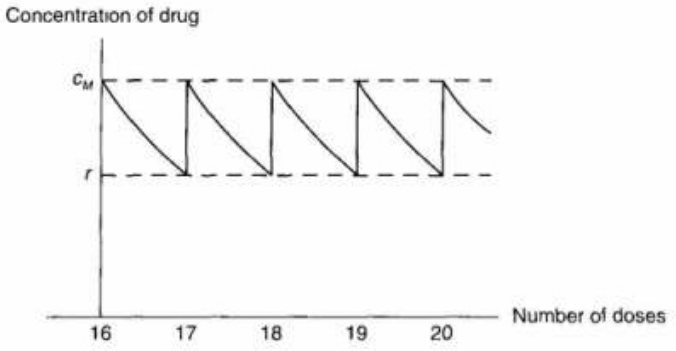


Fig. 2.4. Concentrations of drug after many administrations [109].

Since several antibiotics can have harmful effects until their concentration is below a specific threshold, the oscillatory growth of Figure 2.3 is something to avoid, while the stable oscillation of Figure 2.4 is advisable. In order to quickly reach the second behavior, a large dose c_M is usually administrated at first in order to reach the maximum concentration, and subsequently normal doses of c_0 are given at intervals of Δt.

2.1.3 Modeling biochemical systems

In this section we introduce some notions about the application of differential equations to biochemical systems modeling. Biochemical systems are often represented in standard chemistry by sets of coupled reaction in the form

$$X_1 + X_2 \xrightarrow{k} X_3 \tag{2.14}$$

where molecules X_1 and X_2, called *substrates* or *reactants*, combine to generate molecule X_3, called *product*. Constant k is named *reaction rate* and it represents instantaneous rate of transformation of substrates to products. In other words, a reaction rate is the instantaneous "speed" of a reaction to generate products from substrates. Most kinetic laws relate reaction rates to changes in chemical concentrations, which is equivalent to write differential equations in the form

$$\frac{dX_i(t)}{dt} = \text{instantaneous rate of change in } X_i \text{ at time } t. \tag{2.15}$$

Since change in concentrations are usually functions of substrate concentrations S_i, enzymes E_i, factors F_i and products P_i, Equation (2.15) can be rewritten as [234]

$$\frac{dX_i(t)}{dt} = f(S_1, S_2, \ldots, E_1, E_2, \ldots, F_1, F_2, \ldots, P_1, P_2, \ldots). \tag{2.16}$$

Reaction rate equations (RRE), also called *rate laws*, are well known equations in chemical kinetics. They express reaction rates as functions of reactant concentrations and some constant parameters. For instance, reaction (2.14) may be ruled by the generic rate law

$$rate(X_1, X_2) = kX_1^m X_2^n \tag{2.17}$$

where *rate constant* k and *reaction orders* m and n have to be determined by experiments. These kinetic constants become fundamental for generating differential equation models, since they include everything that affects reaction rate besides concentrations, such as, temperature, pH, pressure, ionic strength, geometric properties of chemicals, light irradiation, and so on. Consequently, kinetic constants are approximations of a more complex (and unknown) functions, and their tuning is often a tricky process.

RREs are, in fact, differential equations. Let us consider, for instance, the well known Michaelis-Menten rate law, which describes the kinetics of many enzymes acting on a substrate S and generating a product P. It is

$$v(S) = -v(P) = \frac{V_{max}\ S}{K_M + S} \tag{2.18}$$

where $v(S)$ and $v(P)$ are, respectively, the instantaneous consumption of substrate S and the instantaneous generation of product P, V_{max} is the maximum rate constant and K_M is the so-called *Michaelis constant*. This equation can be written in the differential form as

$$\frac{dS}{dt} = -\frac{V_{max}\ S}{K_M + S} \tag{2.19}$$

$$\frac{dP}{dt} = \frac{V_{max}\ S}{K_M + S} \tag{2.20}$$

since $\frac{dS}{dt} = -v(S)$ and $\frac{dP}{dt} = -v(P)$.

Given such a differential model, our main target is usually to find the substance concentrations at a specific instant t. It requires to solve differential equations (2.19) and (2.20), that is, to compute functions $S(t)$ and $P(t)$ from these equations. Let us set initial conditions $S_0 = 10\ mol$ and $P_0 = 0\ mol$, and compute the time evolutions of S and P employing rate constants $V_{MAX} = 2$ and $K_M = 4$. We achieve the curves of Figure 2.5.

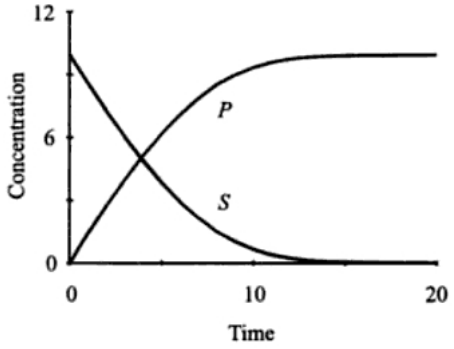


Fig. 2.5. Temporal evolution of substrate S and product P of the Michaelis-Menten model of Equations (2.19) and (2.20). Rate constants are $V_{MAX} = 2$ and $K_M = 4$, initial conditions are $S_0 = 10$ moles and $P_0 = 0$ moles [234].

Coupled differential equations are very common in biological systems modeling since a huge number of different chemical processes occur (even in very simple organisms) which involve interactions and competition. Such a kind of problems require to solve *systems of differential equation*, such as those defined by Equations (2.19) and (2.20).

Kinetics function forms. Given a biochemical system involving chemicals $X_1, \ldots, X_n$, differential equation (2.16) can be rewritten, for each chemical X_i, as

$$\frac{dX_i(t)}{dt} = f_i(X_1, \ldots, X_n) \tag{2.21}$$

if we omit constant parameters.

In order to make such equations able to generate the time evolution of a system, we have to discover suitable functions $f_i,\ i = 1, \ldots, n$. For several systems some information about the form of these functions are available from experiments, but their real forms are usually unknown. Michaelis-Menten rate law, for instance, is an approximation achieved by fitting experimental observations of some enzyme kinetics. This model is relevant to situations where very simple kinetics can be assumed, but in different conditions it becomes not appropriate and more complex kinetic functions are needed.

For generating kinetic function forms for systems of coupled reactions, we consider that each reaction can give to a substance X_i, either a positive contribution (if the reaction generates substance X_i) or a negative contribution (if the reaction consumes substance X_i). To consider this fact we split function f_i of Equation (2.21) into a production term f_i^+ and a consumption term f_i^- [234]

$$\frac{dX_i(t)}{dt} = f_i^+(X_1, \ldots, X_n) - f_i^-(X_1, \ldots, X_n), \quad i = 1, \ldots, n \tag{2.22}$$

where f_i^+ and f_i^- can be, for instance, sums of functions f_{ij} representing the contribution (rate) of each reaction r_j to the transformation of substance X_i. Let us consider a simple example. Given a system of three substances X_1, X_2, X_3, and three reactions

$$\begin{aligned} r_1 &: X_1 \to X_2 \\ r_2 &: X_2 \to X_3 \\ r_3 &: X_3 \to X_1 \end{aligned} \tag{2.23}$$

we write the following system of differential equations

$$\begin{aligned} \frac{dX_1(t)}{dt} &= f_1^+ - f_1^- = f_{13}(X_1, X_2, X_3) - f_{11}(X_1, X_2, X_3) \\ \frac{dX_2(t)}{dt} &= f_2^+ - f_2^- = f_{21}(X_1, X_2, X_3) - f_{22}(X_1, X_2, X_3) \\ \frac{dX_3(t)}{dt} &= f_3^+ - f_3^- = f_{32}(X_1, X_2, X_3) - f_{33}(X_1, X_2, X_3) \end{aligned} \tag{2.24}$$

Now the question is: "which form should functions f_i^+ and f_i^- have to properly represent real-world chemical reactions?" In the following we introduce S-systems, a well known modeling framework for biochemical systems which employs power laws as functions f_i^+ and f_i^-.

2.1.4 S-systems

Let us consider the chemical system of Figure 2.6 which represents the transformation of X_1 to X_2, catalyzed by X_3. The degradation rate of X_1 depends on the the concentrations of X_1 itself and of enzyme X_3, X_2 is produced at the same rate because its production is originated from the degradation of X_1, finally, the degradation of X_2 depends on concentration X_2. Supposing to have a constant generation of X_1 at rate α, we achieve the following system of differential equations [234]:

$$\begin{aligned} \frac{dX_1(t)}{dt} &= \alpha - f_1(X_1, X_3) \\ \frac{dX_2(t)}{dt} &= f_1(X_1, X_3) - f_2(X_2) \end{aligned} \tag{2.25}$$

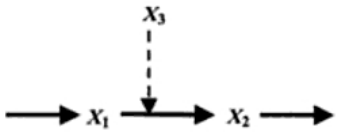


Fig. 2.6. Transformation of X_1 to X_2, catalyzed by X_3 [234].

Many experimental observations and related considerations about kinetic properties of biochemical systems suggest that a convenient mathematical representation for f_i^+ and f_i^- is given by "a product of *power law functions* of those and only those variables that directly affect this process. This product is then multiplied by a rate constant that determines the speed of the process" [205–207, 234]. For a general system of n substances, power-law production and degradation functions are reported in the following, for $i = 1, \ldots, n$

$$f_i^+(X_1, \ldots, X_n) = \alpha_i X_1^{g_{i1}} X_2^{g_{i2}} \ldots X_n^{g_{in}}, \tag{2.26}$$

$$f_i^-(X_1, \ldots, X_n) = \beta_i X_1^{h_{i1}} X_2^{h_{i2}} \ldots X_n^{h_{in}}, \tag{2.27}$$

where *rate constants* $\alpha_i, \beta_i \in \mathbb{R}^+ \cup \{0\}$, and *kinetic orders* $g_{ij}, h_{ij} \in \mathbb{R}$. In particular, powers g_{ij} and h_{ij} are set to 0 for substances X_k that do not affect, respectively, production and degradation of X_i. In the specific case of the catalyzed conversion model (2.25), the power-law representation is

$$\begin{aligned} f_1^+ &= \alpha, \\ f_1^-(X_1, X_3) = f_2^+(X_1, X_3) &= \beta X_1^a X_3^b, \\ f_2^-(X_2) &= \gamma X_2^c. \end{aligned} \tag{2.28}$$

$\alpha, \beta \in \mathbb{R}^+ \cup \{0\}$ and $a, b, c \in \mathbb{R}$. Power law functions are supported only by many experimental evidences [208, 234], but no mathematical proof demonstrates that they are the best possible description of biochemical dynamics. Savageau and Voit shown in [209] that virtually any differentiable nonlinearity can be captured by these equations, including most complex oscillations and even chaos.

S-systems [205–207, 234], where S stands for *synergism* and *saturation*, are defined by substituting functions (2.26) and (2.27) in the general schema (2.22), obtaining differential equations in the following form

$$\frac{dX_i(t)}{dt} = \alpha_i \prod_{j=1}^{n} X_j^{g_{ij}} - \beta_i \prod_{j=1}^{n} X_j^{h_{ij}}, \quad i = 1, \ldots, n. \tag{2.29}$$

We observe that S-systems equations have a standard form. What relates them to specific biological systems are rate constants and kinetic orders. In particular, kinetic orders are interpreted, in elemental chemical kinetics, as the number of molecules involved in a reaction. Recent studies suggest that many enzyme-catalyzed reactions *in vivo* require non-integer and also negative kinetic orders. For instance, Equations (2.28) could have the forms

$$\begin{aligned} f_1^+ &= 1.34, \\ f_1^-(X_1, X_3) = f_2^+(X_1, X_3) &= 0.76 X_1 X_3^{-2.45}, \\ f_2^-(X_2) &= 2X_2. \end{aligned} \tag{2.30}$$

The effect of non-integer and negative parameters in the transformation of a metabolite is easy to understand. For example, constant -2.45 means that enzyme X_3 inhibits the production of X_2 from X_1 by a factor 2.45, thus it is a repressor.

Theoretical justification. A comprehensive analysis of S-systems theoretical basis is beyond the scope of this thesis, but we want to remark that a consequence of S-systems approximation is that relative variations in metabolite concentrations are linearly related to relative changes in production and degradation rates [234]. From biochemical findings it is known that studying *relative* effects in response to relative variations can be more appropriate than studying *absolute* effects. This fact is supported, for instance, by *allometric principle* [236] which states that a specific ratio holds between the relative variations of two related biological parameters, such as, the mass of an organism and its superficial area. It seems to be a general property of living organisms which allows them to keep basic equilibria underlying their internal organization [140]. As shown in [236], many empirical laws on metabolism are also instances of allometry. In [234] Voit shows that S-systems equations can be derived from equations describing relative rate of changes by Taylor's approximation.

Let us define the *flux* F_i of a metabolite X_i as

$$F_i = \alpha_i \prod_n^{j=1} X_j^{g_{ij}} - \beta_i \prod_n^{j=1} X_j^{h_{ij}}, \quad i = 1, \ldots, n. \tag{2.31}$$

where the α-term is called *incoming flux* for X_i and the β-term is named *outgoing flux* for X_i. Experiments suggest that relative variations of metabolite concentrations cause proportional variation in fluxes at the steady state, and S-systems have proved to respond in the same way. In particular, at the steady state we have

$$F_i = \alpha_i \prod_n^{j=1} X_j^{g_{ij}} = \beta_i \prod_n^{j=1} X_j^{h_{ij}}, \quad i = 1, \ldots, n. \tag{2.32}$$

and the mathematical analysis of flux relative variations shows that a one-percent change of concentration X_j induces a g_{ij}-percent change in F_i [234].

2.2 Stochastic models of chemical reactions

Let V be a volume containing a well stirred mixture of n chemical species $X_1, \ldots, X_n$ that interact by means of m chemical reactions $R_1, \ldots, R_m$. In the last section we explained how differential equation models can represent such a kind of systems in a *continuous* and *deterministic* way. Chemical reactions have been regarded as continuous rate processes having a deterministic dynamics in

order to represent their instantaneous contribution by power laws. However, it is known that the time evolution of a real chemical system is not a continuous process because molecules are discrete entities and their quantity can change only by integer amounts. Moreover, the time evolution is not even a deterministic process since microscopic interactions involve uncertainty that cannot be solved unless appealing to quantum considerations about molecular motion, which are often intractable from a computational viewpoint. In other words, although chemical systems evolve deterministically with respect to molecular position, velocity and population size, they would not evolve deterministically with respect to population sizes alone, because quantum indeterminacy unavoidably enters [82].

The continuous and deterministic approach is effective in many cases but, since it deals with *average* molecular population levels, it is unable to describe *fluctuations* and *correlations* that feature every real biological system and become significant when systems involving small amounts of molecules are modeled. In these cases a compromise has to be reached between quantum models, considering position and velocity of every single molecule, and deterministic models, regarding only the average amount of each chemical species. *Stochastic chemical models* attempts to describe time evolution of uniformly distributed chemical systems taking into account system discreteness and stochasticity but avoiding molecular motion details.

2.2.1 Molecular collisions

In a well stirred mixture of molecules in thermal equilibrium, reactions occurs when two or more particles, randomly moving in a medium, *collide* in a proper way. Considering a system composed by a mixture of two gas-phase molecular species X_1 and X_2 (where the same symbols are used to denote the amount of molecules of each species), we assume molecules as hard spheres having radius r_1 and r_2 respectively [79, 80]. A collision between a molecule of type X_1 and a molecule of type X_2 happens whenever the center-to-center distance between the two particles becomes equal to $r_{12} = r_1 + r_2$, as shown in Figure 2.7.

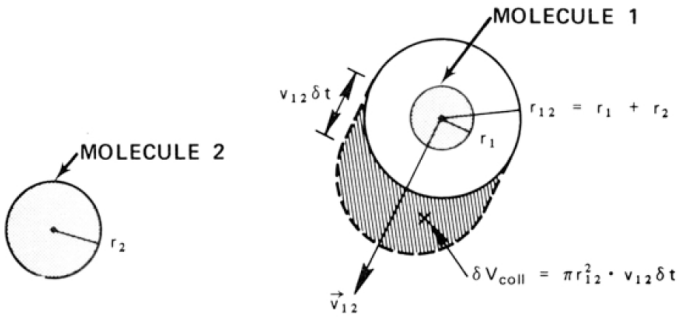


Fig. 2.7. A representation of molecules as hard spheres and the collision volume swept by molecule 1 with respect to molecule 2 in the time interval dt [80].

Being $\vec{v}_{12}$ the relative speed of molecule 1 with respect to molecule 2, the "collision volume" $dV_{coll} = \pi r_{12}^2 v_{12} dt$ represents the volume swept by molecule 1, relative to molecule 2, in the infinitesimal time interval dt (see Figure 2.7). If molecule 2 lies in dV_{coll} during interval $(t, t+dt)$ then a collision occurs. At this point, rather than estimating the *rate* of collision as the number of molecules X_2 whose centers lie in dV_{coll} in the time interval dt, for $dt \to 0$, we compute the *probability* of collision as dV_{coll}/V assuming a random and uniform molecule distribution [80]. In this way we avoid non-rigorous averaging arguments required by the first approach to overcome difficulties related to the limit $dt \to 0$ which brings dV_{coll} to be infinitesimal (the reader may refer to [79] for a more detailed description about this argument). The probabilistic approach instead averages ratio dV_{coll}/V over the molecule velocities, obtaining the average probability that a specific pair of molecules 1 and 2 collide in interval dt

$$\overline{dV_{coll}/V} = \frac{\pi r_{12}^2 \overline{v_{12}}}{V} dt. \tag{2.33}$$

The average velocity $\overline{v_{12}}$ can be easily computed if the mixture of molecules is in thermal equilibrium at absolute temperature T. In this case the velocities of molecules are randomly distributed according to Maxwell/Boltzmann distribution, than Equation (2.33) becomes [79]

$$\overline{dV_{coll}/V} = \frac{\pi r_{12}^2 \sqrt{8kT/\pi m_{12}}}{V} dt. \tag{2.34}$$

where k is the Boltzmann's constant and m_{12} is the reduced mass $m_1 m_2/(m_1+m_2)$ (m_i represents the mass of a molecule of type i).

Considering all the X_1 molecules of type 1 and the X_2 molecules of type 2 contained in V we compute the average probability of collision between two different molecules in V in the infinitesimal time interval $(t, t+dt)$ as $X_1 X_2 \pi r_{12}^2 \overline{v_{12}} V^{-1} dt$. Therefore, even if we cannot compute the number of collisions occurring in V in the infinitesimal time interval dt, we can compute the average probability of a collision in V in the time interval dt.

2.2.2 The stochastic reaction constant c_μ

In this section we explain how the molecular collision model, introduced above, can be employed to describe chemical reaction dynamics. Given the following set of reaction types [79]

$$* \to P, \tag{2.35}$$
$$X_j \to P, \tag{2.36}$$
$$X_j + X_k \to P \quad (j \neq k), \tag{2.37}$$
$$2X_j \to P \tag{2.38}$$
$$X_i + X_j + X_k \to P \quad (i \neq j \neq k \neq i), \tag{2.39}$$
$$X_j + 2X_k \to P \quad (j \neq k), \tag{2.40}$$
$$3X_j \to P. \tag{2.41}$$

where X_i, X_j, X_k represent molecule species and P is a general product term. We characterize each reaction R_μ by a probability constant c_μ (rather than by a rate constant k_μ as done with ODEs) which takes into account the physical properties of molecules involved in the reaction and the temperature of the system. The so called *fundamental hypothesis* states:

$$c_\mu dt \equiv \text{the average probability that a specific combination of } R_\mu \text{ substrate molecules react in the next infinitesimal time interval } dt \text{ [80]} \tag{2.42}$$

Given, for instance, a reaction $R_\mu : X_1 + X_2 \to P$, we have that, if every 1-2 collision leads to an R_μ reaction, than quantity $c_\mu dt$ corresponds exactly to the quantity defined by Equation (2.34). However, real reactions occur only if the kinetic energy of the collision exceeds some "activation energy" u_μ^*, thus the reaction probability must be diminished of a factor $e^{-u_\mu^*/kT}$ [79]. Putting together all these considerations, we conclude that, for a reaction $R_\mu : X_1 + X_2 \to P$, if conditions of thermal equilibrium prevail for species X_1 and X_2, than the quantity defined in (2.42) indeed exists and the reaction parameter c_μ is given by [79]:

$$c_\mu = \frac{\pi r_{12}^2 \sqrt{8kT/\pi m_{12}}}{V} \cdot e^{-u_\mu^*/kT}. \tag{2.43}$$

Notice that, the actual calculation of c_μ is often hard to deal with. Sometimes it is easier to determine this constant experimentally instead of theoretically, but such kinds of experiments require a deep knowledge of microscopic interactions, thus approximations and empirical values are usually employed.

In [79] the relationship between the reaction parameter c_μ and the reaction constant k_μ (employed in the deterministic modeling of chemical kinetics) is analyzed. It turns out that, for a simple bimolecular reaction $R_\mu : X_1 + X_2 \to P$, we have $k_\mu \doteq V \cdot c_\mu$. This is because for (2.42) the probability that an R_μ reaction occurs somewhere in volume V in the next infinitesimal time dt is $X_1 X_2 c_\mu dt$. From this, we may infer that $\overline{X_1 X_2 c_\mu} = \overline{X_1 X_2} c_\mu$ is the average rate at which R_μ reactions occur inside V, where the average is here considered over an ensemble of stochastically identical systems. The average reaction rate per unit volume is thus $\overline{X_1 X_2} c_\mu / V$, or, if we use molecule concentrations $x_i = X_i/V$ instead of number of molecules X_i, it is $\overline{x_1 x_2} V c_\mu$. Now, the reaction rate constant k_μ is defined as the average reaction rate per unit of volume divided by the product of the average densities of reactants, that is, $k_\mu = \overline{x_1 x_2} V c_\mu / \bar{x}_1 \bar{x}_2$. Since in the deterministic formulation it is assumed that $\overline{x_1 x_2} = \bar{x}_1 \bar{x}_2$, the equation $k_\mu \doteq V \cdot c_\mu$ is achieved. If we multiply the expression for c_μ in Equation (2.43) by V we get the well-known formula for the reaction rate constant for hard-sphere bimolecular reaction [175]:

$$k_\mu = \pi r_{12}^2 \sqrt{8kT/\pi m_{12}} \cdot e^{-u_\mu^*/kT}. \tag{2.44}$$

2.2.3 Computing the stochastic dynamics of biochemical systems

Given a stochastic model of a biochemical system, namely, the set of reactions involved in the system and their probability constants c_μ, now we want to compute

the time evolution of each molecular species, starting from some initial condition. Differently from deterministic models, the dynamics generated by a stochastic model is never exactly identical, since it only satisfies the (probabilistic) fundamental hypothesis (2.42) respect to a given set of constants $c_\mu,\ \mu = 1, \ldots, m$ (where m is the number of reactions involved in the system). Two main approaches have been devised in order to characterize the stochastic time evolution, the *master equation* [156] and the *stochastic simulation approach* [79, 80]. They are equivalent, since they both originate from the fundamental hypothesis (2.42), but, as explained in the following, the second approach provides a tractable step-by-step procedure to compute the dynamics, which is not achievable by the first approach.

Master equation approach. Stochastic models of chemical reaction systems usually employ the *grand probability function* $P(X_1, X_2, \ldots, X_n; t)$ to describe the probability that X_1 molecules of type 1, X_2 molecules of type 2, ... , X_n molecules of type n will be in volume V at time t. If this function is known, the state of the system at time t can be characterized by the following equation

$$\overline{X_i(t)} = \sum_{X_1=0}^{\infty} \ldots \sum_{X_n=0}^{\infty} X_i P(X_1, \ldots, X_n; t) \quad i = 1, \ldots, n \tag{2.45}$$

which gives the average number of molecules of type i in V at time t, over many repeated runs from 0 to t starting from the same initial conditions. This average number of molecules is thus computed by multiplying, for each possible state $(X_1, \ldots, X_n)$, the probability to have this state at time t (i.e., $P(X_1, \ldots, X_n; t)$), by the amount of molecules of type i (i.e., X_i) in the same state, and then summing together all these products.

The *master equation* [156] is a first-order differential equation representing the time evolution $\frac{\partial}{\partial t} P(X_1, X_2, \ldots, X_n; t)$ of the grand probability function. The main problem of this equation is that it is analytically solvable only in a few and very simple cases, even fewer than the number of problems for which differential models are analytically solvable. Moreover, the master equation turns to be even numerically intractable in the majority of cases due to the high number of its independent variables [79]. Therefore, despite its exact and elegant formulation, master equation is usually unemployable for numerical simulations.

Stochastic simulation approach. In order to build a tractable procedure for the stochastic simulation of biochemical systems, Gillespie [79] proposed a new approach based on the the *reaction probability density function* instead of on the master equation. While the master equation answers to the question: "given the state $(X_1, X_2, \ldots, X_n)$ at time t, which will be the stochastic state at time $t+dt$?", the reaction probability density function answers to the questions: "given the current state $(X_1, X_2, \ldots, X_n)$, when will the next reaction occurs?" and "what kind of reaction will be?" The reaction probability density function $P(\tau, \mu)$ is a joint probability density function defined as

$$\begin{aligned} P(\tau, \mu)d\tau \equiv\ & \text{probability at time } t \text{ that the next reaction in } V \text{ will} \\ & \text{occur in the next time interval } (t + \tau,\ t + \tau + d\tau) \\ & \text{and will be an } R_\mu \text{ reaction [79].} \end{aligned} \tag{2.46}$$

The simulation procedure we are looking for needs a legitimate method to derive values τ and μ from the fundamental hypothesis (2.42). Let us start by generating an analytical expression for $P(\tau, \mu)$. We define h_μ as the number of combinations of the reactants of R_μ in the state $(X_1, X_2, \ldots, X_n)$, for each reaction R_μ, $\mu = 1, \ldots, m$. This quantity depends on the reaction type, and for reactions (2.35) - (2.41) is, respectively [79]

$$
\begin{aligned}
h_\mu &= 1 && \text{for reaction (2.35),} && (2.47)\\
h_\mu &= X_j && \text{for reaction (2.36),} && (2.48)\\
h_\mu &= X_j X_k && \text{for reaction (2.37),} && (2.49)\\
h_\mu &= X_j(X_j - 1)/2 && \text{for reaction (2.38),} && (2.50)\\
h_\mu &= X_i X_j X_k && \text{for reaction (2.39),} && (2.51)\\
h_\mu &= X_j X_k (X_k - 1)/2 && \text{for reaction (2.40),} && (2.52)\\
h_\mu &= X_j(X_j - 1)(X_j - 1)/6 && \text{for reaction (2.41).} && (2.53)
\end{aligned}
$$

Now, let us compute the probability in (2.46) as the product of two terms, namely, the probability $P_0(\tau)$ that no reaction will occur in the time interval $(t, t + \tau)$, and the probability $h_\mu c_\mu d\tau$ that a reaction R_μ will occur in the next differential time interval $(t + \tau, t + \tau + d\tau)$

$$P(\tau, \mu)d\tau = P_0(\tau) \cdot h_\mu c_\mu d\tau. \tag{2.54}$$

To calculate $P_0(\tau)$ we divide the interval $(t, t + \tau)$ into K subintervals of length $\epsilon = \tau/K$. The probability that none of reactions $R_1, \ldots, R_m$ occurs in the first interval $(t, t + \tau)$ is, by the multiplication theorem for probabilities,

$$\prod_{j=1}^{m}(1 - h_j c_j \epsilon + o(\epsilon)) = 1 - \sum_{j=1}^{m} h_j c_j \epsilon + o(\epsilon) \tag{2.55}$$

The same thing is valid for the second interval $(t + \epsilon, t + 2\epsilon)$, and so on. By multiplying these probabilities for all the K subintervals it turns out that

$$
\begin{aligned}
P_0(\tau) &= (1 - \sum_{j=1}^{m} h_j c_j \epsilon + o(\epsilon))^K && (2.56)\\
&= (1 - \sum_{j=1}^{m} h_j c_j \tau/K + o(K^{-1}))^K && (2.57)
\end{aligned}
$$

which becomes, to the limit of infinitely large K

$$P_0(\tau) = e^{-\sum_{j=1}^{m} h_j c_j \tau}. \tag{2.58}$$

By substituting Equation (2.58) in Equation (2.54) we obtain the following expression for the reaction probability density function:

$$P(\tau, \mu) = \begin{cases} a_\mu e^{-a_0 \tau} & \text{if } 0 \leq \tau < \infty \text{ and } \mu = 1, \ldots, m \\ 0 & \text{otherwise} \end{cases} \tag{2.59}$$

where $a_\mu = h_\mu c_\mu$, for $\mu = 1, \ldots, m$ and $a_0 = \sum_{i=1}^{m} a_i$. Figure 2.8 displays some trends of the reaction probability density function $P(\tau, \mu)$ for different reactions R_μ.

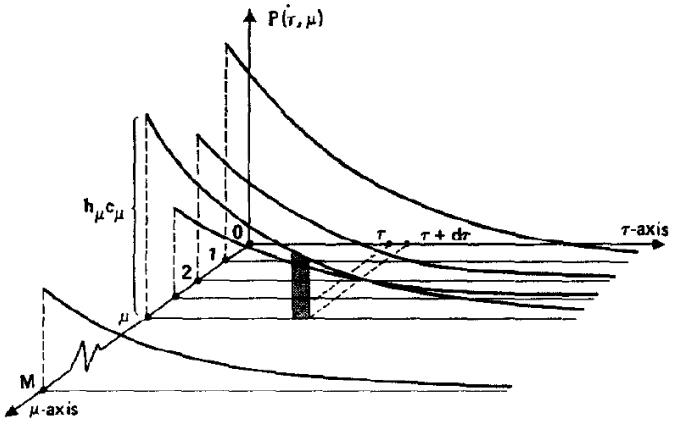


Fig. 2.8. Plotting of reaction probability density function $P(\tau, \mu)$ given in Equation (2.59). The shaded area represents the probability that reaction R_μ occurs in the next time interval $(t + \tau,\ t + \tau + d\tau)$ given the current state $(X_1, X_2, \ldots, X_n)$ [79].

Our goal is now to find a method for simulating the time evolution of biochemical systems according to the reaction probability density function $P(\tau, \mu)$ defined in Equation (2.59). The simulation algorithm proposed by Gillespie in [79] is a Monte Carlo technique based on the generation of random pairs (τ, μ) whose probability density function is $P(\tau, \mu)$. There exists a mathematical rigorous procedure for getting two random numbers r_1 and r_2 from the unit interval uniform distribution, and constructing from them a random pair (τ, μ) from a set described by any specified pair probability density function. For the pair probability density function $P(\tau, \mu)$ in Equation (2.59) the construction procedure turns out to be the following: given two random numbers r_1 and r_2 in the unit interval we can compute τ as

$$\tau = (1/a_0) ln(1/r_1), \tag{2.60}$$

while μ is the integer number for which

$$\sum_{i=1}^{\mu-1} a_i < r_2 a_0 \leq \sum_{i=1}^{\mu} a_i. \tag{2.61}$$

The reader may refer to [79] for a rigorous proof of these formulae. Here we just hint that Equation (2.60) generates a random number τ according to the probability density function $P_1(\tau) = a_0 e^{-a_0 \tau}$, while Equation (2.61) generates a random integer μ according to the probability density function $P_2(\mu) = a_\mu / a_0$. The pair generation follows because $P_1(\tau) \cdot P_2(\mu) = P(\tau, \mu)$ [80].

Since pseudo-random number generators are implemented in many programming languages, this approach brings to an implementable procedure for stochastically generating time evolutions of chemical reaction systems. In the following the

pseudo-code of the Stochastic Simulation Algorithm (SSA) proposed by Gillespie is reported [79].

Stochastic Simulation Algorithm (SSA) [79]

Initialization. Input reaction constants $c_1, \ldots, c_m$, and initial amount of substances $X_1, \ldots, X_n$. Set $t = 0$.

while (Halting condition is not met) **do**

Step 1. Compute $a_1 = h_1 c_1, \ \ldots, \ a_m = h_m c_m$, for the current state $(X_1, \ldots, X_n)$ by using h_μ as defined in (2.47) - (2.53) and calculate $a_0 = \sum_{i=1}^{m} a_i$.

Step 2. Generate random numbers r_1 and r_2 in the unit interval and compute τ and μ according to (2.60) and (2.61).

Step 3. Increase t by τ and update the substance population levels according to the application of reaction R_μ, namely, subtracting reactants and adding products according to their stoichiometry in R_μ.

end while

return time evolution of $(X_1, \ldots, X_n)$.

This algorithm has many *advantages*: *i)* it is exact respect to the fundamental hypothesis (2.42), thus it takes into account stochastic fluctuations and correlations; *ii)* it never approximates infinitesimal time intervals by discrete time steps as numerical methods for differential models do; *iii)* it is very easy to implement and it does not require a big amount of memory. On the other hand, this procedure has also some *drawbacks*: *i)* it is very time expensive since it simulates, one by one, every single reaction. This is a strong limitation on the total amount of molecules that can be simulated, since the larger the number of molecules, the higher the number of reactions that occur in the time unit; *ii)* stochastic simulations have to be repeated many times to achieve average trends. This is also time expensive and, moreover, it seems to nullify the advantage of preserving natural fluctuations. In fact, this is not completely true, since fluctuations keep additional information about the standard deviation of the simulated dynamics, which are not considered by deterministic models.

In order to overcome these drawbacks an approximate procedure, called *τ-leap*, has been presented in [81]. This method computes, in a fast but approximated way, how many times each reaction fires in a discrete time interval τ, where τ has to satisfy the so called *leap condition* [81] in order to reach some given approximation error. In this way the algorithm does not simulate each single reaction but it groups together the effect of all the reactions which fire during each time interval τ. It can be proved that, if enough time intervals contain many reaction events, then the speed of the algorithm can increase substantially [81].

The choice between the deterministic and the stochastic approach depends on the system to be analyzed. In fact, when small molecular populations are involved, fluctuations become fundamental elements of the time evolution, because they yield substantial deviations from the average dynamics. In these cases the stochastic approach is a good choice, since it takes into account fluctuations and correlations, and the simulation algorithm does not require too much time to be executed. Many real-world biochemical systems, however, involve huge amounts of molecules whose dynamics has deterministic average trends which are negligibly influenced by stochastic fluctuations. In such cases the stochastic simulation algorithm requires too much time to be run on standard computers, while deterministic approaches, such as differential models, represent a feasible way to capture average time evolutions.

However, we observe that both the approaches require kinetic constants, namely, k_μ for differential models and c_μ for stochastic models, which are intrinsically related to a microscopic and instantaneous description of the system under investigation. These constants are often difficult to determine since experiments are based on discrete and macroscopic observations of the system. Therefore, kinetic constants are computed as mathematical approximations, to the limit of infinitesimal time intervals, of the system behavior observed in discrete time instants. Moreover, since experimental observations are macroscopic and representative of the behavior of the whole system, another approximation is done when the local internal dynamics of each reaction is deduced from these observations. In Chapter 4 we present a new modeling approach which aims to overcome the drawbacks of using kinetic constants by considering the system at a high abstraction level, but sufficiently low to reveal the logic of observed behaviors.

2.2.4 A simple example: the irreversible isomerization

In this section we report an application of the SSA to the simple case study of irreversible isomerization (or radioactive decay), represented by reaction

$$X \xrightarrow{c} Z. \tag{2.62}$$

This example, firstly presented in [80], copes with a very simple system whose dynamics can be computed analytically by both master equation and reaction rate equation. The *master equation* for this particular case study has the following form [80]

$$\frac{\partial}{\partial t}P(X;t) = c[\epsilon_{X,X_0}(X+1)P(X+1;t) - XP(X;t)] \tag{2.63}$$

where X is the amount of reactant molecules in the system, c is the stochastic constant for isomerization and $\epsilon_{i,j}$ is the 'Kronecker epsilon", defined as $\epsilon_{i,j} = 0$ if $i = j$ and $\epsilon_{i,j} = 1$ otherwise. The solution of this (quite complex) differential equation gives the probability function $P(X;t)$ of having X reactant molecules at time t, for $X = 0, 1, \ldots, X_0$, that is [80]

$$P(X;t) = \frac{X_0!}{X!(X_0 - X)!}e^{-cXt}[1 - e^{-ct}]^{X_0 - X} \quad (X = 0, 1, \ldots, X_0) \tag{2.64}$$

for the initial condition $P(X,0) = \delta_{X,X_0}$.

The mean and standard deviation of the binomial probability function in Equation (2.64) turn out to be, respectively

$$\overline{X(t)} = X_0 e^{-ct} \quad \text{and} \quad \sigma(t) = [X_0 e^{-ct}(1 - e^{ct})]^{1/2} \tag{2.65}$$

On the other hand, by modeling the same chemical reaction by the deterministic approach, we achieve the following *differential equation*

$$\frac{dX}{dt} = -cX \tag{2.66}$$

whose solution is, for $X = X_0$ at time $t = 0$

$$X(t) = X_0 e^{-ct} \tag{2.67}$$

which corresponds exactly to the mean of the master equation solution (2.65).

Now, let us generate a stochastic time evolution of the isomerization process by SSA, employing $c_j = 0.5$ as reaction constant and $X_0 = 1000$ as initial condition. The bold-dotted line of Figure 2.9 represents the trajectory of the stochastic simulation, while dashed and solid lines surround, respectively, the area between $\overline{X(t)} \pm \sigma(t)$ and $\overline{X(t)} \pm 2\sigma(t)$. Notice that, the horizontal separation between the time evolution dots gives a direct measure of the time interval between two successive reactions, which increases as long as the population size X decreases.

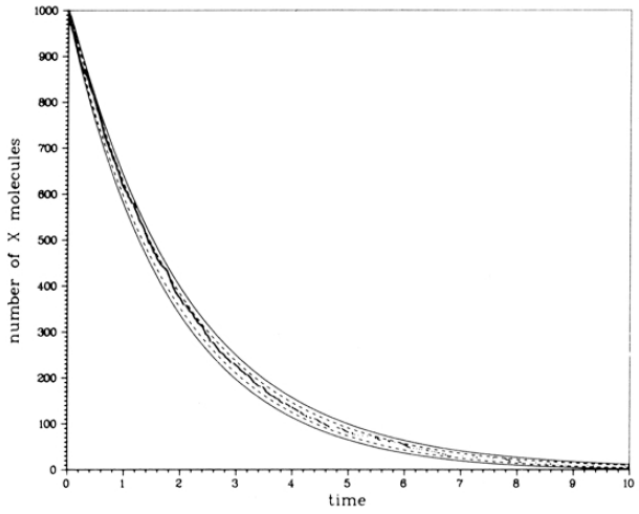


Fig. 2.9. Time evolution (bold-dotted line) of isomerization computed by SSA with $c = 0.5$ and initial condition $X_0 = 1000$. Standard deviation envelope $\overline{X(t)} \pm \sigma(t)$ (dashed line) and two-standard deviation envelope $\overline{X(t)} \pm 2\sigma(t)$ (solid line) [79].

Figure 2.10 displays the simulation trajectory (plotted after every 10 reaction occurrences) and the standard deviation envelopes when the initial condition is

$X_0 = 10000$. We observe that the increased number of molecules makes the simulation chart match the continuous deterministic evolution described by reaction rate Equation (2.66), since the fluctuation contribution becomes negligible when the population size grows. Moreover, stochastic simulations fulfill the temporal behavior predicted by the master equation regardless of the initial number of molecules, indeed bold-dotted time evolutions keep inside the standard deviation envelopes in both Figures 2.9 and 2.10.

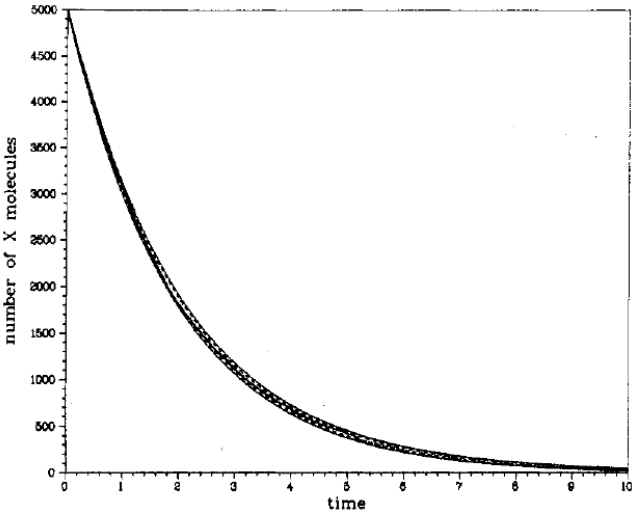


Fig. 2.10. Time evolution (bold-dotted line) of isomerization computed by SSA with $c = 0.5$ and initial condition $X_0 = 10000$. Standard deviation envelope $\overline{X(t)} \pm \sigma(t)$ (dashed line) and two-standard deviation envelope $\overline{X(t)} \pm 2\sigma(t)$ (solid line) [79].

This example shows that deterministic and stochastic approaches converge as the number of interacting molecules increases. However, in nature there exist several biological systems involving low numbers of molecules [212, 246]. For these systems stochastic and deterministic models present inherently different behaviors that do not converge even when averages of the stochastic dynamics are considered. As mentioned at the end of Subsection 2.2.3, stochastic or hybrid models are better suited for analyzing these systems.

3

P systems for modeling biological systems

In this chapter P systems are introduced and some literature is reviewed concerning P systems extensions for biochemical systems modeling. Although these contents do not constitute the original part of this Ph.D. thesis, they have been detailed in order to present to the reader the main contributions in this research field. In Section 3.1 the main elements of a P system are described, namely, membrane structure, multisets of atomic objects, rewriting rules and rule application strategy. A formal definition of *transition P systems* is reported in Subsection 3.1.1, while Subsection 3.1.2 presents the main extensions of this model, concerning respectively, rules, membrane activities, multisets of objects, and membrane arrangements.

P systems have been deeply investigated in two directions, namely, computational power and possible applications as modeling framework. The main results about the first topic are summarized in Subsection 3.1.3, while the second topic is dealt with in Sections 3.2, 3.3 and 3.4. In particular, the first of these sections introduces a P system extension called *stochastic P systems*, the second presents *dynamical probabilistic P systems*, and the third copes with *abstract rewriting systems on multisets*.

3.1 P systems

Membrane computing [44, 184, 185, 223] is a branch of natural computing aiming to abstract computing strategies and models from structure and functioning of the living cell. Membrane systems, usually called *P systems* from the name of G. Păun who devised them in 1998 [182], represent a novel computational model originated from the prominent role played by membranes in the living cell [4]. In fact, membranes do not only act as separation barriers indispensable to create different environments within cell boundaries, but they also constitute fundamental layers whereby the cell communicates with neighboring cells [4, 151, 181], accounts energy [187] and selects chemicals to keep on vital cycles. Moreover, membranes often represent some kind of "working board" in which enzymes can perform their activities. P systems represent an application of the membrane framework to contexts of *formal language theory* [100, 201]. The result is a discrete non-conventional model based on three main elements: *i)* a hierarchical *membrane structure*, *ii) multisets*

of objects and *iii)* sets of *rewriting rules*. Many results about the computational universality of this approach have been achieved [185]. In the following we detail the fundamental elements listed above and we present some strategies for rule application.

Membrane structure. Looking at the cell structure through "mathematical glasses" [44] one observes a hierarchical compartmentalization of the 3D space, which follows the arrangement showed in Figure 3.1. The most external element of this Euler-Venn representation is a *skin* membrane, corresponding to the plasma membrane which separates the environment out of the cell from the environment inside the cell. Membranes arranged inside the skin membrane further compartmentalize the space in a hierarchical way, since each membrane is arranged inside another. A membrane without any other membranes inside it is said to be *elementary*. Each membrane is usually identified by a *label*, which can be a natural number (as in Figure 3.1) or a name string. In some cases a label can be assigned to many membranes of the same "type".

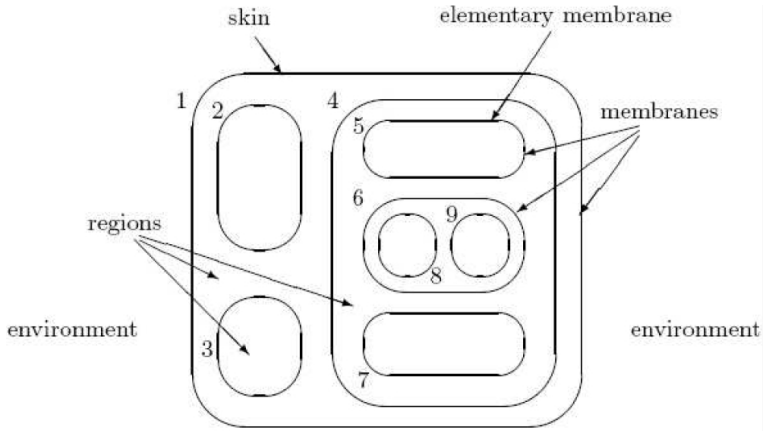


Fig. 3.1. Euler-Venn representation of the hierarchical membrane structure of a P system [44].

Other two useful representations of P system membrane structure are the rooted tree, displayed in Figure 3.2, and the string of labeled matching parentheses, reported in the following

$$[_1\ [_2\ ]_2\ [_3\ ]_3\ [_4\ [_5\ ]_5\ [_6\ [_8\ ]_8\ [_9\ ]_9\ ]_6\ [_7\ ]_7\ ]_4\ ]_1.$$

Each of these the representations described so far highlight different features of membrane structures. In the following we adopt the compact and elegant parentheses representation.

Multisets of atomic objects. Inside cellular membranes, chemical elements (ions, small molecules and macromolecules) freely move in the aqueous solution performing brownian motion. P systems abstract this condition by neglecting the position and the velocity of each particle but considering only the number of copies

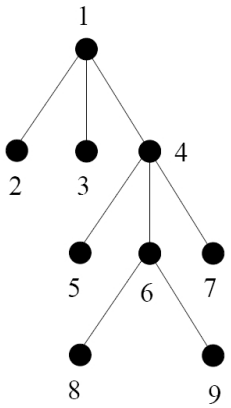


Fig. 3.2. Tree representation of the hierarchical membrane structure of a P system [44].

of each chemical species present in each membrane. This is done by associating a *multiset* of objects to each membrane, namely, a set of objects wherein the multiplicity of each object matters. For instance, if we consider an alphabet of objects $V = \{a, b, c\}$, a possible multiset on V is the string $a^4b^9c^2$, where each exponential represents the multiplicity of the related symbol, thus we have four elements of type a, nine of type b, and two of type c. Notice that multisets can be straightforwardly represented by strings in which the order of symbols does not matter. In the following a formal definition of multiset is reported [17].

Definition 1 (Multiset) *Let V be an alphabet, a (finite)* multiset *over V is a mapping $M : V \to \mathbb{N}$, in which $\mathbb{N}$ denotes the set of natural numbers. For each $a \in V$, $M(a)$ is the multiplicity of a in M.*

Rewriting rules. From biological background we know that chemical reactions among molecules occur in cell compartments. P systems abstract chemical reactions by associating a set of multiset-rewriting rules to each membrane. A rewriting rule is usually represented by the arrow notation $r : u \to v$, where u and v are multisets of objects. For instance, applying the rule $r : \ a^2b \to c^3$ to the multiset $a^4b^9c^2$, two objects of type a and one object of type b are consumed and three objects of type c are produced, thus obtaining the new multiset $a^2b^8c^5$.

Since rules are associated with regions, some kind of matter exchange is needed to make the compartments communicate. For this reason rewriting rules have been equipped with *target* indications, which are labels indicating where each product object has to move after the application of the rule. In the *transition P systems* [44], formally defined in the next section, three target indications have been considered: *here* is associated to objects remaining in the membrane where the rule occurs, *in* indicates that the object has to go into an adjacent membrane nondeterministically chosen among the membranes contained in the compartment where the rule occurs, *out* states that the object has to exit the membrane where the rule is applied. As an example, the rewriting rule $aab \to (a, here)(b, out)(c, in)$ transforms two objects of type a and one object of type b into one object of type a, which remains in the same membrane, one object of type b, which goes out

to the surrounding membrane, and one object of type c, which enters one of the membranes contained in the current one. Notice that, label *here* can be omitted.

A rewriting rule having at least two objects in the left hand side is said *cooperative*, while rules of the form $r : a \to v$, where a is an object and v is a multiset of objects, are called *non-cooperative*. A particular type of cooperative rules are those in the form $r : ca \to cv$, that are called *catalytic* since a catalyst c enables the reaction without being transformed.

Besides the traditional multiset-rewriting rules described so far, two other types of rules have been employed in P systems: *communication rules*, abstracting the symport/antiport trans-membrane communication in cell, and *rules for handling membranes*, which enable the evolution of the membrane structure (e.g., membrane creation, dissolution, division). Here we only consider *membrane dissolution* rules, having the form $u \to v\delta$, where δ indicates that the membrane hosting the rule is dissolved after the rule application, thus the membrane content is released in the surrounding membrane.

Rule application strategy. When many membranes are present in a P system and several rules are associated to each membrane, a strategy is needed to choose which rule to apply at a certain computational step. From this point of view P systems do not abstract any biological law, while rules are chosen in a *nondeterministic and maximally parallel way*, which means that objects are assigned to rules by choosing nondeterministically both objects and rules, until no further assignment is possible [44, 185].

In other words, an evolution step in a given region consists of finding a maximal applicable multiset of rules, removing from the region the objects specified in the left hand sides of the selected rules (multiplied by the number of times each rule has been selected), producing the objects from the right hand side of the rules, and distributing these objects according to the target indications associated to them by each rule. If any dissolution rule has been applied, then the membrane is deleted and its content released into the parent membrane.

3.1.1 Transition P systems

Transition P systems are one of the three main types of cell-like P systems. They employ only multiset-rewriting rules. The other two main classes of cell-like P systems are *P systems with symport/antiport rules* and *P systems with active membranes*, which will be hinted in the next section. A comprehensive introduction to all these models can be found in [44].

The formal definition of transition P systems, reported below, basically involves the three main P system elements, namely, a membrane structure, a multiset of objects for each region, and a set of object-rewriting rules for each region.

Definition 2 (Transition P Systems) *A* transition P system *of degree* $m \geq 1$ *is a construct [44]*

$$\Pi = (O, C, \mu, w_1, w_2, \ldots, w_m, R_1, R_2, \ldots, R_m, i_0)$$

where:

- O *is a (finite and non-empty) alphabet of* atomic objects*;*
- $C \subset O$ *is the set of* catalysts*;*
- μ *is a membrane structure, consisting of* m *membranes labeled* $1, 2, \ldots, m$ *in a system of degree* m*;*
- $w_1, w_2, \ldots, w_m$ *are strings over* O *representing the* multiset of objects *present in regions* $1, 2, \ldots, m$ *of the membrane structure;*
- $R_1, R_2, \ldots, R_m$ *are finite* sets of evolution rules *associated with regions* $1, 2, \ldots, m$ *of the membrane structure. Notice that, rules are in the form* $u \to v$ *or* $u \to v\delta$*, with* $u \in O^+$ *and* $v \in (O \times Tar)^*$*, where* $Tar = \{here,\ in,\ out\}$ *are target indications. Symbol* δ *indicates membrane dissolution;*
- i_0 *is either one of the labels* $1, 2, \ldots, m$*, and the respective region is the* output region *of the system, or it is* 0*, and the result of a computation is collected in the environment of the system.*

Figure 3.3 shows a simple transition P system having three membranes. The *configuration* displayed in the picture, namely, the membrane structure and the multisets of objects enclosed in each compartment, involves three membranes and only two objects placed in membrane 3. Rewriting rules are included in the membrane which they belong to.

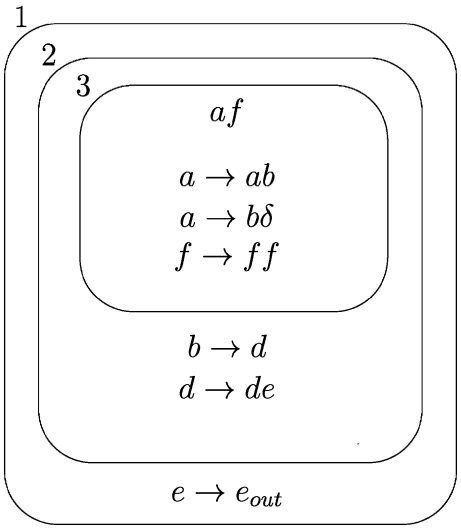


Fig. 3.3. Graphical representation of a transition P system [44].

When P systems are employed as computational tools, an *initial configuration* is arranged and the system is evolved by means of a set of *transitions*, namely, transformations of configurations due to the application of rules in a *nondeterministic and maximally parallel way*. Notice that, in the basic variant of P systems a global clock is assumed which beats the time in all regions.

A sequence of transitions is called *computation*, and a computation is considered successful if it *halts* after a certain number of steps, that is, no rule can be applied any more and the output region i_0 still exists. In that case, the *result* of

the computation can be defined in two ways: if the output region i_0 is an internal region, then we have an *internal result*, which is constituted by the objects present in region i_0 when the halting configuration is reached. If $i_0 = 0$ then the *external result* involves the objects that have exited the system during the computation. Of course, non-halting computations and computations wherein the output region is dissolved, do not provide any result.

Since rules are applied non-deterministically, many halting configurations can be reached from the same initial configuration, therefore several results can be found. Thus, we can say that a P system Π *computes* a language $L(\Pi)$ composed by the set of object multisets it can generate. We observe that, in some cases the result is considered as the number of objects, or the vector of multiplicities of objects, contained in the output region. In those cases the P system generates, respectively, a set of numbers, denoted by $N(\Pi)$, and a set of vectors $Ps(\Pi)$, called Parikh vectors [184]. For instance, a P system Π which generates the set of squares of natural number have $N(\Pi) = \{n^2 \mid n \geq 1\}$.

3.1.2 P systems extensions

The standard formulation of P systems, described so far, has been extended in several ways in order to make it closer to the cell reality. In this section we only give some hints about a few of these extensions, while a more detailed description can be found in [44]. Extensions basically concern the three main elements of P systems, namely, membrane, rules and objects. Notice that the structure wherein membranes are arranged can be modified as well.

Rule extensions

An important type of cell-like P systems is called *P systems with symport/antiport*. In this framework, rules are extended in order to abstract the passage of substances through membranes. Specifically, the following rules are employed [44]:

- *symport rules*, have the form (ab, in) or (ab, out), where $a, b \in O$, and they abstract the process by which two molecules, respectively, enter and exit together the membrane (through a specific protein channel);
- *antiport rules*, have the form $(a, out; b, in)$, where $a, b \in O$, and they abstract the the process wherein two molecules pass simultaneously through the protein channel in opposite directions. In this case object a exits the membrane while object b enters the membrane;
- *uniport rules*, have the form (a, in) or (a, out), where $a, b \in O$, and they abstract the passage of single molecules across the membrane, respectively, in entrance and exit.

Evolution-communication P systems [37] use traditional multiset rewriting rules (without target indications) for object evolution and symport/antiport rules for moving objects among membranes.

Notice that, the target indication $in \in Tar$ introduces a degree of indeterminism in the choice of the internal membrane where to place produced objects. This indeterminism can be removed considering indications in the form in_j, where

$j \in 1, \ldots, m$, is a membrane label. Other extensions assign *electrical polarizations* +, -, 0 to both objects and membranes and force charged objects to go to any inner membrane having an opposite polarization, and neutrally polarized objects to stay in the same region or to exit it, depending if the target indication is *here* or *out*. Further extensions consider *priority* relations among rules, or rules controlled by *promoters* and *inhibitors*, as observed in real biochemical systems [44].

Membrane extensions

Membrane systems considered so far have a fixed membrane structure, but looking at cell membranes one observes they evolve either by changing their features or by dividing. *P systems with active membranes* [178, 183, 251] take into account this feature by introducing the following *developmental rules*, where H is a finite set of membrane labels [44]:

- *object evolution rules*: $[_h a \rightarrow v]^e_h$, for $h \in H,\ e \in \{+, -, 0\},\ a \in O,\ v \in O^*$. They define the transformation of object a to the multiset v in membrane h, if it has charge e;
- *in communication rules*: $a[_h]^{e_1}_h \rightarrow [_h b]^{e_2}_h$, for $h \in H,\ e_1, e_2 \in \{+, -, 0\},\ a, b \in O$. They define the introduction of object a in membrane h if it has a charge e_1. The object is possibly modified to b and the membrane polarization changed to e_2, during the application of the rule;
- *out communication rules*: $[_h a]^{e_1}_h \rightarrow [_h]^{e_2}_h b$, for $h \in H,\ e_1, e_2 \in \{+, -, 0\},\ a, b \in O$. Object a is sent out of membrane h and possibly modified to b. The polarization, which has to be e_1 before the rule application, can be changed to e_2;
- *dissolving rules*: $[_h a]^e_h \rightarrow b$, for $h \in H,\ e \in \{+, -, 0\},\ a, b \in O$. Membrane h, having charge e, is dissolved and internal object a is transformed to b.
- *division rules for elementary membranes*: $[_h a]^{e_1}_h \rightarrow [_h b]^{e_2}_h [_h c]^{e_3}_h$, for $h \in H,\ e_1, e_2, e_3 \in \{+, -, 0\},\ a, b, c \in O$. Membrane h, having charge e_1 and an internal object a, is divided into two membranes having the same label h and polarizations e_2 and e_3, respectively. Object a is replaced by object b in the first new membrane, and by object c in the second new membrane. The remaining objects are duplicated and they may evolve by object evolution rules in the same step.

These rules are applied in the standard nondeterministic maximally parallel way, paying attention to apply first the object evolution rules (first type) and then the rules of the other types. Notice that, when dissolution and division rules are employed the membrane structure is changed during the computation. Other types of rule regarding membrane structure update are considered in [44], among them, rules for membrane creation, rules for merging two membranes, rules of endocytosis/exocytosis and gemmation [15].

Object extensions

The idea of considering structured objects, such as string objects, instead of atomic objects, comes from the observation that in the cell many complex molecules (e.g.,

proteins, DNA molecules and other large molecules) exist, whose structure matters during the system evolution. In general P systems objects could be represented by many complex data structures, such as trees, arrays, etc., however, many valuable properties have been proved for *P systems with string objects* [150,179]. They are defined by means of an alphabet V of strings, a terminal alphabet $T \subseteq V$ and a finite set R_i of string-processing rules for each membrane $i = 1, \ldots, m$, where m is the number of membranes in structure μ. In this approach, object multisets $w_1, \ldots, w_m$ (typical of standard P systems) are replaced by finite sets of strings $M_1, \ldots, M_m$.

In *rewriting P systems* [62] string objects are processed by rules of the form $a \to u(tar)$, where $a \to u$ is a context-free rule over V and tar is a target indication among $\{here, in, out\}$. When such a rule is applied to a string $x_1 a x_2$, where x_1, x_2 are string over V, the result is the string $x_1 u x_2$, which is placed, depending on the target indication, respectively in the same region, in a lower region or in a surrounding region.

Membrane arrangement extensions

The tree membrane structure of traditional cell-like P systems can be generalized by a graph structure, in which membranes are represented by nodes, while arches represent spatial relationships between membranes. In particular, let us consider a tissue-like structure, in which each cell communicates with adjacent cells by common protein channels, enabling molecules transportation. The P system class we obtain by abstracting this framework is called *tissue-like P systems with channels states* [74]. In this approach, m membranes (called cells in this context) containing, respectively, object multisets $w_1, \ldots, w_m$, are connected by a set of synapses $syn \subseteq \{(i,j) | i,j \in \{0,1,2,\ldots,m\}, i \neq j\}$, where at most one of (i,j) and (j,i) is present in syn. Each synapse is, at each computational step, in a state $s_{(i,j)} \in K$, where K is the alphabet of states. Moreover, each synapse $(i,j) \in syn$ has an associated set of rules $R_{(i,j)}$ of the form $(s, x/y, s')$ for some $s, s' \in K$ and $x, y \in O^*$. These rules can occur only if the synapse (i,j) is in state s. The rule moves the multiset x from membrane i to membrane j, and the multiset y from membrane j to membrane i. Moreover, it changes the synapse state from s to s'.

A computation starts from an initial configuration in which multisets $w_1, \ldots, w_m$ are present, respectively, in the m cells. At each step, a rule is applied to each synapse for which a rule can be applied. In this way rules are sequentially applied in every synapse but they are applied in parallel from a system point of view. Result is taken from the output cell, identified by the label $i_0 \in \{1, 2, \ldots, m\}$ when the computation halts.

A further extension of tissue-like P systems, called *neural-like P systems* [185], makes use of more complex cells however arranged in a graph structure. In this model, states are moved from synapses to cells, ensuring a powerful control of object flows through the system, thus making these approach very powerful and efficient. In particular, each cell of the tissue is, at each instant, in a specific state s and it contains a certain multiset of objects. The system configuration evolves by means of rules of the form $sw \to s'(x, here)(y, go)(z, out)$, where s, s' are cell states and w, x, y, z are multisets of objects. This rule can be applied if the cell

is in the state s. It transforms the multiset w to the multisets x, y, z and updates the cell state to s'. Multiset x stays in the same cell, multiset y is communicated to the cells connected (by synapses) with the current cell, and multiset z is sent to the environment (the *out* label can appear only in "output" cells).

The computation takes place by the simultaneous processing of the multiset of every cell according to the local rules of each cell. After being transformed, objects are distributed to other cells along synapses and, possibly, results are sent to the environment. In [44] further strategies are considered for processing multisets of objects (*minimal* mode, *parallel* mode and *maximal* mode) and for distributing objects between cells (*spread* mode, *one* mode, *replicate* mode).

Some of the most recent evolutions in this field concern *population P systems* [13] and *spiking neural P systems* [75, 88, 103]. The first ones are a class of tissue P systems in which the links between cells can be modified by means of a specific set of bond making rules. The second ones incorporate the idea of spiking neurons into the area of membrane computing. An up to date bibliography of new trends in P systems research can be found in [223].

3.1.3 Universality

The computational power of different classes of P systems has been much investigated until now, since the initial goal of membrane computing was to define computability models inspired form the cell. Most of the P systems variants have proved to be Turing complete, that is, their computational power is equivalent to the power of the *universal Turing machine*. As an example, all the P systems extensions considered so far (cell-like, tissue-like, neural-like, with symbol or string objects, etc.) are known to be universal.

An interesting topic of study regards the search of the minimal number of elements needed by each class of P systems in order to keep the universality. Thus, a typical question is the following: "Are P systems with one membrane and two catalysts universal?". In the following we report some of the main results of universality. The reader may refer to [44,185] for a comprehensive description of these results.

The following classes of P systems are universal:

- transition P systems with one membrane and two catalysts [73];
- symport/antiport P systems with three membrane and rules dealing with single objects [5];
- symport/antiport P systems with three membrane and only symport rules dealing with two objects [5];
- P systems with active membranes with three membranes having rules of the first three types (see "membrane extensions" in Section 3.1.2) [163];
- P systems with string objects with three membranes and rules with at most two copies of each string produced by replication [126];
- symport/antiport P systems with four membranes, three objects and rules of arbitrary size [186].

Computational power issues seem to be more related to computer science than to biology, but the generality of a mathematical model (its comparison with Turing

machine and its restrictions) assumes an important role when one aims at algorithmically solving questions about the model dynamics. Typical questions related to dynamical properties of biological systems concern stability, oscillations, periodicity, attractors, reachability, trajectories and answers often depends directly to the generality (and therefore the power) of the model [17].

The main issue related to the use of P systems as a modeling framework concerns the way of making the model evolve. In fact, the nondeterministic strategy is not very meaningful within the context of bio-systems modeling [140], thus some variants based on stochastic, probabilistic and deterministic evolution rules have been proposed in order to overcome this issue [39,44,77,105–108,141,142,170,171, 173,180,196,220]. In these frameworks, the P system formalism is employed as a specification framework for biological systems, while the dynamics is computed according to different strategies, such as the Gillespie's SSA algorithm, probabilistic strategies or difference equations. In the next sections, three of these variants, namely, stochastic P systems, probabilistic P systems and ARMS, will be introduced, while in the next chapter we present metabolic P systems, which is the focus of the rest of this thesis.

3.2 Stochastic P systems

Stochastic P systems [171] are an extension of traditional P systems aiming to model biochemical systems. The non-deterministic and maximally parallel strategy of rule application is replaced, in this framework, with a stochastic algorithm called Multi-Compartmental Gillespie's Algorithm (MGA). In a stochastic P model, symbols or strings represent molecular species, membranes identify compartments of the observed biochemical system and rewriting rules deal with molecule interactions. In this multiscale modeling framework, higher level structures, such as cellular colonies and tissues, are modeled as collections of individual P systems. In a recent work Romero-Campero et al. define Stochastic P systems as follows.

Definition 3 (Stochastic P Systems) *A Stochastic P system is a construct [198]:*

$$\Pi = ((\Sigma_{obj}, \Sigma_{str}), L, \mu, M_{l_1}, \ldots, M_{l_m}, (R_{l_1}^{obj}, R_{l_1}^{str}), \ldots, (R_{l_m}^{obj}, R_{l_m}^{str}))$$

where:

- Σ_{obj} *is a finite alphabet of objects representing molecular species whose internal structure is not relevant in the functioning of the system under investigation;*
- Σ_{str} *is a finite alphabet of objects representing relevant parts of some molecular species in the system. These objects are arranged into strings describing the structure of molecular species;*
- $L = \{l_1, \ldots, l_m\}$ *is a finite alphabet of symbols representing compartment labels used to identify compartment classes. Compartments having the same label share the same class, i.e., set of rewriting rules and initial multisets;*
- μ *is a membrane structure consisting of* $n \geq 1$ *membranes defining compartments identified in a one-to-one way by values from* $\{1, \ldots, n\}$ *and labeled by elements from* L*;*

- $M_{l_t} = (w_t, s_t)$, *for each* $1 \leq t \leq m$, *is the initial state of the compartments of the class identified by label* l_t, *where* $w_t \in \Sigma^*_{obj}$ *is a finite multiset of individual objects and* s_t *is a finite set of strings over* Σ_{str}. *A multiset of objects obj is represented as* $obj = o_1 + o_2 + \ldots + o_p$ *with* $o_1, \ldots, o_p \in \Sigma_{obj}$. *Strings are represented as follows* $\langle s_1 \cdot s_2 \cdot \ldots \cdot s_q \rangle$, *where* $s_1, \ldots, s_q \in \Sigma_{str}$;
- $R^{obj}_{l_t} = \{r^{obj,l_t}_1, \ldots, r^{obj,l_t}_{k_{obj},l_t}\}$, *for each* $1 \leq t \leq m$, *is a finite multiset of rewriting rules on multisets of objects associated with compartments of the type specified by the label* l_t. *The rewriting rules on multisets of objects are of the following form:*

$$r^{obj,l_t}_j : obj_1[obj_2]_l \overset{c^{obj,l_t}_j}{\rightarrow} obj'_1[obj'_2]_l \tag{3.1}$$

with $obj_1, obj_2, obj'_1, obj'_2$ *some finite multiset of objects from* Σ_{obj} *and* l *a label from* L. *These rules are multiset rewriting rules that operate on both sides of membranes, that is, a multiset* obj_1 *placed outside a membrane labeled by* l *and a multiset* obj_2 *placed inside the same membrane can be simultaneously replaced with a multiset* obj'_1 *and a multiset* obj'_2, *respectively. Note that a constant* c^{obj,l_t}_j *is associated specifically with each rule. This constant will be referred to as stochastic constant and it is key to provide P systems with a stochastic extension as it will be used to compute the probability and the time needed to apply each rule. This constant depends only on the physical properties of the molecules and compartments involved in the reaction described by the rule, such as temperature, pressure, pH, volume, etc;*
- $R^{str}_{l_t} = \{r^{str,l_t}_1, \ldots, r^{str,l_t}_{k_{str},l_t}\}$, *for each* $1 \leq t \leq m$, *is a finite set of rewriting rules on multisets of strings and objects associated with compartments of the type defined by* l_t *and of the following form:*

$$r^{str,l_t}_j : [obj + str]_l \overset{c^{str,l_t}_j}{\rightarrow} [obj' + str'; str'_1 + \ldots + str'_s]_l \tag{3.2}$$

with obj, obj' *multisets of objects over* Σ_{obj} *and* $str, str', str'_1, \ldots, str'_s$ *strings over* Σ_{str}. *These rules operate on both multisets of objects and strings. The objects obj are replaced by the objects* obj'. *Simultaneously a substring str is replaced by* str' *whereas the strings* $str'_1, \ldots, str'_s$ *are produced to form part of the content of the compartment. In the same way as for rewriting rules on multisets of objects a stochastic constant* c^{str,l_t}_j *is associated with each rule.*

The time evolution of stochastic P system is computed by the Multi-Compartmental Gillespie's Algorithm [171] which is an extension of the Gillespie's algorithm [79] for multi-compartmental environments. Gillespie's algorithm (SSA), already described in Chapter 2, is a stochastic method proved to be effective, under specific conditions, to simulate biochemical systems. It basically determines, at each computational step, the reaction which has to fire next and the time interval which has to be waited before the firing, according to specific reaction probability density functions. In order to employ this strategy within the stochastic P system framework the Multi-Compartmental Gillespie's Algorithm (MGA) [171], basically runs SSA into each membrane of the system and manages membrane interactions by a biologically-inspired strategy. In the following we outline the main steps of the MGA by using a simplified notation for membranes and rules, as described in [171].

Multi-Compartmental Gillespie's Algorithm (MGA) [171]

Initialization.
- Set the simulation time $t = 0$
- For each membrane i in μ compute the triple (τ_i, j, i) by the Gillespie's equations 3.3 and 3.4, where τ_i is the waiting time of the next reaction r_j firing in membrane i;
- Sort triples $(\tau_i, j, i),\ i = 1, \ldots, n$ according to τ_i;

while (Halting condition is not met) **do**

Step 1. Select the triple (τ_s, j, s) having the lowest τ_s

Step 2. Set time $t = t + \tau_s$;

Step 3. Update waiting times τ_i of the other triples by subtracting τ_s;

Step 4. Apply the rule r_j of membrane s updating the sizes of population affected by the rule;

Step 5. For each membrane s' affected by the rule applied at step 4, update the corresponding triple (τ'_s, j', s') with the triple (τ''_s, j'', s'') generated by the Gillespie's algorithm using the new population sizes;

Step 6. Add the new triples (τ''_s, j'', s'') to the sorted triple list and iterate the process;

end while

return Time evolution of object populations.

We remark that, given membrane i, the triples (τ_i, j, i) is computed by generating two random numbers r_1 and r_2 in the unit interval, and then calculating waiting time as

$$\tau_i = (1/a_0) ln(1/r_1), \tag{3.3}$$

and rule index j such that

$$\sum_{k=1}^{j-1} a_k < r_2 a_0 \leq \sum_{k=1}^{j} a_k \tag{3.4}$$

where $a_k = h_k c_k$ is the firing probability of the k-th rule of membrane i and a_0 is the sum of the firing probabilities of all rules in membrane i (see Section 2.2). Waiting times are then employed to select the rule which fires, in a specific membrane i, at the subsequent step. After rule firing, waiting times of membranes whose objects have been affected by the modification are recomputed by taking into account the new population sizes.

Several biological processes have been modeled by stochastic P systems [194], among them, pseudomonas quorum sensing [23], the epidermal growth factor receptor (EGFR) signaling cascade, quorum sensing system in Vibrio Fisheri [171], the FAS-induced apoptosis [180] and *Lac* operon regulation pathway [197]. In the next section we consider one of these processes as an example of modeling biological systems by stochastic P systems.

3.2.1 An example

In [198] the functioning of the *lac* operon in *Escherichia coli* (*E. coli*) is considered to highlight the main modeling principles of stochastic P systems. Here we report just a few elements of the achieved model in order to show how a stochastic P system looks like. The *lac* operon regulates the glycolitic pathway employed by some organisms to generate carbon from glucose. In particular, *E. coli* bacterium synthesizes carbon from glucose and lactose. If the bacterium grows in an environment with both glucose and lactose, then it consumes glucose, but if the environment contains only lactose, then *E. coli* synthesizes special enzymes that metabolize lactose by transforming it into glucose [4, 53]. The synthesis of these enzymes is controlled by the transcription of the *lac* operon, displayed in Figure 3.4, which consists of three genes, *lacZ*, *lacY* and *lacA*, each related to the production of an enzyme.

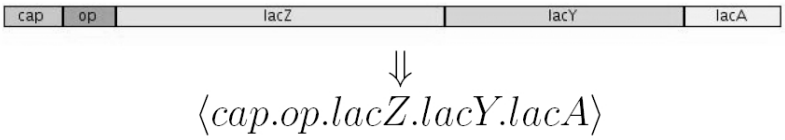


Fig. 3.4. The *lac* operon structure (top) and its representation as a string (bottom) [198].

The biological system described above has been modeled by an appropriate stochastic P system. The set of objects representing molecular species is reported in Table 3.1. The membrane structure involves two membranes identifying, respectively, the cytoplasm region and the cell-surface region, as shown in Figure 3.5. Some of the rules, representing chemical interactions and translocation among membranes, are reported with their stochastic constants, in Table 3.2.

The latest extensions of the stochastic P systems approach attempt also to represent emergent behaviors which arise from the interactions occurring in cell colonies [14,196]. This is achieved by three steps [198]: *i)* by dividing the P system environment into a set of small regions, each satisfying Gillespie's algorithm requirement of well mixed volumes, and defining a connection topology among these regions, *ii)* by distributing a collection of P systems, representing individual cells, over the multi-environment defined above, *iii)* by defining rules representing the diffusion of signals among regions and rules describing P system movements from one environment to another. The reader can find a detailed description of these concepts in [198].

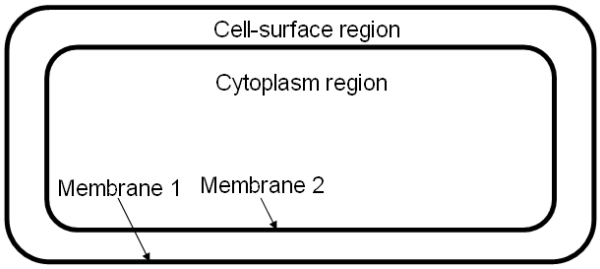


Fig. 3.5. Membrane structure used for the stochastic P model of *lac* operon [198].

Molecular Species	Object
RNA Polymerase	$RNAP$
Ribosome	Rib
Repressor	$LacI$
Activator	CRP CRP^*
LacZ product	$LacZ$
LacY product	$LacY$
LacA product	$LacA$
Lactose	$Lact$
Allolactose	$Allolac$
Glucose	$Gluc$
...	...

Operon site	Object
Activator binding site	cap
Occupied activator binding site	cap^{CRP^*}
Repressor binding site	op
Occupied repressor binding site	op^{LacI}
lacZ gene	$lacZ$
lacY gene	$lacY$
lacA gene	$lacA$
lacZ mRNA	$mlacZ$
...	...

Table 3.1. Molecular species involved in the *lac* operon model and related objects [198].

Number	Rule	Stochastic Constant
r_1 :	$[\, RNAP + \langle cap \rangle \,]_b \xrightarrow{c_1} [\, \langle cap.RNAP \rangle \,]_b$	$c_1 = 5 \times 10^{-3} min^{-1}$
r_2 :	$[\, \langle cap.RNAP \rangle \,]_b \xrightarrow{c_2} [\, RNAP + \langle cap \rangle \,]_b$	$c_2 = 1 min^{-1}$
r_3 :	$[\, CRP^* + \langle cap \rangle \,]_b \xrightarrow{c_3} [\, \langle cap^{CRP^*} \rangle \,]_b$	$c_3 = 16.6 min^{-1}$
r_4 :	$[\, \langle cap^{CRP^*} \rangle \,]_b \xrightarrow{c_4} [\, CRP^* + \langle cap \rangle \,]_b$	$c_4 = 10 min^{-1}$
⋮	⋮	⋮
r_{33} :	$Gluc\, [\, GTS \,]_s \xrightarrow{c_{33}} [\, Gluc\text{-}GTS \,]_s$	$c_{33} = 1 min^{-1}$
r_{34} :	$Gluc\text{-}GTS\, [\ ]_b \xrightarrow{c_{34}} GTS\, [\, Gluc \,]_b$	$c_{34} = 10 min^{-1}$
r_{35} :	$GTS\, [\, CRP \,]_b \xrightarrow{c_6} GTS\, [\, CRP^* \,]_b$	$c_{35} = 6.9 \times 10^{-3} min^{-1}$
r_{36} :	$[\, CRP^* \,]_b \xrightarrow{c_8} [\ ]_b$	$c_{36} = 0.069 min^{-1}$

Table 3.2. Reactions involved in the *lac* operon model and related stochastic constants [198].

Once having generated such a mathematical representation for the system under investigation one can compute its time evolution by setting some initial conditions for object population sizes and running the Multi-compartmental Gillespie's algorithm.

3.2.2 Automatic parameter and structure estimation

One of the main issues of generating novel stochastic P models is the determination of stochastic kinetic parameters c_j, such those reported on the rightmost column of Table 3.2. As seen for S-systems in Chapter 2, this is a common point of many mathematical models, and several authors proposed different optimization techniques able to estimate parameter values that reproduce observed time-series [40, 84, 87, 118]. As for stochastic P system a new methodology for both structure and kinetic parameter estimation has been proposed in [195] by Romero-Campero et al. It is based on a memetic algorithm [125] involving two layers, where the first layer optimizes the model topology by evolving rule structures with genetic algorithms (GA), and the second layer fine tunes stochastic parameters employing GA as well.

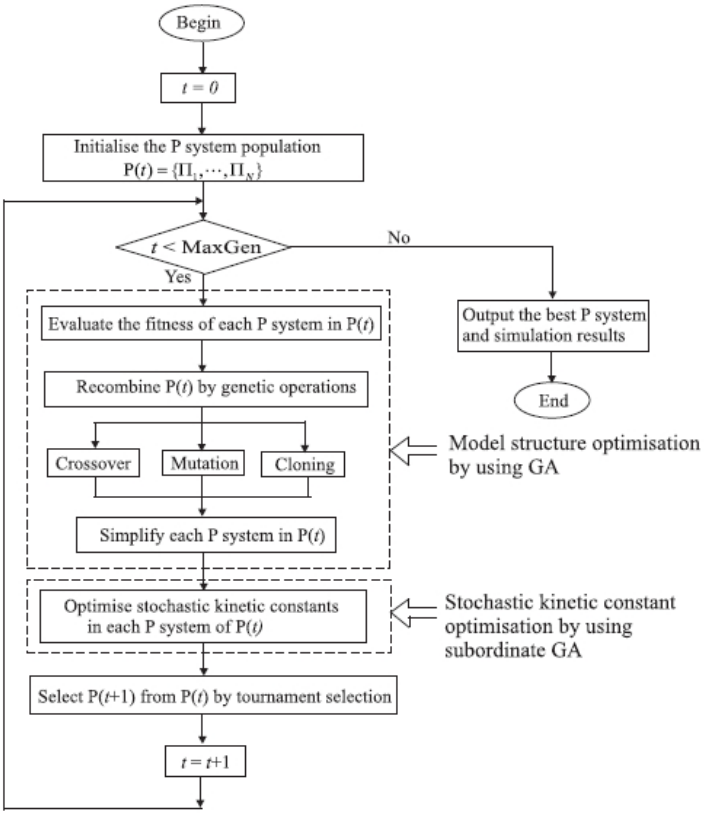


Fig. 3.6. Flowchart of the memetic algorithm employed for structure and parameter estimation in stochastic P systems [195].

This technique is performed on P systems *modules*, namely, sets of rules representing functional separable units which recur in many cell systems. Given an initial population of P system models, each encoded by a set of basic modules,

the memetic algorithm (whose flowchart is displayed in Figure 3.6) firstly evolves individual module structures by applying evolutionary operations of crossover, mutation and cloning. Subsequently, it optimizes the stochastic kinetic constants of the achieved modules by GA, and finally selects the fittest individuals (models) according to a root mean square error fitness function and it iterates the loop.

In [195] an initial library of five P systems modules has been introduced. They represent basic biological functions of complex formation/dissociation, unregulated gene expression, and positive/negative regulated expression. The memetic algorithm has been tested on three case studies, namely, molecular complexation, enzymatic reaction and regulation in transcriptional networks. For each test the memetic algorithm manages to find a topological structure and a set of kinetic parameters that make the resulting P system model reproduce the target time-series. Results were also improved by adding to the elementary library of modules the new modules found during the optimization of previous models, thus suggesting an incremental approach for developing an extended module library. In the last case more than one plausible model has been synthesized, showing the possibility to employ this methodology to support metabolic engineering [60, 157] and synthetic biology [10] in developing new possible design of cellular systems.

3.3 Dynamical probabilistic P systems

Dynamical probabilistic P systems have been firstly introduced by Pescini et al. in [173] and then extended in [172]. In this model, originated from traditional P systems, rules are applied according to a probability-based strategy rather than using the classical non-deterministic and maximally parallel approach. A probability value, depending on the current state of the system, is associated to each rule at each evolution step in order to determine the chance of firing of the rule. In the following the formal definition of a dynamical probabilistic P system (*DPP*) is reported.

Definition 4 (Dynamical Probabilistic P Systems) *A Dynamical Probabilistic P system of degree n is a construct [172]:*

$$\Pi = (V, O, \mu, M_0, \ldots, M_{n-1}, R_0, \ldots, R_{n-1}, E, I)$$

where:

- *V is the alphabet of the system, $O \subseteq V$ is the set of analyzed symbols;*
- *μ is a membrane structure consisting on n membranes labeled with the numbers $0, \ldots, n-1$. The skin membrane is labeled with 0;*
- *$M_i,\ i = 0, \ldots, n-1$, is the multiset of symbols from V representing the initial object configuration in membrane i;*
- *$R_i,\ i = 0, \ldots, n-1$, is a finite set of evolution rules associated with membrane i. The form of each evolution rule is $r : u \xrightarrow{k} v$, where u is a multiset over V, v is a string over $V \times (\{here, out\} \cup \{in_j \mid 1 \leq j \leq n-1\})$ and $k \in \mathbb{R}^+$ is a probabilistic rate constant which defines the probability that, given a simultaneous collision of the objects in u, a reaction r occurs;*

- $E = (V_E, M_E, R_E)$ *is the* environment *consisting of an alphabet* $V_E \subseteq V$*, a feeding multiset* M_E *over* V_E *and a finite set of feeding rules* R_E *of the type* $r : u \to (v, in_0)$*, for* u, v *multisets over* V_E*;*
- $I \subseteq \{0, \ldots, n-1\} \cup \{\infty\}$ *is the set of labels of the analyzed regions (label* ∞ *corresponds to the environment).*

As usual, we have an alphabet of objects V which represents chemical species, a membrane structure μ determining a compartmentalization of the environment and a set of rules R_i for each membrane i. Multisets of objects are employed to describe the presence of multiple copies of any given object inside a membrane, and transformation of object multisets are handled by rewriting rules of the form $r : u \to v$, where u is a multiset of objects and v is a multiset of objects with an associated target identifier tar. Let us consider, for instance, the rule $r : XXY \xrightarrow{k} (YZ, tar)$. If $tar = here$ then objects YZ stay in the same membrane, if $tar = out$ then objects YZ exit from the current membrane, if $tar = in_i$ then objects YZ enter membrane i, placed inside the current membrane (if there exists any inner membrane i). Finally, environment E is equipped with feeding rules R_E which provide the skin membrane (labeled by 0) with new objects.

The state of a probabilistic P model evolves from an initial configuration M_i, $i = 0, \ldots, n-1$, by applying rewriting rules according to a probabilistic strategy. Let $V = \{a_1, \ldots, a_l\}$, M_i be the multiset inside membrane i (thus $M_i(a_h)$ is the multiplicity of object a_h in multiset M_i), and $r_j : u \xrightarrow{k_j} v$ be a rule in the set $R_i = \{r_1, \ldots, r_m\}$, $i = 0, \ldots, n-1$. Let $u = a_1^{\alpha_1} \ldots a_s^{\alpha_s}$, $alph(u) = \{a_1, \ldots, a_s\}$, and $H = \{1, \ldots, s\}$, $s \in \mathbb{N}$. The probability of applying rule r_j, considering all rules in R_i, is [172]

$$p_i(r_j) = \frac{\tilde{p}_i(r_j)}{\sum_{j=1}^{m} \tilde{p}_i(r_j)}, \tag{3.5}$$

where the so called pseudo-probability $\tilde{p}_i(r_j)$, corresponding to the probability of a collision among the reactant objects of r_j, is a weighted product of binomial coefficients each representing the combination of α_h objects of type a_h taken from the multiset M_i

$$\tilde{p}_i(r_j) = \begin{cases} k_j \cdot \prod_{h \in H} \frac{M_i(a_h)!}{\alpha_h!(M_i(a_h)-\alpha_h)!} & \text{if } M_i(a_h) \geq \alpha_h \text{ for all } h \in H, \\ 0 & \text{if } M_i(a_h) < \alpha_h \text{ for some } h \in H. \end{cases} \tag{3.6}$$

Let us consider, for instance, a membrane i containing the multiset $M_i = X^3Y^8Z^2$ and the three rewriting rules $r_1 : X^2Y \xrightarrow{k_1} YZ$, $r_2 : XZ^2 \xrightarrow{k_2} Y$ and $r_3 : Z^3 \xrightarrow{k_3} Y$, with $k_1 = 3.2 \cdot 10^{-1}$, $k_2 = 1.4 \cdot 10^{-1}$ and $k_3 = 4.5 \cdot 10^{-1}$. We firstly observe that rule r_3 cannot fire because it needs at least three objects of type Z while there are only two of such objects in membrane i. By solving Equation (3.6) we obtain the following collision probabilities

$$\tilde{p}_i(r_1) = 3.2 \cdot 10^{-1} \frac{3!}{2!(3-2)!} \cdot \frac{8!}{1!(8-1)!} = 7.68, \tag{3.7}$$

$$\tilde{p}_i(r_2) = 1.4 \cdot 10^{-1} \frac{3!}{1!(3-1)!} \cdot \frac{2!}{2!(2-2)!} = 0.42, \tag{3.8}$$

$$\tilde{p}_i(r_3) = 0 \tag{3.9}$$

therefore, the probabilities of firing of the three rules are

$$p_i(r_1) = \frac{7.68}{7.68 + 0.42} = 0.948, \tag{3.10}$$

$$p_i(r_2) = \frac{0.42}{7.68 + 0.42} = 0.052, \tag{3.11}$$

$$p_i(r_3) = 0 \tag{3.12}$$

In order to compute the time evolution of a probabilistic P system the *simulation algorithm*, reported in the next page, has been proposed in [172, 173]. It assumes that rule application is synchronized among membranes by a universal clock and, at each evolution step, rules are selected according the probability values dynamically computed by Equations (3.5), so that rules having higher probability values are on the average applied more frequently.

Basically, at each evolution step three stages are accomplished on every membrane i: firstly, the probability value of each rule in membrane i is evaluated according to Equation (3.5). Then, rules are selected according to their probability values (while any rule is applicable in each membrane i) and their reactants are removed from the multiset of objects of the membrane i. Finally, products generated by all rules are added to the related target membranes. Notice that along the first two steps there is no interaction among membranes, thus these steps can be performed in parallel. Communication among membranes is instead performed in the third stage by means of the multisets update.

If we denote by R, S and h_i, respectively, the number of rules in the model, the cardinality of the alphabet and the total number of objects appearing in the model at evolution step i, the complexity of the algorithm has been quantified as $T_i(S, R) = O(h_i) \cdot O(R \cdot S)$ [172]. Of course, considering real biological systems the term $O(h_i)$ often turns out to be very time-consuming since the number of molecules involved in such processes can range from a few hundreds to many trillions. In order to overcome this drawback, dynamical probabilistic P systems have been provided with a method, based on the τ-leaping procedure [81], for the simulation of complex systems composed by several communicating regions [38].

Dynamical probabilistic P system simulation algorithm [172]

While (halting condition is not met) **do**

Feed the skin membrane by rules R_E so that multiplicities in M_0 are kept at a constant value

Step 1: probability evaluation
For each membrane $i \in 0, \ldots, n-1$ **do**
For each rule r_j in M_i **do**
evaluate the pseudo-probability $\tilde{p}_i(r_j)$ by Equation (3.6)
end for
For each rule r_j in M_i **do**
evaluate the firing probability $p_i(r_j)$ by Equation (3.5)
end for
end for

Step 2: rule firing (object consumption)
For each membrane $i \in 0, \ldots, n-1$ **do**
For each rule r_j in membrane i **do**
set counter $c_j = 0$
end for
While (any rule is applicable in membrane i) **do**
- Generate a random number *rnd* in [0,1]
- Select a rule r_j by *rnd* according to the firing probabilities $p_i(r_j)$ of the rules in membrane i
- Check the applicability of rule r_j in respect to the current availability of objects in membrane i
- **If** (r_j is applicable) **then** increment by one its counter c_j
- Remove the left-hand side multiset of rule r_j from M_i
end while
end for

Step 3: rule firing (object production)
For each membrane $i \in 0, \ldots, n-1$ **do**
For each rule r_j in M_i **do**
Add c_j times the right side multiset of rule r_j to the multiset M_{tar} of the target membrane *tar* of rule r_j
end for
end for

end while

3.3.1 An example

The so called *predator-prey model* [57, 162] was proposed independently by Alfred J. Lotka [134] and Vito Volterra [235] respectively in 1924 and 1926. It concerns with the interactions among two or more species sharing the same environment and competing for resources, habitat, or territory. Given a species of preys whose

number of individuals is x and a population of predators of y individuals, the following system of differential equations describes the dynamics of the food chain involving these two species:

$$\frac{dx}{dt} = ax - bxy \tag{3.13}$$

$$\frac{dy}{dt} = -cy + dxy \tag{3.14}$$

The term ax represents the prey increment due to their reproduction, the terms $-bxy$ and dxy cope with the interaction between preys and predators, which yields prey decrement (since they are eaten by predators) and predator increment (since it is supposed that predators reproduce each time they eat a prey). Finally, the term $-cy$ concerns with predators death, which naturally happens with an average rate c. Setting proper growth, death and reproduction rates, and starting from suitable initial conditions this model shows interesting oscillations [57, 162].

The investigation of the predator-prey system by means of dynamical probabilistic P systems, in [172], yielded the model $\Pi_{LV} = (V, O, \mu, M_0, R_0, E_{LV}, \{0\})$ described below:

- $V = \{A, X, Y\}$, where A represents sustenance resources for preys, X stands for preys and Y for predators;
- $O = \{X, Y\}$ is the set of symbols whose dynamical variation is analyzed;
- $\mu = [_0]_0$ is the environment membrane where the two species interact;
- $M_0 = X^{p_1}Y^{p_2}$, for some $p_1, p_2 \in \mathbb{N}$, is the multiset representing the initial populations of preys (p_1 individuals) and predators (p_2 individuals). No resources A are available at the starting time;
- $R_0 = \{r_1, r_2, r_3\}$ where

$$r_1 : AX \xrightarrow{k_1} (XX, here)$$
$$r_2 : XY \xrightarrow{k_2} (YY, here)$$
$$r_3 : Y \to (\lambda, here)$$

  for some $k_1, k_2, k_3 \in \mathbb{R}^+$. Rule r_1 represents the prey growth, which is related in this model to the presence of sustenance resources, rule r_2 describes the interaction between predators and prey, which yields the death of a prey and the reproduction of the predator, and r_3 describes the natural death of predators;
- $E = (\{A\}, \{A^s, \text{for some } s \in \mathbb{N}\}, \{r_4 : A \to (A, in_0)\})$, where the first set is the alphabet of input resources, the second set is the feeding multiset and the third set is the set of feeding rules containing only the rule r_4 used to keep the amount of resources inside the system at the constant level s;
- $I = \{0\}$ is the set of the regions of interest for the analysis of this model. It involves only the environment membrane.

By setting the initial configuration to $p_1 = 1000$, $p_2 = 1000$ and $s = 200$, and the probability constants of rules r_1, r_2 and r_3, respectively to $k_1 = 1$, $k_2 = 10^{-2}$ and $k_3 = 10$, the oscillatory dynamics of Figure 3.7 has been found [172].

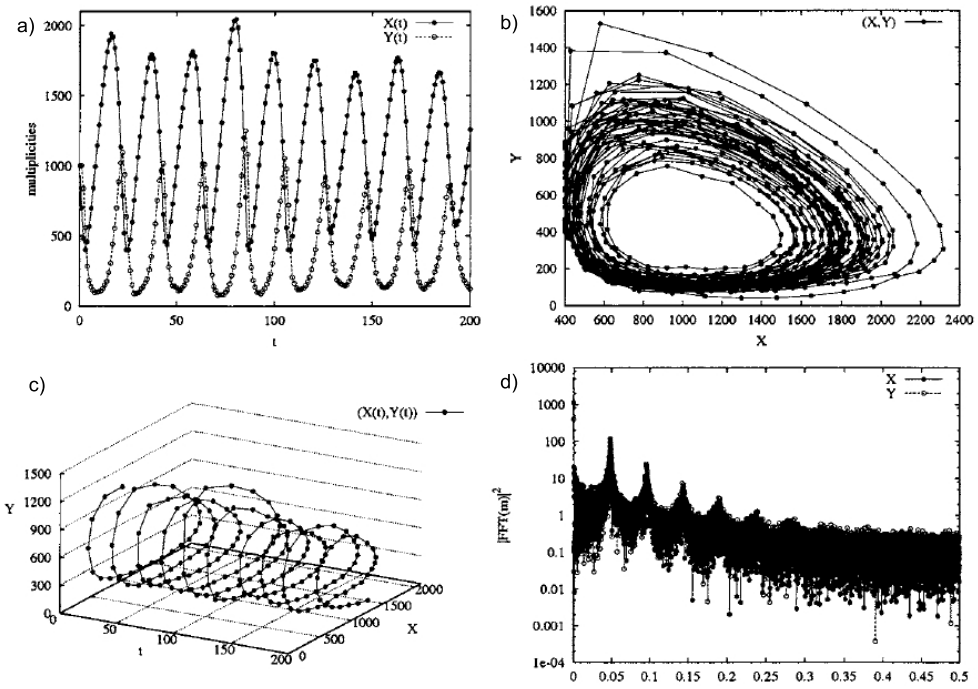


Fig. 3.7. Dynamics charts of the predator-prey model [172]. *(a)* temporal evolution, *(b)* X-Y plane temporal evolution, *(c)* phase diagram and *(d)* FFT frequencies.

3.4 ARMS: abstract rewriting systems on multiset

Abstract Rewriting Systems on Multiset (ARMS) has been proposed for the first time in [221] as an abstract model for chemical reactions in artificial life [52]. An ARMS basically evolves the configuration of a multiset of symbols over an alphabet A by means of a set of rewriting rules R which can be applied according to some evolution strategies. What seems to miss in this framework with respect to P systems is the membrane structure, indeed the first definition of ARMS did not cope with membranes. This feature has been added in successive extensions of this framework in order to satisfy needs of environment compartmentalization. In this context many topics have been investigated, such as, the relationship between ARMS and cellular automata [218] and the emergence of life from chemical evolution [219]. Further applications of this approach range from ecology [217] to medical science [216] and from environment engineering [110] to biochemical systems modeling [220, 230, 231]. The formal definition of ARMS is reported in the following.

Definition 5 (Abstract Rewriting System on Multiset - ARMS) *An ARMS is a construct [230]:*

$$\Pi = (A, w, R)$$

where:

- *A is an alphabet (a finite set of abstract symbols). A multisets over A is defined as a mapping $M : A \mapsto \mathbb{N}$ assigning a multiplicity value $M(a)$ to each symbol a of the alphabet. The set of all the multiset over A is denoted by $A^{\#}$, where the empty multiset is $\emptyset(a) = 0$ for every $a \in A$;*
- *w is the multiset representing the initial configuration of the system;*
- *R is the set of multiset rewriting rules, where a rewriting rule r over A is defined as a couple of multisets (s, u), with $s, u \in A^{\#}$. A rule can also be represented by the arrow notation $r : s \to u$. Given a multiset w, the application of the rule r to w produces a new multiset w' such that $w' = w - s + u$, where multisets w, w', s, u can be represented by vectors of natural numbers $(n_1, \ldots, n_{|A|})$, thus $+$ and $-$ are the classical operator of sum and difference among vectors. The* reaction vector *ν_{ji} denotes the multiplicity variation of symbol $a_i \in A$ due to a single application of reaction $r_j \in R$.*

In order to model chemical kinetics, ARMS have been extended in several ways. A *deterministic ARMS* (DARMS), for instance, employs multisets having real multiplicities instead of natural multiplicities, and the time evolution is computed deterministically by means of the Euler's method usually employed for ODEs. Notice that real multiplicities are very useful for modeling biological systems since they allow to directly handle experimental data of molar concentrations [230].

Let us assume a biological system having $n \in \mathbb{N}$ substances represented, respectively, by the symbols of the alphabet $A = \{a_1, \ldots, a_n\}$ and m reactions represented by rules $R = \{r_1, \ldots, r_m\}$. Being $X : A \mapsto \mathbb{R}$ a multiset of symbols of A having real multiplicities, we denote by $\mathbf{x} = (X(a_1), X(a_2), \ldots, X(a_n))$ the state vector of the system, where $X(a_i)$ is the molar concentration of substance a_i. Finally, let $\mathbf{x}(t)$ be the state of the system at the time t, $t \in \mathbb{R}$. The dynamics of a DARMS is computed by the algorithm displayed in the next page, which starts from an initial state $\mathbf{x}(0)$ and then calculates the subsequent states $\mathbf{x}(t + \Delta)$, with a time interval Δ, by adding to each substance concentration $X(a_i)$ an Euler's change term $\lambda_i \in \mathbb{R}$.

Notice that the time evolution of a deterministic ARMS proceeds by discrete time steps of length Δ. This time interval, however, must be small enough in order to have a good approximation of the reaction rate equation. In fact, if Δ increases, then the state of the system changes considerably during a single computation step and, consequently, the Eulero's method yield a rough approximation of the real reaction rate equation.

Other ARMS variants have been considered in [230], such as *stochastic ARMS* (SARMS) whose dynamics is computed by the stochastic τ-leap method [81], an approximation of the exact stochastic approach proposed by Gillespie in [79]. This Monte Carlo method yields non-deterministic dynamics satisfying the stochastic properties of each reaction (defined by stochastic constants $k_1, \ldots, k_m$). On the other hand, *cellular automata ARMS* (CARMS) are a composition of cellular automata and ARMS in which N DARMS are placed in a grid space. In this variant, each ARMS can communicate with adjacent ARMS in the grid space by means of diffusion rules. The emergence of 1D and 2D chemical waves for the BZ reaction has been shown by means of these tools [230].

DARMS time evolution algorithm [230]

Initialization
- set the simulation time $t = 0$
- for each couple of reaction-substance (r_j, a_i) set the stoichiometric coefficient ν_{ji} of substance a_i with respect to reaction r_j
- set the initial state $\mathbf{x}(0)$ of molar concentrations
- set the rate constants $k_1, \ldots, k_m$ of every reaction
- set the halting time t_{stop}
- set the time interval $\Delta \in \mathbb{R}$

while ($t < t_{stop} \wedge$ halting condition is not met) **do**

Step 1. Calculate the state change vector $\Lambda_t = (\lambda_1, \ldots, \lambda_n)$. Each λ_i is computed by the equation

$$\lambda_i = \sum_{j=1}^{m} \nu_{ji} \cdot f_j(\mathbf{x}(t)) \cdot \Delta$$

where $f_j(\mathbf{x}(t))$ is a reaction rate equation for reaction r_j, involving rate constant k_j and reactant concentrations $\mathbf{x}(t)$ (e.g., $f_j(\mathbf{x}(t)) = k_j a_1(t) a_3(t)$).

Step 2. Update the state vector $\mathbf{x}(t)$ by adding the state change Λ_t:

$$\mathbf{x}(t + \Delta) = \mathbf{x}(t) + \Lambda_t$$
$$t = t + \Delta$$

end while

3.4.1 An example

The Belousov-Zhabotinsky reaction [9, 109, 247, 252] (also known as BZ reaction) is the first discovered example of a class of reactions that show oscillatory behaviors due to non-equilibrium thermodynamics. It has been discovered by B. P. Belousov in 1950 who noted that in a mix of potassium bromate, cerium(IV) sulfate, propanedioic acid and citric acid in dilute sulfuric acid, the ratio of concentration of the cerium(IV) and cerium(III) ions oscillated, causing an oscillation of the color of the solution between a yellow solution and a colorless solution.

In more detail, this interesting oscillatory reaction involves several chemical compounds, among them [17]: malonic acid ($CH_2(COOH)_2$), bromate ions (BrO_3^-), cerium ions (IV) that act as catalysts (Ce^{4+}) and hydrogen ions obtained by the acid reaction environment (H^+). The most important reaction products are carbonic anhydride (CO_2), formic acid (CH_2O_2) and bromomalonic acid ($BrCH(CO_2H)_2$), while the main intermediate compounds are bromous acid ($HBrO_2$), radical (BrO_2), bromide ions (Br^-) and hypobromous acid ($HOBr$). The theoretical importance of this reaction is due to the fact that it is not dominated by equilibrium thermodynamic behavior observed in the majority of the

chemical reactions. Rather, it remains far from equilibrium for a significant length of time.

Several simplified models of the BZ reaction have been proposed, such as the *Oregonator* [64, 65], developed at the University of Oregon by R. J. Field and R. M. Noyes, and the *Brusselator* [177], introduced by I. Prigogine at the Universite Libre de Bruxelles. Here we show a DARMS model $\Pi_o = (A_o, w_o, R_o)$ (from [230]) which translates the classical ODE model of the Oregonator. The alphabet is $A_o = \{X, Y, Z, A, B, W, H\}$, where

$$
\begin{aligned}
X &\equiv HBrO_2 \text{ (bromous acid)},\\
Y &\equiv Br^- \text{ (bromide ions)},\\
Z &\equiv Ce(IV) \text{ (cerium-4)},\\
A &\equiv BrO_3^- \text{ (bromate ion)},\\
B &\equiv CH_2(COOH)_2 \text{ (malonic acid)},\\
W &\equiv HOBr \text{ (hypobromous acid)},\\
H &\equiv H^+ \text{ (hydrogen ion)}.
\end{aligned}
$$

Reaction rules R_o that take part to the three main processes involved in the BZ reaction are the following:

$$
\begin{aligned}
&\text{Process A :}\\
&\qquad r_1 : X, Y, H \xrightarrow{k1} 2W,\\
&\qquad r_2 : A, Y, 2H \xrightarrow{k2} X, W,\\
&\text{Process B :}\\
&\qquad r_3 : 2X \xrightarrow{k3} A, W, H,\\
&\qquad r_4 : A, X, H \xrightarrow{k4} 2X, 2Z,\\
&\text{Process C :}\\
&\qquad r_5 : B, Z \xrightarrow{k5} 0.5Y.
\end{aligned}
$$

where rate constants, taken from the literature, are $k_1 = 10^6\ mol^{-2}sec^{-1}$, $k_2 = 2\ mol^{-3}sec^{-1}$, $k_3 = 2{\cdot}10^3\ mol^{-1}sec^{-1}$, $k_4 = 10\ mol^{-2}sec^{-1}$, $k_5 = B{\cdot}2{\cdot}10^{-2}\ sec^{-1}$.

When the model is simulated, chemicals A and B are continuously supplied in order to keep constant their concentrations, while substances X, Y and Z show the typical BZ oscillations. Notice that oscillations are triggered by the presence of bromous acid (X) which causes the increasing of concentration of cerium-4 (Z). Then, the activation of reaction r_5 determines the consumption of Z and the production of bromide ions (Y), thus a new cycle is ready to start. The time evolution of the achieved DARMS has been computed, by means of the algorithm described in the last section, using five different time intervals $\Delta_1 = 0.0001$, $\Delta_2 = 0.001$, $\Delta_3 = 0.01$, $\Delta_4 = 0.1$, $\Delta_5 = 1.0$. Results shown that time intervals between 0.0001 and 0.01 yield typical BZ oscillations (as displayed in Figure 3.8), while, as the time interval decreases (see Figure 3.9), the oscillation pattern becomes smaller, and smaller until the typical pattern is lost (Figure 3.10). This behavior is due to the approximation made by the the Euler's method which characterizes the simulation strategy of DARMS.

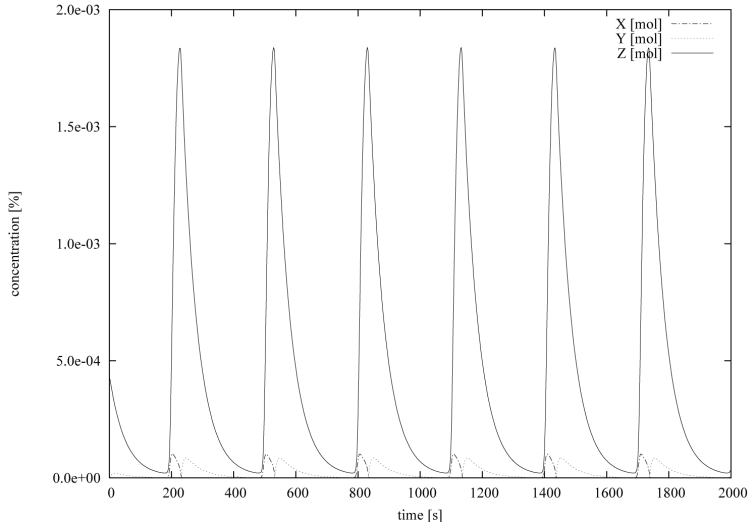


Fig. 3.8. Time evolution of the Oregonator DARMS with $\Delta_1 = 0.0001$. It shows a normal oscillation amplitude (figure from [230] with permission).

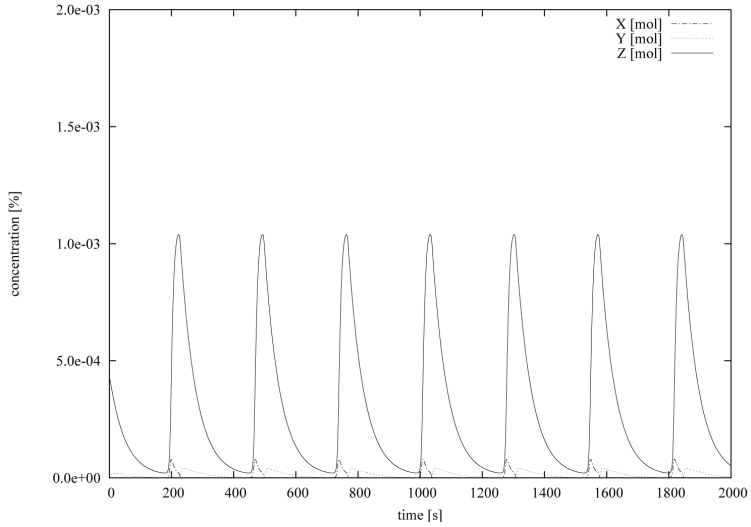


Fig. 3.9. Time evolution of the Oregonator DARMS with $\Delta_4 = 0.1$. It shows a smaller oscillation amplitude in respect to Figure 3.8 (figure from [230] with permission).

3.5 Discussions and comments

In this chapter we have reviewed some of the main extensions of P systems for biological systems modeling. Stochastic P systems take their inspiration from the Gillespie's stochastic approach, which is suited when small amounts of molecules are investigated. Indeed, the dynamics of these models is computed by simulating each single reaction occurring in the system, so that a huge amount of computa-

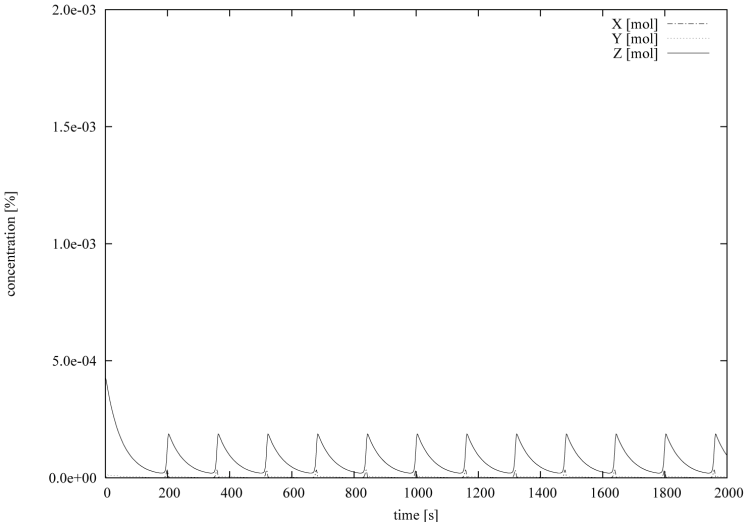


Fig. 3.10. Time evolution of the Oregonator DARMS with $\Delta_5 = 1.0$. The oscillation pattern is different from the typical one (figure from [230] with permission).

tional resources are needed when real biological systems are investigated. Some approximated algorithms are often used to reduce the computational cost of dynamics computation. The τ-leaping approximation algorithm simulates in a single step all the reactions occurring in the system during a time interval τ. This approximation can be employed only if certain ratios are respected between the amount of molecules and the interval length, and it produces averaged dynamics which tend to deterministic ODE dynamics (or corresponding Euler's approximations) as the number of molecules grows.

Deterministic P systems provide a probabilistic approach for computing the time evolution of the model. Equations 3.5 and 3.6 compute the probability of applying each rule according to the probability of collision of rule reactants, multiplied by a reactivity constant assigned to the rule. Notice that no formal way is suggested to define reactivity constants for real biochemical reactions. Moreover, we observe that, when the number of molecules $M_i(a_h)$ is very large, factorials in Equation 3.6 need specific analytic techniques to be computed, otherwise overflow errors arise in normal computers. The strategy for computing the time evolution of ARMS is instead inherited from ODE or Gillespie's algorithm, depending on the type of ARMS to be used. In particular, DARMS employ the Euler's approximation, usually employed with ODE, and SARMS use the Gillespie's algorithm, typical of the stochastic approach.

A key issue of all the three approaches concerns the choice of kinetics constants for each reaction in the system. As for stochastic P systems, for instance, stochastic constants c_j determine the dynamics of the system, since they are employed in Equations 3.3 and 3.4 for choosing the next rule to be applied and the waiting time until its firing. It can be very difficult to estimate these constants from experiments, since they should reflect microscopic, infinitesimal and local behaviors of

molecules which are almost impossible to observe while real systems evolve. The same problem arises also with dynamical probabilistic P systems and ARMS models, which need, respectively, probabilistic constants and rate constants to compute their time evolutions.

What seems to lack in these modeling frameworks is a fresh and original theory for determining the dynamics of each reaction by observing the evolution of the system as a whole. Of course, this is a very difficult task of inverse-engineering, but we think it must be solved in order to overcome some of the main issues of traditional biochemical modeling [66]. A first attempt in this direction has been presented in Subsection 3.2.2 for stochastic P systems. In this context, a memetic algorithm, based on genetic algorithms, has been developed for estimating both model topology and kinetic parameters from observed time series of substance dynamics. Similar techniques, involving genetic algorithms, genetic programming, simulated annealing and other optimization strategies, have been recently employed also for estimating kinetic parameters of other modeling frameworks [16, 40, 84, 87, 118, 232, 234].

In the next chapter we introduce a novel modeling framework for which a brand new theory have been devised expressly for deducing reaction kinetic functions from observed data. This modeling framework, rooted in P systems, is called *metabolic P systems*, and the new theory which makes it particularly attractive is the *Log-gain theory*. The author's contribution in this context concerns three main topics: i) an equivalence between metabolic P systems and hybrid functional Petri nets, ii) the generation of metabolic P models from experimental data, iii) the software MetaPlab, which is a virtual laboratory based on metabolic P systems.

4

Metabolic P systems

Several mathematical formalisms have been proposed to model biological systems and to to overcome some limitations of the traditional ODE models. Some interesting approaches, alternative to ODE, are based on rewriting systems, in the context of formal language theory, where biochemical elements correspond to symbols from a given alphabet and chemical reactions are represented by rewriting of commutative strings (also called multisets). In Chapter 3 P systems [184, 185] have been introduced as a novel computational model inspired by the prominent role played by membranes in living cells [4]. Since the first definition of P system's evolution strategy assumes a *non-deterministic and maximally parallel* application of the rules [184], which is not very meaningful within the context of bio-systems dynamics computation [140], many variants of this formalism were proposed [44]. A class of P systems that proved to be significant and successful for modeling biological phenomena are *metabolic P systems*, also called MP systems [17–22, 29–36, 67–69, 71, 72, 136–149, 168]. Their evolution is computed by a deterministic algorithm, called *metabolic algorithm* [19, 21, 140, 144], based on the *mass partition principle* which defines the transformation rate of object populations according to a suitable generalization of chemical laws. In this thesis we deal with MP systems whose evolution is computed by fluxes [139, 141, 142], and we visualize these models by MP graphs [144], a graphical representation of MP systems. These formalisms are also employed in *MetaPlab*, a Java software which enables to generate, simulate and analyze MP models [18, 22, 34, 146, 241]

A series of significant processes modeled by MP systems include the Belousov-Zhabotinsky reaction (in the Brusselator formulation) [19, 21], the Lotka-Volterra dynamics [19, 69, 145], the SIR (Susceptible-Infected-Recovered) epidemic [19, 21], the circadian rhythms, the mitotic cycles in early amphibian embryos [144] and the lac operon gene regulatory mechanism in glycolytic pathway [31].

4.1 Fundamentals of metabolic P systems

A significant principle of chemistry is the so called *mass action law* which considers two main aspects of chemical reactions: the kinetic aspect, dealing with reaction rate equations for elementary reactions (already defined in Chapter 2), and the

equilibrium aspect, concerning chemical equilibrium as a dynamical process in which rates of forward and backward reactions balance each other out. According to the mass action law, the instantaneous rate of a generic reaction $x_1 + x_2 \stackrel{k}{\rightarrow} x_3$ does not just depend on the chemical nature of reactants (with the rate constant k), but it is also proportional to the active mass of the reactants, each raised to a specific power term

$$rate(X_1, X_2) = kX_1^m X_2^n \tag{4.1}$$

where $k \in \mathbb{R}^+$, $m, n \in \mathbb{R}$. In this way, the instantaneous substance variation produced by a reaction depends on the instantaneous quantities of substances involved in the reaction.

We have seen in Chapter 2 that ordinary differential equation models are often based on the mass action law, indeed each equation of such models expresses a dynamical mechanism at an infinitesimal scale and independently from other equations [141]. The systemic effect then results from the combination of the instantaneous effects of all the equations involved in the model. In this perspective differential models are *infinitesimal, microscopic* and *local*, since kinetic rate constants of each equation have to be evaluated according to a deep understanding of single molecular events.

Metabolic P systems, instead, adopt an *observational, macroscopic*, and *global* approach which is opposite to the perspective of differential models. The idea is to leave unknown the *real* internal dynamics of each reaction but to consider the system at a higher abstraction level which is however sufficiently informative to reveal the logic of the observed system. In particular, no instantaneous kinetic is considered by MP models but the variation of substance amounts is accounted at discrete time instants separated by macroscopic intervals. In this way, the time evolution of the system turns out to be a discrete sequence computed from the knowledge of the contribution of each reaction during every time interval. The way in which these contributions are computed is the key aspect of the MP systems theory which is based on the *log-gain principle* [140–142]. Of course, this abstraction unavoidably causes the loss of some system details, but such a more generic information could be still very useful for discriminating important aspect of the reality under investigation. Moreover, this abstraction sometimes represents the only way for handling some comprehension of very complex systems.

Let us consider now some fundamental chemical principles on which the MP systems theory is based. In metabolic systems, molecules of various types usually float in a liquid medium and they are subjected to reactions when they collide. MP systems omit to consider position and velocity of each single molecule, which is very time and space consuming from a computational point of view, but they represent only the number of molecules of each type, and the reactions which may occur among the various molecule types. More specifically, a *population unit* ν is defined which represents the amount of molecules of a conventional mole, and each substance type is associated, at each time instant, to a real number accounting the number of population units of molecules of this type. For instance, if we set $\nu = 100$ in an MP system having four substance: X_1 with 3.2 population units, X_2 with 5.2 population units, X_3 with 3.0 population units and X_4 with 10.5 population units, then the amounts of molecules represented are, respectively, 320,

520, 300 and 1050, and the *state* of the system can be represented by the vector $(3.2, 5.2, 3.0, 10.5) \in \mathbb{R}^4$. Notice that, the value of population unit ν influences the dynamics only for a scale factor. Indeed if we had chosen $\nu = 10$, instead of $\nu = 100$, then the system state above would be represented by the vector $(32, 52, 30, 105) \in \mathbb{R}^4$, which is $10 \cdot (3.2, 5.2, 3.0, 10.5)$. The population unit is usually set manually just considering the average number of molecules for each substance in the system. Chemical reactions can also be represented by vectors in $\mathbb{R}^n$, where n is the number of substance types. Let us consider, for example, the reaction $r : 3X_1 + X_2 \to 2X_3$. It can be represented by the vector $(-3, -1, 2, 0)$, saying that, each time that reaction r occurs, 3 molecules of type X_1 and 1 molecule of type X_2 are transformed into 2 molecules of type X_3, while substance X_4 is neither generated nor produced.

If we consider the metabolic system along discrete time instants separated by a constant temporal interval $\tau \in \mathbb{R}^+$, then we certainly have that *during each time interval every reaction occurs a certain number of times, transforming molecules from some types to other types*. Moreover, matter can be introduced into the system from the external environment, or expelled from the system. From this perspective, reactions may be considered as agents performing matter transformation according to some ratios which depend on the stoichiometry of the system, as defined by the *Avogadro's principle*. For instance, if reaction $r : 3X_1 + X_2 \to 2X_3$ occurs three times during a certain time interval τ, then 9 molecules of type X_1 and 3 molecules of type X_2 are transformed into 6 molecules of type X_3. MP systems dynamics satisfy the Avogadro's principle since substances are always transformed according to the stoichiometry of some reaction. Furthermore, each reaction r has an associated *flux regulation map* $\varphi_r(q)$ which computes, given the current state q of the system, the *flux* of molecules transformed by the reaction during the next time interval τ. For instance, if at time t the state is $q_t = (3.2, 5.2, 3.0, 10.5)$ and the flux of reaction $r : 2X_1 + X_2 \to X_3$ is $\varphi_r(q) = 0.9$, then 2.7 moles of X_1 and 0.9 moles of X_2 are consumed and 1.8 moles of X_3 are produced, thus the state at time $t + \tau$ will be $q_{t+\tau} = (0.5, 4.3, 4.8, 10.5)$. We manually set parameter τ according to the "speed" of the system. In particular, to analyze very fast processes we set very brief time intervals (e.g., 1 msec), while to model slow processes we employ longer intervals (e.g., 1 sec or 1 min). However, recent studies have shown that time grain τ can influence the complexity and the readability of MP models, since flux regulation functions depend on this parameter. Future work will hopefully provide some formal procedures to estimate parameter τ.

We call this kind of transformation mechanism as *molar multiset rewriting* [141]. It differs from the usual multiset rewriting of P systems, indeed in the classical case a rule $X_1X_1X_2 \to X_3$ replaces two objects X_1 and one object X_2 into one object X_3, thus the rule application is *individual*. On the other hand, in a molar multiset rewriting perspective, the application of the same rule transforms a population of X_1, X_2 and X_3, where the size of these population is given by the flux of the reaction (a value depending on the state of the system) and the number of times this population is taken, is given by the number of occurrences of each symbol (corresponding to the stoichiometry of the reaction). This perspective enables us to tune the time of MP models with the macroscopic time of the observer, rather than with the microscopic time of reaction kinetics. In this way, MP models

can be directly generated from discrete and systemic observations of biochemical system dynamics, while avoiding to compute instantaneous reaction rate constants from discrete observations and then to discretize the achieved equations, as usually happens with ODE.

Another important law of chemistry, referred to Dalton, states the *additivity of the effects* of all reactions involved in a biochemical system. According to this principle, after having calculated the flux of each reaction in state q, we compute the next state by adding the effects of every reaction, that is, by adding to each substance X_i all the amounts of X_1 produced by every reaction, and subtracting from each substance X_i the amounts of this substance consumed by every reaction. For instance, in an MP model having two reactions $r_1 : X_1X_2 \to 2X_3$ and $r_2 : X_2X_3 \to X_4$, if the initial state is $q = (30.3, 40.5, 25.0, 12.3)$ and fluxes are $\varphi_1(q) = 3.3$ and $\varphi_2(q) = 2.1$, then reaction r_1 consumes 3.3 moles of X_1 and 3.3 moles of X_2 and it produces 6.6 moles of X_3, while reaction r_2 consumes 2.1 moles of X_2 and 2.1 moles of X_3 and produces 2.1 moles of X_4. By adding the effects of the two reactions, the next state will be $q' = (27.0, 35.1, 29.5, 14.4)$, where, for instance, X_2 has been decreased of 3.3 moles by reaction r_1 and of 2.1 moles by reaction r_2, while X_3 has been increased of 6.6 moles by r_1 and decreased of 2.1 moles by r_2.

Employing together the Avogadro's and the Dalton's principles, the dynamics of MP systems can be computed by the following two steps:

1. compute the flux of each reaction from the current state;
2. add produced substances and subtract consumed substances to/from substance amounts in the current state q.

In order to ensure the *matter conservation* within MP models the Lavoisier's principle is implicitly assumed. According to this law, for each reaction the mass of matter consumed and produced in the system at each step must be equal. In order to satisfy this principle, MP systems associate to each substance x a real number $\mu(x)$, representing the mass of a mole of x (with respect to some measure unit).

The three laws described so far, namely, Avogadro's, Dalton's and Lavoisier's laws, are fundamental for computing MP dynamics, but what we still need is a strategy for computing the flux of matter transformed by each reaction in each state. In particular, we need a law which extends the (infinitesimal and microscopic) mass action law to the case of discrete and macroscopic time evolution considered by MP system. We will call it *mass partition principle* [140–142], and it states that, during the time interval τ the available mass of a substance X is divided among the reactions which need it, according to the percentage which they consume in the time interval of the observation step. This law bring us to consider a fundamental aspect of MP system modeling, which concerns the synthesis of *flux regulation functions* $\varphi_r(q)$. An new theory, called *log-gain theory* [140–142], has been developed by Prof. Vincenzo Manca and his research group for tackling this problem. It will be described in the following of this chapter, after a formal definition of MP systems and MP graphs.

4.2 MP systems

MP systems are deterministic P systems developed to model dynamics of biological phenomena related to metabolism and signal transduction in the living cell. Here we consider two classes of MP systems, namely, *MP systems with flux maps* (MPF) and *MP systems with reaction maps* (MPR). While in the first approach general functions are employed to compute fluxes, in the second approach flux functions have a standard form involving an implicit mass partition according to the "reactivity" of each reaction.

4.2.1 MP systems with flux maps

The notion of MP system we consider here is essentially that proposed in [141,222]. Given a biological process of interest, all its variables can be split in two different kinds (introduced by item 1 and item 3 of Definition 6): chemical *substances*, which have a mass and transform according to the mass conservation principle, and other *parameter* elements, with no mass (e.g., pressure, pH), that provide some observed values or evolve according to some specific laws. Substances are measured by (conventional) moles, which correspond to suitable population units. For each substance, the mass of a mole specifies the matter quantity (in terms of some mass unit) associated to a mole of the substance. The biochemical reactions of the process (R, in item 3) are seen as rewriting rules. The observation time interval (τ, in item 8) is established on the basis of either the kind of process or the kind of study in which one is interested; faster biological processes require shorter time intervals.

Definition 6 (MPF system) *An MP system with flux regulation maps, shortly an MPF system, is a discrete dynamical system specified by a construct [142]:*

$$M = (X, R, V, Q, \Phi, \nu, \mu, \tau, q_0, \delta)$$

where X, R, V are disjoint sets and the following conditions hold, with $n, m, k \in \mathbb{N}$:

1. *$X = \{x_1, x_2, \ldots, x_n\}$ is a set of **substances** (molecule types);*
2. *$R = \{r_1, r_2, \ldots, r_m\}$ is a set of **reactions** over X. A reaction r is represented in the arrow notation by a rewriting rule $\alpha_r \to \beta_r$ with α_r, β_r strings over X. The **stoichiometric matrix** $\mathbb{A}$ stores reactions stoichiometry, that is, $\mathbb{A} = (\mathbb{A}_{x,r} \mid x \in X, r \in R)$ where $\mathbb{A}_{x,r} = |\beta_r|_x - |\alpha_r|_x$, and $|\gamma|_x$ is the number of occurrences of the symbol x in the string γ;*
3. *$V = \{v_1, v_2, \ldots, v_k\}$ is a set of **parameters** (such as pressure, temperature or pH) equipped with a set $\{h_v : \mathbb{N} \to \mathbb{R} \mid v \in V\}$ of **parameter evolution functions**, where, for any $i \in \mathbb{N}$, $h_v(i) \in \mathbb{R}$ is the value of parameter v at simulation step i;*
4. *Q is the set of **states**, namely functions $q : X \cup V \to \mathbb{R}$ from substances and parameters to real numbers. A state q is identified by the vector $q = (q(x_1), \ldots, q(x_n), q(v_1), \ldots, q(v_k))$ of the values that the function q associates to the elements of $X \cup V$. We denote by $q_{|X}$ the substance set state, and by $q_{|V}$ the parameter set state;*

5. $\Phi = \{\varphi_r : Q \to \mathbb{R} \mid r \in R\}$ *is a set of* ***flux regulation functions****, where for any* $q \in Q$*,* $\varphi_r(q)$ *states the amount of moles consumed/produced, in the state* q*, for every occurrence of a reactant/product of* r*. The* ***flux vector*** *at state* q *is defined as* $U(q) = (\varphi_r(q) \mid r \in R)$*;*
6. ν *is a natural number, called* ***population unit****, which specifies the number of molecules of a (conventional) mole of* M*;*
7. μ *is a function which assigns to each* $x \in X$*, the* ***mass*** $\mu(x)$ *of a mole of* x *(with respect to some measure unit);*
8. τ *is the* ***temporal interval*** *between two consecutive observation steps;*
9. $q_0 \in Q$ *is the* ***initial state****;*
10. $\delta : \mathbb{N} \to Q$ *is the* ***dynamics*** *of the system. It can be identified as the vector* $\delta = (\delta(0), \delta(1), \delta(2), \ldots)$*, where* $\delta(0) = q_0$*, and* $\delta(i) = (\delta(i)_{|X}, \delta(i)_{|V})$ *is computed by the following autonomous first order difference equations:*

$$\delta(i+1)_{|X} = \mathbb{A} \times U(\delta(i)) + \delta(i)_{|X} \tag{4.2}$$

$$\delta(i+1)_{|V} = (h_v(i+1) \mid v \in V) \tag{4.3}$$

where $\mathbb{A}$ *is the stoichiometric matrix of* R *over* X*, of dimension* $n \times m$*, while* $\times$*,* $+$ *are the usual matrix product and vector sum.*

In other words, if we denote by $X[i] = (x_1[i], x_2[i], \ldots, x_n[i])$ the vector of substance amounts at step i, by $V[i] = (v_1[i], v_2[i], \ldots, v_k[i])$ the vector of parameter values at step i, and by $U[i] = (u_1[i], u_2[i], \ldots, u_m[i])$ the vector of fluxes at step i, the dynamics of M can be defined by the following vector recurrent equation [141, 222], called **Equational Metabolic Algorithm** ($EMA[i]$):

$$X[i+1] = \mathbb{A} \times U[i] + X[i] \tag{4.4}$$

and parameters are updated at each step according to equation

$$V[i+1] = (h_1(i+1), h_2(i+1), \ldots, h_k(i+1)). \tag{4.5}$$

Thus, the P system formalism is here employed as a specification framework for biological systems while the dynamics of the model is computed by solving a set of difference equations. With a slight abuse of notation, in the following we sometimes use $h_j(i)$, instead of $h_{v_j}(i)$, to represent the evolution function of a parameter v_j, $j = 1, \ldots, k$, and we use $\varphi_j(q)$, instead of $\varphi_{r_j}(q)$, to represent the flux regulation function of a reaction r_j, $j = 1, \ldots, m$. Moreover, in this thesis notation "MP systems" is often used to refer to MPF systems. Only in a few cases we will use MP systems with reaction maps. In these cases we will explicitly adopt the notation MPR systems.

In order to clarify the notions introduced so far, we consider a toy MPF system. The elements of this system have no relationships with any specific biological system, but its simple structure is useful for understanding the key mechanisms of MP dynamics. Let M be an MPF system with three substances, i.e., A, B, C, having molar masses, respectively, of $\mu(A) = 1.5\ g/mol$, $\mu(B) = 2.8\ g/mol$ and $\mu(C) = 0.9\ g/mol$, where the conventional mole is defined as $\nu = 1000$ molecules. The system has a single parameter P representing the system pressure and having an evolution function $h_P(i) = 0.001 \cdot C[i]^2$. It has four reactions, i.e.,

$R = \{r_1, r_2, r_3, r_4\}$, having stoichiometry and flux regulation maps as reported in Table 4.1. Notice that, both parameter and flux regulation maps are set to evolve with a time interval $\tau = 2$ *sec*.

Reactions	**Flux regulation maps**
$r_1 : \lambda \to A$	$\varphi_1(q) = 10 \cdot \frac{C+P}{C \cdot P}$
$r_2 : \lambda \to B$	$\varphi_2(q) = 1.2$
$r_3 : A + B \to CC$	$\varphi_3(q) = 0.1 \cdot \frac{A^2+B}{B}$
$r_4 : C \to \lambda$	$\varphi_4(q) = \frac{B+C}{C}$

Table 4.1. Reactions and related flux regulation functions of a toy MPF system (λ is the empty multiset, + denotes the multiset sum in the left column, while the arithmetic sum in the right column).

If we want to compute the system dynamics from the initial state $q_0 = (A[0], B[0], C[0], P[0]) = (100.00, 120.30, 30.80, 9.49)$, according to EMA (Equation (4.4)), then we first need to generate the stoichiometric matrix, which is

$$\mathbb{A} = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 2 & -1 \end{pmatrix}, \tag{4.6}$$

since: reactions r_1 (in the first column) produces one substance A (first row); reaction r_2 (in the second column) produces one substance B (second row); reaction r_3 (in the third column) consumes one A and one B and produces two C; reaction r_4 (in the forth column) only consumes one C. The first step of system dynamics, from instant 0 to instant 1, can be therefore computed by the following three steps:

1. compute fluxes by means of flux regulation maps:

$$\begin{aligned} \varphi_1(100.00, 120.30, 30.80, 9.49) &= 1.30\ mol, \\ \varphi_2(100.00, 120.30, 30.80, 9.49) &= 1.20\ mol, \\ \varphi_3(100.00, 120.30, 30.80, 9.49) &= 8.41\ mol, \\ \varphi_4(100.00, 120.30, 30.80, 9.49) &= 4.90\ mol; \end{aligned}$$

2. apply each reaction according to the related flux, that is, reaction r_1 generates 1.30 moles of A, reaction r_2 produces 1.20 moles of B, reaction C consumes 8.41 moles of A, 8.41 moles of B and produces $2 \cdot 8.41$ moles of C (since C has multiplicity 2 in the right side of r_3), reaction r_4 consumes 4.90 moles of C. Employing the additivity of the effects we obtain:

$$\begin{aligned} A[1] &= 100.00 + 1.30 - 8.41 = 92.89\ mol, \\ B[1] &= 120.30 + 1.20 - 8.41 = 113.09\ mol, \\ C[1] &= 30.80 + 2 \cdot 8.41 - 4.90 = 42.72\ mol, \end{aligned}$$

   where $X[i]$ is the amount of substance X at instant i;

3. compute the new value of parameter P, that is, in our case

$$h_P(1) = 0.01 \cdot C[1]^2 = 18.25$$

By iterating this procedure for t times, the dynamics of the system is computed from time 0 to time $t \cdot \tau$.

4.2.2 MP systems with reaction maps

MP systems with reaction maps, shortly MPR systems, are a second class of MP systems wherein fluxes are not computed by generic flux regulation functions, but they are rather calculated by a specific strategy implicitly based on mass partition. In fact, MPR systems have the same elements of MPF systems but each reaction $r \in R$ is provided with a *reaction map* $f_r : Q \to \mathbb{R}$, instead of a flux map $\varphi_r : Q \to \mathbb{R}$, and each substance $x \in X$ is provided with an *inertia function* $\psi_x : Q \to \mathbb{R}$. The reaction map $f_r(q)$ computes the *reactivity* of rule r given the system state q, while inertia function $\psi_x(q)$ determines the (molar) quantity of substance x which is not consumed in state q. Fluxes are thus calculated, by using reaction and inertia functions, by means of two steps. First of all, for each reaction r and each substance x, we compute the *partial pressure* $w_{r,x}(q)$ applied by r to x (at state q), that is:

$$w_{r,x}(q) = \frac{f_r(q)}{\psi_x(q) + \sum_{r' \in R_\alpha(x)} f_{r'}(q)}, \tag{4.7}$$

where $R_\alpha(x)$ is the set of reactions having x as a reactant. Second, the flux $\varphi_r(q)$ of each reaction r (at state q) is calculated by a regulation functions having the following form:

$$\varphi_r(q) = \begin{cases} f_r(q) & \text{if } \alpha_r = \lambda; \\ min\{\frac{w_{r,y}(q) \cdot q(y)}{|\alpha_r|_y} \mid y \in \alpha_r\} & \text{otherwise.} \end{cases} \tag{4.8}$$

where $q(y)$ represents the amount of substance y in state q, and $|\alpha_r|_y$ is the the multiplicity of object y in the reactant multiset α_r of reaction $r : \alpha_r \to \beta_r$.

In other words, as the system evolves, reactants are partitioned among all the rules according to a competition strategy which assigns to each reaction a portion of the available reactants proportional to its reactivity $f_r(q)$ [17]. The reactivity of a rule in a certain state, given by its reaction map, measures the capability of the rule to acquire its reactants. Non transformed matter is taken into account by fake reactions of the form $x \to x$, having a reactivity given by the inertia function $\psi_x(q)$. Equation (4.7) provides the relative portion (a number between 0 and 1) of substance x which r is allowed to consume, while Equation (4.8) computes the flux of r as the minimum substance portion available among all the reactants of r.

Let us consider the example proposed in Section 4.2.1, and replace the reactions and the flux regulation functions of Table 4.1 with the reactions, the reaction maps and the inertia maps of Table 4.2.

In order to compute the first step of the system dynamics, namely, substance and parameter variations between instant 0 and instant 1, we perform the following five steps:

Reactions	Reaction maps	Inertia maps
$r_1 : \lambda \to A$	$f_1(q) = 10 \cdot \frac{C+P}{C \cdot P}$	$\psi_A(q) = 0.1 \cdot A$
$r_2 : A \to B$	$f_2(q) = 1.2$	$\psi_B(q) = 0.3 \cdot \sqrt{B}$
$r_3 : A + 2B \to 2C$	$f_3(q) = 0.1 \cdot \frac{A^2+B}{B}$	$\psi_C(q) = \frac{10 \cdot C}{A+B+C+P}$
$r_4 : C \to \lambda$	$f_4(q) = \frac{B+C}{C}$	

Table 4.2. Reactions, reaction maps and inertia maps of a toy MPR system (λ is the empty multiset, + denotes the multiset sum on the left, while the arithmetic sum on the right).

1. compute the reactivity of each reaction and the inertia of each substance at state $q_0 = (A[0], B[0], C[0], P[0]) = (100.0, 120.3, 30.8, 9.49)$:

$$\begin{aligned} f_1(100.00, 120.30, 30.80, 9.49) &= 1.30, \\ f_2(100.00, 120.30, 30.80, 9.49) &= 1.20, \\ f_3(100.00, 120.30, 30.80, 9.49) &= 8.41, \\ f_4(100.00, 120.30, 30.80, 9.49) &= 4.90, \\ \psi_A(100.00, 120.30, 30.80, 9.49) &= 10.00, \\ \psi_B(100.00, 120.30, 30.80, 9.49) &= 3.29, \\ \psi_C(100.00, 120.30, 30.80, 9.49) &= 1.18, \end{aligned}$$

2. compute, by means of Equation (4.7), the partial pressure applied by each reaction on each of its reactants (notice that reactions r_1 does not have any pressure value since its reactant set is empty):

$$\begin{aligned} w_{2,A}(100.00, 120.30, 30.80, 9.49) &= \frac{1.20}{10.00 + 1.20 + 8.41} = 0.06, \\ w_{3,A}(100.00, 120.30, 30.80, 9.49) &= \frac{8.41}{10.00 + 1.20 + 8.41} = 0.43, \\ w_{3,B}(100.00, 120.30, 30.80, 9.49) &= \frac{8.41}{3.29 + 8.41} = 0.71, \\ w_{4,C}(100.00, 120.30, 30.80, 9.49) &= \frac{4.90}{1.18 + 4.90} = 0.80; \end{aligned}$$

3. compute fluxes by means of Equation (4.8):

$$\begin{aligned} &\varphi_1(100.00, 120.30, 30.80, 9.49) = f_1(100.00, 120.30, 30.80, 9.49) = 1.30\ mol, \\ &\varphi_2(100.00, 120.30, 30.80, 9.49) = min\left\{\frac{0.06 \cdot 100.00}{1}\right\} = 6.00\ mol, \\ &\varphi_3(100.00, 120.30, 30.80, 9.49) = min\left\{\frac{0.43 \cdot 100.00}{1}, \frac{0.71 \cdot 120.3}{2}\right\} = 42.70\ mol, \\ &\varphi_4(100.00, 120.30, 30.80, 9.49) = min\left\{\frac{0.8 \cdot 30.8}{1}\right\} = 24.64\ mol; \end{aligned}$$

4. apply each reaction according to the related flux, that is, reaction r_1 generates 1.30 moles of A, reaction r_2 consumes 6.00 moles of A and produces 6.00 moles

of B, reaction r_3 consumes 42.70 moles of A, $2 \cdot 42.70$ moles of B (since B has multiplicity 2 in the left side of reaction r_3) and produces $2 \cdot 42.70$ moles of C (since C has multiplicity 2 in the right side of reaction r_3), reaction r_4 consumes 24.64 moles of C. Employing the additivity of the effects we obtain:

$$A[1] = 100.00 + 1.30 - 6.00 - 42.70 = 50.00\ mol,$$
$$B[1] = 120.30 + 6.00 - 2 \cdot 42.70 = 40.90\ mol,$$
$$C[1] = 30.80 + 2 \cdot 42.70 - 24.64 = 91.56\ mol,$$

where $X[i]$ is the amount of substance X at instant i;

5. compute the new value of parameter P:

$$h_P(1) = 0.01 \cdot C[1]^2 = 18.25$$

Notice that, when two or more reactions compete for a substance $x \in X$, such as reactions r_2 and r_3 do for substance A (see Figure 4.1), matter is partitioned among the reactions according to their partial pressure value at the current state $w_{r,x}(q)$. These values are computed through the reaction maps of reactions involved in the competition and the inertia map of substance x. In the example above, reaction r_2 has a reactivity $f_2(q_0) = 1.20$, reaction r_3 has a reactivity $f_3(q_0) = 8.41$ and substance A has an inertia (i.e., a tendency to keep untransformed) $\psi_A(q_0) = 10.00$. This means that r_2 produces a partial pressure $w_{2,A} = 0.06$ on substance A while r_3 produces, on the same substance, a partial pressure $w_{3,A} = 0.43$, which is about seven time greater than $w_{2,A}$. Moreover, since reaction r_3 has also B as a reactant, we need to compute the partial pressure of r_3 on B, which is $w_{3,B} = 0.71$. Given the partial pressures, we compute, for every reaction $r \in R$, the amount of acquired reactant $y \in X$ as

$$w_{r,y}(q) \cdot q(y) \mid y \in \alpha_r, \tag{4.9}$$

that is, by multiplying the partial pressure $w_{r,y}(q)$ by the amount of substance y, i.e., $q(y)$. In our example, reaction r_2 acquires 6.00 *mol* of its single reactant A, while reaction r_3 acquires 43.00 *mol* of A and 85.40 *mol* of B. In this way, the flux of r_2 is simply 6.00 *mol*, while the flux of r_3 is the minimum between 43.00 *mol* and $85.40/2 = 42.70$ *mol* (since B has multiplicity 2 in α_{r_3}). This minimum is 42.7 *mol*. We can conclude that the 100.00 *mol* of substance A are partitioned as follows: 6.00 *mol* are transformed by reaction r_2, 42.7 *mol* are transformed by reaction r_3 and the remaining 51.30 *mol* are not transformed.

4.2.3 Equivalence between MPF systems and MPR systems

MP systems with flux maps and MP systems with reaction maps have been compared in [142] and an equivalence has been proved. The following definition of (dynamical) equivalence between two MP systems is used.

Definition 7 (Equivalence between MPF and MPR [142]) *Two MP systems with the same sets X (substances) and R (reactions), and the same ν, μ, q_0, τ of Definition 6 are equivalent when in both systems the dynamics of substances $\delta_{|X}(i)$ provides the same values for any $i > 0$.*

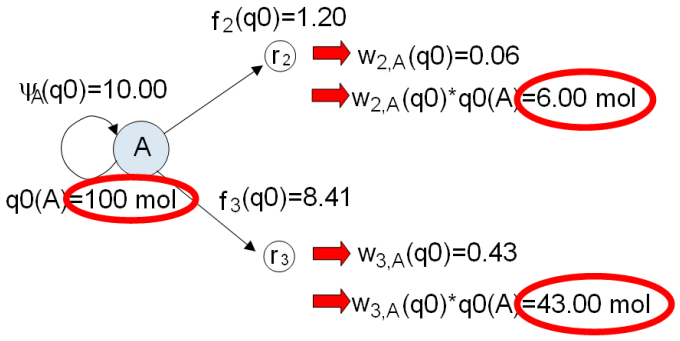


Fig. 4.1. Matter partition in MP systems with reaction maps.

Given an MPR system, the definition of an equivalent MPF system turns out to be quite straightforward. The following proposition gives a constructive way to generate flux regulation functions from reaction and inertia maps.

Proposition 1 *For any MPR system there is an equivalent MPF system with the same sets X (substances) and R (reactions), and the same ν, μ, q_0, τ of Definition 6, having for any $r \in R$ the following flux regulation map [142]:*

$$\varphi_r(q) = \begin{cases} f_r(q) & \text{if } \alpha_r = \lambda; \\ min\{\frac{w_{r,y}(q) \cdot q(y)}{|\alpha_r|_y} \mid y \in \alpha_r\} & \text{otherwise} \end{cases} \tag{4.10}$$

where

$$w_{r,x}(q) = \frac{f_r(q)}{\psi_x(q) + \sum_{r' \in R_\alpha(x)} f_{r'}(q)}. \tag{4.11}$$

The proof of this proposition is directly contained in the definition of MPR systems and in their explicit use of the mass partition principle. The definition of an MPR systems equivalent to a given MPF system requires, indeed, a few more passages to be proved.

Definition 8 (Monic and non cooperative MP system [142]) *An MP system is **monic** if $|\alpha_r|_x \leq 1$ for every $r \in R$ and for every substance $x \in X$. An MP system is **non-cooperative** if $|\alpha_r| \leq 1$ for every $r \in R$.*

We remember that $|\alpha_r|_x$ is the the multiplicity of object y in the reactant multiset α_r of reaction $r : \alpha_r \to \beta_r$, while $|\alpha_r|$ is the sum of multiplicities of all objects in the reactant multiset α_r.

Lemma 4.1. *For any MP system there exists a monic MP system which is dynamically equivalent to it [142].*

Proof. Any rule $r : \alpha_r \to \beta_r$ with $|\alpha_r|_x > 1$ can be split into many rules satisfying monic requirements, if all these rules share the flux map of rule r. For instance, given the rule $r : aa \to b$, which is not monic due to the presence of two objects a in α_r, we can split r into two rules $r_1 : a \to b$ and $r_2 : a \to \lambda$ sharing the same flux map. By proceedings in this way for each rule which

does not satisfy monic requirements we obtain an equivalent monic MP system. □

The equivalence between MPR and MPF systems now can be easily proved by the following theorem.

Theorem 4.2. *For any MPF system there exists an MPR system with the same sets X (substances) and R (reactions), and the same ν, μ, q_0, τ of Definition 6, which is equivalent to it [142].*

Proof. Let M be a monic MPF system, we define an MPR system M' having the same sets of substances (i.e., $X' = X$) and reactions (i.e., $R' = R$), wherein any rule r has a reaction map $f'_r(q) = \varphi_r(q)$. Moreover, we associate to each substance $x \in X$ an inertia function $\psi_x(q) = q(x) - \sum_{r \in R_\alpha(x)} f'_r(q)$. By employing these reaction and inertia maps in Equation (4.10) of Proposition 1 we obtain that, for any rule $r \in R$, the flux map $\varphi'(r)$ in M' is trivially equal to $\varphi(r)$ if $\alpha_r = \lambda$. On the other hand, if $\alpha_r \neq \lambda$, we substitute functions $f'_r(q)$ and $\psi_x(q)$, just defined, into Equation (4.11), obtaining

$$w_{r,x}(q) = \frac{\varphi_r(q)}{q(x)} \tag{4.12}$$

and then we employ this equation within the second branch of Equation (4.10), achieving the following function:

$$\varphi'_r(q) = min \left\{ \frac{\varphi_r(q)}{|\alpha_r|_y} \mid y \in \alpha_r \right\}. \tag{4.13}$$

We can thus conclude that $\varphi'_r(q) = \varphi_r(q)$ since M is monic and therefore $|\alpha_r|_y = 1$ for every $r \in R$ and for every $x \in X$. □

Notice that, dynamics generated by MPR systems are necessarily non negative, since matter is partitioned among rules. On the other hand, MPF systems could generate unrealistic dynamics having negative substance amounts, if no constraint is employed within flux regulation functions.

4.3 MP graphs

Biological and biochemical systems are characterized by a high complexity level which needs specific representation tools in order to be handled. Macroscopic systems usually involve big amounts of individuals and species interacting together within ecosystems. Microscopic systems also include an impressive variety of biochemical elements, such as, DNA and RNA molecules, proteins, enzymes and so on. For instance, a simple bacterium, which is an unicellular microorganism, contains more than ten thousand proteins and thousands of other constituents needed for life under ever varying conditions [234], while biochemical pathways, that seems quite simple if investigated alone, are actually part of complex networks of metabolic processes. The first step for modeling such an ubiquitous complexity

concerns the choice of a meaningful representation of the system under investigation. It should include the key elements of the system while excluding unnecessary features in order to convey a clear picture of the system functioning.

There exist several ways to describe biochemical pathways. In the simplest case of a system with a few reactions a brief verbal description can contain all the information needed to understand the process. For instance, the statement "5-phosphoribosyl-α-1-pyrophosphate (PP-ribose-P) is formed by transfer of the terminal pyrophosphate group of ATP to the carbon 1 of ribose-5-phosphate in a reaction catalyzed by PP-ribose-P synthetase" [214] well describes one of the initial steps of purine synthesis *de novo* [234]. The sentence lists the chemicals involved in the system and explains some of the mechanisms by which the process evolves. Natural language has, however, several disadvantages in describing pathways with parallel branches and simultaneous events, since it is a linear stream. In order to overcome these problems, two-dimensional graphical representations of biochemical pathways are almost always employed which offer greater flexibility in visualizing biochemical processes than a verbal description.

Figure 4.2 shows a *biochemical map* of the purine biosynthesis pathway explained above. Notice that, flow of material and interactions among different compounds, represented respectively by bold and thin arrows, can be clearly grasped from this simple picture, which enables biochemists and modelers to understand at a glance the main features of the pathway. Other kinds of symbols can be also added to these maps in order to increase their expressiveness. The reader can refer to [234] for a full set of rules for constructing proper biochemical maps.

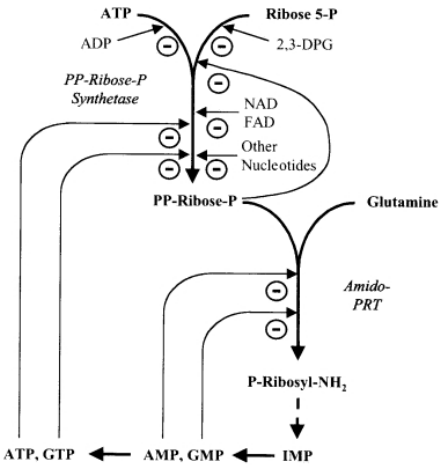


Fig. 4.2. A biochemical map of the purine biosynthesis pathway [234].

In systems biology many formalisms have been defined to graphically represent biopathways. One of the most important of these notations is the Systems Biology Graphical Notation (SBGN) [242], which is briefly described in Section 6.1. Within the framework of MP systems a specific representation of biochemical systems, named *MP graphs*, has been introduced in [144]. MP graphs represent biochemical

reactions as graphs with two levels: the first level describes the *stoichiometry* of reactions, while the second level expresses the *regulation*, which tunes the flux of every reaction (i.e., the quantity of chemicals transformed at each step) depending on the state of the system. MP graphs provide an intuitive way to model biological pathways overcoming the rather complicated use of traditional mathematical descriptions, such that of ODE systems, which need a strong mathematical background in order to be understood.

Definition 4.3 (MP graph). *An MP graph is a construct [144]:*

$$G = (X_G, V_G, R_G, \Phi_G, I, E, o)$$

where:

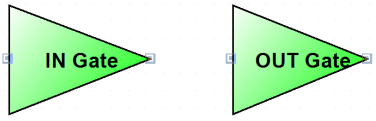
- X_G *is a finite set of* ***substance nodes*** *representing substance types. These nodes are visualized by blue circles or ellipses containing the substance name;*

- V_G *is a finite set of* ***parameter nodes*** *representing system variables (i.e., pressure, pH, temperature, etc.). Each parameter node is visualized by an orange rectangle or square containing the parameter name. It is usually associated to an analytical expression $h_v(i)$ which computes the parameter evolution function at each step i (as reported in Definition 6);*

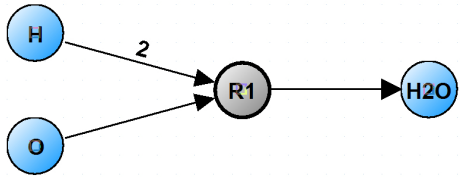
- R_G *is a finite set of* ***reaction nodes*** *representing biochemical reactions among substances. These nodes are visualized by gray circles or ellipses labeled by the name of a reaction and they act as hubs for all the substances nodes involved in a reaction (reactants and products). Each reaction node is connected with a flux node (explained at the next point) which regulates the amount of matter transformed by the reaction itself.*

- Φ_G *is a finite set of* ***flux nodes*** *one-to-one related with reaction nodes. Each flux node is visualized by a red rounded-corner rectangle or square containing the flux name. It is usually associated to an analytical expression $\varphi_r(q)$ which computes reaction fluxes in accordance with the current state q of the system (as reported in Definition 6);*

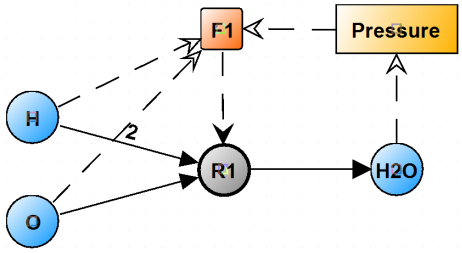
- *I is a set of two nodes: the* ***input gate node*** *and the* ***output gate node****. These nodes are visualized by green triangles labeled respectively by the "in gate" or the "out gate" string. They mark reaction nodes in which substances are created or destroyed;*



- *E is a set of direct edges between nodes. There are two kinds of edge: stoichiometric edges and regulatory edges.* ***Stoichiometric edges*** *are plain edges that connect substance nodes (reactants) to reaction nodes, or reaction nodes to substances nodes (products). They possibly have labels denoting reaction stoichiometry if it is different from* 1 *(e.g., label* $'2'$ *on the edge from* H *to* R_1*, in the picture below).*



Regulatory edges *are dashed edges of two types: the first type, characterized by a black arrow, connects each flux node to the reaction node it regulates (e.g., edge from flux node* F_1 *to reaction node* R_1*, in the picture below); the second type, characterized by a white arrow, connects* i) *substance and parameters nodes involved in a regulation function to the flux node containing the function itself (e.g., edge from substance node* H *and parameter node* $Pressure$ *to flux node* F_1*, in the next picture),* ii) *substance nodes involved in a parameter evolution function to the related parameter node (e.g., edge from substance node* H_2O *to parameter node* $Pressure$*, in the next picture).*



- *o is an* ***organism node*** *that contains the organism name, the molar unit* ν *of the MP system, its time unit* τ *and an additional organism description.*

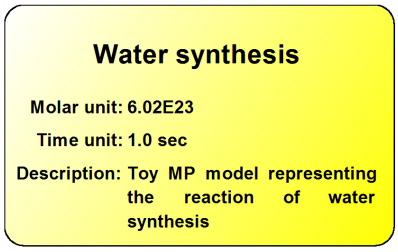


Figure 4.3 shows the MP graph corresponding to the toy MPF system of Table 4.1. It has three substance nodes represented by the blue circular nodes A, B and C. Reaction nodes $R1$, $R2$ and $R4$, connected with input and output gates (green triangles), represent respectively the entrance of substances A and B into the system and the expulsion/degradation of substance C. Reaction node $R3$ represents the chemical rule $A + B \rightarrow CC$. Each reaction node has an input dashed arrow from the only flux node that regulates it (e.g., flux node F_1 for reaction node R_1). The only parameter involved in the MP graph is *Pressure*, whose value evolves with a function depending on the substance C. This is showed by the dashed edge from node C to node *Pressure*. Finally, the isolated node, on the top, represents the organism: it contains the organism name, i.e., "Toy MPF system", the molar unit $\nu = 1000.0$ of the MP system, its time unit $\tau = 2.0$ *sec*, and an additional organism description. Fluxes are computed by means of algebraic formulae contained in flux nodes. MP graphs show the arguments of each flux/parameter regulation function by means of dashed edges from substances and parameters to flux/parameter nodes.

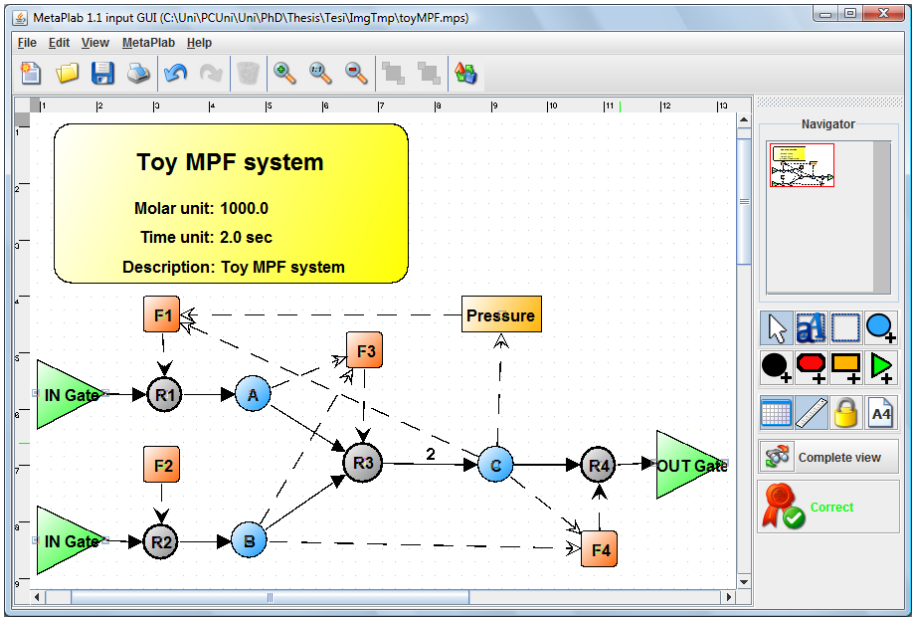


Fig. 4.3. The MP graph representation of the MPF toy example described in Table 4.1.

The graphical user interface (GUI) used for drawing the MP graph of Figure 4.3 is part of the software *MetaPlab* [18,34,146], a tool developed to provide modelers, as well as biologists, with a reliable and easy-to-use simulation environment for computing systems dynamics. The last release of this software, available at the MetaPlab website [241], is presented Chapter 6.

4.4 Equivalence between MP systems and ODEs

The equivalence between MPR systems and ODEs has been investigated in [68], where two procedures for mapping, respectively, MPR systems to ODEs, and ODEs to MPR systems, have been proposed. The first result shows that MPR systems translate naturally to ODE systems based on mass action law. Let us refresh the following notation, which will be used in the next definitions:

Notation 1 (MP notation)

- *Each reaction $r \in R$ is denoted by $r : \alpha_r \to \beta_r$, where α_r identifies the multiset of reactants of reaction r, and β_r identifies the multiset of products of reaction r;*
- *$|\alpha_r|_x$ is the number of occurrences of x in α_r;*
- *$|\beta_r|_x$ is the number of occurrences of x in β_r;*
- *$R_\alpha(x) = \{r \in R \mid x \in \alpha_r\}$;*
- *$R_\beta(x) = \{r \in R \mid x \in \beta_r\}$;*
- *$\mathbb{A}_{x,r} = |\beta_r|_x - |\alpha_r|_x$;*
- *$\mathit{\Pi}(\alpha_r) = \prod_{x \in \alpha_r} q(x)^{|\alpha_r|_x}$;*

The next definition formalizes the transformation of an MPR system to an ODE system [68]:

Definition 9 (MP-ODE transformation [68]) *Let $M = (X, V, R, Q, \tau, q_0, F, \Psi, H, \nu, \mu)$ be an MPR system, where F is the set of reaction maps, Ψ is the set of inertia maps, and the other elements are as in Definition 6. For every $x \in X$, the following is the ODE-transformation of M:*

$$x' = \sum_{r \in R} \mathbb{A}_{x,r} \cdot f_r(q) \cdot \mathit{\Pi}(\alpha_r). \tag{4.14}$$

A comprehensive description and motivation of this formula is reported in [68].

On the other side, given a set of reaction rules and an ODE system S describing their differential dynamics, an MPR system having S as ODE-transformation can be derived [68]. The procedure for generating such an MPR system involves the synthesis of reaction and inertia maps from differential equation terms. Many MPR models of real biological processes have been generated from ODE models by means of this mapping procedure. A few examples are the Belousov-Zhabotinsky reaction (in the Brusselator formulation) [19, 21], the Lotka-Volterra dynamics [19, 69, 145] and the mitotic cycles in early amphibian embryos [144].

Another significant result concerns the identification of a class of MP systems whose temporal evolution asymptotically converges, at the limit of infinitesimal

time, to the solution of ODE systems of Equation (4.14). For this class of MP systems, metabolic and differential perspectives meet each other, so that "mass partition" (characterizing MP systems) and "time partition" (characterizing numerical solution of ODE) turn out to be strictly related one to the other [68].

Definition 10 (Uniformly transparent MPR system [68]) *For some $c \in \mathbb{R}$, an MPR system is $c-$uniformly transparent if every substance $x \in X$ has a constant inertia $\psi_x(q) = c$.*

Definition 11 (Input closed MPR system [68]) *An MPR system is input closed if no rules $r \in R$ exists in the system such that $\alpha_r = \lambda$.*

Theorem 4.4. *The computation of a non-cooperative, input closed c-uniformly transparent MP system converges, as $c \to \infty$, to the solution, when it is unique, of the ODE system provided by the MP-ODE transformation.*

The reader can refer to [68] for a formal proof of this theorem. Another significant result reported in the same paper ensures the existence, for every MP system M, of a non-cooperative MP system M' having the same ODE-transformation of M. However, this non-cooperative MP system is not uniquely determinable, and even if all the possible non-cooperative MP systems obtained from M provide a solution that converges to the same ODE system, the speed of this convergence depends on the specific non-cooperative MP system. From this point of view MP systems can be seen as machines for computing approximations of ODE system solutions.

4.5 Equivalence between MP systems and hybrid functional Petri nets

In this section some original results are presented, about an equivalence between MPF systems and a special class of Petri nets. These results, presented for the first time in [31], are here reported and expanded.

Petri nets were introduced in 1962 by Carl Adam Petri [174] as logic circuits to describe concurrency in artificial systems (i.e., operative systems or event-driven systems) [191]. They have been recently employed to model biological pathways [96, 189] and in particular metabolic processes [95]. The recent development of MP systems theory, based on fluxes, shows deep similarities with a novel extension of Petri nets, named *hybrid functional Petri nets* (HFPN) [154]. They have been introduced to overcome some drawbacks of traditional Petri nets for modeling biochemical pathways. A software has also been developed to compute biological simulations of HFPN models [53, 164, 237].

In the following, a thorough comparison between the formalism of MP systems and that of hybrid functional Petri nets is presented, in order to highlight similarities and differences as well as feasibilities to model biochemical systems. A first investigation about this comparison can be found in [30, 35], while the results reported in the following have been published in [31]. A formal description of the HFPN graphical model is introduced in the next Section 4.5.1. Section 4.5.2 and Section 4.5.3 respectively show the equivalence between the two formalisms

and point out two mapping procedures (from one to the other and vice versa). In Section 4.5.4 these procedures have been successfully employed for modeling and simulating the *lac* operon gene regulatory mechanism and glycolytic pathway.

4.5.1 Hybrid functional Petri nets: a formalization

A *Petri net* [174, 191] is a network mainly consisting of four kinds of elements: *i) places*, *ii) transitions*, *iii) arcs*, *iv) tokens* (see Figure 4.4). According to the traditional notation, a *place* P_γ can hold a nonnegative integer number of tokens m_γ as its content. The amount of tokens in all places identifies the *state* of a Petri net. Transitions (e.g., T in Figure 4.4) are elements equipped with firing rules specified by: *i)* the *arcs* connecting the places, *ii*) the number of tokens (expressed as arc labels) to move from input to output places, and *iii)* the *speed* of the token transition.

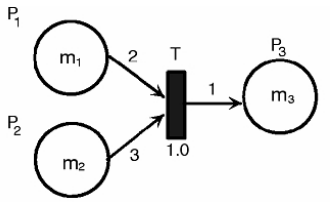


Fig. 4.4. A simple Petri net [154]: P_1 and P_2 are input places and P_3 is an output place of the transition T; m_1, m_2 and m_3 are the amount of tokens held by P_1, P_2, and P_3; the arc labels 2, 3, 1 state that T can fire if $m_1 \geq 2$ and $m_2 \geq 3$, by removing two tokens from P_1 and three from P_2 and adding one token to P_3. The firing speed is assumed to be constant and the label 1.0 under the transition symbol means that T must wait 1.0 step before firing.

Hybrid functional Petri nets [154] extend traditional Petri nets by introducing continuous places and transitions, and adding special arcs, to overcome Petri net drawbacks in modeling biopathways. Figure 4.5 shows in detail the main elements of this model, which are basically: *i)* the *discrete places and transitions* inherited by the traditional formalism, *ii)* the *continuous places and transitions*, and *iii)* three kinds of *arcs* to connect places and transitions. Namely, a discrete (continuous) *normal* input arc has an integer (real) label, which states a strict lower bound for the place amount that causes the transition activation; a discrete (continuous) *inhibitory* input arc has an integer (real) label, which states an upper bound of the place amount, that causes the transition activation but it does not remove any token; a discrete (continuous) *test* arc works as a normal input arc but it does not remove any token from the input place.

HFPN discrete components have been employed (in glycolytic pathway simulation [53] and in circadian rhythms of Drosophila simulation [154]) for pathways regulatory mechanisms modeling. In fact, transcription switches, feedbacks and promotion/inhibition mechanisms can intuitively be modeled by discrete elements that can namely stand for DNA binding sites or trigger conditions.

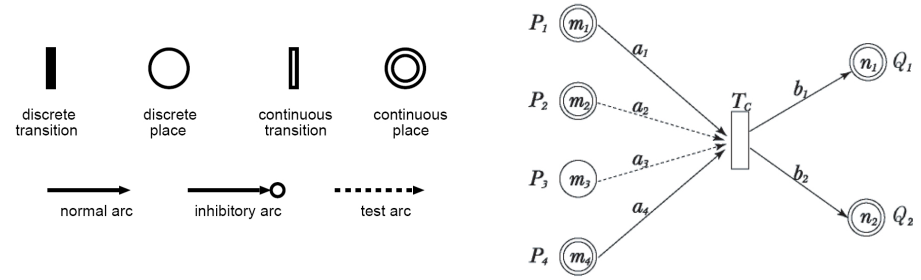


Fig. 4.5. On the left, the basic elements of HFPN [154]. Only normal arcs are able to move tokens, and their labels represent strict lower bound conditions for the transition firing. Inhibitory and test arcs represent respectively upper bound and lower bound conditions that must be satisfied for the transition firing. **On the right**, an HFPN continuous transition T_C [154]. P_1, P_2, P_4, Q_1, Q_2 are continuous places, P_3 is a discrete place; m_1, m_2, m_3, m_4, n_1 and n_2 represent the content of the corresponding places. Labels a_2, a_3 denote test arcs, the other are normal arcs.

In general [154], a (continuous or discrete) transition T specifies three functions (on the right side of Figure 4.5):

- the **firing condition** given by a predicate $c(m_1(i), \ldots, m_n(i))$ (if T is continuous, as long as this condition is true T fires continuously; if T is discrete, whenever the condition is true T gets ready to fire),
- a nonnegative function $f_j(m_1(i), \ldots, m_n(i))$ for each input arc a_j, called **consumption firing speed** which states the (real, integer) number of tokens removed by firing from P_j through arc a_j,
- a nonnegative (continuous, or integer) function $g_j(m_1(i), \ldots, m_n(i))$ called **production firing speed** for each output arc b_j, that specifies the number of tokens added by firing to Q_j through arc b_j.

If a_j is a test or an inhibitory input arc, then we assume $f_j \equiv 0$ and no amount of tokens is removed from P_j through the input arc a_j.

Moreover, for discrete transitions, the **delay function** is given by a nonnegative integer valued function $d(m_1(i), \ldots, m_n(i))$. If the firing condition gets satisfied at time i, the corresponding transition acquires the chance of firing after a number of steps equal to its delay $d(m_1(i), \ldots, m_n(i))$. The transition actually fires if and only if the firing condition does not change during the delay time.

A time unit is assumed, called **Petri time**, in terms of which the firing speeds and the discrete transition *delays* (waiting time before firing) are given. In the case that simulation granularity must be increased, a fraction of Petri time named **sampling interval** is considered.

An interesting difference between discrete and continuous transitions is the firing policy. A discrete transition moves the expected amount of tokens (equal to the firing speed value) in only one step of the sampling interval, while the continuous transition moves it in one Petri time unit, though step by step with respect to the sampling interval.

The following definition outlines the mathematical structure of HFPN.

Definition 12 (HFPN) *An HFPN is a construct*

$$N = (P, T_1, T_2, I, E, S, s_0, Pt, SI, D, C, F, G)$$

where:

1. $P = \{p_1, \ldots, p_n\}$ *is a finite set of* **discrete** *or* **continuous places**;
2. $T_1 = \{t_1, \ldots, t_k\}$ *and* $T_2 = \{t_{k+1}, \ldots, t_z\}$ *are finite sets of* **continuous** *and* **discrete** ***transitions*** *respectively, where* $T_1 \cap T_2 = \emptyset$ *and we define* $T = T_1 \cup T_2$;
3. $I \subseteq P \times T$ *and* $E \subseteq T \times P$ *are respectively the* ***normal input arcs*** *and the* ***output arcs*** *of the transitions, given by the stoichiometry of the modeled system;*
4. S *is the set of* ***states****, that are functions* s *from the places of* P *to real numbers. If we assume some order among the elements of* P*, and an instant* i *ranging in the set of natural numbers, then the state* s *at the time* i *can be identified as the real vector* $s_i = (m_1(i), m_2(i), \ldots, m_n(i))$*, also denoted by P[i];*
5. $s_0 \in S$ *is the* ***initial state****, that is,* $s_0 = (m_1(0), m_2(0), \ldots, m_n(0))$;
6. Pt *is the time unit of the model (related to that one of the modeled real system), called* ***Petri time****, whereas* SI *is the* ***sampling interval****, which represents the number of petri times between two computational steps;*
7. $D = \{d_{k+1}, \ldots, d_z\}$ *is a finite set of* ***delays****, given by nonnegative functions* $d_j : S \to \mathbb{N}$ *which specify the time that the corresponding discrete transition* t_j *must wait before firing;*
8. $C = \{c_1, \ldots, c_z\}$ *is a set of* ***firing conditions****, given by boolean functions on the states* $c_j : S \to \{0, 1\}$ *which control the activation of the corresponding transition* t_j;
9. $F = \{f_1, \ldots, f_{|I|}\}$ *is a set of* ***consumption firing speeds****, given by non-negative functions* $f_x : S \to \mathbb{R}$*, where* $x = (p_\gamma, t_j)$*,* $\gamma = 1, \ldots, n$*,* $j = 1, \ldots, z$*, that specify the quantity which the transition* t_j *can remove from the place* p_γ *for each state* s*. For all* $x = (p_\gamma, t_j)$ *in which* p_γ *is a discrete place, we have* $f_x : S \to \mathbb{N}$;
10. $G = \{g_1, \ldots, g_{|E|}\}$ *is a set of* ***production firing speeds****, given by non-negative functions* $g_y : S \to \mathbb{R}$*, where* $y = (t_j, p_\gamma)$*,* $\gamma = 1, \ldots, n$*,* $j = 1, \ldots, z$*, that specify the quantity which the transition* t_j *can add to the place* p_γ *for each state* s*. For all* $y = (t_j, p_\gamma)$ *in which* p_γ *is a discrete place, we have* $g_y : S \to \mathbb{N}$.

Given an HFPN $N = (P, T_1, T_2, I, E, S, s_0, Pt, SI, D, C, F, G)$, the algorithm presented in Table 4.3 computes the first h consecutive states, for $h \in \mathbb{N}$, according to the following strategy.

As a first computational step, the algorithm initializes the functions l_j for all the discrete transitions t_j (instructions 2 and 3). These functions are initially defined by the algorithm to control the waiting of discrete transition delays, their null value corresponds to a positive delay in the initial state.

In order to compute the state $P[h]$ of the HFPN system, it is necessary to check how the discrete transitions behave in the previous states of the system. In fact, each of them can be waiting either for its condition to be true, or for the delay to pass in order to fire, or it can be ready to fire. This information is kept by the function l_j for each discrete transition t_j, then all the values $l_j(1), l_j(2), \ldots, l_j(h-1)$

HFPN-Evolution(N,h)

1. **begin**
2. **for** $j = k+1, \ldots, z$ **do** // $t_j \in T_2$
3. **if** $d_j(P[0]) = 0$ **then** $l_j(0) = 1$ **else** $l_j(0) = 0$;
   od
4. **for** $i = 0, \ldots, h-1$ **do**
5. **for** $\gamma = 1, \ldots, n$ **do** $m_\gamma(i+1) := m_\gamma(i)$; **od**
6. **for** $j = 1, \ldots, k$ **do** // $t_j \in T_1$
7. **if** $c_j(P[i]) = 1$ **then**
8. **for** $\gamma = 1, \ldots, n$ **do**
9. $x := (p_\gamma, t_j)$;
10. $y := (t_j, p_\gamma)$;
11. **if** $x \in I$ **then** $m_\gamma(i+1) := m_\gamma(i+1) - f_x(P[i]) \cdot SI$;
12. **if** $y \in E$ **then** $m_\gamma(i+1) := m_\gamma(i+1) + g_y(P[i]) \cdot SI$;
    od
    od
13. **for** $j = k+1, \ldots, z$ **do** // $t_j \in T_2$
14. **if** $l_j(i) = 0$ **then**
15. **if** $c_j(P[i]) = 1$ **then** $l_j(i + \frac{d_j(P[i])}{SI} - 1) := 1$;
16. **for** $k = 1, \ldots, \frac{d_j(P[i])}{SI} - 2$ **do**
17. $l_j(i+k) := 2$;
    od
18. **else** $l_j(i+1) := 0$;
19. **if** $l_j(i) = 1$ **then** // t_j is ready to fire
20. **if** $c_j(P[i]) = 1$ **then** //firing
21. **for** $\gamma = 1, \ldots, n$ **do**
22. $x := (p_\gamma, t_j)$;
23. $y := (t_j, p_\gamma)$;
24. **if** $x \in I$ **then** $m_\gamma(i+1) := m_\gamma(i+1) - f_x(P[i])$;
25. **if** $y \in E$ **then** $m_\gamma(i+1) := m_\gamma(i+1) + g_y(P[i])$;
    od
26. **if** $d_j(P[i]) > 0$ **then** $l_j(i+1) := 0$;
27. **else** $l_j(i+1) := 1$;
    od
28. **write** $(m_1(i+1), m_2(i+1), \ldots, m_n(i+1))$
    od
29. **end**

Table 4.3. HFPN Algorithm to compute the first h consecutive states of N.

need to be computed, and consequently all the states $P[1], P[2], \ldots, P[h-1]$. For each of the instants $1, \ldots, h-1$ (instruction 4), the state $P[i+1]$ is initialized by the current state $P[i]$ (instruction 5), and then processed by the firings as in the following.

There are two main for cycles (instructions 6 and 13): the first one controls the application of the continuous transitions (from T_1) and the second one con-

trols the application of the discrete transitions (from T_2) which are equipped with delay functions. By the first of these for cycles (instructions 6-12), each continuous transition with true condition fires, and the contents of the involved places are consequently modified (by instructions 11 and 12). The content of each place connected to the continuous transition by an arc of I decreases of $f_x(P[i]) \cdot SI$ (instruction 11). The content of each place connected to the continuous transition by an arc of E increases of $g_y(P[i]) \cdot SI$ (instruction 12).

In the last for cycle (instructions 13-27), the firing of discrete transitions t_j is ruled firstly by the value of a corresponding function l_j (instructions 14 and 19) and secondarily by the boolean value of the corresponding condition c_j (instructions 15 and 20). If a transition t_j has a null delay function, its function l_j is identically equal to 1 (instruction 27).

In the case of delay $d_j(P[i])$ greater than zero, l_j (instruction 14) has value 0 (initially and) whenever the transition has just fired and until the condition c_j is not true. As soon as the condition is true (instruction 15), the function l_j is set to 2 (instruction 17) for each time of the next d_j steps, and it is set to 1 (instruction 15) at the time the transition will be ready to fire (after the delay). Then, the transition fires only if the condition is true (instruction 20). At this point the function l_j is set to 0 (denoting that the transition is waiting for the condition to be true).

HFPN is an intrinsically parallel model which assumes the simultaneous firing of all the transitions that can fire in every computational step. The algorithm presented in Table 4.3, however, sequentially obtains an equivalent evolution of the system if, for every step, the quantity of tokens contained in each place p_γ is greater than or equal to the sum of the quantities actually taken from that place. This is an assumption considered more or less explicitly in all the HFPN literature.

We want to remark the complex structure of the algorithm with respect to the simplicity of the finite difference system which computes the MP dynamics (Equation (4.2)). This aspect is not only a matter of elegance. In fact, within the framework of MP systems, a theory was developed [140] which allows us to define an MP model of an observed dynamics. The key point of this theory is the discovery of flux regulation maps by means of suitable algebraic manipulations of time series of observed states.

4.5.2 Mapping HFPN to MP systems

MP systems and HFPN have been both developed for biological dynamics modeling, but they have a very different theoretical background. Both the formalisms are mainly based on *i)* reactant elements (*substances* and *places*), *ii)* rules (*reactions* and *transitions*) that state how the substances can interact with each other, and *iii)* dynamics elements (*fluxes* and *firing speeds/conditions*) that control the system evolution by computing the amount of substances moved in the network at each computational step. Nevertheless, in order to model biological processes, the firing speeds $f_{(p_\gamma,t_j)}$ and $g_{(t_j,p_\gamma)}$ have to be linked to transitions rather than to arcs. This argument has been confirmed also by analyzing some HFPN models and testing of an HFPN simulator called *Cell Illustrator*^{TM} [164, 237]. We can assume arc firing speeds f_j to have this form:

$$f_{(p_\gamma,t_j)}(s_i) = \bar{f}_j(s_i) \cdot w_{\gamma,j} \qquad \text{and} \qquad g_{(t_j,p_\gamma)}(s_i) = \bar{f}_j(s_i) \cdot w_{\gamma,j} \tag{4.15}$$

where $w_{\gamma,j} \in \mathbb{N}$, $\gamma = 1, \ldots, |P|$, $j = 1, \ldots, |T|$. In (4.15), $\bar{f}_j$ denotes a function common to the consumption and production firing speeds of the transition t_j, while $w_{\gamma,j}$ represents the weight of the arc (p_γ, t_j) or (t_j, p_γ).

The following mapping associates to a given HFPN $N = (P, T_1, T_2, I, E, S, s_0, Pt, SI, D, C, F, G)$ an MP system $M(N) = (X, R, V, Q, \Phi, \nu, \mu, \tau, q_0, \delta)$ that will be proved to be dynamically equivalent to N.

HFPN-to-MP mapping procedure

1. $X = P$. Places $p_1, \ldots, p_n$ of N are one to one mapped into substances $x_1, \ldots, x_n$ of M;

2. $R = T$. Transitions of N are mapped into reactions of M. Stoichiometric matrix $\mathbb{A}$ is generated according to Equations (4.15), because firing speeds can be related to transitions rather than to arcs. In this way, every weight $w_{\gamma,j}$ of a consumption firing speed $f_{(p_\gamma,t_j)} = \bar{f}_j \cdot w_{\gamma,j}$ (see Equations (4.15)) generates the element $a_{\gamma,j} = -w_{\gamma,j}$ of the stoichiometric matrix $\mathbb{A}$, while every weight $w_{\gamma,j}$ of a production firing speed $g_{(t_j,p_\gamma)} = \bar{f}_j \cdot w_{\gamma,j}$ generates the element $a_{\gamma,j} = w_{\gamma,j}$;

3. $V = \{v_1, \ldots, v_{z-k}\}$. Elements of T_2 (discrete transitions t_j) are mapped into parameters (v_{j-k}), having initial values related to the corresponding delays from D (see item 8 below) and evolving according to functions $\{h_1, \ldots, h_{z-k}\}$ defined as follows. Each function h_j controls the evolution of the parameter v_j as a "counter of waiting", set by the delay $d_{j+k} \in D$, which supports the simulation of the discrete transition t_{j+k}. Each function h_j can be defined as:

$$\begin{aligned} h_j(q) = {} & (q(v_j) \leq 0)(d_j(q)/SI - 1) + \\ & + ((c_j(q) \wedge q(v_j) = d_j(q)/SI - 1) \vee \\ & \vee (0 < q(v_j) < d_j(q)/SI - 1))(q(v_j) - 1) \end{aligned}$$

   where the first term of the sum reinitializes the counter v_j to $d_j(q)/SI - 1$ if $q(v_j) \leq 0$, while the second term decreases the counter by one if either it is equal to $d_j(q)/SI - 1$ and the condition $c_j(q)$ is true, or the counter value is between 0 and $d_j(q)/SI - 1$. The initial value $d_j(q)/SI - 1$ corresponds to the number of steps that M performs during the time interval $d_j(q)$;

4. $Q = \{q \mid q : X \cup V \to \mathbb{R},\ q_{|X} \in S\}$ where $q_{|X}$ is the restriction of q to the set X;

5. $\Phi = \{\varphi_1, \ldots, \varphi_k, \varphi_{k+1}, \ldots, \varphi_z\}$. Regulation functions are deduced from T which has k continuous transitions and $z - k$ discrete transitions. They are defined as $\varphi_j(q) = c_j(q) \cdot \bar{f}_j(q) \cdot SI$ for $j \in \{1, \ldots, k\}$ and as $\varphi_j(q) = (c_j(q) \wedge q(v_{j-k}) \leq 0) \cdot \bar{f}_j(q)$ for $j \in \{k+1, \ldots, z\}$, where $q \in Q$, and $\bar{f}_j$ is determined by Equations (4.15). Boolean values have been mapped to integer numbers 0 (true) and 1 (false);

6. The value ν and the mass function μ of M are chosen with respect to the modeled system; indeed they do not have any corresponding component in N. They work as biological data measurements and they do not affect the system dynamics;

7. The time interval τ is the time between two computational steps of N, that is $\tau := SI \cdot Pt$;

8. $q_0 = (s_0, V[0])$, denoting the juxtaposition of the vectors s_0 and $V[0]$, where s_0 is the initial state of N and $V[0] = (\frac{d_{k+1}(s_0)}{SI} - 1, \ldots, \frac{d_z(s_0)}{SI} - 1)$, with $d_{k+1}, \ldots, d_z \in D$.

Finally, we notice that HFPN model defines a delay value for each discrete transition while MP systems can simulate the delays by using parameters. Actually, the delays do not have a biological counterpart in the real systems, and they are just computational tricks to describe some complex low-level processes (which needs a particular time interval to be performed) as black-boxes that return particular outputs after a delay time.

Both MP systems and HFPN dynamics are characterized by the temporal evolution of quantities that represent significant entities of the biological systems of interest. Therefore, to compare the two formalisms we consider the evolution of quantities related to substances and parameters in the MP systems, and related to places in the HFPN.

Theorem 1: [HFPN to MP] Given an HFPN $N = (P, T_1, T_2, I, E, S, s_0, Pt, SI, D, C, F, G)$, the MP system $M(N) = (X, R, V, Q, \Phi, \nu, \mu, \tau, q_0, \delta)$ obtained by applying the HFPN-to-MP mapping procedure has the same time evolution of N.

Proof: For every (discrete or continuous) place p_γ of P in N (with γ ranging from 1 to n) there exists a corresponding substance x_γ of X in M, such that, the content of p_γ is equal to the value of x_γ for every instant, that is, $m_\gamma(i) = x_\gamma[i]\ \forall i \in \mathbb{N}$. The proof is given by induction on the computational step i. The evolution of N is computed by the HFPN algorithm reported in Table 4.3 while Equations (4.2) and (4.3) compute the evolution of M. Since discrete places of N evolve only by discrete transitions (instructions 6-12 of the HFPN algorithm) and continuous places evolve by both discrete and continuous transitions (instructions 6-27), we analyze the two cases separately, while the base of the induction is common to the two cases.

Base. The HFPN-to-MP mapping procedure states that $q_0 = (s_0, V[0])$. As a consequence, for every $\gamma = 1, \ldots, |P|$ we have $x_\gamma[0] = m_\gamma(0)$.

Inductive step for discrete places. The content of a discrete place p_γ is modified only by discrete transitions (instructions 13-27 of the HFPN algorithm). Instruction 5 assigns $m_\gamma(i)$ to each γ-th component of the state at the instant $i+1$, and instructions 24-25 update $m_\gamma(i+1)$ for every input or output transition t_j related to p_γ, that satisfies both the conditions $l_j(i) = 1$ (instruction 19) and $c_j(s_i) = 1$ (instruction 20). In particular, the quantity $f_x(s_i)$ is removed from $m_\gamma(i+1)$ for

every output arc x of p_γ, while the amount $g_y(s_i)$ is added to $m_\gamma(i+1)$ for every input arc y of p_γ. The evolution equation is the following:

$$m_\gamma(i+1) = m_\gamma(i) - \sum_x f_x(s_i) + \sum_y g_y(s_i) \tag{4.16}$$

where $x \in \{(p_\gamma, t_j) \mid (p_\gamma, t_j) \in I,\ t_j \in T_2,\ c_j(s_i) = 1 \text{ and } l_j(i) = 1\}$ and $y \in \{(t_j, p_\gamma) \mid (t_j, p_\gamma) \in E,\ t_j \in T_2,\ c_j(s_i) = 1 \text{ and } l_j(i) = 1\}$. By replacing production and consumption firing speeds by means of Equations (4.15) and arc weights $w_{\gamma,j}$ with the related stoichiometric coefficients $a_{\gamma,j}$ (as defined at point 3 of the HFPN-to-MP mapping procedure), the next equation follows:

$$m_\gamma(i+1) = m_\gamma(i) + \sum_{j=k+1,\ldots,z} (l_j(i) = 1 \wedge c_j(s_i)) \cdot \bar{f}_j(s_i) \cdot a_{\gamma,j} \tag{4.17}$$

On the other hand, HFPN-to-MP mapping procedure maps discrete places p_γ to substances x_γ and discrete transitions t_j to reactions r_j having regulation functions $\varphi_j(q) = (c_j(q) \wedge q(v_{j-k}) \leq 0) \cdot \bar{f}_j(q)$. The evolution of a "discrete" substance x_γ is thus computed by the Equation (4.2) as:

$$x_\gamma[i+1] = x_\gamma[i] + \sum_{j=k+1,\ldots,z} (c_j(s_i) \wedge q(v_{j-k}) \leq 0) \cdot \bar{f}_j(s_i) \cdot a_{\gamma,j} \tag{4.18}$$

Equations (4.17) and (4.18) compute the same dynamics, i.e., $m_\gamma(i+1) = x_\gamma[i+1]$, indeed *i)* $m_\gamma(i) = x_\gamma[i]$ from the inductive hypothesis, *ii)* the two summations are equal; in fact, the first one adds the contribution $\bar{f}_j(s_i) \cdot a_{\gamma,j}$ for $j \in \{k+1, \ldots, z\}$ if $l_j(i) = 1$ and $c_j(s_i)$ is true, while the second one adds the same contribution if $q(v_{j-k}) \leq 0$ and $c_j(s_i)$ is true. The two conditions are equivalent since $l_j(i) = 1 \Leftrightarrow q(v_{j-k}) \leq 0$, in fact the function $l_j(i)$ is initialized (instructions 2-3) and updated (instructions 14-18 and 27) by the algorithm of Table 4.3 in order to be:

- $l_j(i) = 0$, if (at the step i) the transition t_j is waiting for the conditions c_j to be true,
- $l_j(i) = 1$, if (at the step $i+1$) the transition t_j will be ready to fire,
- $l_j(i) = 2$, if (at the step i) the transition t_j is waiting for the delay d_j to pass.

Finally, every parameter v_j is initialized to $\frac{d_{k+j}(s)}{SI} - 1$ and then it is decreased by one, at each step, by the regulation function h_j, in order to be zero (or less) as soon as the delay $d_{k+j}(s)$ is elapsed and the reaction r_j is ready to fire. This is a way to force the discrete transition (and corresponding reactions) to fire only if the delay is passed.

Inductive step for continuous places. A continuous place can be connected with both continuous and discrete transitions. Thus, its temporal evolution is computed by the for cycles of instructions 6-12 and 13-27 according to the following equation:

$$\begin{aligned} m_\gamma(i+1) = m_\gamma(i) &- \sum_{x_d} f_{x_d}(s_i) + \sum_{y_d} g_{y_d}(s_i) + \\ &- \sum_{x_c} f_{x_c}(s_i) \cdot SI + \sum_{y_c} g_{y_c}(s_i) \cdot SI \end{aligned} \tag{4.19}$$

where

- $x_d \in \{(p_\gamma, t_j) \mid (p_\gamma, t_j) \in I,\ t_j \in T_2,\ c_j(s_i) = 1 \text{ and } l_j(i) = 1\}$,
- $y_d \in \{(t_j, p_\gamma) \mid (t_j, p_\gamma) \in E,\ t_j \in T_2,\ c_j(s_i) = 1 \text{ and } l_j(i) = 1\}$,
- $x_c \in \{(p_\gamma, t_j) \mid (p_\gamma, t_j) \in I,\ t_j \in T_1 \text{ and } c_j(s_i) = 1\}$,
- $y_c \in \{(t_j, p_\gamma) \mid (t_j, p_\gamma) \in E,\ t_j \in T_1 \text{ and } c_j(s_i) = 1\}$.

Basically, Equation (4.19) appends to Equation (4.16) two terms (second row) related to continuous transitions.

By replacing production and consumption firing speeds by Equations (4.15), and arc weights $w_{\gamma,j}$ with the related stoichiometric coefficients $a_{\gamma,j}$ (as defined at point 3 of the HFPN-to-MP mapping procedure), we have:

$$\begin{aligned} m_\gamma(i+1) = m_\gamma(i) + \sum_{j=1,\ldots,k} c_j(s_i) \cdot \bar{f}_j(s_i) \cdot a_{\gamma,j} + \\ + \sum_{j=k+1,\ldots,z} (l_j(i) = 1 \wedge c_j(s_i)) \cdot \bar{f}_j(s_i) \cdot a_{\gamma,j} \end{aligned} \tag{4.20}$$

On the other hand, HFPN-to-MP mapping procedure maps continuous places p_γ to substances x_γ, discrete transitions t_j to reactions r_j having regulation functions $\varphi_j(q) = (c_j(q) \wedge q(v_{j-k}) \leq 0) \cdot \bar{f}_j(q)$, and continuous transitions t_j to reactions r_j having regulation functions $\varphi_j(q) = c_j(q) \cdot \bar{f}_j(q) \cdot SI$. The evolution of a "continuous" substance x_γ according to Equation (4.2) gives:

$$\begin{aligned} x_\gamma[i+1] &= x_\gamma[i] + \sum_{j=1,\ldots,z} a_{\gamma,j} \cdot \varphi_j(s_i) \\ &= x_\gamma[i] + \sum_{j=1,\ldots,k} c_j(s_i) \cdot \bar{f}_j(s_i) \cdot a_{\gamma,j} + \\ &\quad + \sum_{j=k+1,\ldots,z} (c_j(s_i) \wedge q(v_{j-k}) \leq 0) \cdot \bar{f}_j(s_i) \cdot a_{\gamma,j} \end{aligned} \tag{4.21}$$

Equations (4.20) and (4.21) compute the same dynamics, i.e., $m_\gamma(i+1) = x_\gamma[i+1]$, since *i)* $m_\gamma(i) = x_\gamma[i]$ from the inductive hypothesis, *ii)* the two summations having indices between 1 and k (related to continuous transitions) are equal, and *iii)* the summations having indexes between $k+1$ and z provide identical values for the considerations previously made for discrete places.

□

4.5.3 Mapping MP systems to HFPN

Given an MP system $M = (X, R, V, Q, \Phi, \nu, \mu, \tau, q_0, \delta)$, an hybrid functional Petri net $N(M) = (P, T_1, T_2, I, E, S, s_0, Pt, SI, D, C, F, G)$ having the same dynamics can be obtained by the following mapping.

MP-to-HFPN mapping procedure

1. $P = X \cup V$. All the substances and parameters are mapped into places, thus we can identify the following vectors $(p_1, \ldots, p_n) = (x_1, x_2, \ldots, v_1, v_2, \ldots)$;

2. $T_1 = R \cup \{t_{|R|+1}, \ldots, t_{|R|+2|V|}\}$, $T_2 := \emptyset$, then $T = T_1 \cup T_2 = T_1$. All the reactions of M are mapped into continuous transitions. Furthermore, $2|V|$ continuous transitions have been defined to control the places $p_{|X|+1}, \ldots, p_{|X|+|V|}$, related to parameters;

3. $I = \{(p_\gamma, t_j) \mid \gamma$ and j are positive natural numbers such that, either $a_{\gamma j} < 0 \wedge \gamma \leq |X| \wedge j \leq |R|$ or $|X| < \gamma \leq |P| \wedge j = |R| - |X| + \gamma\}$. The transition *input* arcs of N include both arcs corresponding to negative elements of the stoichiometric matrix $\mathbb{A}$ and arcs corresponding to transitions $t_{|R|+1}, \ldots, t_{|R|+|V|}$, respectively related to places $p_{|X|+1}, \ldots, p_{|X|+|V|}$ which map parameters of M;

4. $E = \{(t_j, p_\gamma) \mid \gamma$ and j are positive natural numbers such that, either $a_{\gamma j} > 0 \wedge \gamma \leq |X| \wedge j \leq |R|$ or $|X| < \gamma \leq |P| \wedge j = |R| + |V| - |X| + \gamma\}$. The transition *output* arcs of N include both arcs corresponding to the positive elements of the stoichiometric matrix $\mathbb{A}$ and arcs corresponding to transitions $t_{|R|+|V|+1}, \ldots, t_{|R|+2|V|}$, respectively related to places $p_{|X|+1}, \ldots, p_{|X|+|V|}$ which map parameters of M;

5. $S = Q$;

6. $s_0 = q_0$. Since $P = X \cup V$ we can easily identify the initial states of the two systems;

7. $Pt = \tau$, $SI := 1$. This setting is chosen in order to suitably compare the computational behaviors of the two systems. It allows N to evolve along the same computational steps of M, and to get simulations with respect to the same time interval τ. Furthermore, with this choice, firing speeds are the number of tokens moved in a single computational step, and delays are the number of steps to wait before the firing;

8. $D = \emptyset$. Delays are not included into the MP model M, thus the delay set of N turns out to be empty;

9. $C = \{c_j \mid c_j : S \to \{0, 1\},\ c_j(s) = 1$, where $j = 1, \ldots, |T|\}$. Firing conditions are identically set to 1 (*true*), because in M they are included into regulation functions Φ and evolution functions $\{h_v : \mathbb{N} \to \mathbb{R} \mid v \in V\}$, which are mapped to the firing speeds, as in points 10 and 11;

10. $F = \{f_{(p_\gamma, t_j)} \mid f_{(p_\gamma, t_j)} : S \to \mathbb{R},\ f_{(p_\gamma, t_j)}(s) = -\varphi_j(s) \cdot a_{\gamma,j}$ for $\gamma = 1, \ldots, |X|$, $j = 1, \ldots, |R|$, $(p_\gamma, t_j) \in I$ and $f_{(p_\gamma, t_j)}(s) = q(v_{\gamma - |X|})$ for $\gamma = |X| + 1, \ldots, |P|$ and $j = |R| + 1, \ldots, |R| + |V|\}$, where $q(v_{\gamma - |X|}) = s(p_\gamma)$ for the above definition of S. Consumption firing speeds are defined from the

regulation functions Φ and the stoichiometric matrix $\mathbb{A}$ for arcs connecting substances to reactions in M. They are defined as $s(p_\gamma)$ for arcs which control the places p_γ related to parameters $v_{\gamma-|X|}$, in order to remove the entire content of these places at each step;

11. $G = \{g_{(t_j,p_\gamma)} \mid g_{(t_j,p_\gamma)} : S \to \mathbb{R},\ g_{(t_j,p_\gamma)}(s) = \varphi_j(s) \cdot a_{\gamma,j}$ for $\gamma = 1,\ldots,|X|,\ j = 1,\ldots,|R|,\ (t_j,p_\gamma) \in E$ and $g_{(t_j,p_\gamma)}(s) = h_{\gamma-|X|}(s)$ for $\gamma = |X|+1,\ldots,|P|,\ j = |R|+|V|+1,\ldots,|R|+2|V|\}$. Production firing speeds are defined from the regulation functions Φ and the stoichiometric matrix $\mathbb{A}$ for arcs connecting reactions to substances in M. They are defined as $h_{\gamma-|X|}(s)$ for arcs which update places related to parameters $v_{\gamma-|X|}$. In this way, at each step, they set the new parameter value $h_{\gamma-|X|}(s)$ as content of these places.

Theorem 2: [MP to HFPN] Given an MP system $M = (X, R, V, Q, \Phi, \nu, \mu, \tau, q_0, \delta)$, the HFPN $N(M) = (P, T_1, T_2, I, E, S, s_0, Pt, SI, D, C, F, G)$ obtained by applying the MP-to-HFPN mapping procedure has the same time evolution of M.

Proof: For every element y_γ of $X \cup V$ in M (with γ ranging from 1 to n) there exists a corresponding place p_γ of P in N, such that, the value of y_γ is equal to the content of p_γ for every instant $i \in \mathbb{N}$, that is, $\forall i \in \mathbb{N}\ y_\gamma[i] = m_\gamma(i)$. The proof is given by induction on the computational step i and the evolution of N is computed by the algorithm reported in Table 4.3. Since substances and parameters in M evolve independently by Equations (4.2) and (4.3) respectively, we analyze separately these two elements, while the base of the induction is common to the two cases.

Base. The mapping procedure (4.5.3) states that $s_0 = q_0$. As a consequence, for every $\gamma = 1,\ldots,|X|$ we have $x_\gamma[0] = m_\gamma(0)$, and for every $\gamma = 1,\ldots,|V|$ we have $v_\gamma[0] = m_{\gamma+|X|}(0)$.

Inductive step for substances. The evolution of substances in M is computed by the Equation (4.2), whose γ-th component can be written as:

$$x_\gamma[i+1] = x_\gamma[i] + \sum_{j=1,\ldots,|R|} a_{\gamma,j} \cdot \varphi_j(q_i) \tag{4.22}$$

where, $\gamma = 1,\ldots,|X|$, and $a_{\gamma,j} > 0$ if x_γ is a reactant of the reaction r_j, $a_{\gamma,j} < 0$ if x_γ is a product of the reaction r_j, and $a_{\gamma,j} = 0$ otherwise.

Since N has only continuous transitions t_j, having conditions $c_j(s)$ identically true, its evolution from the state s_i to s_{i+1} is essentially computed by the instructions 5-12 of the HFPN algorithm (Table 4.3), starting from the assignment of $m_\gamma(i)$ to each γ-th component of the state at the instant $i+1$ (instruction 5). The *for* cycle of instructions 6-12 updates the content of every place p_γ by adding $g_y(s_i)$ from any input arc y or subtracting $f_x(s_i)$ from any output arc x, since $SI = 1$. The evolution equation of a place p_γ is the following:

$$m_\gamma(i+1) = m_\gamma(i) - \sum_x f_x(s_i) + \sum_y g_y(s_i) \tag{4.23}$$

where $x \in \{(p_\gamma, t_j) \mid (p_\gamma, t_j) \in I\}$ and $y \in \{(t_j, p_\gamma) \mid (t_j, p_\gamma) \in E\}$.

Given that $m_\gamma(i) = x_\gamma[i]$ for inductive hypothesis, and the firing speeds $f_{(p_\gamma,t_j)}(s)$ and $g_{(t_j,p_\gamma)}(s)$ have been defined by the procedure 4.5.3 as

$$f_{(p_\gamma,t_j)}(s) = -\varphi_j(s) \cdot a_{\gamma,j} \qquad \text{and} \qquad g_{(t_j,p_\gamma)}(s) = \varphi_j(s) \cdot a_{\gamma,j}$$

for $\gamma = 1, \ldots, |X|$ and $j = 1, \ldots, |R|$, the comparison between Equations (4.22) and (4.23) shows the equivalence between the evolution of each substance x_γ and the related place p_γ.

Inductive step for parameters. Parameters v_γ, with $\gamma = 1, \ldots, |V|$, evolve in M according to their regulation functions:

$$v_\gamma[i+1] = h_\gamma(q_i). \tag{4.24}$$

The evolution of the related places $p_{|X|+\gamma}$ is still computed by the instructions 5-12 of the HFPN algorithm (Table 4.3) because only continuous transitions, having identically true conditions, are involved. Instruction 5 initializes $m_{|X|+\gamma}(i+1)$ with $m_{|X|+\gamma}(i)$ and the *for* cycle at the instructions 6-12 update $m_{|X|+\gamma}(i+1)$ by the amounts computed by firing speeds $f_{(p_{|X|+\gamma},t_{|R|+\gamma})}(s_i)$ and $g_{(t_{|R|+|V|+\gamma},p_{|X|+\gamma})}(s_i)$. They are designed by the mapping procedure 4.5.3 to remove the current place content $m_{|X|+\gamma}(i)$, and to add the next value of v_γ, namely $h_\gamma(q_i)$. Since $SI = 1$, the new amount of $p_{|X|+\gamma}$ is:

$$\begin{aligned} m_{|X|+\gamma}(i+1) &= m_{|X|+\gamma}(i) - f_{(p_{|X|+\gamma},t_{|R|+\gamma})}(s_i) + g_{(t_{|R|+|V|+\gamma},p_{|X|+\gamma})}(s_i) \\ &= h_\gamma(s_i) \end{aligned} \tag{4.25}$$

which is identical to the value of $v_\gamma[i+1]$ computed by Equation (4.24).

□

4.5.4 The *lac* operon gene regulatory mechanism and glycolytic pathway

Glycolytic pathway is the network by which some organisms obtain carbon from glucose. The E. coli bacterium can synthesize carbon from glucose and lactose. If the bacterium grows in an environment with both glucose and lactose, then it consumes glucose, but if the environment contains only lactose, then E. coli synthesizes special enzymes that metabolize lactose by transforming it into glucose [4, 53].

A specific regulation mechanism placed in the *lac* operon allows the expression of the genes for each of these situations. Figure 4.6 shows the dual control of *lac* operon (represented by the horizontal line) along which *i*) gene *I* produces a *repressor* protein at a constant rate, *ii*) the *promoter* region allows the RNA polymerase to trigger for the operon transcription, *iii*) the *operator* region matches with the repressor protein to inhibit the transcription, and *iv*) the *Z*, *Y* and *A* genes produce the enzymes for the synthesis of glucose from lactose inside the cell [154].

Main elements. The main substances involved in the pathway (and reported in Figure 4.6) are *lac* repressor, allolactose, catabolite gene activator protein (*CAP*), cyclic AMP (*cAMP*) and β-galactosidase (*LacZ protein*).

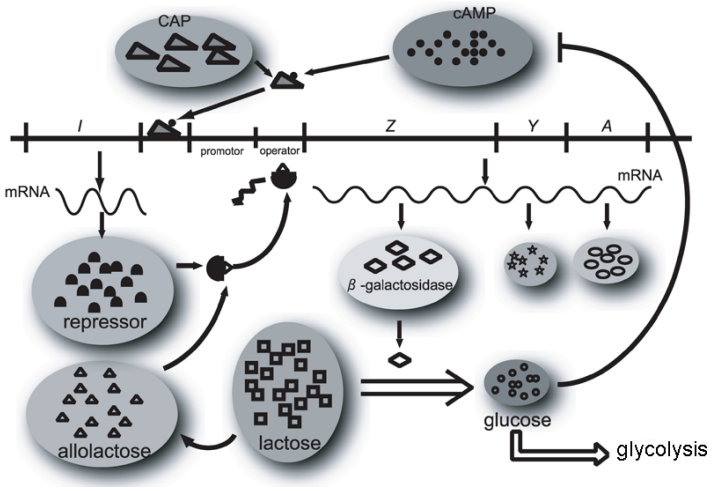


Fig. 4.6. The regulatory mechanism of glycolytic pathway in E. Coli [154].

Process description. In presence of glucose or in absence of lactose the operon transcription does not start, as the two control mechanisms described in the following allow the lac operon transcription (and the consequent glucose production) only in presence of lactose or absence of glucose.

The presence of glucose in the environment decreases the concentration of *cAMP* which no longer binds with *CAP*, while the absence of the gene activator *CAP-cAMP* turns off the operon transcription. The glucose decreasing enhances the concentration of *cAMP*, which promotes the operon activation by binding to *CAP*. The transcription of genes *Z*, *Y*, *A* starts only if the repressor protein is removed from the operator region, and this takes place only when lactose increases, producing the allolactose protein which binds to the repressor by removing it from the DNA. Operon genes produce both the β-galactosidase permease (LacY protein) and the β-galactosidase (LacZ protein) which allow, respectively, the lactose recruitment and its transformation to glucose, in order to keep on the glycolysis. Finally, glycolysis pathway breaks down the glucose by means of enzymes while releasing energy and pyruvic acid.

The pathway described above was modeled first by traditional Petri nets [189] and recently by means of HFPNs [53, 154]. The HFPN of Figure 4.8 has been designed in [53], where every substance has been modeled by a place and every chemical reaction by a transition with firing speeds and firing conditions (Figure 4.7).

This HFPN has been mapped, by the HFPN-to-MP mapping procedure, to the MP system of Figure 4.9, which resulted to have an equivalent dynamics. MP dynamics was computed by the simulator *MetaPlab* [34, 146, 241], while the software *Cell Illustrator*^{TM} [164,237] has been employed to compute the dynamics of the HFPN. A detailed description of the HFPN model and a complete analysis of its dynamics are proposed in [53].

The temporal evolutions of the two models have been compared by the six case studies considered in [53] (wild type, $lacZ^-$, $lacY^-$, $LacI^-$, $lacI^s$ and $lacI^{-d}$) and

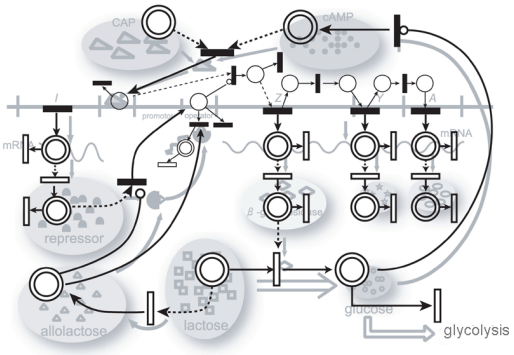


Fig. 4.7. A sketch of the modeling process by HFPN [154].

the equivalent results of Figures 4.10 and 4.11 have been achieved. Each column shows the dynamics of five substances (i.e., lactose outside the cell, lactose inside the cell, glucose, LacZ protein and LacY protein) for a specific case study. Figure 4.10 displays the results achieved for the HFPN dynamics, while Figure 4.11 is related to the equivalent MP dynamics.

"*Wild type*", showed in the first columns, represents the healthy case. It starts with the degradation of the glucose by the glycolytic pathway, finally allowing the lactose homing and its transformation to glucose. In this way, the lactose outside the cell decreases and the lactose inside the cell increases together with the glucose concentration which re-activates the glycolytic pathway. In the second columns, $lacZ^-$ is a mutant which disables the transformation of lactose in glucose. To perform this test the reaction (transition) involved in the translation of β-galactosidase has been deleted from the MP system (HFPN), thus, the lactose enters the cell but it is not transformed into glucose. The glycolytic pathway is not re-activated and the operon transcription keeps on producing a great amount of LacY protein. The $lacY^-$ mutation disables the capability of the cell to recruit lactose from the environment. The third columns in both the figures show that LacZ starts to grow when glucose is finishing, while LacY remains absent because the mutation inhibits the LacY mRNA transcription.

Figures 4.10 and 4.11 show equivalent results also for *lac* repressor mutants. In the case study of $lacI^-$ mutant, the repressor function is disabled and the Lac operon transcription keeps on even in absence of glucose and lactose. The behavior of $lacI^{-d}$ mutant, is similar to $lacI^-$, indeed in both cases the repressor inhibition forces the LacZ and LacY production, even after the complete consumption of lactose, leading to a great amount of these proteins. Finally, $lacI^s$, in the fifth column, is a mutant for which the *lac* operon transcription is disabled, thus preventing the synthesis of glucose from lactose. The graphical results show that lactose, LacZ and LacY are just slightly degraded while glucose is not reproduced after its termination.

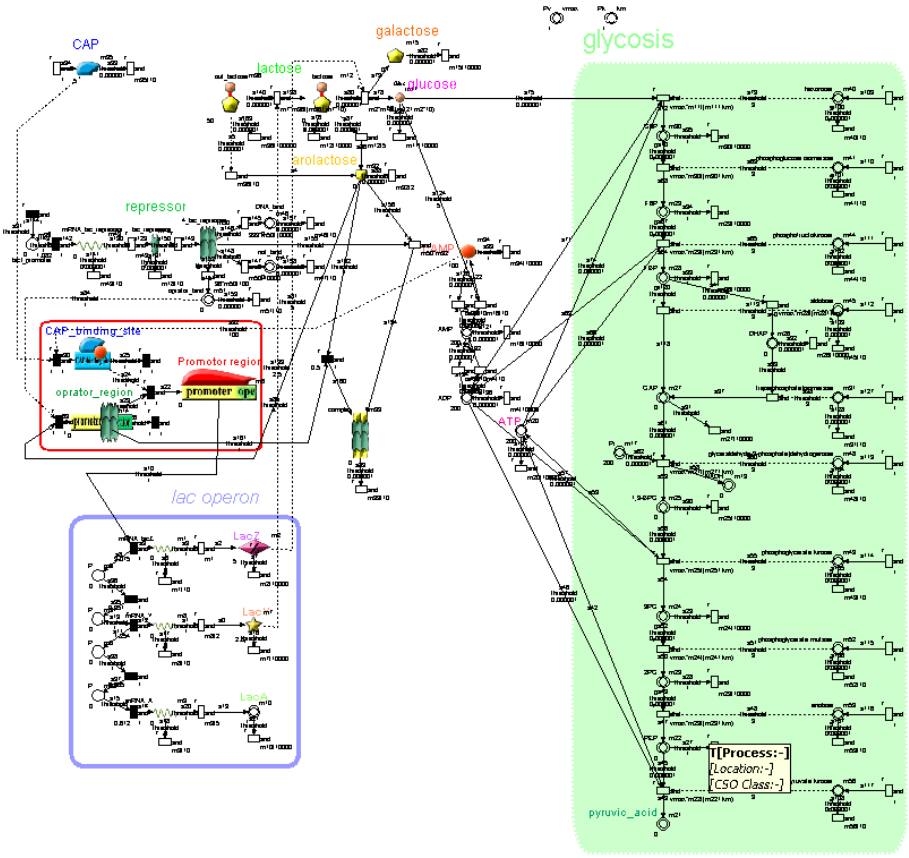


Fig. 4.8. The *lac* operon gene regulatory mechanism and glycolytic pathway [154] modeled by HFPN. The picture has been obtained by the graphical user interface of the software Cell Illustrator^{TM}.

4.6 Flux discovery: the metabolic log-gain theory

MP systems proved to be relevant in the analysis of metabolic processes, since they enable to simulate pathway behaviors under different environmental conditions and to understand internal mechanisms of biological systems. However, a key problem of MP modeling concerns models design. In order to generate a new MP model of a biochemical system we basically need to know: *i)* the substance types and the parameters involved in the process (i.e., sets X and V), *ii)* the reactions occurring among substance types (i.e., set R and stoichiometric matrix $\mathbb{A}$), *iii)* the flux of matter transformed by each reaction during the system evolution (i.e., set Φ). Even if substance types, parameters, and reactions are often available in the literature, the microscopic nature of chemical elements, their huge amount in real systems and their complex interactions make it difficult to measure reaction fluxes. The *log-gain*

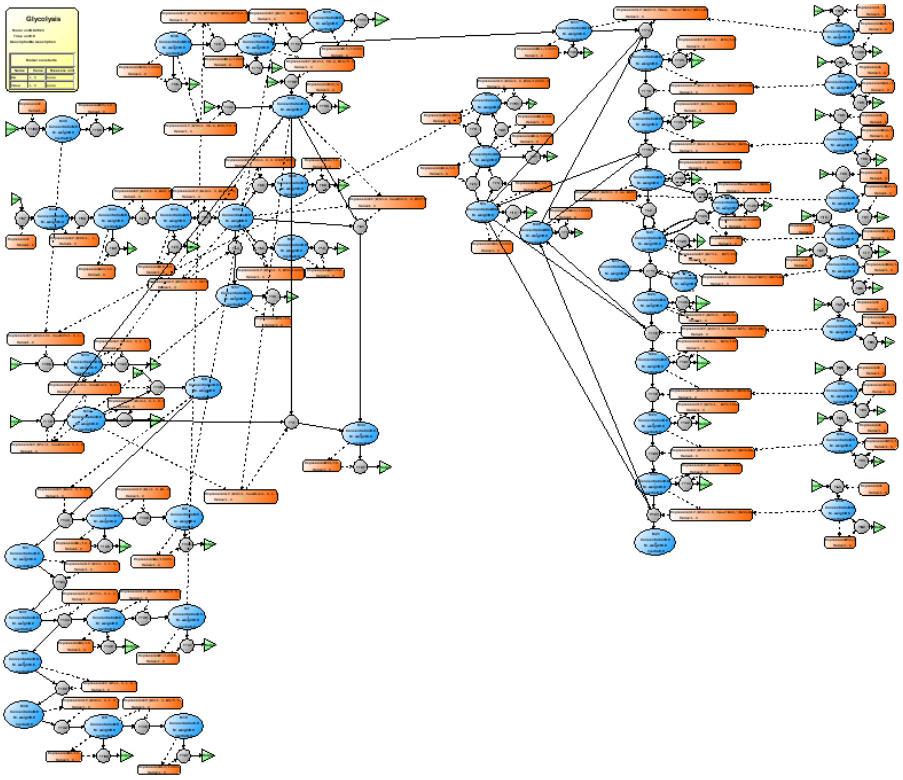


Fig. 4.9. The *lac* operon gene regulatory mechanism and glycolytic pathway modeled by MP systems (figure from [31] with permission). The network has been mapped from the HFPN of Figure 4.8 by the HFPN-to-MP mapping procedure. An enlarged version of this picture is published at the web page [153].

theory[1] [140–142] is a tool for deducing fluxes, within an acceptable approximation, from observed time evolutions of the system. In this context, to "observe" time evolutions means to measure, with a sufficient accuracy, the (molar) quantities of all different kinds of molecules and the values of parameters over a sequence of time instants.

Nowadays biologists have several high-throughput experimental techniques able to provide time-series of substance quantities and chemo-physical parameters [133, 233]. The theoretical framework provided by the log-gain theory, based on some acknowledged biological principles, aims to interpret this huge amount of data and to employ them into MP models for regulation functions synthesis. In fact, two main steps have to be performed in order to infer flux regulation functions from experimental data: *i)* to compute flux time-series from observed substance and parameter time-series, *ii)* to synthesize regulation functions from substance,

[1] Notice that the term "log-gain" is due to the fact that in differential notation (with respect to the time) the relative variation of a value, i.e., $\Delta(x)/x$ becomes $\frac{dx}{dt}/x$, which is the same as $\frac{d(lgx)}{dt}$.

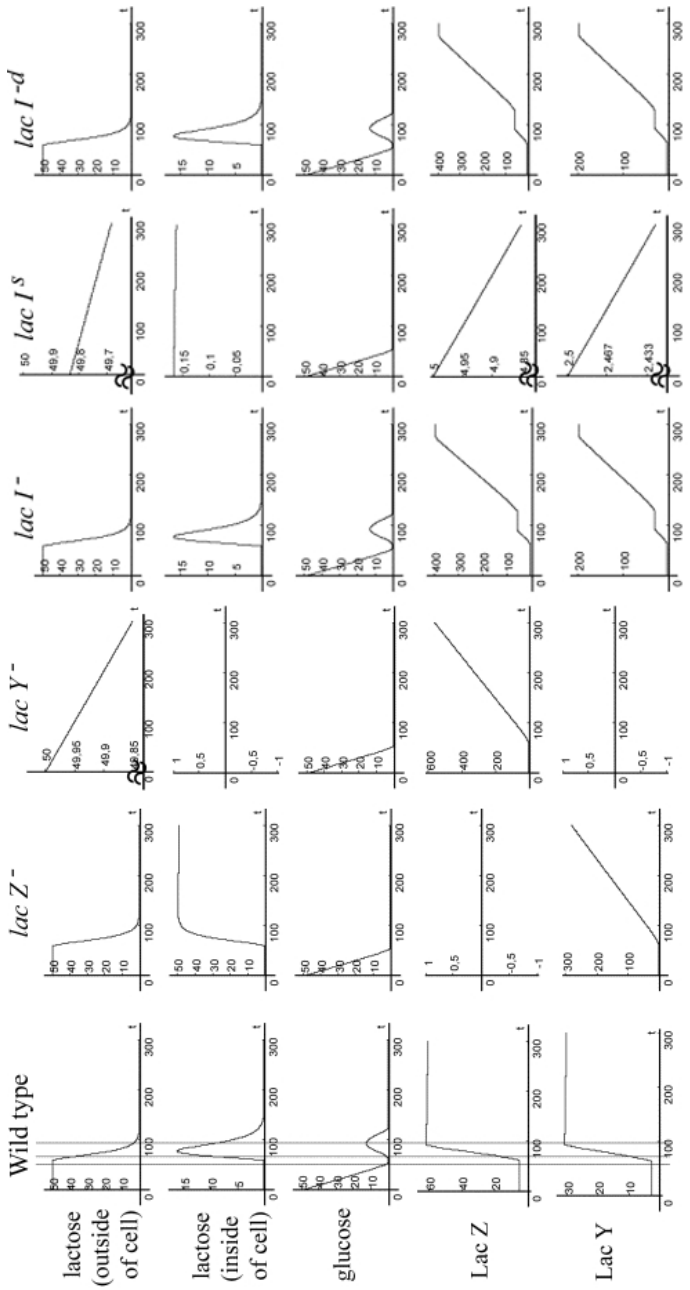


Fig. 4.10. HFPN simulation results of the *lac* operon gene regulatory mechanism and glycolytic pathway described in [53]. The *wild type* column describes the evolution of lactose outside of the cell, lactose inside of the cell, glucose, LacZ and LacY proteins in a healthy cell. The other columns plot the temporal evolutions of the same substances for mutants $lacZ^{-}$, $lacY^{-}$, $lacI^{-}$, $lacI^{s}$ and $lacI^{-d}$.

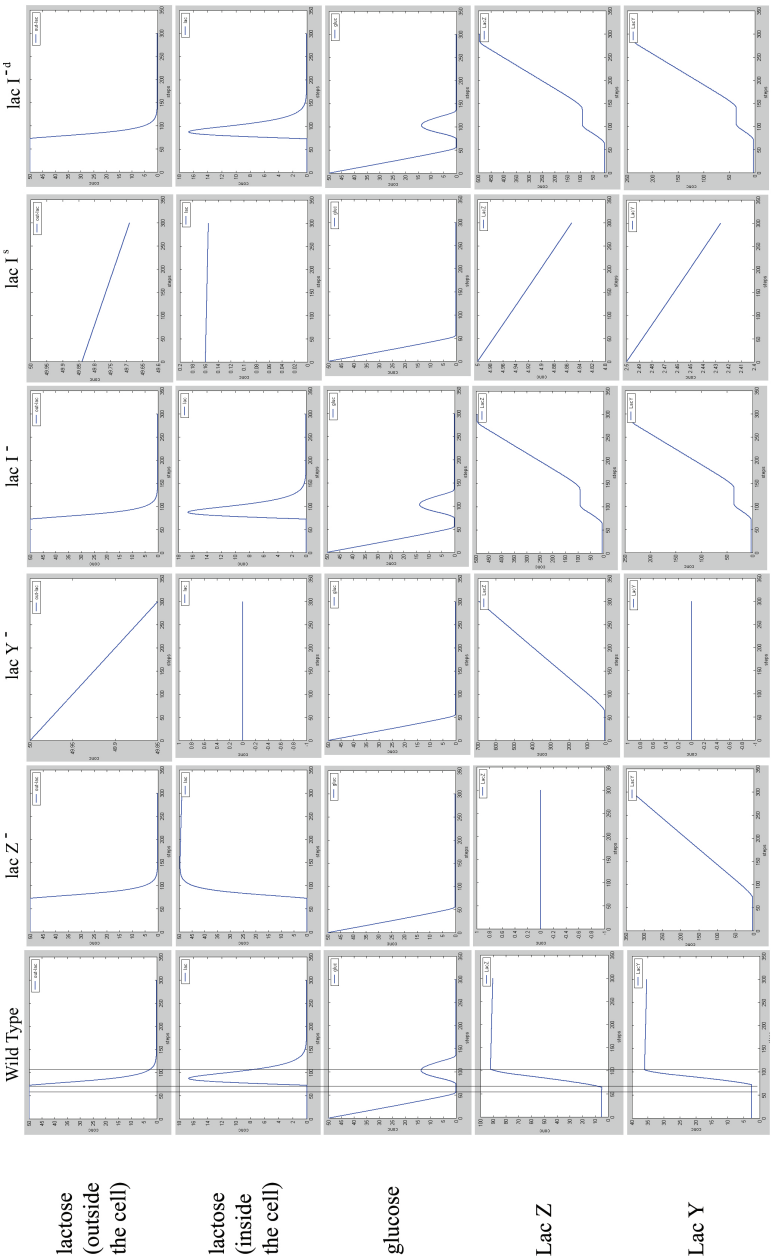


Fig. 4.11. MP systems simulation results of the *lac* operon gene regulatory mechanism and glycolytic pathway (figure from [31] with permission). The *wild type* column describes the evolution of lactose outside of the cell, lactose inside of the cell, glucose, LacZ and LacY proteins in a healthy cell. Subsequent columns describe the temporal evolutions of the same substances for mutants $lacZ^-$, $lacY^-$, $lacI^-$, $lacI^s$ and $lacI^{-d}$. An enlarged version of this picture is published at the web page [153].

parameter and flux time-series. Specifically, the log-gain theory supports the first step by deducing the time-series of flux values from the corresponding time-series of substances and parameters of an observed dynamics. After this first step a regression technique is needed to compute sound flux regulation functions from flux time-series. In the following we report the main notions of the log-gain theory, while Chapter 5 collects some original results about the synthesis of regulation functions by neural networks and other regression techniques.

Let us consider the problem of flux discovery in a very simple MPR system called *Sirius* [140,142]. This model does not have any biological counterpart but it is interesting as well because of the oscillations it generates when specific regulation functions are employed. As displayed in Figure 4.12, Sirius has three substances, A, B and C, five reactions $R_1, \ldots, R_5$. It generates the oscillatory dynamics reported at the bottom of the same figure when reaction maps of Table 4.4 are employed with inertia maps $\psi_a(q) = \psi_b(q) = \psi_c(q) = 100$ and initial state $a[0] = 100, b[0] = 100, c[0] = 0$.

Reactions	**Reaction maps**
$r_1 : a \to aa$	$f_1(q) = k_1$
$r_2 : a \to b$	$f_2(q) = k_2 \cdot c$
$r_3 : b \to \lambda$	$f_3(q) = k_3$
$r_4 : a \to c$	$f_4(q) = k_4 \cdot b$
$r_5 : c \to \lambda$	$f_5(q) = k_5$

Table 4.4. Sirius's reactions and reaction maps, where $k_1 = k_3 = k_5 = 4$, $k_2 = k_4 = 0.02$.

Sirius' differential formulation, according to the MP-ODE transformation described in Section 4.4, is given by the following differential equation system:

$$\begin{aligned} \frac{da}{dt} &= k_1 a - k_2 ca - k_4 ba \\ \frac{db}{dt} &= k_2 ac - k_3 b \\ \frac{dc}{dt} &= k_4 ab - k_5 c. \end{aligned} \tag{4.26}$$

Now, if we suppose to observe the dynamics $\delta(i)$ of the system for t steps, we achieve a set of vectors $\{(a[i], b[i], c[i]) \mid i = 0, \ldots, t\}$, where $a[i], b[i], c[i]$ are the quantities (expressed in moles) of substances a, b, c, respectively, at time $i = 0, \ldots, t$. From the stoichiometry of the system we know that, at each step, substance a is produced by the reaction r_1, while it is consumed by reactions r_2 and r_4; substance b is produced by reaction r_2 and it is consumed by reaction r_3; finally, substance c is produced by reaction r_4 and it is consumed by reaction r_5. Accordingly, considering the transition between two consecutive instants i and $i+1$ we get the following system of equations, we call $SD[i]$ (*Substance Differences at step* i):

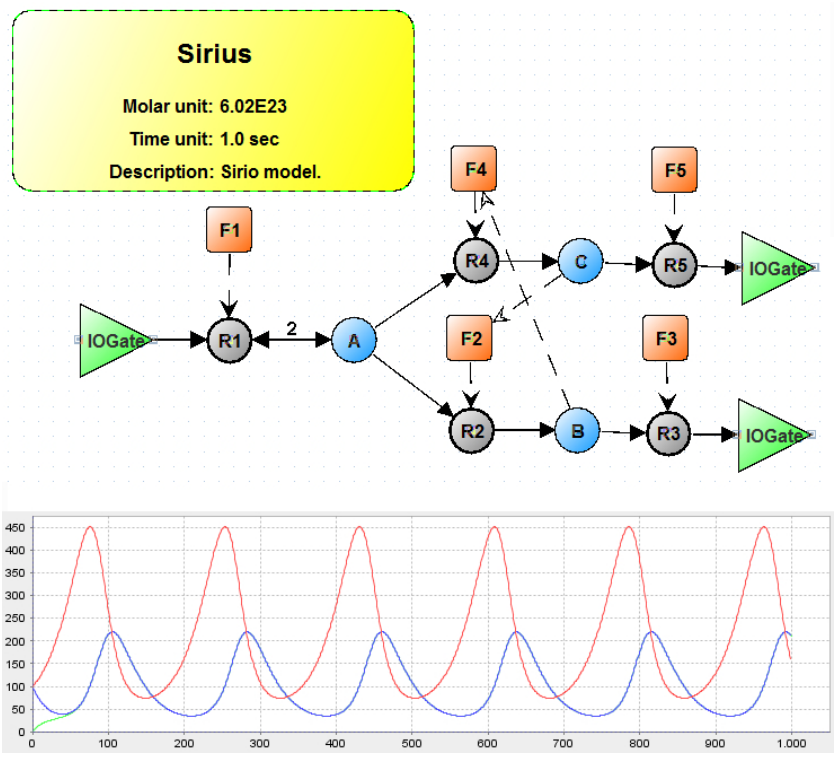


Fig. 4.12. On top: Sirius' MP graph. At the bottom: Sirius' dynamics

$$\begin{aligned} a[i+1] - a[i] &= u_1[i] - u_2[i] - u_4[i] \\ b[i+1] - b[i] &= u_2[i] - u_3[i] \\ c[i+1] - c[i] &= u_4[i] - u_5[i] \end{aligned} \tag{4.27}$$

where each $u_r[i], r = 1, \ldots, 5$, represents the flux of reaction r at step i, namely, $u_r[i] = \varphi_r(\delta(i))$. In order to solve the problem of flux discovery we should compute fluxes $u_r[i], r = 1, \ldots, 5$, for $i = 0, \ldots, t$, but the system $SD[i]$ does not permit to do it, since it has three equations (assuming them to be linearly independent) and five variables. In general, whenever the maximal rank of the system $SD[i]$ is lesser than the number of reactions, the equation system $SD[i]$ becomes indeterminate and it needs more constraints in order to be solved. According to the log-gain theory, this issue can be overtaken by adding new equations based on a generalization of the *allometric principle* [236]. It is a mathematical law, introduced by L. von Bertalanffy, which states that a specific ratio holds between the relative variations of two related biological parameters (e.g., the mass of an organism and its superficial area). This principle seems a general property of living organisms which enables them to keep basic equilibria underlying their internal organization. As reported in [236], many empirical laws on metabolism are instances of *allometry* and also the abundance of power laws in biological systems is related to this principle [140, 142]. Therefore, the application of allometric constraints deriving

from the allometric principle to the equation system $SD[i]$ seems to make sense in the perspective of identifying a biochemically meaningful solution of the system.

The *log-gain principle*, formally described below, constrains the relative variation of each flux, during the step i, to be a linear combination of the relative variation of some substances and parameters along the same step. In the following, given a dynamics δ of an MP system, we will use the simplified notation, for $i \in \mathbb{N}, r \in R$, and $w \in X \cup V$:

$$\begin{aligned} u_r[i] &= \varphi_r(\delta(i)), \\ w[i] &= (\delta(i))(w). \end{aligned}$$

Principle 1 (Log-gain [142]) *For $i \in \mathbb{N}$ and $r \in R$, let us call*

$$Lg(u_r[i]) = (u_r[i+1] - u_r[i])/u_r[i]$$

the log-gain of the flux unit u_r at the step i, and analogously,

$$Lg(w[i]) = (w[i+1] - w[i])/w[i]$$

the log-gain of the substance or parameter w at step i. There exists a subset T_r of $X \cup V$ of elements called (log-gain) tuners *of r such that $Lg(u_r[i])$ is a linear combination, in a unique way, of the tuners of r:*

$$Lg(u_r[i]) = \sum_{w \in T_r} p_{r,w} Lg(w[i]). \tag{4.28}$$

In order to understand this principle in action, let us apply it to Sirius. We firstly identify a set of tuners for each reaction, say, the reactants of the reaction plus the arguments of its flux regulation function, i.e., $T_1 = \{a\}, T_2 = \{a, c\}, T_3 = \{b\}, T_4 = \{a, b\}, T_5 = \{c\}$. Afterward, we write the Equation 4.28 for every reaction $r_1, \ldots, r_5$ in order to generate the following system of equations, we call *log-gain module* $LG[i]$:

$$\begin{aligned} Lg(u_1[i]) &= p_1 Lg(a[i]) \\ Lg(u_2[i]) &= p_2 Lg(a[i]) + p_3 Lg(c[i]) \\ Lg(u_3[i]) &= p_4 Lg(b[i]) \\ Lg(u_4[i]) &= p_5 Lg(a[i]) + p_6 Lg(b[i]) \\ Lg(u_5[i]) &= p_7 Lg(c[i]) \end{aligned} \tag{4.29}$$

Now, putting together the systems 4.27 and 4.29, at step $i+1$ and i, respectively, we obtain the *observation log-gain module* at step i, we indicate by $LG[i]+SD[i+1]$. If we consider this module at step 0 and we assume to know the vector $U[0] = (u_1[0], u_2[0], u_3[0], u_4[0], u_5[0])$ (some methods for discovering it have been hinted in [72, 140, 168]), then we achieve a system of 8 equations and 12 variables (i.e., 5 fluxes at time i=1 and 7 log-gain coefficients $p_1, \ldots, p_7$). Moreover, let us set to 1 the log-gain coefficients while adding an *offset* coefficient p_r to each equation of $LG[i]$, which accounts the error introduced by setting log-gain coefficients to 1. The equation system we get in this way has still 8 equations, but the number

of its variables is 10, since 7 log-gain coefficients have been set to 1 and 5 offset coefficients have been introduced, decreasing the total number of variables by two. The equation system obtained in this way is called *offset log-gain module*, ($OLG[i]$) and it can be determined according to the following principle.

Principle 2 (Offset log-gain [142]) *There exists a subset T_r of $X \cup V$ of elements called* (log-gain) tuners *of r and a unique value p_r, called offset log-gain, such that*

$$Lg(u_r[i]) = \sum_{w \in T_r} Lg(w[i]) + p_r. \tag{4.30}$$

The offset log-gain module has always $n+m$ equations and $2m$ unknown variables, but in the following we show that the number of variables can be decreased to $n+m$, obtaining a univocally solvable equation system [142]. In fact, by considering the stoichiometric module $SD[i+1]$, we observe that the sum of offset coefficients of reactions consuming or producing a given substance x is constrained to be equal to a fixed value. Let us consider, for instance, the case of Sirius, wherein the module $SD[i+1]$ is:

$$\begin{aligned} a[i+2] - a[i+1] &= u_1[i+1] - u_2[i+1] - u_4[i+1] \\ b[i+2] - b[i+1] &= u_2[i+1] - u_3[i+1] \\ c[i+2] - c[i+1] &= u_4[i+1] - u_5[i+1]. \end{aligned} \tag{4.31}$$

It can be rewritten, in terms of (offset) log-gain, as:

$$\begin{aligned} &a[i+1]Lg(a[i+1]) - u_1[i] + u_2[i] + u_4[i] = \\ &\qquad = u_1[i]Lg(u_1[i]) - u_2[i]Lg(u_2[i]) - u_4[i]Lg(u_4[i]) \\ &b[i+1]Lg(b[i+1]) - u_2[i] + u_3[i] = \\ &\qquad = u_2[i]Lg(u_2[i]) - u_3[i]Lg(u_3[i]) \\ &c[i+1]Lg(c[i+1]) - u_4[i] + u_5[i] = \\ &\qquad = u_4[i]Lg(u_4[i]) - u_5[i]Lg(u_5[i]) \end{aligned} \tag{4.32}$$

and distinguishing, in the log-gain offset linear combination, the non-offset part

$$Lg_{u_r}[i] = \sum_{w \in T_r} Lg(w[i]) \tag{4.33}$$

we can easily obtain the following form:

$$\begin{aligned} &a[i+1]Lg(a[i+1]) - u_1[i] + u_2[i] + u_4[i] = \\ &\qquad = u_1[i]Lg_{u_1}[i] - u_2[i]Lg_{u_2}[i] - u_4[i]Lg_{u_4}[i] + p_1 - p_2 - p_4 \\ &b[i+1]Lg(b[i+1]) - u_2[i] + u_3[i] = \\ &\qquad = u_2[i]Lg_{u_2}[i] - u_3[i]Lg_{u_3}[i] + p_2 - p_3 \\ &c[i+1]Lg(c[i+1]) - u_4[i] + u_5[i] = \\ &\qquad = u_4[i]Lg(u_4[i]) - u_5[i]Lg(u_5[i]) + p_4 - p_5 \end{aligned} \tag{4.34}$$

that is, for suitable linear operators $K_1, K_2, K_3, H_1, H_2, H_3$:

$$\begin{aligned} K_1(Lg_{u_1}[i], Lg_{u_2}[i], Lg_{u_4}[i]) &= H_1(p_1 - p_2 - p_4) \\ K_2(Lg_{u_2}[i], Lg_{u_3}[i]) &= H_2(p_2 - p_3) \\ K_3(Lg_{u_4}[i], Lg_{u_5}[i]) &= H_3(p_4 - p_5). \end{aligned} \tag{4.35}$$

From 4.35 it is clear that, if we set to 0 the offset coefficients of reactions r_2 and r_4, namely, p_2 and p_4, then the log-gain gap can be covered by the offset of reaction r_1, which is p_1, in the first equation. Analogously, in the second and third equations, respectively, offsets p_3 and p_5 cover the the log-gain gap since p_2 and p_4 have been set to 0. Therefore, the number of log-gain offsets has been reduced from 5 (the number of reactions) to 3 (the number of substances), and in the final system OLG[i]+SD[i+1] the number of equations (i.e., $n + m$) equals the number of variables. This reduction is possible since, at each step, the sum of the fluxes of all the reactions competing for a substance are constrained to algebraically equate the total variation of the substance itself. In this way, only one offset can be chosen for any set of competing reactions, that is, one offset for any substance. The set R_0 of reactions whose offset coefficients are not null, is said to have the *covering log-gain property*. This property and the procedure of *offset log-gain adjustment*, formerly applied to Sirius, have been formally generalized to any offset log-gain module in [142].

In conclusion, the log-gain theory enables to compute flux vectors $U[1], U[2], \ldots, U[t-1]$, which yield an observed dynamics $\delta(0), \delta(1), \delta(2), \ldots, \delta(t)$, given *i)* a vector $U[0]$ of initial fluxes, *ii)* a set R_0 satisfying the covering log-gain property, and *iii)* a set T_r of tuners for each reaction $r \in R$. Some preliminary methods for systematically discovering $U[0]$ and R_0 have been outlined in [140, 168] and [72, 142] respectively. They represent crucial topics of our current research, together with tuners discovery.

Once flux vectors $U[i], i = 0, \ldots, t-1$ have been computed, the last task remaining to be performed concerns the synthesis of flux regulation functions $\varphi_r(q), r = 1, \ldots, m$, which fit these flux data. The problem requires regression tools able to deduce functions representing complex biochemical reaction mechanisms from time-series. In Chapter 5 we propose some original methodologies for flux regulation functions and tuners discovery. All of them have been also implemented by software tools for MetaPlab, as described in Chapter 6.

Author's publications for this chapter

- A. Castellini, G. Franco, and V. Manca. Hybrid functional Petri nets as MP systems. *Natural Computing,* 9121, 2009. DOI: 10.1007/s11047-009-9121-4. In print.
- A. Castellini, G. Franco, and V. Manca. Toward a representation of hybrid functional Petri nets by MP systems. In Y. Suzuki, M. Hagiyra, H. Umeo, and A. Adamatzky, editors, *Natural computing*, volume 1 of *Proceedings in Information and Communications Technology*, pages 28-37. Springer Japan, 2009.
- A. Castellini, V. Manca, and L. Marchetti. MP systems and Hybrid Petri Nets. In *Studies in Computational Intelligence 129 (NICSO 2007)*, pages 53-62. Springer, 2008.
- A. Castellini and G. Franco. On modeling signal transduction networks. In *Sixth Brainstorming Week on Membrane Computing, BWMC6*, pages 67-77, Sevilla, Spain, 2008. Fenix Editora.

5

Statistical and optimization perspectives in MP modeling

Models of biological systems are becoming increasingly crucial for tackling the current challenges of biology and medicine. The aim of these tools is to provide new insight on biological processes by abstracting their main features from sets of observations. Many different approaches have been employed so far in order to develop effective models, but complex systems usually show different characteristics when viewed from different "distances", and the majority of models now available seem to be either very low level (too detailed), or very high level (too coarse grain). Better understanding [233] and synthesis [10] of biological systems probably need an intermediate level of abstraction. According to the approach of *executable biology* [66] such a kind of abstraction could be achieved by *computational models*, namely a new class of models that resemble computer programs and mimic natural phenomena by executing algorithm instructions, rather than using computer power to analyze mathematical relationships among the elements of biological systems.

As seen in Chapter 4, metabolic P systems suggest a deterministic strategy based on the generalization of chemical laws for computing the dynamics of metabolic phenomena [140, 144, 145]. Being intrinsically discrete and based on string rewriting, MP models are able to give a different viewpoint on biological processes respect to ODE models.

The difficulty of MP modeling concerns the synthesis of MP regulation functions from experimental data. Nowadays biologists have several high and low-throughput experimental techniques able to provide time series of substance quantities and chemo-physical parameters [26, 133, 233], but this huge amount of data needs ad-hoc mathematical methodologies in order to be interpreted and employed into MP models for regulation functions. The *log-gain theory* [142] supports the first step of the regulation function synthesis by deducing the time series of flux values from the corresponding time series of substances and parameters of an observed dynamics. After this first step a regression technique is needed to compute sound flux regulation functions from flux time series. The choice of such a regression method deeply depends on the knowledge one has about the form of the expected flux regulation function.

In general, if the function is known to be a linear combination of its numerical parameters then *linear regression* analysis is used [1,94], i.e., least squares, while if

the function is a nonlinear combination of its parameters then *nonlinear regression* analysis is employed [211]. These two kinds of regression are known as *parametric regression* since they estimate a set of parameters allowing the function to fit a data set. When, conversely, the functional form is unknown, it cannot be parameterised in terms of any basis function. In this case a smooth function can be estimated by using *nonparametric regression* [91]. A preselected function form sometimes might be too restricted or too low-dimensional to fit unexpected features, whereas the nonparametric smoothing approach is a flexible tool for analyzing unknown regression relationships. On the other hand, nonparametric regression usually requires more complex computations and larger number of samples than parametric regression because data must supply the function form as well as the function parameters.

In Section 5.1 we introduce the inverse-engineering problem of the synthesis of MP flux regulation functions from observed data, and we present four mathematical representations for these functions. Sections 5.2 and 5.3 are dedicated to the definition of two regression techniques employed for solving our problem, namely, linear regression and artificial neural networks. For both the methodologies, specific optimization techniques are described for the estimation of parameters which make functions fit observed data. In Section 5.4 we propose a case study in which flux regulation functions of the mitotic cycle in amphibian embryos have been synthesized by neural networks. The learning technique employed in this case study has been extended in Section 5.5 to allow also an automatic selection of the variables, i.e., substances and parameters, required by each regulation functions. The chapter ends with the presentation, in Section 5.6, of a complete pipeline coping with the process of flux regulation function synthesis, from data preparation to model validation.

5.1 Synthesis of MP flux regulation maps from data: an inverse-engineering problem

One of the fundamental targets of a mathematical model is to contribute towards the understanding of a real-world process, in order to forecast and control its future behavior. As for metabolic processes, this target is usually achieved when all the substances involved in a system are detected and the typology of their interactions, i.e., the reactions occurring in the process and their rates, are identified. Let us suppose to know all the substances and the reactions involved in a biological process. In this case the transformation rate of every reaction is needed to forecast the dynamics of the system. MPF systems define the rate of each reaction $r \in R$, which depends on the state $q \in Q$ of the system, by means of a flux regulation map $\varphi_r(q)$ (see Definition 6).

Two main steps have to be performed in order to infer flux regulation functions from experimental data, as shown in Figure 5.1: *i)* to compute the flux time series which yield the observed substance and parameter time series, *ii)* to synthesize regulation functions from substance, parameter and flux time series. In the following we analyze these two steps.

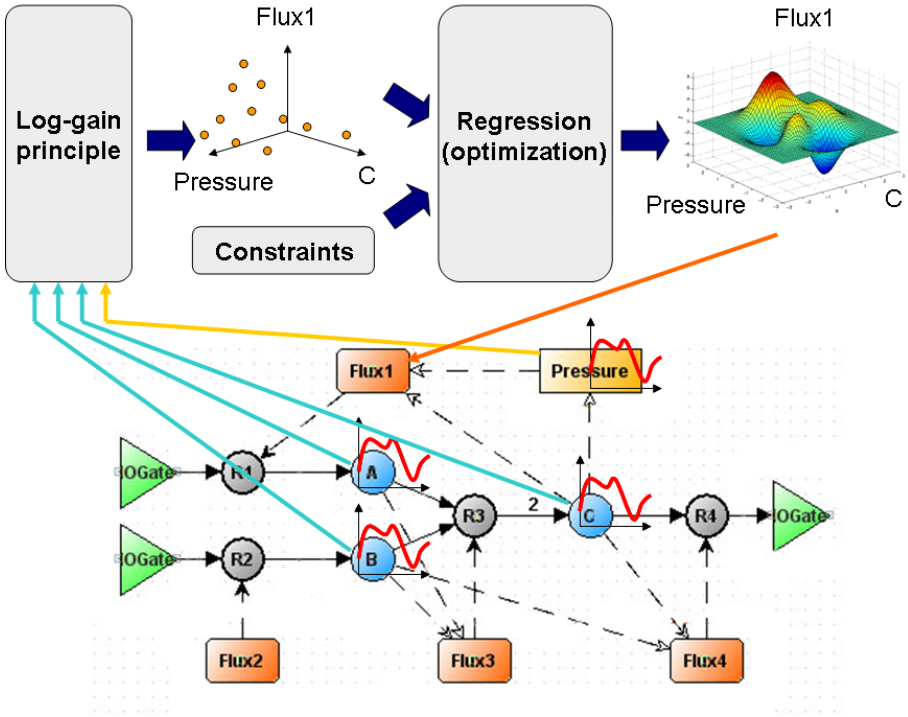


Fig. 5.1. Regulation function synthesis is performed by two main steps: *i)* the application of the log-gain principle (the box on top-left) to compute flux time series from substance and parameter time series, *ii)* the employment of regression analysis (the box on top-right) to generate regulation functions from flux time series and further constraints.

Flux time series computation. One way to compute flux time series from substance time series relies on the log-gain theory [142]. For each step i of substance time series, a system $SD[i]$ of n equations and m variables, i.e., fluxes $U[i]$, is initially generated from the stoichiometric matrix $\mathbb{A}$ and the substance values at step i. Readers can refer to [142] and to Chapter 4 for a detailed description of this equation system. If the number of linearly independent columns of $\mathbb{A}$ (reactions) is less than or equal to the number of linearly independent rows (substances) then the number of variables of $SD[i]$ is less than or equal to the number of equations. In this case flux time series at step i can be computed from substance time series at step i by just solving the equation system $SD[i]$, which has a unique solution. If, on the other hand, the number of lineraly independent columns of $\mathbb{A}$ is greater than the number of linearly independent rows, then the equation system is indeterminate and it needs more constraints in order to be solved. According to the log-gain theory, this problem can be overtaken by adding new equations based on a sophisticated generalization of the allometric principle [236], a mathematical law introduced by L. von Bertalanffy describing relationship among biological attributes. This new method integrates some natural proportions among system attributes and specific statistical analysis of the system dynamics in order to gen-

erate an univocally solvable equation system.

Regulation function synthesis. For each flux time series computed at the previous step, a function which interpolates these data points has to be synthesized by means of suitable regression techniques. The following constraints should be also considered in order to get effective and biologically-significant functions:

1. $\varphi_r(\delta(i)) \geq 0, \forall i \in \mathbb{N}, \forall r \in R$. Regulation functions should be nonnegative since they represent reaction rates.
2. $\delta(i)_{|X} \geq 0, \forall i \in \mathbb{N}$. Substance quantities should keep nonnegative during the dynamic computation.
3. *Function form.* Some knowledge about reactions can be exploited in order to constraint the form of regulation functions. For instance, if it is known that a regulation function $\varphi_r(q)$ is a linear combination of some substances and parameters, then linear regression analysis should be employed to synthesize function parameters rather than using a nonparametric regression technique which generates both form and parameters of the function from scratch.

The integration of such constraints into a classical regression technique is often a nontrivial process. Moreover, real-world biological problems often require nonlinear regression techniques, rather than linear methodologies, in order to capture all the features of complex dynamics. In these cases, often no closed-form expression exists for the best-fitting parameters, as there exists for linear regression, thus *numerical optimization algorithms* [27] are usually employed to determine the best-fitting parameters. Optimization techniques make use of *evaluation functions* that express the quality of solutions. For instance, in the case of MP regulation function synthesis, an evaluation function should estimate the quality of a set of parameters to be used by the target regulation function. In general, the optimization process aims at finding the solution which maximizes the evaluation function and these functions usually rate candidates according to their error (in our case a "distance" between known flux samples and the regulation function), but even further constraints can be rated. Notice that, complex evaluation functions may be nondifferentiable and have many local optima, thus several different techniques may be used to climb that functions up to a global optima without getting stuck into local optima.

5.1.1 Mathematical representations of complex biochemical reaction mechanisms

One of the key issues of regulation maps generation concerns the choice of function forms able to represent the biochemical mechanisms of the reactions they regulate. Indeed, only using proper regulation function forms the final MP model will be suited to interpret the phenomenon under investigation and to forecast its future behaviors. In this subsection we briefly review some mathematical techniques for representing biochemical reactions.

We firstly notice that, the choice of a correct form for a flux regulation function deeply depends on the abstraction level of the entire model. This is because chemical reactions usually involve many elementary steps, namely, steps that cannot be

decomposed to reveal reaction intermediates. But given a set of concentration time series observed with a certain timescale, only those reaction steps which take place on the observation timescale can be "viewed" in the data. Accordingly, flux regulation functions should summarize the reaction steps "visible" at the observation timescale. Depending on the timescale of the available data and, consequently, on the observation level of the model, different approaches to deriving kinetic functions are commonly adopted, such as, mass action kinetics, the Michaelis-Menten equation and allosteric kinetics, the power-law approximation and nonlinear kinetics approximations [46]. In the following we briefly review these approaches and we present some advantages and disadvantages of using each mathematical form to represent biochemical transformation processes. From the first representation, based on the mass action law, to the last one, involving nonlinear function forms, the degree of abstraction increases, therefore the first representations should be used when one needs regulation functions which consider low-level details of the biochemical transformation process, while the last one should be employed when one wants to capture the logic of a biochemical transformation process from a high abstraction level.

The law of mass action. Often used with ODE systems, the law of mass action enables to compute the (instantaneous) rate of elementary reactions supposing that it is proportional to the product of the concentrations of the species reacting in the elementary process. This law has been postulated more than a century ago to describe observations about the rate of elementary chemical reactions, thus it is empirical in its origin, but it has been shown to be consistent with results in non-equilibrium thermodynamics [115].

The rate of change $F_{i,j}$ of a specie x_i due to a reaction r_j satisfying the law of mass action has the form [46]:

$$F_{i,j} = (|\beta_{r_j}|_{x_i} - |\alpha_{r_j}|_{x_i}) k_j \prod_{l=1}^{n} x_l^{|\alpha_{r_j}|_{x_i}} \quad i = 1, \ldots, n,\ j = 1, \ldots, m, \tag{5.1}$$

where $|\alpha_{r_j}|_{x_i}$ and $|\beta_{r_j}|_{x_i}$ are the stoichiometric coefficients for species i appearing as a reactant and as a product, respectively, in the j-th reaction, k_j is a rate constant specific for reaction r_j, and x_i is the concentration of the i-th specie.

This function form is consistent for several kinds of elementary reactions but it has two main drawbacks: first, the rate constant is sensible to reaction conditions (temperature, pH, etc.), thus data must be collected by preserving these conditions to avoid higher-order complexities, and second, this law could be not satisfied by "complex" reactions, namely, reactions that summarize more than a single elementary reaction. The second issue is much related to MP systems, since they aim to model biochemical systems from a discrete-time and macroscopic perspective. MP reactions can thus represent many elementary reactions of the real system, and fluxes must take into account all the amount of matter transformed, by MP reactions, during a discrete time interval. Consequently, MP system reactions are often complex and they require more general function forms for regulating their fluxes.

Michaelis-Menten equation. One way to represent enzyme-catalyzed reactions involves the description of all the elementary steps of the enzyme-substrate association-dissociation, isomerization of intermediates and formation of products. However, such a representation often brings to highly nonlinear models with many kinetic and stoichiometric parameters, which are typically stiff, computationally hard to solve numerically and have multiple timescales [46].

A simplified kinetic function for enzymatic reactions, however, can be obtained if the overall reaction is studied under the quasi-steady-state conditions. The approximated function is called Michaelis-Menten equation and has the following form:

$$\frac{d[P]}{dt} = -\frac{d[S]}{dt} = \frac{v_{max}[S]}{K_M + [S]'} \tag{5.2}$$

where V_{max} and K_M are, respectively, the maximum rate and the so-called *Michaelis constant*, $[P]$ represents the product concentration, $[S]$ the substrate concentration of the enzymatic reaction $S + E \rightleftharpoons_{k_{-1}}^{k_1} C \stackrel{k_2}{\rightarrow} E + P$, E is an enzyme, and C the intermediate enzyme-substrate complex.

This is an example in which all the elementary steps of a reaction (i.e., the enzyme-catalyzed reaction) have been embedded in a single complex reaction, and an approximated kinetic function (i.e., Equation (5.2)), has been employed to compute its dynamics. This is an interesting approach but it can still be used only with specific reaction types and in presence of particular environmental conditions. In order to overcome these issues, alternative approaches have been developed for modeling reactions following non-ideal kinetics.

The power law approximation: S-systems. In 1969 Savageau [205–207] proposed a new approach which, in contrast to the previous ones, assumes that the rate of change of a state variable is equal to the difference of two products of variables raised to non-integer powers [46], that is:

$$\frac{dx_i}{dt} = \alpha_i \prod_{j=1}^{n} x_j^{g_{i,j}} - \beta_i \prod_{j=1}^{n} x_j^{h_{i,j}} \quad i = 1, \ldots, n, \tag{5.3}$$

where $\alpha_i \in \mathbb{R}^+$ is a rate constant pertaining to all reactions which produce substance x_i, $\beta_i \in \mathbb{R}^+$ is a rate constant pertaining to all reactions which consume substance x_i, and powers $g_{i,j} \in \mathbb{R}$ and $h_{i,j} \in \mathbb{R}$ are kinetic orders which account the influence of substance x_j to the transformation of substance x_i. These models, called *synergistic-systems* (S-systems) and already presented in Chapter 2, are based less on physical principles and more on mathematical concepts, and they are achieved by linearizing enzymes kinetic rate expressions in terms of concentrations or in terms of reaction parameters. The idea here is to write the network reaction rate $F_i = \sum_{j=1}^{m} F_{i,j}$ for the i-th chemical specie (see Equation (5.1)) as a polynomial or a rational function of concentrations $x_j,\ j = 1, \ldots, n,$, and then to take the logarithmic transform and truncate a Taylor series expansion about an arbitrary point at linear order. In this way we find the form of Equations (5.3) [46, 205].

It is not always possible to pass from a general mass action or Michaelis-Menten rate law to an equivalent S-system form since production and consumption terms are not in general separable under this transformation. Moreover, we observe that,

in contrast to the previous approaches, Equations (5.3) do not show the contribution of each reaction in the computation of $\frac{dx_i}{dt}$, thus hiding some aspects of the system stoichiometry. S-systems can approximate many properties of reaction kinetics but they fail to describe some important biochemical effects, such as saturation and sigmoidicity [93]. For all the considerations outlined above, S-systems seem to be appropriate whenever we need mathematical expressions able to approximate complex biochemical data without requiring a detailed understanding of kinetics mechanisms [46].

Nonlinear forms. The most general way to model kinetic functions is represented by nonlinear forms. Notice that, the term "nonlinear" is here referred to the use of any kind of functions, such as, powers, polynomials, exponentials, logarithms, and their linear or nonlinear combinations. However, in literature the term "linear regression" is usually referred to the discovery of function coefficients occurring linearly (i.e., not as exponential, powers, etc.) in a function. On the other hand "nonlinear regression" is referred to techniques able to compute function coefficients occurring also as exponentials, powers and other nonlinear functions, or as nonlinear combinations of these functions. The use of nonlinear forms is significant in those cases wherein the real internal mechanisms of a reaction are unknown, because of a lack of complete information or because the reaction actually includes many elementary reactions. The use of global nonlinear forms provides a method for establishing a mathematical description of the behavior of a biochemical reaction, which is valid over a specific range of observed data. Typically for this approach, the type of nonlinearity to be used is chosen empirically. Biochemical reactions are constrained by the fundamental laws of chemistry and physics, and so this is a good starting point for mathematical representation of reaction kinetics. Employing these constraints to select function forms increases the likelihood of obtaining an accurate mathematical representation of the underlying reaction mechanism, by restricting the model class (the number of type of kinetic functions) that may be used to describe the biochemical reaction. Once a good nonlinear form has been selected, parameters are estimated in order to make the function fit observed data. The amount of experimental data available, their timescale and the complexity of the biochemical reaction under investigation are deciding factors when selecting an appropriate nonlinear form [46]. However, the choice of a model is somewhat arbitrary since different approaches better capture different features of the observed data, and they represent the underlying hypothesis of a theory more or less faithfully.

Multivariate polynomial models [1,46] are a restrictive class of functions, which is frequently used in biochemical reaction analysis since, for instance, it can capture the structure of equations deriving from mass action kinetics. According to this approach a set of basis functions (monomials) involving substance concentrations, parameter values, their powers and cross products, are summed together to obtain a polynomial in which parameters enter linearly. This means that the model may be fitted to data using least squares for computing a set of parameters minimizing the sum of squared residuals. The main advantages of using this method are that it is known to converge (via the Weierstrass approximation theorem), it is computationally light, its statistical properties are well established and the achieved

functions can be subsequently interpreted in terms of reaction mechanisms. However, when the nonlinearity of the kinetics is significant, multivariate polynomials tend to assume high degrees and to employ many variables, thus limiting their usefulness [192].

Artificial neural networks (ANNs) are an extremely flexible tool which may be convenient in this case. Being nonlinear in their parameters, ANN models are capable of approximating very complicated functions [24], which is useful in order to approximate complex biochemical mechanisms or if the data set is of very poor quality, so that only patterns and correlations in the data are sought. ANNs are modular and parsimonious in their structure and they provide an accurate approximation of recorded time series, but a significant disadvantage (and a common criticism) of their use is that the obtained models are often difficult to interpret from a biochemical point of view. Moreover, to make an ANN fit a data set, a nonlinear minimization step must be accomplished, which is generally fraught with many local minima in parameter space. For this reason, large data sets are usually needed and, however, one cannot be certain that the function found during the learning step is the best one for fitting observed data. As a consequence a large amount of computational effort must be dedicated to this step to have good results. Learning techniques include the iterative error backpropagation algorithm, or any other general nonlinear optimization approach, such as the conjugate gradients, simulated annealing [27, 176], genetic algorithms [97, 250] and other techniques described in the following.

Nonlinear forms have been selected as the best way to represent MP flux regulation functions, since these functions must compute flux values by considering the overall effects of complex regulation mechanisms occurring over discrete time intervals. Moreover, such functions can sometimes include in their structure the logic of some unknown parts of a the system under investigation. Other times they can summarize complex regulation mechanisms involving many elementary reactions, but in any case, their form must be sufficiently general to capture the logic of a biochemical transformation process from a high abstraction level. We have employed two kinds of nonlinear models, namely, polynomial models, introduced in Section 5.2, and artificial neural networks, described in Section 5.3.

5.2 Regulation function synthesis by linear regression

Linear regression and multiple regression are both statistical methods for generating mathematical functions describing sets of data. Our aim is to infer, for each reaction r_j of an MP system, a function $\varphi_{r_j}(q)$ which regulates the reaction flux depending on the system state q. Since we want to make these functions fit a dataset of observations $O = \{(\delta^{oss}(i), u_j^{oss}[i]) \mid i = 0, \ldots, t\}$, where $\delta^{oss}(i)$ is the system state observed at time i and $u_j^{oss}[i]$ is the flux of reaction r_j observed at time i (these values are usually computed by the log-gain theory), linear regression techniques represent proper tools for inferring our functions.

Our aim is to generate functions able to explain as much as possible about the biochemical (reaction) process underlying observed data. However, due to the uncertainty inherent in all real world situations, our functions will probably not

explain everything, and some errors will certainly remain. They are due to unknown outside factors that affect the process generating our data. The functions we are looking for should capture the systematic behavior of the data, leaving out the factors that are nonsystematic and cannot be foreseen, namely, the errors. The final target is, thus, to break down the data into a nonrandom, systematic component, described by a function, and a purely random component representing the error [1] (as shown in Figure 5.2). Linear regression models assume that the random errors, denoted by ϵ, are *additive*, *normally distributed* with mean of zero and constant variance σ^2, and they are *independent* of one another (Figure 5.3).

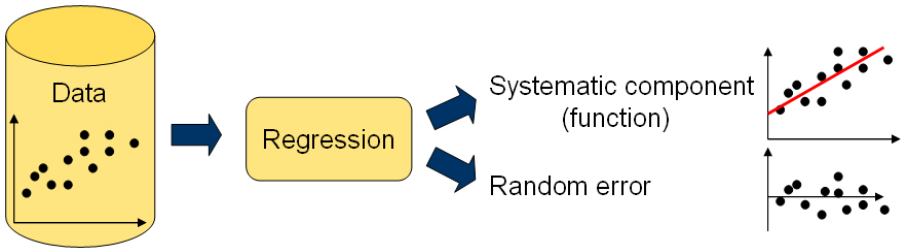


Fig. 5.2. Breaking down the data into a systematic component and a random component by linear regression.

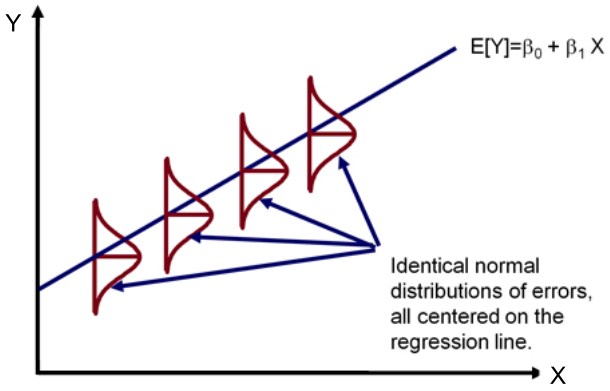


Fig. 5.3. Linear regression assumptions [1].

In order to build a statistical model describing a certain dataset, some steps have to be performed, as displayed in Figure 5.4. First, a specific function form is defined. Then the parameters of the function are estimated from the dataset by means of optimization techniques. Subsequently, *residuals*, i.e., the errors resulting from the fit of the model to the data, are examined in order to quantify the information in the data not explained by the model. If the residuals are found to contain some nonrandom, systematic component, then the function form defined at the first step has to be adjusted by incorporating this systematic component.

Otherwise, the function achieved is employed for its intended purpose, which is usually: variable prediction, variable control, or explanation of the relationships among variables. We are interested in predicting flux values and in explaining the biochemical mechanisms underlying flux regulation.

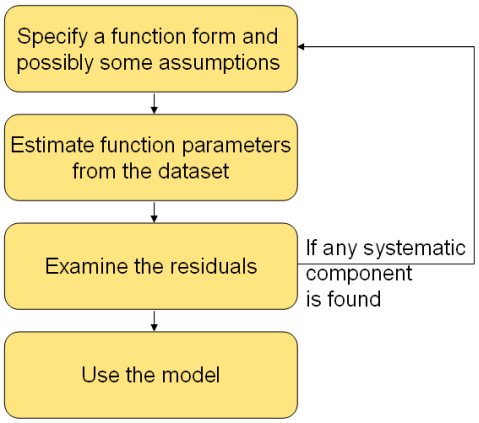


Fig. 5.4. Some basic steps for building a statistical model.

5.2.1 Simple linear regression

One of the simplest function forms which may be used for describing the systematic component of a dataset is the straight line. The relationship between an independent variable X and a dependent variable Y is represented by the equation:

$$Y = \beta_0 + \beta_1 X + \epsilon, \tag{5.4}$$

where the two parameters to be estimated are β_0, i.e. the Y *intercept*, and β_1, i.e. the *slope* of the line, as shown in Figure 5.5.

In this model the nonrandom, systematic component is represented by the line, while the purely random component is represented by the error term ϵ. If we adopt such model to represent the relationships between an independent variable X and a dependent variable Y, we obtain the equation

$$\hat{Y} = b_0 + b_1 X, \tag{5.5}$$

where $\hat{Y}$ is the approximated value of Y (which does not consider the random error ϵ), b_0 and b_1 are the estimators of the regression parameters β_0 and β_1, computed from a dataset of experimental observations $O = \{(x^{oss}[i], y^{oss}[i]) \mid i = 0, \ldots, t\}$. The *method of least squares* is used to compute the *best linear unbiased estimators* of β_0 and β_1, which are the estimators that minimize the sum of square error (SSE) between the straight line and the data points:

$$SSE = \sum_{i=0}^{t} (y[i] - \hat{y}[i])^2 \tag{5.6}$$

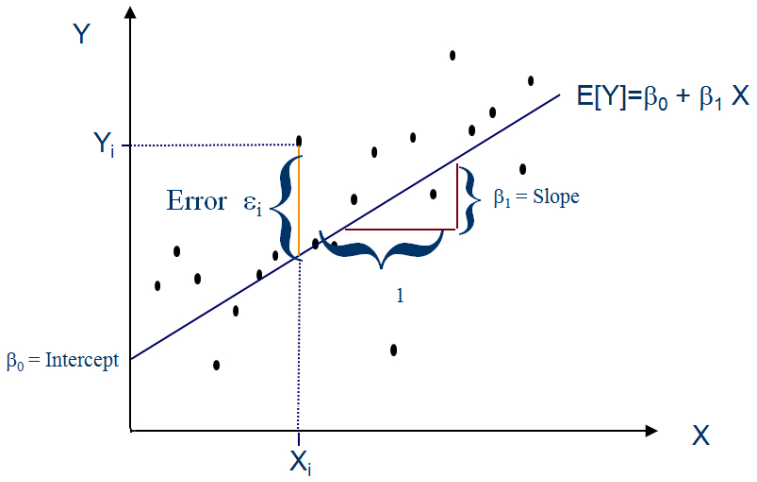


Fig. 5.5. The main elements of a simple linear regression model [1].

Estimators b_0 and b_1 are computed by the following equations:

$$b_1 = \frac{\sum_{i=0}^{t}(x[i] - \bar{x})(y[i] - \bar{y})}{\sum_{i=0}^{t}(x[i] - \bar{x})^2} \tag{5.7}$$

$$b_0 = \bar{y} - b_1\bar{x} \tag{5.8}$$

where $\bar{x}$ and $\bar{y}$ are, respectively, the mean value of x and y in the dataset O.

If we apply a simple linear regression model to represent the relationships between a flux time series $u_j[i], i = 1, \ldots, t$, and a substance time series $x_l[i], i = 1, \ldots, t$, we obtain a flux regulation function of the form

$$\varphi_j(x_l) = b_0 + b_1 x_l. \tag{5.9}$$

This form often results too weak to represent multiple regulations characterizing real biochemical networks, thus in the following we present an extension of linear regression which enables to consider functions of more than just one variable instead of straight line functions.

5.2.2 Multiple linear regression

In [46], Crampin et al. show that multiple regression models often provide proper mathematical descriptions of biochemical kinetics since, according to the *Weierstrass approximation theorem*, their flexible polynomial form enables them to approximate, in principle, any function. A p-variable multiple regression model is an extension of the simple linear regression model, having the following form:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon, \tag{5.10}$$

where the regression surface Y (see Figure 5.6) is in general an hyperplane with intercept β_0 and slope parameters $\beta_i,\ i = 0, \ldots, p$.

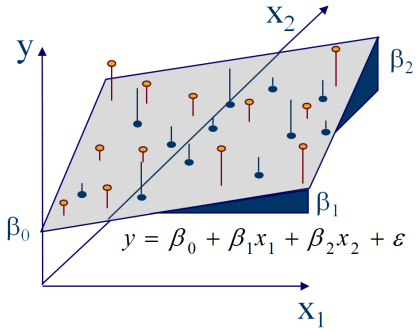


Fig. 5.6. The regression surface of a multiple regression model having two variables [1].

As for the simple linear regression model, we assume that the error terms ϵ are normally distributed with mean zero and constant variance σ^2, and they are independent for each observation. In this case the method of least squares can be applied to estimate parameters $\beta_i,\ i = 1, \ldots, p$. Let us assume, for instance, a model having two variables:

$$\hat{Y} = b_0 + b_1 X_1 + b_2 X_2. \tag{5.11}$$

Estimators b_0, b_1 and b_2 which minimize the square error between observation data and regression surface can be computed by solving the following system of normal equations [1]:

$$\sum_{i=0}^{t} y[i] = (t+1)\ b_0 + b_1 \sum_{i=0}^{t} x_1[i] + b_2 \sum_{i=0}^{t} x_2[i] \tag{5.12}$$

$$\sum_{i=0}^{t} x_1[i]y[i] = b_0 \sum_{i=0}^{t} x_1[i] + b_1 \sum_{i=0}^{t} x_1[i]^2 + b_2 \sum_{i=0}^{t} x_1[i]x_2[i] \tag{5.13}$$

$$\sum_{i=0}^{t} x_2[i]y[i] = b_0 \sum_{i=0}^{t} x_2[i] + b_1 \sum_{i=0}^{t} x_1[i]x_2[i] + b_2 \sum_{i=0}^{t} x_2[i]^2 \tag{5.14}$$

where t is the number of samples in the observation dataset. This system can be generalized to any number of variables, but we observe that its solvability is ensured only if the variables are independent one another (so that the determinant of the system matrix is not null). In practice, this condition is often difficult to ensure since explanatory variables are often interrelated in some way.

A key problem of multiple regression models concerns the selection of a proper set of explanatory variables. Although it seems logical to incorporate as many variables as possible to obtain the maximum prediction power, this approach has many limitations. Section 5.5 focuses on this problem and presents some techniques of variable selection that have been employed for the synthesis of flux regulation models.

For what said so far, we assume that an MP flux regulation function φ_r can be expressed as a weighted sum of p linearly independent basis functions ψ_j, with $p \leq l$:

$$\varphi_r(q) = \sum_{j=1}^{p} b_j \psi_j(q) \tag{5.15}$$

where each basis function is a monomial including a component of the system state (a substance or a parameter), their powers or cross products between state components. From a biological point of view, each weight b_j represents the influence of the basis ψ_j on the flux function φ_r, with a positive or negative sign indicating activation or repression.

This approach has been applied for the first time to MP systems in [148, 149], where the flux regulation functions of a Non Photochemical Quenching model have been generated by means of multiple regression. The polynomials achieved in this way have quite simple forms which can be also analyzed from a biochemical point of view in order to understand the contribution of each substance in flux regulation. Notice that, in [148, 149] polynomial's variables, i.e. substance and parameters, have been manually chosen according with the knowledge of the process owned by modelers and acquired from the literature. However, the process of variable selection can be automate by using variable selection techniques presented in Section 5.5. In Section 5.6 we present some results achieved by employing these techniques on the NPQ and the mitotic cycle models.

5.3 Regulation function synthesis by optimized neural networks

The problem of MP regulation function synthesis is here tackled in the very general case in which the form of these functions is very complex and involves several nonlinearities. In this case linear regression models tend to assume high degrees and to employ many variables, therefore their usefulness and applicability are limited [192]. On the other hand, artificial neural networks (ANNs) [24] turn out to be a convenient approach, because their modular structure enable them to provide accurate approximations of very general functions just nonlinearly combining simple seed functions.

The assumptions are the same as for linear regression, that is, given a set of reactions $r_1, \ldots, r_m$ of an MP model, and given a time series of $t+1$ observations for each substance, parameter and flux, we aim to synthesize a flux regulation function $\varphi_j(q)$, for each reaction r_j, which fits the flux values observed for that reaction r_j. In the following we briefly introduce the main elements of ANNs, the algorithms employed to train the networks with data, and finally we suggest two possible ways to "connect" ANNs to MP systems [33].

5.3.1 The choice between linear and neural models

A preliminary evaluation of the regression complexity is very important for selecting the right regression model. We recommend to start with a linear model, i.e., a polynomial, since if it satisfies the performance requirements then it should be adopted. The reason is that linear models are computationally easier than ANNs and they have unique solutions which can be easily computed by the least square

method. Moreover, linear models often have a simple (readable) form and their statistical properties (i.e., statistical tests, confidence intervals, etc.) are founded in a solid mathematical theory.

On the other hand, when the non-linearity of the regression is significant and complex polynomials with many high-degree monomials are needed to fit the data, neural networks become useful. With this approach, the higher modeling power has, however, the cost of a higher model complexity and a computationally more expensive procedure for computing the non-unique set of parameters that optimize the model. Tuning ANN parameters involves, indeed, iterative optimization techniques which require more time than least squares to find solutions. Furthermore, these solutions could be local minima instead of a global minima, depending on the starting point of the search.

Rivals and Personnaz [192] propose a strategy for choosing between polynomial and neural models. In Figure 5.7 we report the flow chart of this strategy. When the number of model input variables is large, say greater than 30, it is almost impossible to cope with monomials of degrees larger than two, since the number of possible monomials would be too high. In this case we try to generate a polynomial of degree less or equal than two. Fortunately, cross-products of two variables are often able to model interactions between systems variables, and squared terms may represent nonlinearities. However, sometimes it happens that polynomials of degree two are not enough powerful and they achieve bad performance. In that case neural networks may be employed to find a better model. On the right side of Figure 5.7, is represented the case in which the number of potentially significant input variables is small, say less than 30. We therefore generate a multiple linear model of (theoretically) any degree and then we check the performance and the complexity of the model, namely, the number of monomials employed and their degrees. If the performance is good and the complexity is acceptable in respect to the purpose the model is built (i.e., the model is readable and allows some understanding about the process being investigated) then the linear model is kept, otherwise a neural model is generated.

5.3.2 Artificial neural networks

ANNs are a paradigm of information processing inspired by the networks of interconnected neurons constituting the biological nervous systems and, in particular, by the brain. The key elements of this mathematical model are *i)* a set of processing units, called *neurons*, computing *activation functions*, *ii)* a set of weighted interconnections, called *synapses*, conveying information among neurons. ANNs are usually depicted by graphs where nodes represent neurons and edges symbolize synapses, as displayed in Figure 5.8. Every neuron u_j (be careful not to confuse neuron notation, i.e., u_j, with flux values notation, i.e., $u_j[i]$) computes its output y_j by means of the equation $y_j = f(\sum_i w_{ji} y_i)$, where f is the activation function of neuron u_j, y_i is the output value of neuron u_i and w_{ji} is a real number related to the synapse which connects u_i to u_j. Activation functions are often nonlinear functions, such as the *logistic sigmoid*, $f(x) = \frac{1}{1+e^{-x}}$, or *tanh*, $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, but also other kind of function can be considered. *Feed-forward* neural networks have no feedback loops, thus, neurons are usually arranged in layers, where the *input-layer*

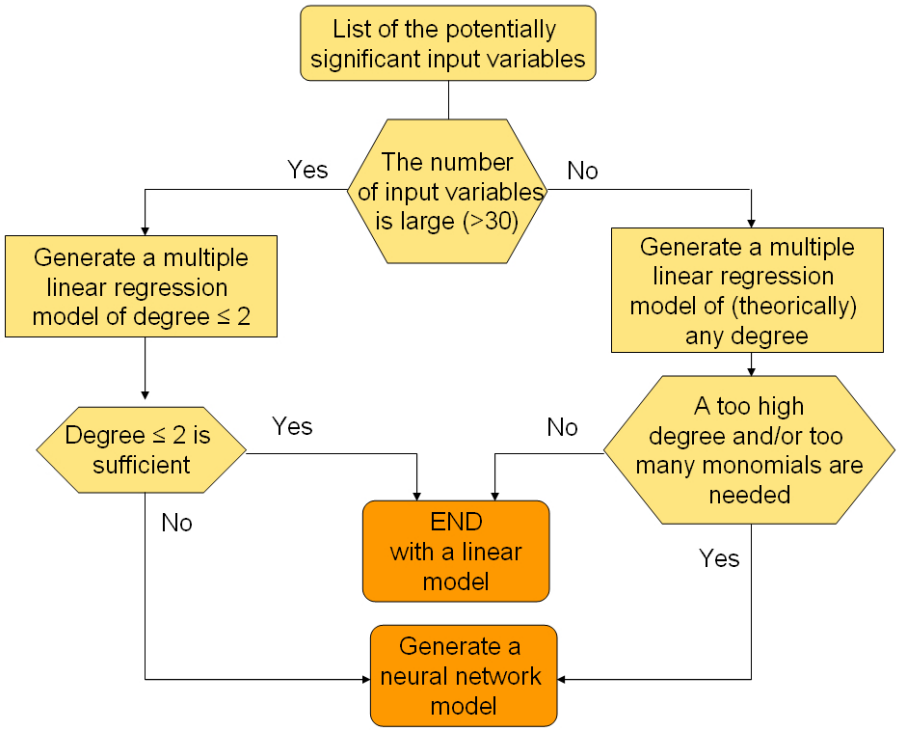


Fig. 5.7. A strategy for choosing between linear and neural models [192].

includes neurons receiving input from the environment, the *output-layer* contains neurons returning output to the environment, and *hidden layers* include internal neurons (see Figure 5.8).

The power of neural networks lies in their ability to represent both linear and nonlinear relationships between a set of input variables and a set of output variables, and in their ability to directly learn these relationships from the data being modeled. This skill is achieved by representing functions of many variables in terms of composition of nonlinear functions having a single variable. Many studies focused on the representational capabilities of an ANN depending on the number of neurons, the number of layers and the type of activation functions [24]. In particular, it has been proved [76] that networks having just one hidden layer of sigmoid neurons are able to approximate any continuous functional mapping if no limit is given on the number of hidden neurons. Accordingly, a regression technique based on this model can, in principle, be employed even when the form of the required function is completely unknown. Arbitrary network topologies can be considered (not necessary having a simple layered structure), the only restriction to be satisfied is that the topology must be *feed-forward* in order to ensure the translation of the network into an explicit function.

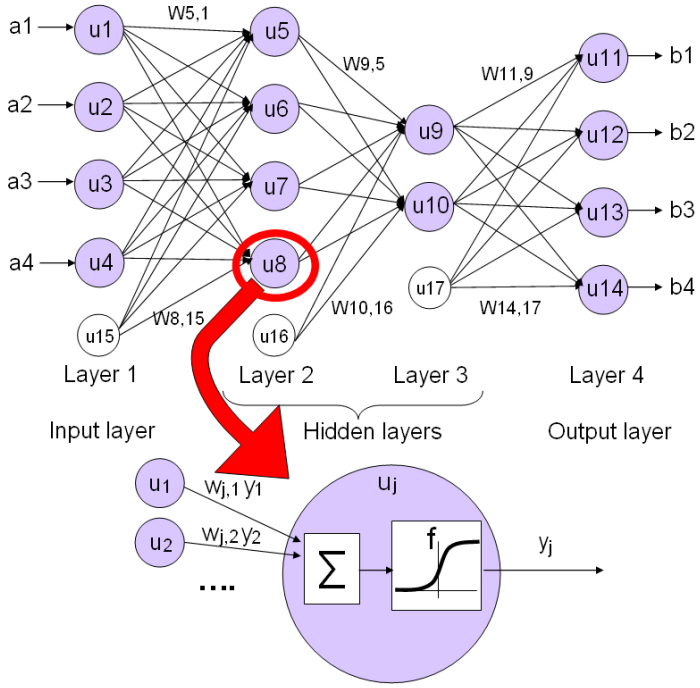


Fig. 5.8. On top: feed-forward neural network having four layers: an input layers with four neurons, an output layer with four neurons and two hidden layers with, respectively, four and two neurons. White circles represent bias neurons. At the bottom: the computation performed by a single neuron (figure from [36] with permission).

5.3.3 Traditional and evolutionary optimization algorithms for training ANNs

ANNs store knowledge within inter-neuron connection strengths, namely, synaptic weights. The process of tuning these parameters is called *training* and it is performed by *learning algorithms*, namely, optimization techniques that search for a set of weights able to give to the network a behavior defined by a set of examples, called *training set* (see Figure 5.9). The central goal in network training is not to memorize the training data (*memorization*), but rather to model the underlying generator of data (*generalization*) in order to get the best possible predictions for the new values presented to the network after the training. This is particularly important in systems and synthetic biology modeling as one is interested in coming up with robust models that could predict, in silico, a large number of unseen behaviors while accurately matching observed experimental behaviors. Generalization can be achieved by choosing the right number of hidden neurons, and specially by splitting the training set T into two subsets T_1 and T_2, and using the first one as a normal training set and the second as a *validation set*, namely, a set of patterns used to validate the network trained by T_1. If the error is computed on the validation set, rather than on the training set, the network tends to better fit new inputs [24].

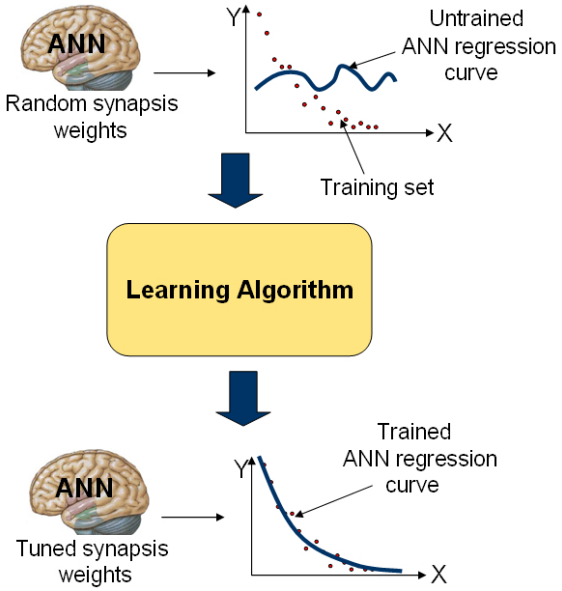


Fig. 5.9. Learning algorithms tune ANN weights in order to make the network fit a training set.

Learning algorithms can be roughly split in two main classes, *i) gradient-based* techniques employ information about the gradient of the evaluation function in order to find its global optimum; *ii) non-gradient-based* techniques do not make use of gradient information. Usually gradient-based learning algorithms have good time performance but they can be applied only if the evaluation function is differentiable and not multimodal. Nevertheless, when applied without any metaheuristic, gradient-based techniques may get stuck into local optima. Non-gradient-based methods can often overcome this problem by performing the searching process in accordance with alternative rules. For evolutionary algorithms [97] and memetic algorithm [125] these rules are inspired by natural phenomena, such as natural evolution and swarm intelligence.

In the following we present four techniques for learning neural networks, namely, backpropagation, genetic algorithms, particle swarm optimization, and a memetic algorithm. Given a training set $T = \{(\bar{a}, \bar{t}) \mid \bar{a}$ network input, $\bar{t}$ target output$\}$, our aim is to estimate the synapse weights which make a network fit these data. Notice that, notation $\bar{x}$ is here employed to represent vectors of neuron inputs, which has no relations with notation x_j used previously to represent biochemical substances.

Backpropagation

Backpropagation is a learning algorithm using gradient descent to find, in the hypothesis space of all possible weight vectors, the weights that best fit a set of training samples T. Let us consider the task of training a simple *linear neuron*, whose output $y(\bar{x})$ is given by:

$$y(\bar{x}) = \bar{w} \cdot \bar{x}, \tag{5.16}$$

where $\bar{w}$ is the vector of input synapse weights and $\bar{x}$ is the vector of neuron input values. In order to derive a weight learning rule for this simple unit, we start by specifying the error function, relative to the training samples, with respect to the weight vector $\bar{w}$, that is [160]:

$$E(\bar{w}) = \frac{1}{2} \sum_{d \in T} (t_d - y_d)^2 \tag{5.17}$$

where T is a training set for the linear neuron, t_d is the target output for training sample $d \in T$, and y_d is the output of the linear neuron for the training sample d. Error $E(\bar{w})$ is simply half the squared difference between the target output t_d and the linear unit output y_d, summed over all training samples. Notice that, given a fixed training set T, the error $E(\bar{w})$ depends on the weight vector $\bar{w}$, since values y_d are computed according to Equation 5.16.

To understand the gradient descent procedure, we show in Figure 5.10 the entire hypothesis space of possible weight vectors (for a simple neuron having only two inputs) and their associated error values. For each couple of weights (w_0, w_1) in the hypothesis space, a point in the error surface is assigned. Since the neuron under investigation is linear, the error surface must always be parabolic with a single global minimum. The specific parabola form depends on the particular set of training samples.

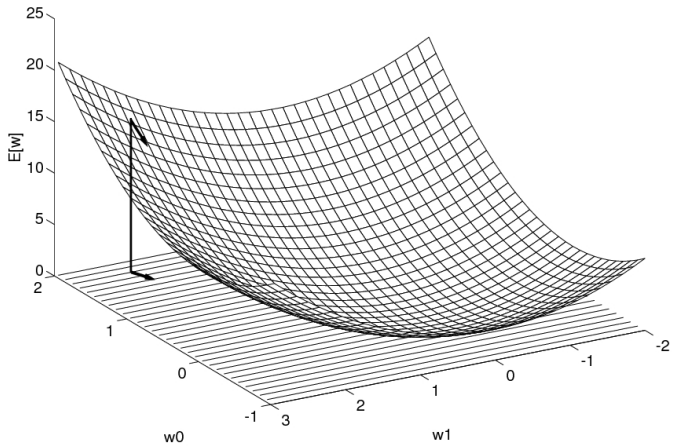


Fig. 5.10. Error function $E(\bar{w})$ for a linear neuron with two weights [160].

The aim of gradient descent search is to find the weight vector which minimizes the error function $E(\bar{w})$ by starting from an arbitrary point in the hypothesis space. To accomplish this task, it iteratively moves the weight vector in the direction that produces the steepest descent along the error surface $E(\bar{w})$, until the global minimum is reached. This direction corresponds to the negative gradient of $E(\bar{w})$ with respect to $\bar{w}$. In a general n-dimensional hypothesis space, this is:

$$\nabla E(\bar{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]. \tag{5.18}$$

The training rule for gradient descent is therefore:

$$\bar{w} = \bar{w} - \eta \nabla E(\bar{w}) \tag{5.19}$$

where η, called *learning rate*, determines the step size in direction $\nabla E(\bar{w})$. The problem of gradient descent search thus translates into the computation of gradient components $\frac{\partial E}{\partial w_i}$. For a linear unit, by differentiating E from Equation 5.17 we find [160]:

$$\frac{\partial E}{\partial w_i} = \sum_{d \in T} (t_d - y_d) x_{id} \tag{5.20}$$

where x_{id} is the i-th input component of training sample d. However, a big disadvantage of using linear units is that networks with multiple layers of cascaded linear units can produce only linear functions. As a consequence, for real applications *sigmoid units* are usually employed, which are based on a smoothed, differentiable threshold function. The computation of the error gradient for multilayer networks of sigmoid units is harder than that for linear units, thus specific methodologies have been implemented to speed up this task.

Backpropagation is a powerful and computationally efficient method for computing the derivatives of the error function, with respect to weights, for multilayer networks with a fixed set of units and interconnections. The aim of employing this technique is to minimize, by gradient descent, the overall error of a neural network, that is, the sum of the errors over all the network output neurons:

$$E(\bar{w}) = \frac{1}{2} \sum_{d \in T} \sum_{k \in outputs} (t_{kd} - y_{kd})^2 \tag{5.21}$$

where *outputs* is the set of output neurons, and t_{kd} and y_{kd} are, respectively, the target and the output values of the k-th output for the d-th sample of T. The surface of error $E(\bar{w})$ is highly dimensional, since the number of weights is usually high in multilayer networks. Moreover, this surface can have multiple local minima, in contrast to the case of Figure 5.10. For these reasons, gradient descent techniques cannot ensure to converge to one of the global minima of the error surface, but they only ensure to find a local minimum. Despite this issue, the application of backpropagation to real-world problems has often yielded good results [160].

The *backpropagation algorithm* for feedforward networks having two layers of sigmoid neurons (with each neuron connected to all the neurons of the next layer) can be summarized as follows [160]:

Initialization: n=0;
While the termination condition is not met, *Do*

- *For each* $(\bar{a}, \bar{t}) \in T$
  1. Input forward-propagation:
     - Feed the input neurons with $\bar{a}$ and apply the activation functions of every neuron until the output y_k of each output neuron u_k is computed.

2. Error back-propagation:
   - *For each* output neuron u_k, compute the error term e_k

$$e_k \leftarrow y_k(1 - y_k)(t_k - y_k) \tag{5.22}$$

   - *For each* hidden neuron u_h compute the error term e_h

$$e_h \leftarrow y_h(1 - y_h) \sum_{k \in outputs} w_{kh} e_k \tag{5.23}$$

     where *outputs* is the set of output neurons.
3. Weights update:
   - Update each weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}(n) \tag{5.24}$$

     where

$$\Delta w_{ji}(n) = \eta e_j y_i + \sigma \Delta w_{ji}(n-1), \text{ for } n \geq 0 \tag{5.25}$$

$$\Delta w_{ji}(-1) = 0 \tag{5.26}$$

     and y_i is the output of the neuron u_i.
4. Iteration update: $n = n + 1$.

The algorithm input is a network having random weights with small values. The main loop is the *for* cycle, which iterates over each training sample. In particular, for each sample $(\bar{a}, \bar{t}) \in T$, it firstly computes the output $b(\bar{a})$ of the network (forward propagation), then it calculates the error terms e for each output and hidden neuron according to gradient descent, and it finally updates weight values by means of Equation (5.24) (back propagation). The weight variation, computed by Equation (5.25), consists of two terms: a gradient descent term, i.e., $\eta e_j y_i$, and a momentum term, i.e., $\sigma \Delta w_{ji}(n-1)$. The first determines the step in the gradient-descent direction, the second, which is optional, adds an inertial contribution to the search movement, and it is useful to avoid local optima. The *for* cycle is iterated many times, until the error $E(\bar{w})$ falls under a specific threshold or a predefined number of iterations have been performed.

Notice that, the gradient descent update, i.e., $\eta e_j y_i$, is quite similar to the update term of Equations (5.19) and (5.20) for the linear unit. The main differences are the following:

- in the backpropagation algorithm presented above, weights are updated by considering one sample at a time, instead of summing over all the training samples of T. This is an approximated way to perform the weight descent search, which is computationally faster.
- the error $t_d - y_d$ of Equation (5.20) is replaced by a more complex term e_j for each neuron u_j, which follows from the derivation of the error function

$$E(\bar{w}) = \frac{1}{2} \sum_{k \in outputs} (t_k - y_k)^2 \tag{5.27}$$

  namely, the network error given a single sample of the training set.

To understand intuitively how the e_j terms of Equations (5.22) and (5.23) are conceived, let us consider the first equation. Given an output neuron u_k, the error term e_k is simply the term $(t_k - y_k)$ of linear units multiplied by the factor $y_k(1 - y_k)$, which is the derivative of the sigmoid function. As for hidden units u_h, the error term e_h of Equation (5.23) has a similar form but, since no target value is given for hidden units, the error term $(t_k - y_k)$ is substituted by the sum $\sum_{k \in outputs} w_{kh} e_k$. This accounts for the contribution of the error e_k of each unit influenced by the neuron u_h, multiplied by w_{kh}, that is, the weight from the hidden neuron u_h and the output neuron u_k. For the complete derivation of backpropagation rules readers may refer to [24, 160].

Genetic algorithms

Among non-gradient-based optimization techniques, *genetic algorithms* (GAs) [97] have been often used as a learning methodology for neural networks [250]. GAs are a population based search strategy inspired by principles of natural selection and genetics. Individuals in a population compete and exchange information with each other in order to perform certain tasks.

GAs usually encode the solutions of a search problem by finite-length strings called *chromosomes*. The variables of the problem are referred as *genes* and the possible values of variables as *alleles* [27]. For instance, in the ANN learning problem, a chromosome represents the set of all the weights, a gene represents a single weight and an allele represents a weight value. In order to evolve good solutions, natural selection is implemented. An evaluation function is defined which rates the fitness of every chromosome of the population according to its error and/or other rating criteria. Chromosomes are indeed selected for reproduction with a probability proportional to their fitness, thus better solutions transfer with a higher probability their characters to the offspring.

A standard *genetic algorithm* can be summarized as follows [27]:

Initialization: randomly generate an initial population of N chromosomes across the search space.

While the termination condition is not met, *Do*

1. Selection: select the chromosomes to be reproduced according to a selection strategy which employs the survival-of-the-fittest mechanism.
2. Recombination: combine, with a probability called *recombination rate*, parts of two or more parental chromosomes to create new and possibly better solutions (offspring).
3. Mutation: locally and randomly modify genes of every chromosome with a probability called *mutation rate*.
4. Replacement: replace the original parental population with the offspring population generated so far by selection, recombination and mutation operators.

After being selected, solutions are recombined by *crossover operators* with a probability called *recombination rate*, and then each allele is mutated by a *mutation operator* with a probability called *mutation rate*. The new solutions finally

replace the offspring population and the cycle is iterated until a termination condition is met. The four biological-inspired operators used by these algorithms, namely, selection, crossover, mutation and replacement, may be implemented by many different strategies which are summarized in the following. For a detailed description of these strategies the reader may refer to [27].

- *Selection strategies. Roulette-wheel selection* assigns to each chromosome a likelihood to be selected, which is proportional to its fitness. *Tournament selection* chooses $s \in \mathbb{N}$ chromosomes at random, enters them in a tournament against each other and select the fittest. *Rank selection* assigns a rank from 1 to N (number of chromosomes) to every chromosome of the population according to its fitness and then applies the roulette-wheel strategy using ranks as fitness values.
- *Recombination strategies. Single-point crossover* selects a random crossover site over the chromosome length and exchanges the alleles of one side between two chromosomes. *Double-point crossover* selects two random crossover sites over the chromosome length and exchange the alleles between these two sites, between two chromosomes. *Uniform crossover* exchanges every allele of a couple of randomly selected chromosomes with a specific probability.
- *Mutation strategies. Random mutation* substitutes an allele with a new random value. Depending on the particular problem, other ad-hoc strategies can be applied. For instance, if alleles represent numbers, a random value can be added to an allele rather than to substitute it.
- *Replacement strategies. Delete-all replacement* substitutes all the elements of the parent population with the same number of new chromosomes. *Elitist replacement* preserves the best parent chromosomes. *Steady-state replacement* substitutes $s \in \mathbb{N}$ old chromosomes with s new members.

Recombination and mutation rates must be suitable tuned in order to achieve the best performance from the algorithm. The termination condition of GA is usually satisfied when the fitness of at least one chromosome of the population reaches a desired threshold and/or when a maximum number of epochs have been performed. We notice that, conversely to backpropagation error functions, GA evaluation functions can be nondifferentiable, thus, this technique is able to evaluate the fitness of a solution according to several constraints.

Particle swarm optimization

Another non-gradient-based technique we consider in order to train a neural network is *particle swarm optimization* (PSO) [116]. The roots of this metaheuristic lie in the observation of the social behavior of birds flocking and fish schooling, more generally known as swarming. During coordinated search for food, each bird makes its moving decisions based on cognitive aspects (awareness of itself and its position) and social aspects (awareness of other individuals and their positions). This idea inspired Kennedy and Eberhart to develop a method for function optimization [27]. Basically, a PSO algorithm maintains a population of N particles, called *swarm*, where each particle represents a location in a D-dimensional search space, where D is the number of parameters to be optimized. Particles start at random

locations and search for the minimum (or maximum) of a given objective function by moving through the search space according to cognitive and social rules. In particular, each particle i keeps track of its current position $\bar{x}_i = (x_{i1}, \ldots, x_{iD})$ (where each $x_{ij}, j = 1, \ldots, D$ is a parameter to be tuned in order to minimize an objective function $\gamma(\bar{x})$), its current velocity $\bar{v}_i = (v_{i1}, \ldots, v_{iD})$ (where each $v_{ij}, j = 1, \ldots, D$ is the current rate of change of parameter x_{ij}) and its best position $\bar{p}_i = (p_{i1}, \ldots, p_{iD})$, i.e., the position on its past trajectory where it scored the best fitness value. The best position $\bar{p}_g = (p_{g1}, \ldots, p_{gD})$ reached by any particle is considered as well.

The *particle swarm optimization algorithm* can be summarized as follows [27]:

Initialization: randomly initialize location and velocity of N particles across the search space.

Repeat

1. *For each* particle $i \in [1, N]$
   - evaluate the objective function $\gamma(\bar{x})$ on the particle current location $\bar{x}_i$
2. *For each* particle $i \in [1, N]$
   - if $\gamma(\bar{x}_i) < \gamma(\bar{p}_i)$ then $\bar{p}_i = \bar{x}_i$ (particle best update)
3. If $\gamma(\bar{p}_i) < \gamma(\bar{p}_g)$ then $g = i$ (global best update)
4. *For each* particle $i \in [1, N]$
   - *For each* dimension $d \in [1, D]$

$$v_{id} = w \cdot v_{id} + c_1 \cdot r_1 \cdot (p_{id} - x_{id}) + c_2 \cdot r_2 \cdot (p_{gd} - x_{id}) \tag{5.28}$$

$$x_{id} = x_{id} + v_{id} \tag{5.29}$$

until the termination condition is met.

The main parameters employed by this algorithm are *inertia* w, *cognitive coefficient* c_1 and *social coefficients* c_2. Inertia is a nonnegative real number determining the influence of the old velocity on the new particle movement, while cognitive and social coefficients are nonnegative real numbers affecting, respectively, the influence of the local and the global best on the particle movements. A formal description of the equations exploited by this technique in order to move the particles through the searching space is included in [27].

PSO can be straightforwardly applied to train a neural network by mapping each synapse weight into a particle dimension $d \in D$. According to this approach particle movements across the search space correspond to synapses weight tuning. As well as genetic algorithms, this non-gradient-based technique is able to find the global bests of nondifferentiable and multiobjective functions, taking into account both an error function and additional constraints.

A memetic algorithm for learning ANNs

Being gradient-based, backpropagation has good time performance but it can be applied only if the evaluation function (usually the mean square error) is differentiable and not multimodal, otherwise it may get stuck into local optima. On the

other hand, non-gradient-based methods such as GA and PSO, can often overcome this problem by steering the searching process in accordance with evolutionary rules, but sometimes they have lower performance. In order to overcome the drawbacks of both gradient-based and non-gradient-based optimization techniques we have implemented a *memetic algorithm* [125] employing GA for the global search and backpropagation for the local search. In particular, the algorithm initially generates a population of random solutions and then it evolves the population by means of two stages [33]:

- a global search performed by running GA for a certain number of epochs,
- a local search performed on every individual of the population by running backpropagation for a certain number of epochs.

These two stages are iterated until a termination condition is met. In this way, GA guarantees a spread search of global optima over the search space, while backpropagation ensures quite a fast convergence towards local optima. Notice that, both the GA and backpropagation parameters have to be set before the employment of this memetic algorithm. Furthermore, the number of GA steps and backpropagation steps of each iteration have to be suitable selected in order to achieve the best performances.

5.3.4 ANNs for the synthesis of flux regulation functions

We have employed feed-forward neural networks as "observers" of the experimental data collected from metabolic systems in order to capture some rules that govern those systems [33]. Neuron activation functions are set to sigmoid, a very used function for modeling population growth in biology. The nonlinear combination of such functions enables to fit a large range of function forms. Moreover, since sigmoid output ranges in $(0, 1)$, regulation functions generated by them are nonnegative, as required by constraints of Section 5.1.

Given an MP system having n substance, k parameters and m fluxes, each associated to a reaction, we have two ways to connect it to neural networks: *i)* by employing only one neural network having $n + k$ input neurons and m output neurons, as shown in Figure 5.11, *ii)* by using one neural network for each reaction, having $n + k$ input neurons and one output neuron, as displayed in Figure 5.12. In the first case we connect each substance and parameter node of the MP system to a different input neuron of the neural network and each output neuron is connected to a different flux node. In the latter case input neurons of each network still "observe" the MP system state (substances and parameters) but their only output neuron is connected to a specific flux node. Both the configurations work as regressors but, while the first one emphasizes a systemic regulation of the system, the second one tends to preserve the functional independence among the regulation functions, thus some difference between the two approaches could be noticed.

The number of hidden layers and hidden neurons depends on the complexity of the sought regulation functions. As a rule of thumb, the more "complex" the regulation functions, the higher the number of hidden layers and hidden neurons.

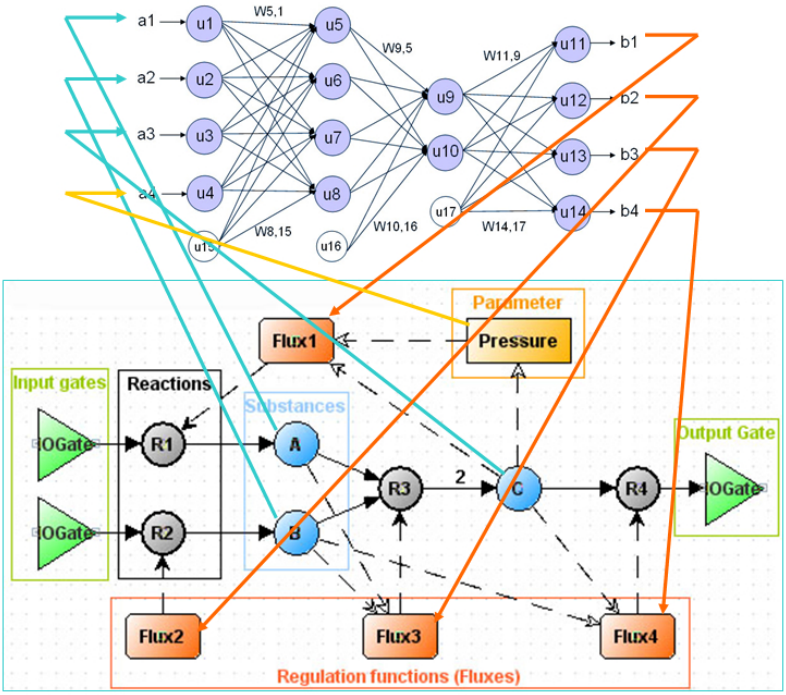


Fig. 5.11. MP systems fluxes computed by a single ANN. Substances and parameters are connected to input neurons while output neurons are connected to fluxes (figure from [33] with permission).

Since sometimes no information is available about the regulation function complexity, few networks having different topologies may be tested, until a good set of regulation functions is found.

A training set is represented by time series of substances and parameters, generally collected by observations, and flux time series computed by the log-gain method [140]. Training data are cyclically "observed" by neural networks which update their weight values at each training epoch (according to some learning rules) in order to minimize the mean square error between their outputs and the outputs of the training sets. We have developed a Java library for training neural networks where four kinds of learning algorithms are available: *backpropagation*, *genetic algorithms* (GA), *particle swarm optimization* (PSO) and the *memetic algorithm* described above. The library has been linked to *NeuralSynth*, a Java software which can be plugged-in to the MetaPlab virtual laboratory in order to automatically learn neural networks from experimental data stored in MP model files. This software, joins the rack of tools developed so far to generate, simulate and analyze MP models within the framework of *MetaPlab*, a plug-in based software for MP systems modeling [34, 241]. Both MetaPlab and the NeuralSynth plugin may be downloaded from [153, 241] and they are described in Chapter 6.

After ANNs has been trained on the experimental data of a specific biological system, they enclose some information about the behavior of that system in some

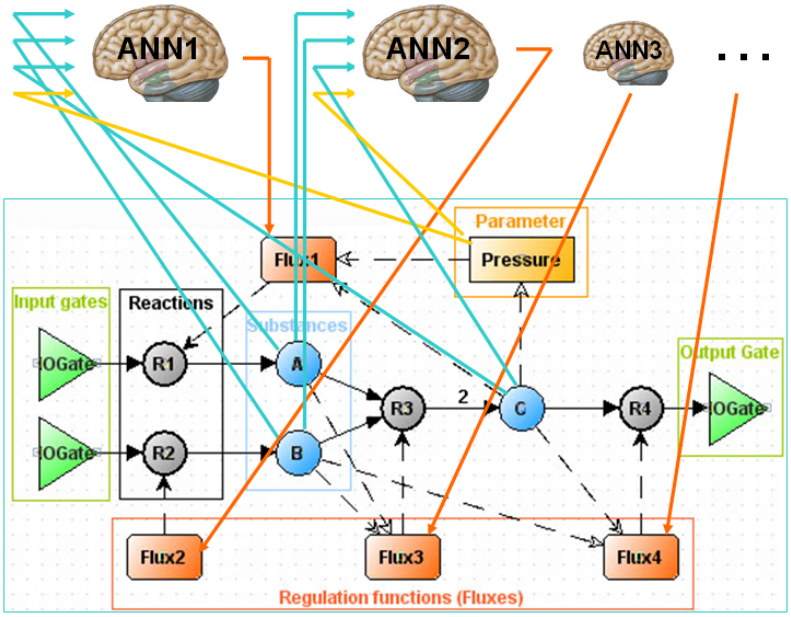


Fig. 5.12. MP system fluxes computed by one neural network for each reaction. Substances and parameters are connected to input neurons while the only output neuron of each network is connected to a specific flux (figure from [33] with permission).

situations. Moreover, the training process employs the information included in the training set in order to guess new information about the system behavior in different states and it stores this information in the network structure and weights. The question which naturally arises at this point is: what do we want to do by these networks? We have two main targets: *i)* to extract and analyze the information they contain in order to achieve a deeper insight of the observed biological systems, *ii)* to simulate, in silico, the dynamics of the system by computing MP fluxes through the trained ANNs. As for network analysis, in Section 5.5 we focus on the application of weight elimination techniques [24] to simplify the network structure and to determine substances and parameters having a major influence on the regulation of each flux. On the other hand, some of the results we achieved by employing ANNs as regulation functions in a real biological system are showed in the next section.

5.4 A case study: mitotic oscillator in early amphibian embryos

In this section we present an application of ANN regression to the synthesis of flux regulation functions in a real biological system, namely, the mitotic oscillator in early amphibian embryos. The results, reported after a brief description of the biological process and an overview of the related MP model, are encouraging, since they highlight the capability of ANNs to learn regulation rules by samples [33].

5.4.1 Process description

Mitosis, the process of cell division, aims at producing two identical cells from a single parent cell. Some studies on yeast and embryonic cells pointed out the existence of an universal mechanism regulating the onset of this process. In particular, many cellular changes related to mitosis are triggered by the fluctuations in the activation state of a protein kinase produced by *cdc2* gene in fission yeast and by homologs in other eukaryotes [85]. In the following we consider the simplest form of this mechanism, which has been observed in early amphibian embryos. In these organisms the process involves two cyclic loops. The first one starts by the progressive accumulation of a protein signal, named *cyclin*, which causes the activation of *cdc2 kinase* when it goes beyond a threshold. The kinase activation is performed by the generation of a complex known as M-phase-promoting (*MPF*) from cyclin and cdc2 kinase. The complex triggers mitosis and promotes cyclin degradation which consequently generates a negative feedback loop that inactivates cdc2 kinase and brings back the cell to the initial state. In the second cycle cdc2 kinase activates a cyclin protease which promotes cyclin degradation.

In 1991 Goldbeter proposed a minimal model, based on ordinary differential equations (ODE), for the mitotic oscillator [85]. According to this model, graphically depicted in Figure 5.13, cyclin is produced at a constant rate v_i and it controls the activation rate of cdc2 kinase, which is considered by the model in its inactive (M^+) and active (M) form. Cdc2 kinase deactivation is instead performed at a rate V_2 by another kinase which is not considered. The model also represents inactive (X^+) and active (X) forms of cyclin protease whose activation is promoted by active cdc2 kinase (dashed arrow from M). In Figure 5.13 the arrow connecting X to X^+ takes into account the protease deactivation while the dashed arrow starting from X denotes the control performed by X on cyclin degradation.

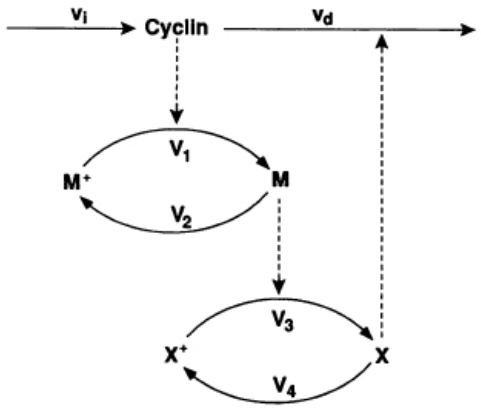


Fig. 5.13. Graphical model, devised by Goldbeter, of the mitotic oscillator [85].

The mathematical representation conceived by Goldbeter to represent this model consists on the following ODE system:

$$\begin{aligned}\frac{dC}{dt} &= v_i - v_d X \frac{C}{K_d + C} - k_d C \\ \frac{dM}{dt} &= V_1 \frac{(1-M)}{K_1 + (1-M)} - V_2 \frac{M}{K_2 + M} \\ \frac{dX}{dt} &= V_3 \frac{(1-X)}{K_3 + (1-X)} - V_4 \frac{X}{K_4 + X}\end{aligned} \tag{5.30}$$

where $V_1 = \frac{C}{K_c+C} \cdot V_{M1}$, $V_2 = 1.5$, $V_3 = M \cdot V_{M3}$, $V_4 = 0.5$, $V_{M1} = 3.0$, $V_{M3} = 1$, $K_i = 0.005$, $i = 1, \ldots, 4$, $v_i = 0.025\ \mu mol \cdot min^{-1}$, $v_d = 0.25\ \mu mol \cdot min^{-1}$, $K_d = 0.02\ \mu mol$, $K_c = 0.5\ \mu mol$, $k_d = 0.01\ min^{-1}$. A detailed description of this model is out of the scope of this thesis but we recall some notation meanings for the easy of reading. Symbol C denotes cyclin concentration, while M and X represent the fraction of active cdc2 kinase and cyclin protease, thus $(1 - M)$ and $(1 - X)$ are the inactive fraction of the same proteins (corresponding, respectively, to M^+ and X^+). Parameters v_i, v_d and k_d are constant rates of cyclin synthesis and degradation, K_d and K_c are Michaelis constants for cyclin degradation and cyclin activation of the phosphatase, V_i and K_i $(i = 1, \ldots, 4)$ are the maximum rates and the Michaelis constants that characterize the kinetics of enzymes E_i $(i = 1, \ldots, 4)$ which activate and deactivate cdc2 kinase and cyclin protease. The numerical solution of the ODE system (5.30) from initial conditions $C = 0.01 \mu mol$, $M = X = 0.01$ shows the oscillatory behavior displayed in Figure 5.14. Readers may refer to [85] for a more accurate description of the model.

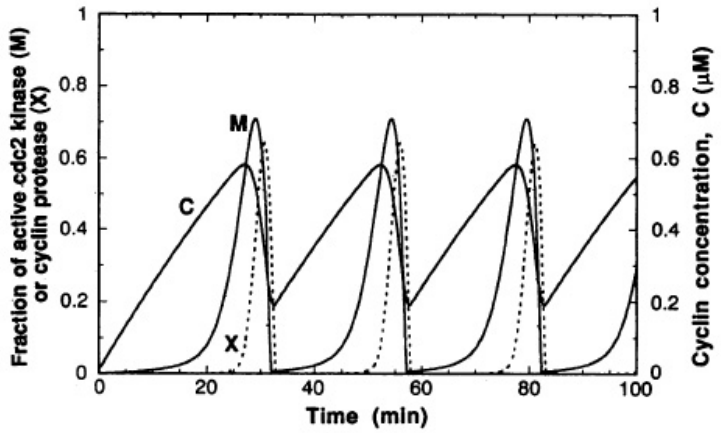


Fig. 5.14. Numerical solution of the ODE system (5.30) [85].

5.4.2 MP model

In [144] Manca and Bianco proposed three translations of the ODE model (5.30) to MP systems that exhibit an equivalent oscillatory behavior. In that work regulation functions have been directly generated from differential equations by means of the

mapping procedure defined in [68], which guarantees the model equivalence under specific conditions. Here, starting from a simplification of a model given in [144], our aim is to automatically synthesize, by means of neural networks, the flux regulation functions of an MP model from the observed time series of substances C, M, M^+, X, X^+.

Let us start by considering an MP system we call *AMC* (Amphibian Mitotic Cycle) in which, *i)* substance symbols C, M, M_p, X, X_p have been already described above (M_p and X_p coincide, respectively, with M^+ and X^+); *ii)* reactions and regulation functions are listed in the following:

$$\begin{aligned}
&R1 : \lambda \rightarrow C && , \varphi_1(Q) = v_i \\
&R2 : C \rightarrow \lambda && , \varphi_2(Q) = k_d C \\
&R3 : C\ M_p \rightarrow C\ M && , \varphi_3(Q) = V_1 \tfrac{M_p}{K_1 + M_p} \\
&R4 : C\ X \rightarrow X && , \varphi_4(Q) = \tfrac{v_d X C}{K_d + C} \\
&R5 : M \rightarrow M_p && , \varphi_5(Q) = \tfrac{V_2 M}{K_2 + M} \\
&R6 : X_p\ M \rightarrow M\ X && , \varphi_6(Q) = \tfrac{V_3 X_p}{K_3 + X_p} \\
&R7 : X \rightarrow X_p && , \varphi_6(Q) = \tfrac{V_4 X}{K_4 + X}
\end{aligned}$$

where constant values and initial conditions are as in (5.30); *iii)* the temporal interval $\tau = 0.06$ *sec*. An MP graph representation of this model is displayed in Figure 5.15.

We generate the dynamics of this model for 4000 steps obtaining five time series, one for each substance, and we use these data as training set for ANN synthesis. These are synthetic data, but of course, the same methodology could be applied to data coming from experimental observations. Flux time series have been generated by means of the log-gain principle, as described in Section 4.6. Readers can generate the time series of both substances and fluxes by means of two MP plug-ins contained in the MetaPlab free package (for the sake of simplicity, the entire training set can be downloaded from [153] within an MP model file).

Once collected all the training set data we launch the plugin NeuralSynth and select one of the training techniques among backpropagation, GA, PSO and the memetic algorithm introduced above. Depending on the training technique the software displays a slight different interface asking for the corresponding training parameters and the topology of the neural networks to be trained (other options can be selected but they are described in Chapter 6). At the end of the training process each neural network provides a function which is stored into a flux node.

5.4.3 Tests

We performed five training tests for each of the four learning techniques (backpropagation, GA, PSO and memetic algorithm). In every test we connected seven independent neural networks to the MP model, one for each reaction according to the general method shown in Figure 5.12. The aim of each network is to learn the regulation function of a specific reaction from training set samples. Since the lower error has been achieved by the memetic algorithm (e.g., Figure 5.16), in the following we detail only the results obtained by such technique.

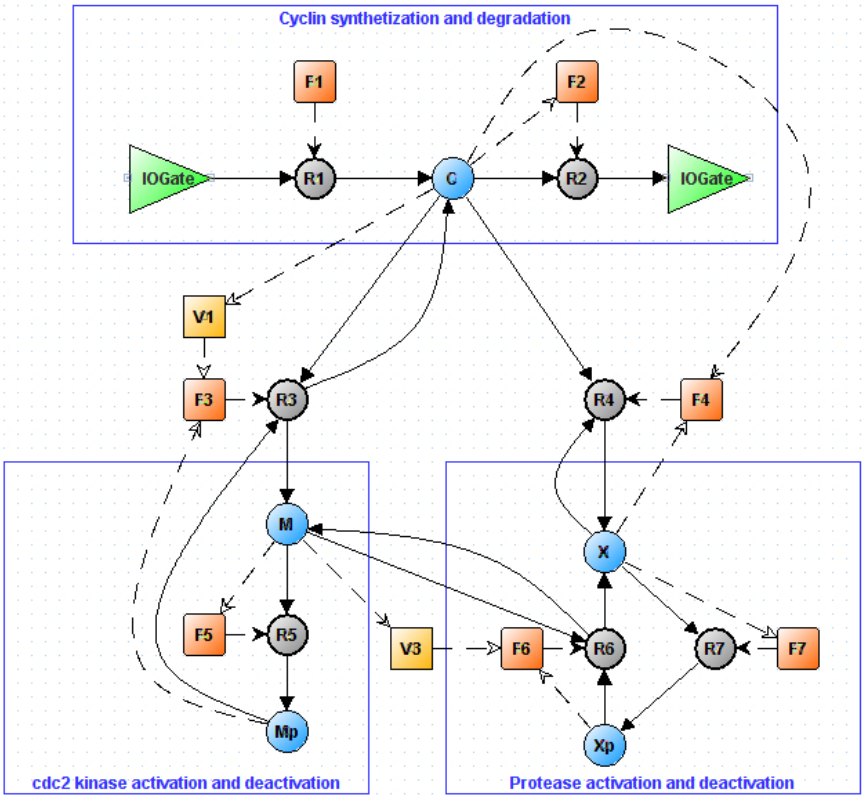


Fig. 5.15. An MP graph of the mitotic cycle (figure from [33] with permission).

We start by initializing a population of 4 individuals, each encoding a neural network having 1 hidden layer with 3 sigmoid neurons and random weights in $[-1, 1]$. This number of individuals, even if quite small, ensures a more spread search than the classical backpropagation though keeping good learning performance. As for the *global search*, the population is evolved by GA for 100 epochs using rank selection with elitist replacement, double point crossover with a recombination rate of 0.8 and random mutation with a mutation rate of 0.1 and mutated values in [-1,1]. After every global search a *local search* is performed by running backpropagation on each of the four individuals for 900 epochs by employing a learning rate of 1.0 and a momentum rate of 0.1. Then the optimization loop starts again by a new global search, until 10000 training epochs have been performed. The same process is repeated for every neural network connected to the MP model.

Training parameters have been accurately selected as they achieved good performance for some benchmark functions (such as, Michalewicz's function) during a previous validation of the learning algorithms (see Section 6.2.7). A detailed description of GA, backpropagation and their parameters can be found, respectively, in [27] and [24].

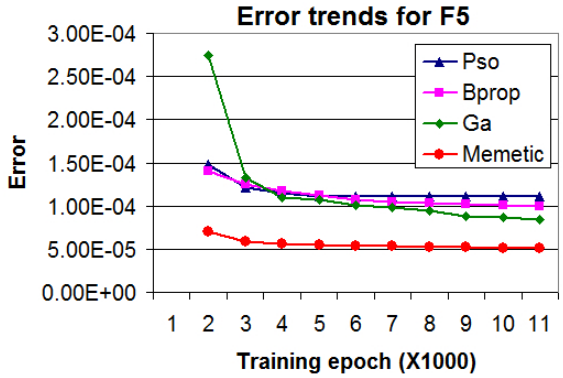


Fig. 5.16. Error trends of backpropagation, GA, PSO and memetic algorithm achieved from the best training processes of flux $F5$. Memetic algorithm reached the best results even for all the other fluxes (figure from [33] with permission).

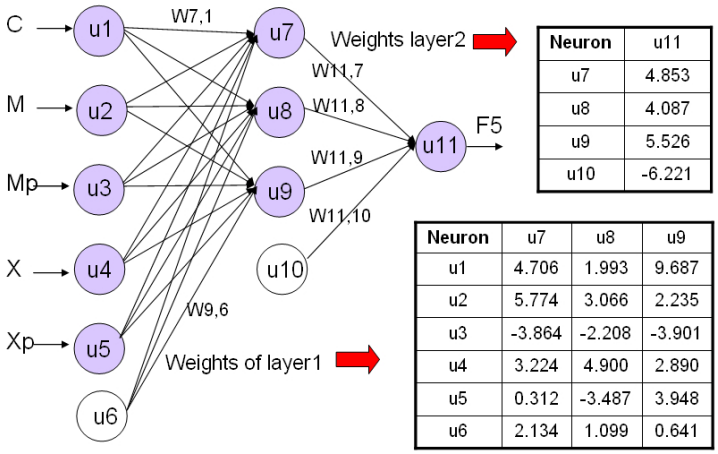
5.4.4 Results

For each reaction R_i, $i = 1, \ldots, 7$, Table 5.1 shows *i)* the mean error $\bar{e}$ (over the five tests) between fluxes computed by the i-th neural network and the related flux values for R_i in the training set, *ii)* the standard deviation σ of these errors. We randomly split the set of 4000 samples into a training set containing the 20% of samples and a validation set including the remaining 80% of them, and we computed errors $\bar{e}$ on the validation set. Different mean errors have been achieved for different reactions. This is quite normal since some reactions, such as R_1, have very simple regulation functions in the original model while other reactions have quite complex regulation functions that are harder to learn. However, we remark that neural networks having just one hidden layer with three neurons achieved very low errors and they managed to fit quite precisely the observed data points. Moreover, time performances have been fairly good, since every single test has been performed in about 30 minutes by a laptop equipped with a processor Intel(R) Core(TM)2 Duo CPU T7250, 2.00GHz and 2038 MB of memory.

Flux	$\bar{e}$	σ
R_1	8.806E-09	5.675E-09
R_2	7.540E-06	6.238E-06
R_3	2.533E-05	1.792E-05
R_4	2.223E-05	6.798E-06
R_5	7.750E-05	3.987E-05
R_6	6.258E-06	5.066E-06
R_7	2.935E-04	1.095E-04

Table 5.1. Mean training errors $\bar{e}$ of regulation functions computed by ANNs (over five training tests) and related standard deviations σ, for each reaction [33].

Figure 5.17 shows, as an example, a neural network trained for reaction $R5$ and its weight values. The translation of such a kind of networks to mathematical functions yields quite long strings that we avoid to mention here for a matter of space. Readers can visualize these functions downloading the final MP model from [153] and, eventually, they can compute the related dynamics by MetaPlab.



Neuron	u11
u7	4.853
u8	4.087
u9	5.526
u10	-6.221

Neuron	u7	u8	u9
u1	4.706	1.993	9.687
u2	5.774	3.066	2.235
u3	-3.864	-2.208	-3.901
u4	3.224	4.900	2.890
u5	0.312	-3.487	3.948
u6	2.134	1.099	0.641

Fig. 5.17. On the left: the best ANN trained for reaction $R5$. On the right: ANN weight values (figure from [33] with permission).

Figure 5.18.a displays the training set dynamics of substances C, M and X, while the dynamics generated by using trained ANNs as regulation functions is displayed in Figure 5.18.b. In particular, ANNs employed to generate this chart have been trained during the third test on the memetic algorithm. The high similarity between the two evolutions, highlighted by Figure 5.18.c in the case of active cdc2 kinase, shows the capability of ANNs to learn regulation functions from sample data. The mean error between the two curves of Figure 5.18.c is equal to 0.01216. This error is equal to 0.00624 for cyclin curves (C) and 0.01046 for active cyclin protease (X).

5.5 Variable selection for flux regulation functions

The problem we tackle in this section concerns the automatic discovery of flux tuners from observed data [36]. In the following we will call *tuners* of a flux regulation function φ_i, the variables (i.e., substances and parameters) involved in the function [140]. In fact, it is known that every reaction of a biochemical system transforms reactants into products with a rate depending on the instantaneous value of some substances and parameters. Discovering these elements provides key understanding about the system and it may suggest new experiments.

From regression theory it turns out that regulation functions should have as few independent variables as possible in order to give to MP models the best

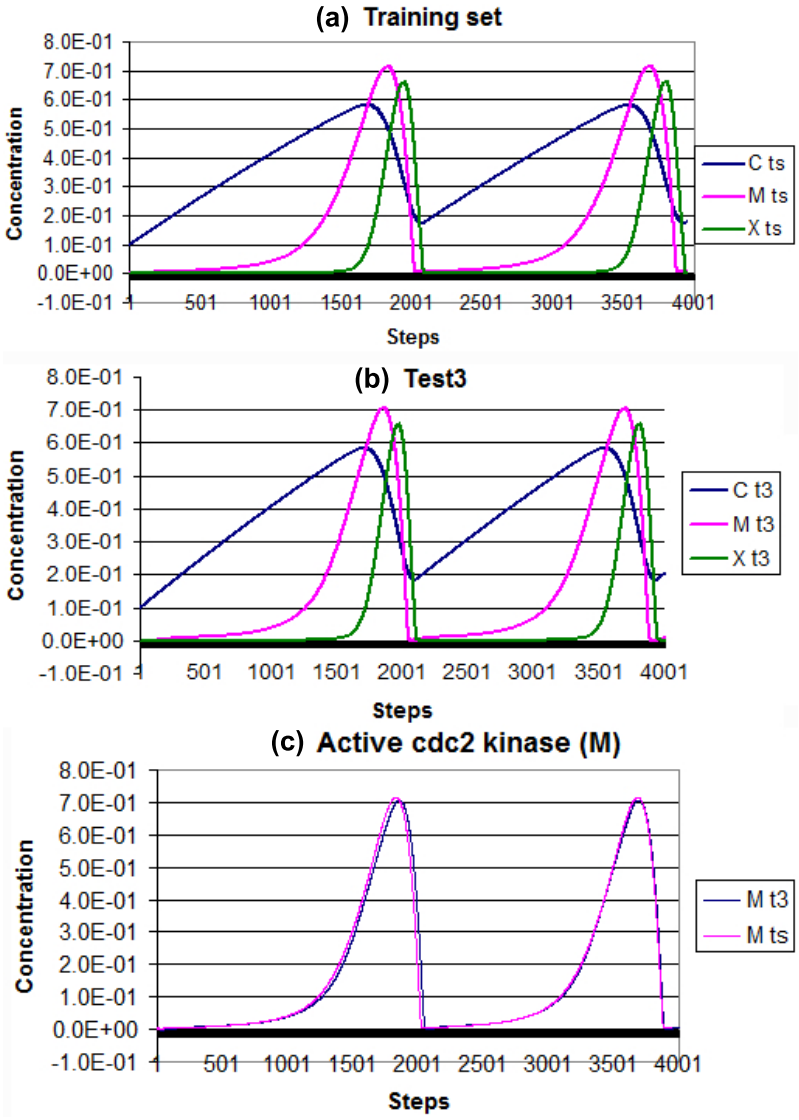


Fig. 5.18. a. Training set dynamics of the mitotic oscillator (4000 samples); b. Dynamics computed by means of the regulation functions synthesized in the third test of the mitotic algorithm; c. Comparison between the two evolutions of active cdc2 kinase (figure from [33] with permission).

predictions capabilities [1]. This statement could sound a bit counterintuitive since it seems logical that, if a regulation function incorporates as many variables as possible, then its flux prediction should be more accurate. As a matter of fact, this is true only if the number of data points to be fitted has no limitations (which is not realistic), indeed, because of the *curse of dimensionality* [8], as the dimensionality of the fitting surface increases also the degrees of freedom of this

surface increase, and the number of points needed to achieve a good fitting surface increases as well. Therefore, functions generated by regression methods have to be parsimonious in the number of independent variables in order to capture the systematic trend of data while avoiding uncertainty and overfitting typical of high-dimensional functions [1].

A critical point of regulation function synthesis is thus the choice of a proper set of basis functions (i.e., variables) for the flux under investigation. A model involving many basis functions may provide excellent approximations but it could cause problems of overfitting. On the other hand, a model having not enough basis functions is usually unable to capture all the features of the systematic component in the flux data. In both cases, the ability of the model to generalize to new data would be relatively poor and prediction on new data would be of very low quality.

Our aim is to identify the relationships between each flux dataset and every substance/parameter dataset, in order to select a minimal subset of variables involved in the regulation maps. Functions employing these subsets of variables have three main advantages: *i)* they provide a better understanding of the functional role of each substance/parameter in the regulation of fluxes, *ii)* they improve the prediction performance of MP models, since variable parsimony enhances the signal-to-noise ratio, and *iii)* they reduce the computational complexity of learning and simulation algorithms [89, 165].

The problem of variable selection for building a good prediction model often contrasts with the problem of finding all potentially relevant variables involved in a process [122]. Indeed, good predictors should employ a minimal number of variables (parsimony) to reach better performance, while variables that influence the process but do not convey new information may possibly be discarded. In our specific case, the flux regulation functions generated by using variable selection techniques will involve a set of substances/parameters which may not correspond to the entire set of substances and parameters regulating the reaction in the real system. Some substances/parameters may be discarded because the information they convey is redundant.

There exist several techniques for variable selection, which can be summarized in three main classes [89]:

- *filters* select variables by ranking them according to specific criteria, such as, *correlation* coefficients [1, 89, 213] or *mutual information* coefficients [89, 213, 227] between variables and fluxes. These tools can be applied, in principle, to every kind of regression model although the significance of their results may be different. For instance, correlation methods best fit linear models but sometimes they are applied also to nonlinear models in order to gain some hints of possible relationships among variables. Many variable selection algorithms include variable ranking as preprocessing step, independent of the choice of the prediction model, because of its simplicity, scalability, and good empirical results. A common use of these techniques is, for instance, microarray analysis for discovering sets of drug leads [86], where a ranking criterion is employed to identify genes that discriminate between healthy and disease patients;
- *wrappers* assess subsets of variables according to their usefulness to a given regression model [89]. The regression model of interest is used here as a black box to score subsets of variables according to their predictive power, thus on-

the-shelf prediction tools can be used as variable testers. In order to employ wrapper methods, we need a strategy for exploring the search space of variable subsets, a predictor for testing variable subsets, and a methodology for assessing the predictor performance. Since the search space becomes quickly large as the number of variables increases, many search strategies have been proposed that ensure to find good variable subsets while avoiding exhaustive searches [122]. As for predictors, the most used regression and feature selection models are usually employed, such as, least-square linear predictors, decision trees, naive Bayes classifiers and support vector machines. Performance assessments are usually performed by means of a validation set of data removed from the initial training set, or by cross-validation. Even if wrappers are often criticized because of their brute force approach requiring massive computational capabilities, this is not always the case. Indeed, when efficient search strategies are employed, such as greedy strategies (forward selection, backward elimination, etc.), good results may be achieved without sacrificing prediction performance;

- *embedded methods* are dependent on specific regression models, since they perform the selection of variable subsets during the training stage. Specifically, variable selection becomes part of the optimization process in which model parameters are tuned, so that a multiobjective optimization is performed. This is often achieved by using objective functions that consist of two terms competing with each other: a term accounting for the goodness of fit, and a term coping with the number of variables (to be minimized). The search is performed by estimating changes in the objective function produced by moving first in the variable subset space, and afterward in the corresponding space of model parameters. The main advantages of embedded methods with respect to wrappers concern efficiency. By avoiding to retrain the predictor from scratch for every variable subset investigated, embedded methods are often faster than wrappers. Moreover, embedded methods often exploit better the information available in the datasets since they do not need to split the training data in a training and a validation set [89].

Notice that, whilst filters are quite simple tools that enable to rank variables according to their *individual* predictive power, wrappers and embedded methods can involve quite complex tools able to select *subsets* of variables that *all together* have a good predictive power. Indeed variables that are useless by themselves can be useful together. A more detailed description of all the methods introduced above is available in [89].

In the next two subsections we present two methodologies for variable selection, namely, stepwise regression for multivariate linear functions and an approach based on weight elimination for neural networks, that can be seen as an implementation of the minimum description length principle. These techniques have been chosen for their simplicity and rich statistical background, but many other approaches may be tested [28], such as, the Akaike information criterion (grounded in the concept of entropy) and bayesian model comparison (rooted in Bayes's theory).

5.5.1 Variable selection with linear models

The linear regression techniques introduced in Section 5.2 employed a trial and error method for deciding which monomials ψ_j to include in a polynomial regulation function of the form:

$$\varphi_r(q) = \sum_{j=1}^{p} b_j \psi_j(q). \tag{5.31}$$

The idea, according to the schema of Figure 5.4, was to start with a subset $s' \subseteq S$ of monomials, where S is the set of all possible monomials, then to test the achieved model (by analyzing residuals, multicollinearity and other properties), and, if the performance are not acceptable, to select other subset of monomials, until good performance are reached. Since for systems with many variables it is very difficult to find manually a good subset of monomials, some computational procedures are often employed for tackling this problem in a systematic way. In the following we report some of this methods.

All possible regressions

This method consists of testing all the possible models achievable from a set S of monomials. Let us consider, for instance, a system having four variables, i.e., x_1, x_2, x_3, x_4. If we want to test all the models having a degree less than or equal to two, we must put in the set S: *i)* the four variables $x_1, \ldots, x_4$, *ii)* their powers $x_1^2, \ldots, x_4^2$, etc., and *iii)* all their cross products $x_1x_2, x_1x_3, \ldots$, thus the total number of possible monomials is 14. Since each monomial may be either included or not in the model and the model involves also an intercept term, we should test $2^{14} = 16384$ models. Of course, this method can be used only if the number of variables of the system and the degree of the considered polynomials are very small, otherwise it takes too much time to be accomplished.

Forward selection and Fisher's test

To overcome the performance issue of the previous technique we need a search procedure able to find a good subset of monomials without exploring the whole space of monomial subsets. Forward selection starts with a model with no variable and it adds one monomial at a time, according to a statistical test, until a predefined performance level is reached.

The statistical test employed is *Fisher's test*, also called *F-test*, and it is briefly explained in the following. Let us consider a multiple linear model of the form

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5 + \epsilon \tag{5.32}$$

and suppose we want to test the significance of the subset of variables $s' = \{X_2, X_4\}$ in the model. This information is very important since if it turns out that variables X_2 and X_4 are not significant, then a more parsimonious model can be employed, using only variables X_1, X_3 and X_5 to achieve the same prediction performance. In statistical terms, we want to test the following hypothesis:

$$H_0 : \beta_2 = \beta_4 = 0 \tag{5.33}$$

$$H_1 : \beta_2 \text{ and } \beta_4 \text{ are not both zero.} \tag{5.34}$$

The test statistic for checking this kind of hypothesis test is the partial F statistic [1]:

$$F_{[r,n-(p+1)]} = \frac{(SSR_R - SSR_F)/r}{\hat{\sigma}_F^2} \tag{5.35}$$

where SSR_F is the sum of squared residuals of the full model (having p variables), SSR_R is the sum of squared residuals of the reduced model (having $p-r$ variables), $\hat{\sigma}_F^2$ is the residual mean square of the full model, r is the number of variables dropped from the full model to create the reduced model, and n is the number of samples in the dataset. In our example, if we have a dataset with $n = 100$ samples, than we have to use the F statistic $F_{[2,94]}$, to find the threshold value over which the null hypothesis H_0 should be rejected. This topic requires some statistics skills, thus we explain it by a simple example. Statistic tables, that may be consulted in [1], give us a critical value close to 5.2 for the F distribution $F_{[2,94]}$ with a significance level $\alpha = 0.025$. This means that if the right side of Equation (5.35) returns a value greater than 5.2, then we have only less than the 5% of chance that the null hypothesis be true. Consequently the null hypothesis should be rejected and variables X_2 and X_4 should not be dropped from the model. On the other hand, if the value returned by Equation (5.35) is less than 5.2, then the null hypothesis should not be rejected and the reduced model should be kept.

When the F test is employed in forward selection, it suggests to add to the current polynomial, having p terms, the i-th monomial of a set S, such that [94]:

$$F_i = max_i(\frac{SSR_R - SSR_{F_i}}{\hat{\sigma}_{F_i}^2}) > F_{in} \tag{5.36}$$

where SSR_R is the sum of squared residuals of the original model, having p monomials, SSR_{F_i} is the sum of squared residuals of the model in which the i-th monomial has been added, $\hat{\sigma}_{F_i}^2$ is the residual mean square of the same model and F_{in} is a threshold, computed by means of the F statistic, resulting in a rule for terminating the variable search.

The selected (i-th) monomial is therefore the monomial which maximizes the difference between the residuals sum of squares before and after its insertion (averaged on the residual mean square of the model after the insertion), if it is greater than the threshold F_{in}. In other words, the procedure keeps adding the monomials which maximize the performances of the model, until the performance increasing falls under the threshold F_{in}.

Backward elimination

An opposite way to select a subset of monomials for a linear model involves the elimination of unnecessary terms from a starting polynomial including all the possible monomials. At each step, the partial F statistic is computed for assessing the elimination of each monomial i and the term with the smaller F_i value is eliminated if it does not exceed a specified threshold. That is, the backward elimination procedure drops, at each step, the monomial i, such that [94]:

$$F_i = min_i(\frac{SSR_{R_i} - SSR_F}{\hat{\sigma}_F^2}) < F_{out} \tag{5.37}$$

where SSR_{R_i} is the sum of squared residuals of the model in which the i-th monomial has been dropped, SSR_F is the sum of squared residuals of the original model, having p variables, $\hat{\sigma}_F^2$ is the residual mean square of the original model and F_{out} is a threshold, computed by means of the F statistic, resulting in a rule for terminating the variable search.

In this case, the monomial selected for deletion is the monomial which minimizes the performance decreasing, in term of difference between the sum of squared residuals after and before its elimination (averaged on the residual mean square of the model before the elimination). The process iterates until all variables in the model are significant in term of their partial statistic, and consequently the performance decreasing for dropping any monomial is greater than the threshold F_{out}.

Stepwise regression

The necessity to overcome some typical shortcomings of both forward selection and backward elimination suggested a number of combination between the two techniques. Stepwise regression, first presented by Efroymson [58, 59], is probably the most commonly used method for variable selection in linear models. It basically entails an alternate application of forward and backward steps by starting from an initial model and ending when no step is applicable anymore. In this way, after a variable has been entered in the model (by means of forward step) it can be also dropped (by a backward step) if a subsequent reevaluation of the model shows that the variable's significance is decreased, because of the presence of other variables inserted more recently. On the other hand, variables eliminated from the model at a certain step can be re-inserted subsequently if their significance has increased after some insertion and deletion of other variables.

By combining forward selection and backward elimination, stepwise regression reevaluates the significance of every variable at every step. Thus it tends to avoid multicollinearity, i.e., the presence of redundant variables in the model, typical of forward selection, and it minimizes the chance of leaving out relevant variables, which is a drawback of backward elimination. The main steps of the stepwise regression procedure can be summarized as follows:

1. identifying an initial model and a set S of basis functions (monomials) available for inclusion,
2. adding to the current model the variable i of set S yielding the largest *F-test* index F_i according to Equation (5.36).
3. removing from the current model the variable i yielding the smallest *F-test* index F_i according to Equation (5.37),
4. repeating step 3 until no further variables can be dropped, then go to step 2,
5. terminating the search when neither step 2 nor step 3 can be performed, or when a maximum number of steps has been executed.

Notice that, parameters F_{in} and F_{out} must be carefully tuned, since they determine the termination conditions of the procedure. To ensure the termination we must have $F_{in} > F_{out}$ [55, 203]. It is important also to note that stepwise

algorithms may not find the best model since there is order dependence in the selection process, thus we may not always arrive to the same model. However, there is evidence of many good results achieved by this method for real-world regression problems.

When applied to MP systems, stepwise regression enables to compute simultaneously a polynomial model and a set of tuners for each regulation function. The tuners found do not necessarily correspond to the complete set of substances and parameters involved in the regulation of a flux but they tend to be a minimal set of variables which suffice for regulating the flux by means of a polynomial model. That is, variables conveying redundant information, which are typical in nature to preserve system robustness, could be discarded in order to increase the parsimony (and consequently the performance) of the prediction model.

5.5.2 Variable selection with neural networks

ANNs have been introduced in Section 5.3 as models for MP flux regulation functions. The techniques presented in that section for training ANNs do not consider the selection of input variables, but they only cope with weight tuning of networks having pre-defined sets of variables. This approach yields good models if the system under investigation is well known, since in this case input variables can be manually selected. However, if the process has unclear features which we want to elucidate by neural models (such as, unknown relationships between substance concentrations and reaction fluxes) then the variable selection stage should be automatized by selection algorithms specific for neural networks. These techniques enable to generate meaningful neural models and to exploit these models for unveiling new relationships among the elements of the system.

Variable selection techniques for neural networks usually enclose the variable selection process in the training process, since variable subsets are sought as a further way to optimize model performance. The aim of this global optimization is to maximize the prediction capabilities of the model while employing a minimal set of variables having biological relevance for the flux function being modeled. However, since the overall optimization depends on the model parameters (i.e., synapse weight values), it may be necessary to train the network with different sets of variables: some selection procedures alternate between variable selection and retraining of the model parameters [128].

It is important to note that neural models have some specificities that must be taken in consideration by variable selection algorithms. First of all, ANN are nonlinear in their parameters. This means that the majority of the methods assuming linear relationships between inputs and outputs, such as correlation-based methods, are ill fitted for neural models. Second, the search space of ANN weights has many local minima. Since variable relevance depends on the minimum the ANN has converged to, relevance measures should be averaged over several training runs.

In [128] a complete overview of the main algorithms for feature selection with ANNs is given. Here we report a summary of the principal criteria used to rank and select variables:

- *zero order methods* use only the network parameter values (i.e., weight values) to rank each variable. Yacoub and Bennani [249] proposed an heuristic which

exploits both weight values and the network topology of multilayer perceptron to rank variables according to a saliency measure;

- *first order methods* employ the first derivatives of network weights. That is, to evaluate the relevance of an input variable these techniques measure the variation of the output with regard to the variation of the input, when the other inputs are kept fixed. Since these derivatives are not constant, like in linear models, they must be averaged over the training set in order to obtain meaningful relevance measures. Moody and Utans [161] proposed *saliency based pruning* (SBP), in which the relevance of a variable x_i is measured by evaluating the variation of the learning error when the variable is replaced by its sample mean $\bar{x}_i$. This is a direct measure of the usefulness of the variable for computing the output. Several authors have proposed to measure the sensitivity of a neural model with regard to an input variable x_i by computing the mean value of output derivatives with respect to x_i over the whole training set [92, 190, 202];
- *second order methods* use the second derivatives of network weights. Specifically, several variable selection methods in this class evaluate the relevance of an input variable x_i by applying some *weight pruning criteria* to the set of weights of the related input neuron u_i. MacKey [135] has proposed an approach based on bayesian learning. *Optimal cell damage* has been introduced by Cibas et al. [42,43] and it is inspired from the weight pruning technique developed by LeCun [47] and named *optimal brain damage*. In these techniques, the saliency of an input neuron is usually defined as the sum of its output weight saliences. The saliency of each weight can be computed in several ways, one of these is the variation of the model error (over the training set) with respect to the variation of the weight itself. Other weight pruning techniques are called, respectively, *early brain damage* and *early brain surgeon*, and have been proposed by Tresp et al. [228].

The majority of these techniques perform a backward search, since they start from a large set of possible variables and they identify, in different ways, a subset of variables to be kept in the model and another subset of variables to drop.

The methodology we have presented in [36] for discovering flux tuners by means of neural networks, consists of two steps:

1. application of the *weight elimination* technique [24, 245], during the network training, for removing unnecessary synapse weights,
2. assignment, to each substance (parameter) of the MP system, of a *tuning index* for each flux, rating the saliency of the substance (parameter) to tune the flux itself.

We can thus classify this technique as a zero-order variable ranking methodology, according to the classification reported above.

Weight elimination

Weight elimination [24, 245] is a technique aiming to find a neural network which fits a specific training set by using the smallest number of weights. The hypothesis on which this method is based states that "if several networks fit the data equally well, then the network having the smallest number of weights will on average

provide the best generalization", that is, it will get the best predictions for new data.

The idea is to add to the backpropagation cost function (usually a square error), a term which "counts" the number of weights, obtaining the new cost function [24]:

$$E = \sum_{k \in \mathcal{T}} (target_k - output_k)^2 + \lambda \sum_{i \in \mathcal{C}} \frac{w_i^2}{\hat{w}^2 + w_i^2}. \tag{5.38}$$

and then to minimize this function by means of backpropagation. The first term of Equation (5.38), called *performance term*, represents the square error between network outputs, i.e. $output_k$, and target outputs, i.e. $target_k$, over the entire training set $\mathcal{T}$. The second term, named *complexity term*, deals with the network size. Its sum, which extends over all the synapses $\mathcal{C}$, adds a penalty value close to unity (times λ) to each weight $w_i \in \mathbb{R}$ such that $|w_i| >> \hat{w}$, for some threshold $\hat{w}$ while it adds a penalty term approaching to zero to each weight w_i such that $|w_i| << \hat{w}$. The parameter $\lambda \in \mathbb{R}^+$ represents the relative importance of the network simplicity with respect to the network performance.

When the classical backpropagation learning algorithm is employed with the cost function of Equation (5.38), weights are updated at each step according to the gradient of both the performance and the complexity terms, thus a trade-off between a small fitting error and a small number of weights is found. In other words, the complexity term tends to "push" every weight to zero with a strength proportional to weight magnitudes and to λ, while the performance term keeps far from zero the weights actually needed to fit training data. Notice that, parameter λ is a sensitive factor in this procedure, since if it is too small, then the complexity term has no effect, while if it is too large then all the weights are driven to zero. Moreover, the value of λ usually changes depending on the problem. In [245] some heuristic rules are presented for dynamically tuning the value of λ during the training process in order to find a minimal network while achieving a desired level of performance on training data.

The weight-elimination technique has been implemented in the NeuralSynth plug-in [33], a Java software, described in Section 6, which can be employed within the MetaPlab virtual laboratory to automatically learn neural networks from experimental data. By employing a specific feature of the plug-in, neural networks are trained on time series data and, at the same time, their unnecessary weights are removed.

Tuning indexes assignment

The second step of the tuners discovery strategy proposed in [36] involves the analysis of the neural networks achieved at the first step, with the aim to evaluate the sensitivity of each flux to the variation of each substance and parameter. Given a trained (and minimized) neural network encoding a regulation function $\varphi(q)$, we assign to each input neuron x (which is connected to a substance or a parameter node according to the schema of Figure 5.12) a *tuning index*:

$$\xi(x) = \sum_{p \in path(x,o)} \prod_{w \in p} |w| \tag{5.39}$$

where $path(x, o)$ is the set of all paths from the input neuron x to the (only) output neuron o (connected to a flux node according to the schema of Figure 5.12), and each path $p \in path(x, o)$ is, in turn, the set of weights of synapses on the path from x to o. In other words, the tuning index $\xi(x)$ rates the saliency of the substance (parameter) connected to the input neuron x to tune the flux connected to the output neuron o. This index is computed by summing, for every path from the input neuron x to the output neuron o, the product of weights in the path. Similar techniques [128, 249] have been already employed in several fields, such as in finance [56].

The idea behind this heuristic for computing tuning indexes is informally explained by means of Figure 5.19. In that picture, red thin arrows represent synapses having weights with small absolute values, green thick arrows stand for synapses having weights with large absolute values, and orange medium-thickness arrows represent synapses having weights with medium size absolute values. From Figure 5.19 it is evident that the contribution of a single path from the input neuron u_1 (related to substance A) to the output neuron u_9 (connected to flux F_1), is proportional to the product of the absolute values of weights on the path between u_1 and u_9. Moreover, the overall contribution of input A in tuning output F_1 is related to the sum of the contributions of every path. This is because each neuron computes a sigmoid function of the weighted sum of its inputs, as already described in Section 5.3.

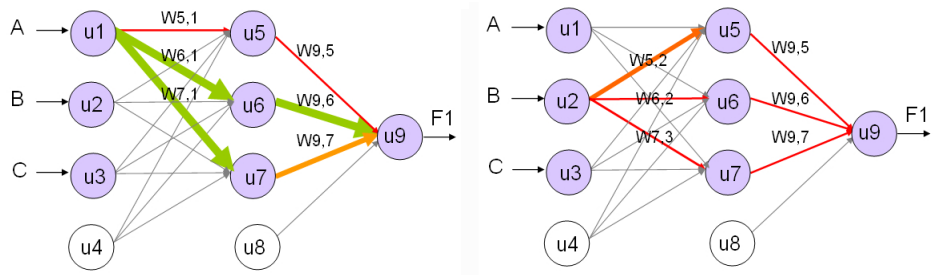


Fig. 5.19. Weight analysis of paths from the input neurons u_1 (on the left) and u_2 (on the right), to the output neuron u_9 for computing the tuning indexes of, respectively, substance A and B in respect of flux F_1 (figure from [36] with permission).

Let us consider a simple example. On the left side of Figure 5.19, the contribution of path $u_1 \to u_5 \to u_9$, that is $|w_{5,1}| \cdot |w_{9,5}|$, is smaller than the contribution of path $u_1 \to u_6 \to u_9$, that is, $|w_{6,1}| \cdot |w_{9,6}|$, since $|w_{5,1}|$ and $|w_{9,5}|$ are smaller than $|w_{6,1}|$ and $|w_{9,6}|$. The tuning index of substance A with respect to flux F_1 is the sum $|w_{5,1}| \cdot |w_{9,5}| + |w_{6,1}| \cdot |w_{9,6}| + |w_{7,1}| \cdot |w_{9,7}|$. On the right side of the same picture it is showed that the contribution of substance B in tuning flux F_1 is almost insignificant, since the absolute values of all the weights on the paths between the input neuron u_2 (connected to B) and the output neuron u_9 have small or medium sizes. Accordingly, the tuning index of substance A will be greater than the tuning index of substance B.

A case study: the Sirius model

In this section we report some preliminary results of the application of the tuners discovery strategy explained above to a simple case study. The MP system we investigate, called Sirius, has been already defined in Section 4.6 where it has been employed to explain the log-gain principle. Here, we report an MPF version of this model, which involves the following flux regulation functions:

$$\begin{aligned} F1 &= \frac{k_1 a}{k_1 + k_2 c + k_4 b + k_a} \\ F2 &= \frac{k_2 ac}{k_1 + k_2 c + k_4 b + k_a} \\ F3 &= \frac{k_3 b}{k_3 + k_b} \\ F4 &= \frac{k_4 ab}{k_1 + k_2 c + k_4 b + k_a} \\ F5 &= \frac{k_5 c}{k_5 + k_c} \end{aligned} \tag{5.40}$$

where $k_1 = k_3 = k_5 = 4$, $k_2 = k_4 = 0.02$, and $k_a = k_b = k_c = 100$. Notice that, functions F_1, F_2 and F_4 have the same denominator but the numerator of F_1 is characterized by the tuner A, numerator of F_2 by the tuners A and C, and numerator of F_4 is characterized by the tuners A and B. On the other side, functions F_3 and F_5 are characterized, respectively, by the tuners B and C. The oscillatory dynamics generated by these functions, displayed in Figure 5.20, is featured by a very similar trend for substances B and C, which differ only in the first fifty steps.

We have sampled the dynamics of Figure 5.20 in order to obtain three substance time series (one for each substance), each having 1000 values, and we have computed the related five flux time series (one for each flux) by the log-gain theory. Subsequently, these time series have been employed to train five neural networks (one for each regulation function) by means of backpropagation with weight elimination. Specifically, substance values have been used as inputs and flux values as target outputs during the training process performed by the software NeuralSynth. We run the computation of the tuning indexes of each flux for five times and, subsequently, we have calculated the mean and the standard deviations of these indexes for each flux regulation function. The best results, reported in Table 5.2, have been achieved by employing $\lambda = 0.0001$ and $w_0 = 1.0$ for weight elimination and neural networks having one hidden layer with three neurons. This value of parameter w_0 tends to eliminate weights between (about) -5.0 and 5.0, which is consistent with the random initialization of neural network weights between -1.0 and 1.0. The parameter λ has been manually tuned for this case study but some heuristics [245] will be considered to dynamically tune its value during the training process. The network topology has been adapted to the complexity of the searched regulation function.

Let us analyze the results of Table 5.2. The first row reports the mean relative tuning indexes of flux F_1 and, in brackets, the standard deviation of the relative tuning indexes over the five tests performed. Value 0.918 in the first column, states

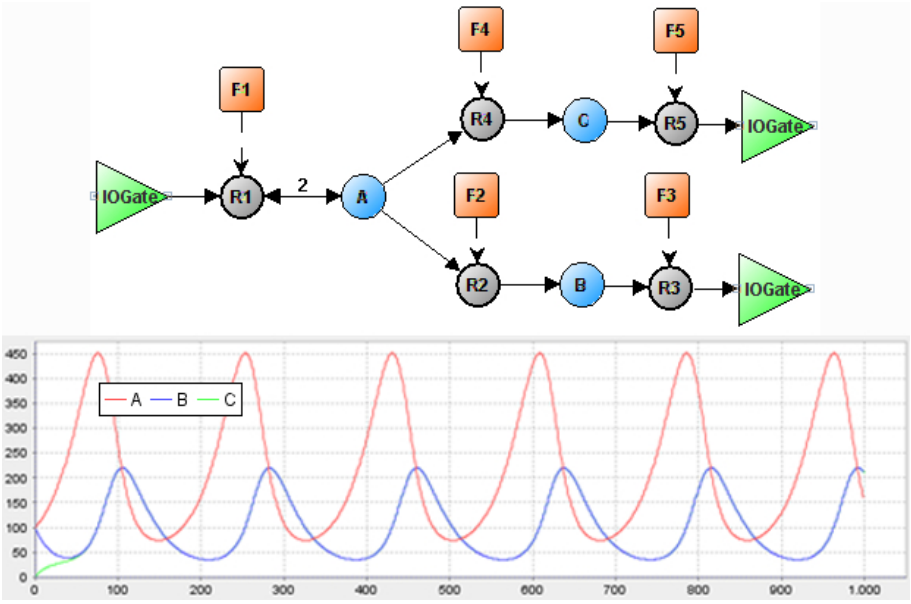


Fig. 5.20. On top: Sirius model. At the bottom: Sirius dynamics

	A	B	C
$\mathbf{F_1}$	0.918 (0.044)	0.043 (0.026)	0.038 (0.017)
$\mathbf{F_2}$	0.336 (0.001)	0.301 (0.209)	0.362 (0.209)
$\mathbf{F_3}$	0.018 (0.017)	0.971 (0.020)	0.010 (0.009)
$\mathbf{F_4}$	0.337 (0.006)	0.525 (0.292)	0.136 (0.292)
$\mathbf{F_5}$	0.020 (0.027)	0.084 (0.112)	0.895 (0.111)

Table 5.2. Mean tuning indexes and related standard deviations (in brackets) of substances A, B and C with respect to fluxes F_1, F_2, F_3, F_4, F_5. These results have been computed by performing five tests for each flux.

that substance A have obtained a mean tuning index of 91.8% for flux F_1 over the five tests. Substances B and C, respectively in the second and third columns, have achieved mean tuning indexes of 4.3% and 3.8%. This result completely agrees with the form of function F_1, by which dynamics data have been generated, indeed function F_1 is deeply related with substance A, which appears in the numerator of this function. By analyzing the third row of Table 5.2, related to flux F_3, we observe that substance B, which appears in the numerator of function F_3, has achieved a mean tuning index of 97.1%, while substances A and C, which are not arguments of function F_3, have scored only 1.8% and 1.0%. Quite good results have been achieved for flux F_4 (in the forth row), indeed the variables appearing in its numerator, namely A and B, have scored mean tuning indexes of, respectively, 33.7% and 52.5% in contrast to the 13.6% scored by substance C. Flux F_5, in the last row of the table, has mean tuning indexes of 2.0% for A, 8.4% for B and 89.5% for C, according to the form of function F_5 which includes only substance C amomg its arguments. Instead, the result related to flux F_2 (in the second row)

deserves further investigations, since the mean tuning indexes turned out to be not informative enough. Indeed, they are 33.6 for A, 30.1 for B and 36.2 for C, and the values are so close to each other that we cannot deduce A and C to be the only tuners for F_2 (as it clearly appears in the numerator of function F_2). We believe that this problem can be due to the high similarity between the dynamics of substance B and C, which makes it difficult to distinguish between the two inputs. This seems to be confirmed also by the high standard deviation values achieved for substances B and C for both fluxes F_2 and F_4, which points out a large variance in the relative tuning indexes computed over the five tests. The dynamics trend of the model obtained by this approach, which is displayed in [33], is very similar to the original one, showed in Figure 5.20.

5.6 A pipeline for statistical data analysis and modeling: from laboratory to MP models

In this section we introduce a methodological pipeline, of five steps, which outlines the entire process of flux regulation map synthesis from raw experimental data, given as time series of observed substance concentrations and parameter values [32].

The first two steps concern, respectively, the pre-processing (preparation) of raw experimental data and the computation of flux time series from pre-processed data. Afterwards, substance, parameter and flux data are analyzed by means of some statistical techniques aiming to discover reciprocal relationships, while two regression techniques are suggested for synthesizing regulation maps, namely, stepwise regression and neural networks. These methodologies are finally tested and compared by means of suitable statistical indexes [1]. In the following, we describe, one by one, each of the five steps introduced in [32], and graphically described in Figure 5.21.

1. Initially, it is necessary a phase of *data preparation* and *preprocessing* [188]. It involves the elimination of both noise and artifacts from experimental data. Statistical tools are here employed for removing outliers and instrumental noise from raw data, moreover, correlations among all substances and parameters are computed.
2. Log-gain theory is here employed to infer flux time series from the preprocessed time series of substances and parameters.
3. Flux time series from the previous step are analyzed by statistical tools, such as scatter plots, correlation analysis, and t-tests, in order to discover their relationships with the preprocessed time series of substances and parameters. These tests are introductory to a phase of *feature selection* [89] in which a proper subset of substances and parameters is identified as for the choice of variables of each flux regulation map.
4. This step concerns the synthesis of flux regulation maps by means of mathematical regression. Two main techniques from literature are considered, *(linear) stepwise regression* [1, 94] and *(non-linear) neural networks* [24, 33], and some hints are given about the choice of the appropriate technique for specific case studies.

5. As a final step, the validity and the performance of each flux regulation map is evaluated by means of statistical tools, such as, coefficient of determination R^2 and residual analysis [1]. These tests help us to see if an appropriate regression model has been applied, and if a specific technique (between stepwise regression and neural networks) achieves better performance for the specific system under investigation.

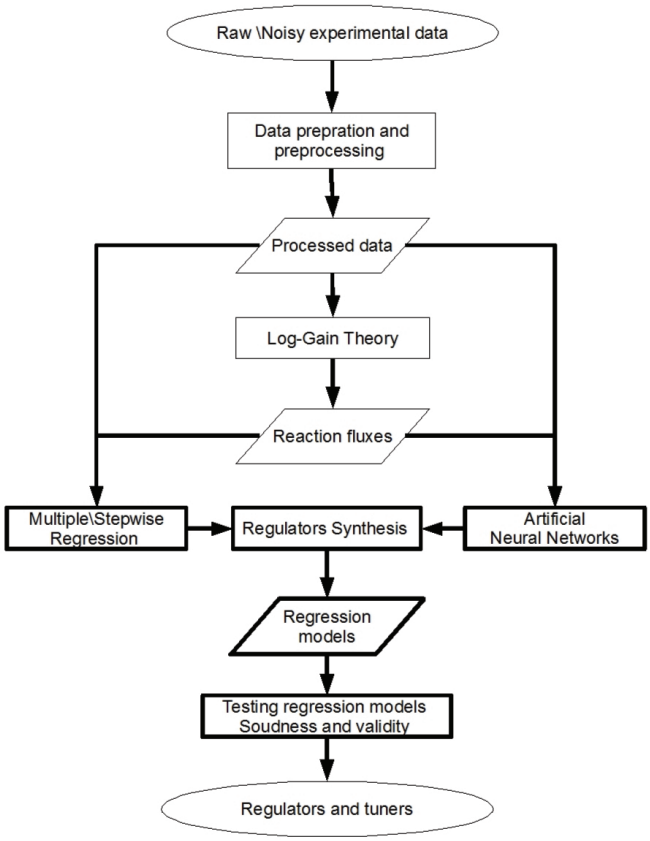


Fig. 5.21. Main steps of data analysis pipeline, where flux regulation maps are shortly called *regulators*, their variables *tuners*, and the areas in bold include the contribution of this thesis (figure from [32] with permission).

We have applied this workflow to two case studies: the mitotic cycle in early amphibian embryos [85] here reported very briefly, and the non photochemical quenching phenomenon (NPQ) [148], which is analyzed in more detail as an example of pipeline application. Namely, both stepwise regression and neural networks have been employed, and their performance has been compared by means of statistical indexes. The results discussed in the final part of the chapter show

new interesting developments in the framework of data analysis for modeling by metabolic P systems.

5.6.1 MP model of the NPQ phenomenon

Let us briefly describe the photosynthetic process we employ as a case study for our pipeline [32]. It is an interesting biological system, which received wide investigation by biologists but was never modeled by conventional mathematical techniques, such as differential equations. It has been successfully modeled by an MP system in [148], where the reader may find even references about the biological studies of this system, and where a few open modeling problems were addressed in the last sections. In the following we improve the regulative part of that model, based on the rules reported in Table 5.3 (describing the dynamics we are going to explain in biological terms), and solve those open problems, to systematically find (simpler) flux regulation maps from flux time series together with their relevant system variables (sections 5.6.2 and 5.6.2).

Photosynthetic organisms need to maximize the amount of absorbed light but avoid the damages that follow from an excess of excitation energy, which is the main cause for the formation of reactive oxygen species, shortly ROS[1]. The phenomenon that helps to deal with quick light excess is called *non-photochemical quenching*, shortly NPQ. Through this phenomenon the excess of light can be dissipated by using non-chemical ways, when the excitation is transmitted to particular molecules that can pass to their unexcited state by emitting heat.

$r_1 : c \to o + 12h + p$
$r_2 : c \to c + q^+$
$r_3 : c \to c + f^+$
$r_4 : o \to c$
$r_5 : h \to \lambda$
$r_6 : p \to \lambda$
$r_7 : x + 100v \to x + 100z$
$r_8 : y + h \to x$

Table 5.3. NPQ reactions, according to the following abbreviations: c = closed photosytems, o = open photosytems, h = hydrogen ions, p = NADPH, q^+ = cumulative heat, f^+ = cumulative fluorescence, x = active VDE, y = inactive VDE, v = violaxanthin, z = zeaxanthin. The first three reactions model the possible fates of excited chlorophylls, the fourth models the turning of the opened into closed photosystems. The fifth and sixth rules represent the decreasing of unused r_1-products, leading to the dark phase of photosynthesis. Finally, the seventh and eighth rules represent xanthophylls cycle. f^+ and q^+ are substances introduced to apply the log-gain method, that account for the cumulation of fluorescence and heat. Four parameters (which one may see in Figure 5.22) are r = reactivity, l = light (both effecting the process), f = fluorescence, q = heat (output of the system) [148].

[1] ROS are chemical species producing a dangerous effect known as photooxidative damage.

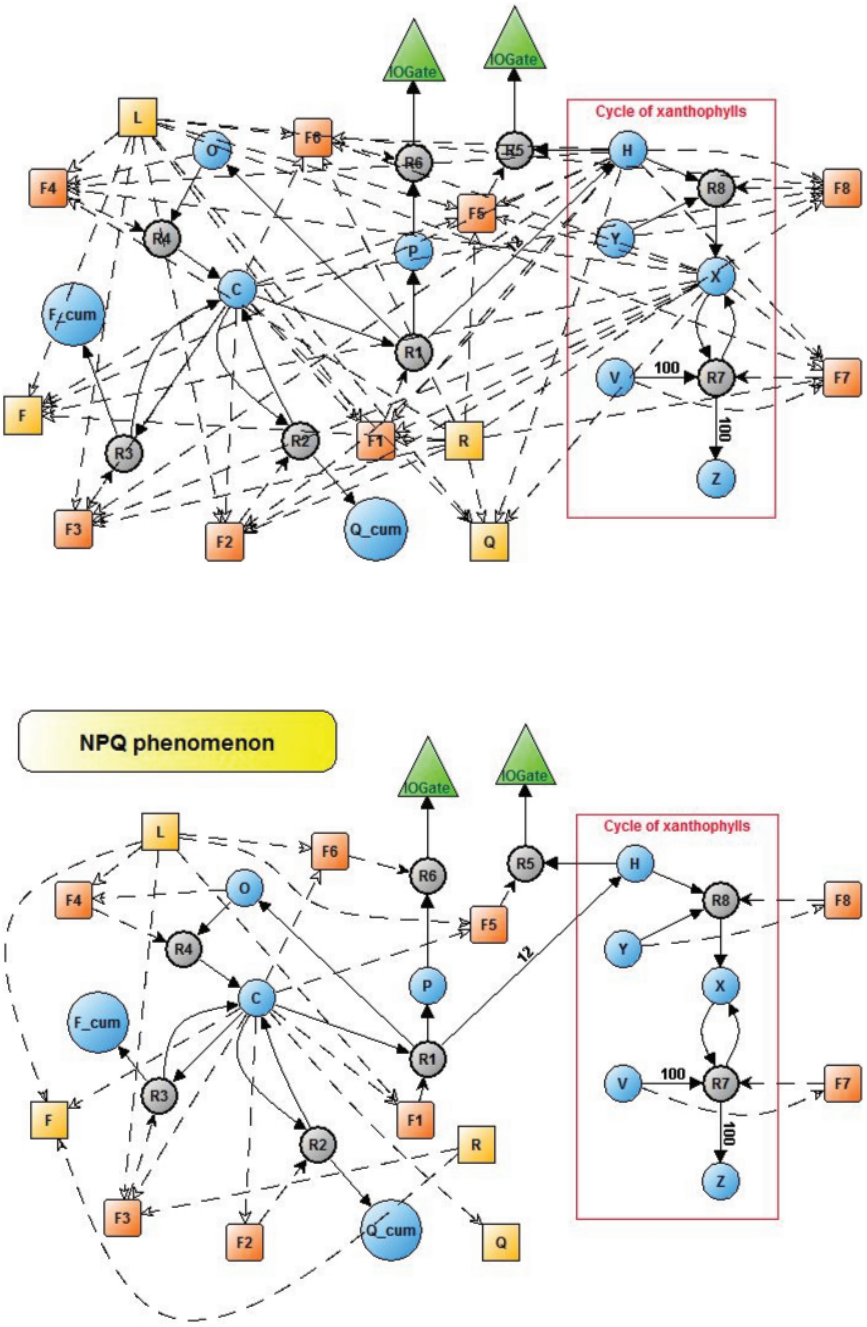


Fig. 5.22. MP graphs of NPQ phenomenon visualized by a graphical user interface of MetaPlab. On top: the NPQ model proposed in [148] - At the bottom: the simpler model obtained as an output of our data analysis workflow (figures from [32] with permission).

According to this phenomenon, light energy is absorbed by plants mainly by means of protein complexes called Light Harvesting Complexes (LHC), which bind many chlorophyll molecules (Chl), that are excited by light radiation. LHC are connected to the photosystems, protein super-complexes that host structures called reaction centers where the first phases of photosynthetic process occur. Structurally, photosystem super-complexes include both reaction centers and LHC, and may assume either an *open* or a *closed* state, respectively when they are able to accept further photons or not. When a chlorophyll molecule absorbs a photon, it passes to the excited state. Excited states may be transferred to the reaction centers where the oxidation of two water molecules produces a stoichiometric amount of electrons, oxygen and hydrogen ions. Electrons are then carried to enzymes which synthesize high energy molecules, like ATP and NADPH, involved in the so-called dark phase of photosynthesis. Moreover, excited chlorophyll molecules can be de-excited by passing energy to molecules that emit the heat resulting in NPQ phenomenon, as well as fluorescence radiation, or can decade in the triplet state, that can be transferred to oxygen atoms, thus generating ROS.

LHC and chlorophyll molecules bind other molecules, called *caroteoinds*, which absorb energy from excited chlorophyll molecules and dissipate it by heat generation. Two caroteoinds of great interest in NPQ phenomenon are *violaxanthin* and *zeaxanthin*. When the absorption of solar radiation exceeds the capacity of the organism to use it, an increase of hydrogen ions provides a signal of over-excitation that triggers a regulative feed-back process. The Violaxanthin De-Epoxidase (VDE), once activated by hydrogen ions, catalyzes the *cycle of xanthophylls*, which transforms violaxanthin to zeaxanthin. Zeaxanthin bound to LCH favors the NPQ fluorescence and heat production [3].

5.6.2 A pipeline to synthesize flux regulation maps from observed data

In this subsection we describe in detail the single steps of our workflow, by showing how we applied them to model NPQ by an MP system, starting from raw data [32]. The data required are typically time series on the response of a biochemical system to different conditions and stimuli, as its behavior in response to small perturbations can be used to determine the properties of the system near to a steady state [46]. In our case, experimental measurements on Arabidopsis thaliana wild type plants, complemented by literature data, made it possible to estimate the concentrations of the species involved in modeling the NPQ phenomenon. In particular, it was possible to measure the fraction of closed (c) and opened photosystems (o). The presence of many closed photosystems induces the inability of canalizing further amount of energy through the photochemical way: this is the ideal situation to measure the ability of the NPQ phenomenon. To induce such a condition strong light flashes were used. With closed photosystems, the reduction of the fluorescence (f) yield and the efficiency of non photochemical quenching were measured. The rate of fixation of CO_2 during a suitable (to get measurements) condition of NPQ gave an index for the reactivity (r) of the system, which is strictly connected to the capacity to reach equilibrium after light energy absorption. The fluorescence (f) and heat (q) values were deduced from measurements on the sample [155]. Produced NADPH (p) was estimated through

laboratory measures, while pH value was deduced by combining data from literature [61, 111, 229] and applying the rate of change to the estimated pH values during the NPQ measure. VDE state (x, y) was set in relation to various pH values [83]. The change over time of violaxanthin (v) and zeaxanthin (z) was obtained with lab measurements during VDE activity.

Data preparation

Let us consider a set of experimental data obtained by sampling (possibly at a constant rate τ) substance concentrations and chemo-physical parameter values of a certain biochemical system. The analysis of experimental time series is necessarily complicated by uncertainty due to measurement errors, natural fluctuations, noise, unexpected external variations effecting the experiment, and missing data [46].

To remove artifacts from substance and parameter time series, we consider curve fitting, which is a simple technique, often employed by biologists to find a smooth curve which fits noisy data by reducing their random component while preserving the main trend of the dynamics under investigation. Of course, if data are affected by other kinds of errors regarding, for instance, consistency, integrity, or outliers, then ad hoc techniques must be used [188], but it is out of the scope of this work to consider particular methods to process raw data. After such a preprocessing of experimental data, we assume that fluctuations and measurement errors are normally distributed around the average trend of the system dynamics, therefore each observed substance and parameter time series is fitted by a smooth function using least square theory (as reported in Figure 5.23).

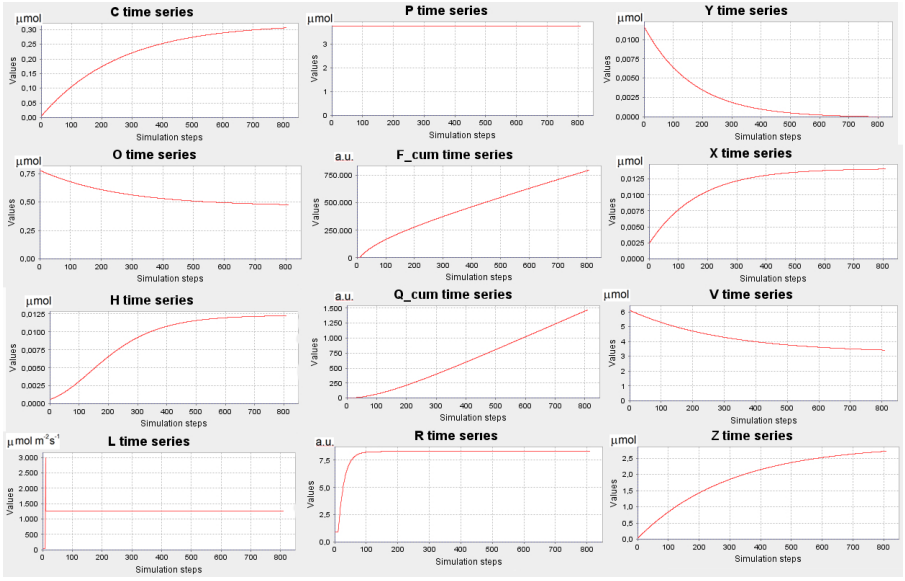


Fig. 5.23. Pictures reporting the pre-processed laboratory data of all substances and parameters in NPQ phenomenon (see caption of Table 5.3). Figure from [32] with permission.

The analysis of *Pearson's correlations* (PC, also called *estimator* R) gives a measure of the degree of linear relationship between two random variables $x(t)$ and $y(t)$, $t = 0, \ldots, n$. It is defined as

$$R(x, y) = \frac{\sum_{t=0}^{n}(x(t) - \overline{x})(y(t) - \overline{y})}{(n-1)s_x s_y}, \tag{5.41}$$

where $\overline{x}$, s_x and $\overline{y}$, s_y are sample means and standard deviations, respectively, of variables $x(t)$ and $y(t)$ over the n measurements. Correlation values close to 1 indicate positive linear relationships between substance/parameter monomial x and y, correlations equal to zero indicate no linear associations, while correlations near to -1 indicate negative linear relationships.

Correlation indexes are computed for each couple of elements out of the set of substances and parameters $X \cup V$. The consequent *correlation matrix*

$$\mathcal{C} = \{c_{x,y}\}_{x,y \in X \cup V}, \tag{5.42}$$

turns out helpful to select subsets of variables relevant for flux regulation maps (see the third step of this pipeline). Indeed, regression techniques are assumed to be applied to variables with a mutual low correlation, and for the success (and the significance) of the model it is important paying attention not to select variables which are highly correlated with each other. In the case of NPQ phenomenon, such a matrix is reported in Table 5.4, where f and q are not considered because their significance is to account for the system output while parameter $l \cdot r^{-1}$ was suggested by biological knowledge of the system [148].

	c	v	h	p	x	o	z	y	l	r	$\frac{l}{r}$
c	1.00	-1.00	0.99	0.00	0.99	-1.00	1.00	-0.99	0.23	0.58	-0.21
v	-1.00	1.00	-0.99	0.00	-0.98	1.00	-1.00	0.98	-0.22	-0.55	0.20
h	0.99	-0.99	1.00	0.00	0.98	-0.99	0.99	-0.98	0.20	0.52	-0.20
p	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
x	0.99	-0.98	0.98	0.00	1.00	-0.99	0.98	-1.00	0.27	0.66	-0.25
o	-1.00	1.00	-0.99	0.00	-0.99	1.00	-1.00	0.99	-0.23	-0.58	0.21
z	1.00	-1.00	0.99	0.00	0.98	-1.00	1.00	-0.98	0.22	0.55	-0.20
y	-0.99	0.98	-0.98	0.00	-1.00	0.99	-0.98	1.00	-0.27	-0.66	0.25
l	0.23	-0.22	0.20	0.00	0.27	-0.23	0.22	-0.27	1.00	0.53	0.45
r	0.58	-0.55	0.52	0.00	0.66	-0.58	0.55	-0.66	0.53	1.00	-0.41
$\frac{l}{r}$	-0.21	0.21	-0.20	0.00	-0.25	0.21	-0.21	0.25	0.45	-0.41	1.00

Table 5.4. NPQ substances and parameters correlations [32].

Flux discovery by Log-gain theory

Log-gain theory, described in [140–142] and already presented in Section 4.6, allows to infer a set of flux time series which yields an observed dynamics. Indeed, according to the mass partition principle, during one time interval between two

observations, each reaction transforms a certain amount of reactants into products [140], and these quantities (fluxes) may be computed, with a good approximation, for each step of the observed dynamics, by solving system OLG[i]+SD[i+1] (see Section 4.6).

At each step, this system, having a time constant block matrix of dimension equal to the number of rules (and fluxes), turns out to be univocally solvable [72]. Hence, MP fluxes may be efficiently computed from the states, given by observed time series, once flux values at an initial observational step is computed (namely by means of the algorithm reported in [168]).

One could think of computing flux time series directly from noisy data (thus keeping all the information contained in the laboratory dataset, but also propagating the noise to the fluxes), but in this case a fitting of noisy fluxes by a smooth function should be followed by a synthetic production of data by means of MP dynamics employing the resulting fluxes (because we need a dynamical correspondence between states and fluxes), with an extra computational cost.

Flux analysis and variable selection

Our aim at this point is to identify functional relationships between each flux time series and every substance/parameter time series, in order to select a set of significant variables for regulation maps. Several techniques for variable selection have been already presented in Section 5.5. We have implemented one filter tool based on correlation measurements and two embedded methods, one for stepwise regression and another for neural network regression. The filter tool has been implemented in a Matlab® function, which performs a statistical analysis of each flux time series (obtained as described at the previous step) along the following three points:

1 Generation of a *scatter plot*, which is a sketch of data on two variables [1], of each substance/parameter time series (or possibly their products up to a specific degree) and the flux time series.

By plotting an observed dynamics in such a chart the general trend or some simple relationships between variables can be caught. In Figure 5.24 two scatter plots regarding NPQ phenomenon are showed, where we may visualize an "almost linear" dependence of flux u_2 on its rule reactant c, which is not the case for the reactivity substance r.

2 Computation of *correlation* (together with *partial correlation*) index between each substance/parameter time series (and their products up to a specific degree) and the flux time series.

Namely, given a flux u, correlation index $R(z_j, u)$ are computed, for each z_j ranging in $X \cup V$ (according to the formula given at the first step), where fluxes u and state components z_j are considered random variables, whose measured values are denoted, respectively, as $z_j(t)$ and $u(t)$, for $t = 0, \ldots, n$.

Unfortunately, correlation cannot always distinguish between direct interactions (e.g., substances or parameters which directly regulate a flux) and indirect interactions (e.g., substances or parameters which regulate a flux by means of an

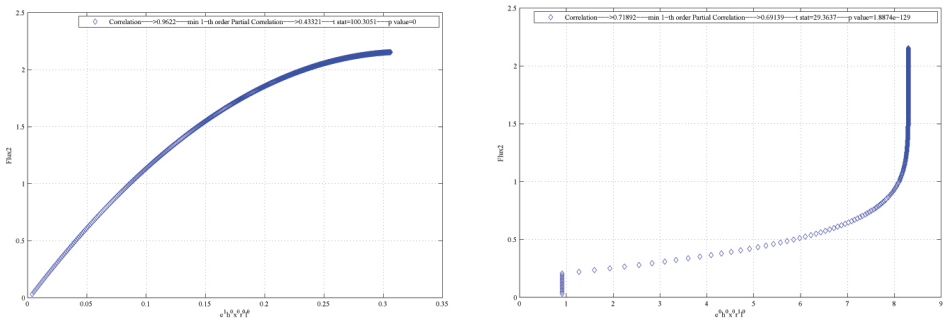


Fig. 5.24. Scatter plots produced by our Matlab® function, where flux u_2 is plotted with c (left) and r (right). Figures from [32] with permission.

intermediate substance/parameter), thus our analysis has been improved by computing also partial correlations. The formula for computing the *first order partial correlation* between two random variables $x(t)$ and $y(t)$, $t = 0, \ldots, n$ is the following [213]:

$$R_{C_1}(x, y) = min_{z \neq x,y} |R(x, y|z)|, \tag{5.43}$$

where

$$R(x, y|z) = \frac{R(x, y) - R(x, z)R(y, z)}{\sqrt{(1 - R^2(x, z))(1 - R^2(y, z))}}. \tag{5.44}$$

If the correlation between a substance and a flux is high but the first order partial correlation between the same variables is close to zero, then the substance does not directly regulate the flux but it probably influences an intermediate substance which regulates the flux.

3 Application of *t-test* to evaluate, with a certain confidence level, when to drop a variable from a regulation map.

Our Matlab® function employs the *t-test* to decide whether a substance/parameter z_i is correlated to the flux u. Intuitively, given a specific pair of variables x, y, measured along n steps, the t-test answers to the question: "is it possible to say, with a certain confidence, that x and y are in some way correlated?" In statistical terms, this test is based on the following hypothesis test:

$$H_0 : \rho(x, y) = 0$$
$$H_1 : \rho(x, y) \neq 0$$

where H_0 is the so called *null hypothesis* and $\rho(x, y)$ is the real (unknown) correlation coefficient, which $R(x, y)$ estimates by using the set of n observations available for x and y. If, given a predefined confidence threshold $\alpha \in [0, 1]$ (a common value for α is 0.025), the test concludes that there is sufficient evidence against the null hypothesis H_0, than we can reject it with a probability less than α to make a mistake. The test statistic for determining the rejection or nonrejection of the null hypothesis H_0 is the following:

$$t_{(n-2)} = \frac{R(x, y)}{\sqrt{(1 - R(x, y)^2)/(n - 2)}}, \tag{5.45}$$

where $t_{(n-2)}$ refers to a t distribution with $n-2$ degrees of freedom. The null hypothesis should be rejected if the value of $t_{(n-2)}$ is greater than the critical point for a t distribution with a confidence level α (this values are defined in specific tables or they can be computed by means of the *p-value*). A more detailed description of these concepts can be found in [1]. In our case study, in which the t-test is performed between a substance/parameter z_i and a flux u, if the null hypothesis is not rejected, than we can conclude with a certain confidence level that no linear correlation exists between the two variables, therefore we can drop variable z_i from the regulation function of flux u.

In conclusion, when our Matlab® function is launched on a specific dataset of substance, parameter and flux time series, it produces a statistical analysis about linear relationships between substance/parameter time series (and their products up to a specific degree) and flux time series. On the basis of the information reported in Table 5.4, a subset $\mathcal{N}$ may be extracted of variables carrying non-redundant information about the regulative mechanisms. For each reaction, one considers its set of reactants and the set of substances/parameters which are not correlated to any reactant. The last one is processed by non-deterministically choosing one out of the sets of elements mutually correlated, and then united to the set of reactants to form $\mathcal{N}$. In case of NPQ, besides this procedure, we extracted sets $\mathcal{N}_1$ (reported, correspondingly to each rule, in second column of Table 5.5), also by deleting p from the variables (because it has an observed constant value), and by considering some biological constraints [148], according to which all the rules are regulated by l and r but r_3, which is regulated by parameter $l \cdot r^{-1}$. As a result, for each reaction, we have a reduced set of (possibly regulative) variables (which gives new biological information beside reactants), and the correlation itself is a measure of importance of each variable in the reduced set.

Reactions	$\mathcal{N}_1$	$\mathcal{N}_2$
$r_1 : c \to o + 12h + p$	c, h, x, r, l	c, h, l
$r_2 : c \to c + q^+$	c, h, x, r, l	c, h, x
$r_3 : c \to c + f^+$	$c, h, x, l \cdot r^{-1}$	$c, h, x, l \cdot r^{-1}$
$r_4 : o \to c$	o, h, x, r, l	o, h, l
$r_5 : h \to \lambda$	c, h, x, r, l	c, h, l
$r_6 : p \to \lambda$	c, h, x, r, l	c, h, l
$r_7 : x + 100v \to x + 100z$	v, h, x, r, l	v, h
$r_8 : y + h \to x$	c, h, y, r, l	c, h, y

Table 5.5. NPQ reactions, equipped with a set $\mathcal{N}_1$ of corresponding regulative non-correlated variables (tuners), and a set $\mathcal{N}_2$ of variables selected from $\mathcal{N}_1$ by correlation and scatter plot analyses with corresponding fluxes [32]. $\mathcal{N}_2$ has been employed as set of input variables for stepwise regression generating the regulators in Table 5.8, which give the reduced MP graph in Figure 5.22.

Regression analysis

Building a model of a flux regulation map given a dataset of system states (time series of variables) and fluxes (time series of map values), entails the employment

of regression analysis. In Sections 5.2 and 5.3 we have proposed two regression techniques, namely, linear regression, which uses polynomial functions, and neural network regression, employing neural networks to represent regulation maps. The reader may refer to those sections for a detailed description of both the techniques.

Multiple regression. Three main experiments were performed by applying stepwise regression to estimate the MP regulators of the rules reported in Table 5.3 modeling the NPQ phenomenon. In a first experiment, multiple stepwise regression has been applied starting from sets $\mathcal{N}_1$ of tuners in Table 5.5, and gave the successful results reported in Table 5.6. In a second experiment stepwise regression was applied by enforcing the presence of reactants among the tuners, and the regulators reproducing the correct dynamics are reported in Table 5.7. Finally, a parsimonious model has been found as the result of a third experiment, where stepwise regression was applied to variables from $\mathcal{N}_2$ selected by correlation and scatter plot analyses.

In some cases (such as, mitotic oscillator discussed in the final part of this chapter), multiple regression models tend to assume high degrees and to employ many variables, meaning the regulators have a high level of complexity, and the employed regression method to estimate them is not appropriate [192]. In these situations, artificial neural networks (ANNs) may be convenient, because of their modular and parsimonious structure, which allows us to obtain accurate approximations of any function [24].

Artificial neural networks. We have performed several experiments, by different network topologies, sets of input variables, and learning techniques. Let us here notice an important difference between stepwise and neural networks regression methods: the first one transforms the set of input variables into a set of output variables (which are tuners for the estimated regulators, as in Tables 5.6, 5.7, 5.8, while the second does not. Then, the choice of input variables for ANNs actually gives a choice of tuners for the associated MP model. We have trained networks with one hidden layer having three, five or seven hidden neurons. Three sets of variables have been processed by each network: *i)* that of tuners selected in the original MP model of NPQ [148], *ii)* the set $\mathcal{N}_1$ reported in Table 5.5, and *iii)* the set $\mathcal{N}_5$ of tuners selected by the stepwise regression technique as in Table 5.8. The cardinality of these sets decreases from the first to the third case. In very few tests we obtained networks able both to fit the observed data and to reproduce the observed dynamics, while best results have been achieved by improving the method along with generalization and normalization, at least for data on the parameter l representing light.

To better understand the cause of such a kind of inadequacy, an analysis has been performed on the initial dataset. We firstly observed that a normalization is needed in order to feed each input neuron with data belonging to a certain interval of values. Indeed, observed NPQ dynamics have very different scales: light (l), for instance, fluctuates between 40 and 3000 $\mu mol\ m^{-2}\ s^{-1}$, while hydrogen ions (h) oscillates between $6 \cdot 10^{-4}$ and $122 \cdot 10^{-4}$ μmol [148]. The normalization module we added to the NeuralSynth plugin linearly rescales each substance and parameter dataset to the interval $[0, 1]$, in order to give to every input the same relative

importance in determining the required outputs [24]. Another issue concerns the generalization, which is a bipartition of the dataset in a training and a validation set. Better performance is usually obtained by training the networks on a subset of randomly chosen data (training set) and, subsequently, validating them on the remaining subset of data (validation set). Our dataset has, however, a particular evolution, since at the tenth step a peak of light is observed which affects the subsequent dynamics of the system. For this reason, in the training set (usually taken as half of the dataset) we put *i)* some observations of the dynamics before the light peak, *ii)* some observations of the dynamics after the light peak and *iii)* the observation gathered during the peak, otherwise the network hardly learns how to behave in all these three conditions. Accordingly, we generate our training set by enforcing the first 15 steps of the observed dynamics, and then randomly choosing a certain number of observations. The validation set has been generated by merging the first 15 steps of the observed dynamics (again) with the observations which have not been inserted in the training set.

Testing regression model soundness and validity

Once a regression model has been generated, its validity and its performances have to be tested, in order to understand if the values it predicts are reliable and if it is able to generalize on new data. There exist several techniques and statistical indexes aiming to score regression models, such as *root mean square error* (shortly RMSE) and the *adjusted coefficient of determination* $\bar{R}^2$. The first is the average distance (i.e., error) between the observed data points y_j and the related regression estimations $\hat{y}_j$, and it is defined as

$$RMSE = \sqrt{\frac{\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}{n-(p+1)}}, \tag{5.46}$$

where n is the number of observations and p is the number of basis functions. A high RMSE indicates that the regression model produces wrong estimations, while values of RMSE close to zero are usually associated to good models.

This index, however, depends on the magnitude of data, while we prefer relative measures of variation degree of the data on the regression hyperplane, that is an helpful information to compare different models [1]. The gap between observed data $y_j, j = 1, \ldots, n$ and their average value $\bar{y}$ is usually assumed to be given by the sum of an *unexplained deviation* $y_j - \hat{y}_j$ (i.e., error) and a *deviation explained* by the variation of the independent variables $\hat{y} - \bar{y}$ (see Figure 5.25). The percentage of total variation explained by the regression model is measured by the *coefficient of determination* R^2, as in the following:

$$R^2 = \frac{\sum_{j=1}^{n}(\hat{y}_j - \bar{y})^2}{\sum_{j=1}^{n}(y_j - \bar{y})^2} = 1 - \frac{\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}{\sum_{j=1}^{n}(y_j - \bar{y})^2}. \tag{5.47}$$

An improved version of R^2 which accounts also for degrees of freedom of the regression model, and therefore it does not always increase as new basis functions are added, is the *adjusted coefficient of determination*:

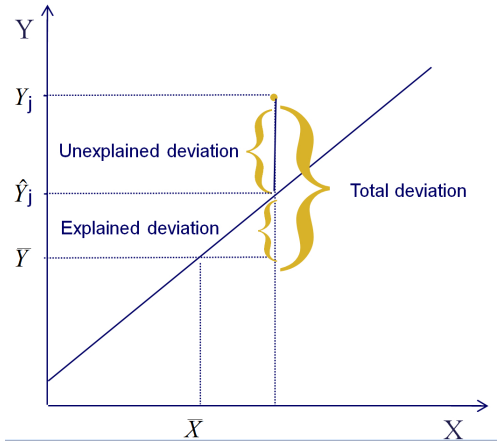


Fig. 5.25. Unexplained deviation and explained deviation in the computation of the coefficient of determination R^2 [1].

$$\bar{R}^2 = 1 - \frac{\sum_{j=1}^{n}(y_j - \hat{y}_j)^2(n-1)}{\sum_{j=1}^{n}(y_j - \bar{y})^2[n-(p+1)]}. \tag{5.48}$$

This coefficient represents a reliable index of goodness for regression models. Its value is 1 if the regression hyperplane perfectly fits all the data, while it decreases to 0 as the variation of the data unexplained by the model increases.

Estimation errors $y_j - \hat{y}, j = 1, \ldots, n$, also called *residuals*, are the most important elements to analyze for discovering model biases. Residual analysis involves also *residual plots*. We may plot residuals versus predicted values of dependent variables, versus each independent variable, versus time, an so on, in order to point out if either the error is constant or has some pattern. If any patter is revealed, then the model performance to compute the regression function was not good enough. In Figure 5.26 we report residual plots of stepwise regression models (left-hand side) and neural models (right-hand side) for functions F_3 (on the top) and F_8 (on the bottom) of NPQ model.

5.6.3 Experimental results

Table 5.6 presents the regulators obtained by applying stepwise procedure to sets $\mathcal{N}_1$ reported in Table 5.5, beside the statistical tests associated with the regression model. These functions reproduce quite accurately experimental results, while statistical indexes point out that stepwise regression has learned the regulators with good precision. In all the following experimental results, the regulator φ_3 has residuals higher than the other regulators, but they are relatively low considering that its range reaches values around 3000. Also, particular attention has to be paid to regulators φ_2 and φ_3, which state, respectively, the rate of heat (q) and the amount of fluorescence (f) at each evolution step. These parameters are not input of the system, but are important because they account (by means of MP modeling) for information about non-chemical ways plants have to dissipate excessive energy produced by light.

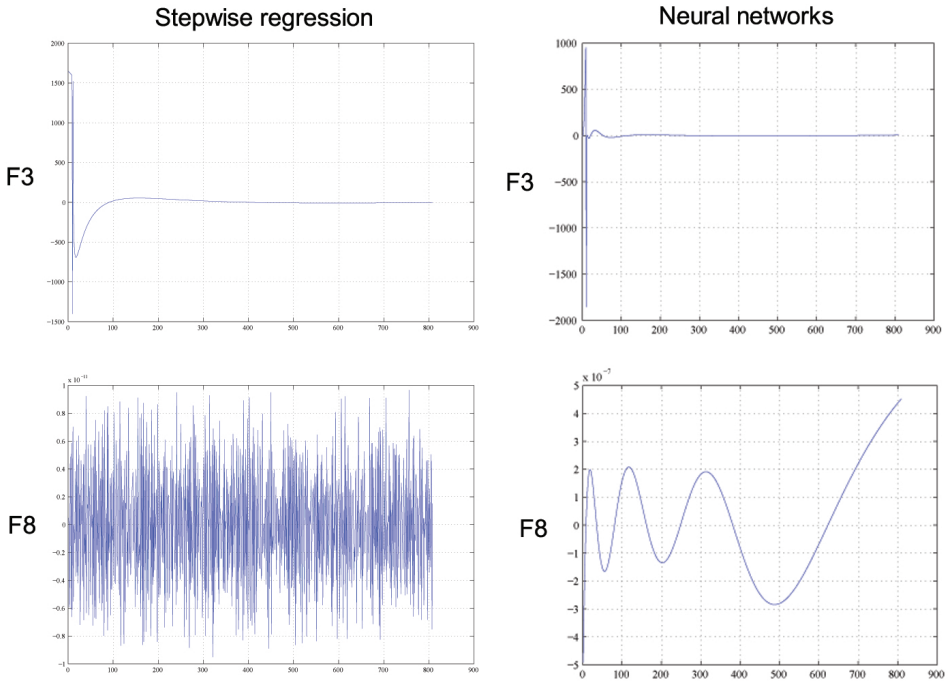


Fig. 5.26. Residual plots of φ_3 and φ_8 in the more parsimonious model we have found ($\mathcal{N}_5$), by stepwise regression (left) and by neural networks (right). Figures from [32] with permission.

	Regulators	$\mathcal{N}_3$ (Tuners set)	$\overline{R}^2$	RMSE
φ_1	$\alpha_1 - \beta_1 c + \gamma_1 h - \eta_1 x + \vartheta_1 r + \rho_1 l$	c, h, x, r, l	0.99997	1.4998e-04
φ_2	$-\alpha_2 - \beta_2 c - \gamma_2 h + \eta_2 x + \vartheta_2 l$	c, h, x, l	0.99999	1.3811e-03
φ_3	$\alpha_3 - \beta_3 x + \gamma_3 l \cdot r^{-1}$	$x, l \cdot r^{-1}$	0.65384	2.1567e+02
φ_4	$-\alpha_4 + \beta_4 o + \gamma_4 h - \eta_4 x + \vartheta_4 r + \rho_4 l$	o, h, x, r, l	0.99997	1.4998e-04
φ_5	$\alpha_5 - \beta_5 c + \gamma_5 h - \eta_5 x + \vartheta_5 r + \rho_5 l$	c, h, x, r, l	0.99997	1.8000e-03
φ_6	$\alpha_6 - \beta_6 c + \gamma_6 h - \eta_6 x + \vartheta_6 r + \rho_6 l$	c, h, x, r, l	0.99997	1.4998e-04
φ_7	$-\alpha_7 + \beta_7 v$	v	1.0	4.1058e-11
φ_8	$\alpha_8 + \beta_8 y$	y	1.0	4.2496e-12

Table 5.6. NPQ regulators obtained by applying stepwise regression to sets $\mathcal{N}_1$ of variables reported in Table 5.5, and relative statistical tests [32]. The basis functions weights can be downloaded from [153].

In the first experiment, stepwise regression did not select all the rule reactants as tuners of corresponding fluxes, namely, for reactions r_3, r_7, and r_8. In order to build a model closer to the biological reality, where reactants are assumed to be tuners for a reaction, a second experiment has been carried out, where stepwise was applied by including the reactants in the initial regression models for regulators of reactions r_3, r_7, and r_8. Also in this case, the new set of functions, reported in Table 5.7 with its statistical tests, allowed us to reproduce the observed dynamics.

	Regulators	$\mathcal{N}_4$ (Tuners set)	$\overline{R}^2$	RMSE
φ_1	$\tilde{\alpha}_1 - \tilde{\beta}_1 c + \tilde{\gamma}_1 h - \tilde{\eta}_1 x + \tilde{\vartheta}_1 r + \tilde{\rho}_1 l$	$c, h, x, r,$	0.99997	1.4998e-04
φ_2	$-\tilde{\alpha}_2 - \beta_2 c - \tilde{\gamma}_2 h + \tilde{\eta}_2 x + \tilde{\vartheta}_2 l$	c, h, x, l	0.99999	1.3798e-03
φ_3	$\tilde{\alpha}_3 + \tilde{\beta}_3 c - \tilde{\gamma}_3 x + \tilde{\eta}_3 l \cdot r^{-1}$	$c, x, l \cdot r^{-1}$	0.65369	2.1572e+02
φ_4	$-\tilde{\alpha}_4 + \tilde{\beta}_4 o + \tilde{\gamma}_4 h - \tilde{\eta}_4 x + \tilde{\vartheta}_4 r + \tilde{\rho}_4 l$	o, h, x, r, l	0.99997	1.4998e-04
φ_5	$\tilde{\alpha}_5 - \tilde{\beta}_5 c + \tilde{\gamma}_5 h - \tilde{\eta}_5 x + \tilde{\vartheta}_5 r + \tilde{\rho}_5 l$	c, h, x, r, l	0.99997	1.8000e-03
φ_6	$\tilde{\alpha}_6 - \tilde{\beta}_6 c + \tilde{\gamma}_6 h - \eta_6 x + \tilde{\vartheta}_6 r + \tilde{\rho}_6 l$	c, h, x, r, l	0.99997	1.4998e-04
φ_7	$-\tilde{\alpha}_7 + \tilde{\beta}_7 v - \tilde{\gamma}_7 x$	v, x	1.0	4.1083e-11
φ_8	$\tilde{\alpha}_8 + \tilde{\beta}_8 y + \tilde{\gamma}_8 h$	y, h	1.0	4.2522e-12

Table 5.7. NPQ regulators, obtained by applying stepwise regression to sets of variables $\mathcal{N}_1$ (Table 5.5) and by adding the reactants as tuners of φ_3, φ_7, φ_8, and corresponding statistical tests [32]. The basis functions weights can be downloaded from [153].

Finally, in order to achieve more parsimonious models, we applied stepwise regression starting from $\mathcal{N}_2$ and we obtained tuner sets $\mathcal{N}_5$, along with the results (regulators and statistical tests) reported in Table 5.8. This has been a significant point, even for a biological interest in NPQ, because we drastically reduced (with respect to the original MP model in [148]) the set of regulative variables, and then we have found the key tuners of each regulator of the NPQ phenomenon. Excitingly enough, with tuners $\mathcal{N}_5$ from Table 5.8 we perfectly reproduce the system dynamics, along with statistical results comparable with those of the previous experiments (see Tables 5.6 and 5.7).

	Regulators	$\mathcal{N}_5$ (Tuners set)	$\overline{R}^2$	RMSE
φ_1	$\bar{\alpha}_1 - \bar{\beta}_1 c + \bar{\gamma}_1 l$	c, l	0.99829	1.0826e-003
φ_2	$-\bar{\alpha}_2 + \bar{\beta}_2 c - \bar{\gamma}_2 c^2$	c	1.00000	4.8559e-004
φ_3	$\bar{\alpha}_3 - \bar{\beta}_3 c + \bar{\gamma}_3 c^2 + \bar{\eta}_3 l \cdot r^{-1}$	$c, l \cdot r^{-1}$	0.65188	2.1629e+002
φ_4	$-\bar{\alpha}_4 + \bar{\beta}_4 o + \bar{\gamma}_4 l$	o, l	0.99832	1.0826e-003
φ_5	$\bar{\alpha}_5 - \bar{\beta}_5 c + \bar{\gamma}_5 l$	c, l	0.99829	1.2995e-002
φ_6	$\bar{\alpha}_6 - \bar{\beta}_6 c + \bar{\gamma}_5 l$	c, l	0.99829	1.0826e-003
φ_7	$-\bar{\alpha}_7 + \bar{\beta}_7 v$	v	1.0	4.1083e-011
φ_8	$\bar{\alpha}_8 + \bar{\beta}_8 y$	y	1.0	4.2522e-012

Table 5.8. NPQ stepwise third experiment, where the more parsimonious model was found [32]. NPQ tuners are obtained by applying the stepwise regression to sets $\mathcal{N}_2$ of variables reported in Table 5.5. The basis functions weights can be downloaded from [153].

In Table 5.9, adjusted coefficient of determination $\bar{R}^2$ and the RMSE are reported of experiments, where functions have been estimated by employing neural networks having one hidden layer, with five neurons in the case of $\varphi_1, \ldots, \varphi_6$, and three neurons for φ_7 and φ_8. Training has been performed, as for all the other tests, by means of backpropagation, using half data as training test and half as validation set. The dataset has been split as described in s Section 5.6.2. Moreover, each network has been retrained for three times and only the network achieving the best results has been used for simulating the system dynamics.

Tuners
c, h, l, o, p, r, v, z
c, h, l, r, z
$c, l \cdot r^{-1}, v, h$
c, h, l, o, p, r, v, z
c, h, l, o, p, r, v, z
c, h, l, o, p, r, v, z
v, x
h, y

	$\overline{R}^2$	RMSE
φ_1	1.0	8.71e-05
φ_2	0.99989	4.97e-03
φ_3	0.91246	105.96e00
φ_4	0.99998	1.10e-04
φ_5	1.0	9.66e-004
φ_6	0.99998	1.05e-004
φ_7	0.99964	4.56e-07
φ_8	0.99977	2.61e-07

	$\overline{R}^2$	RMSE
φ_1	1.0	1.17e-05
φ_2	0.99995	3.43e-03
φ_3	0.94515	83.88e00
φ_4	1.0	1.08e-05
φ_5	1.0	2.15e-04
φ_6	1.0	1.02e-05
φ_7	0.99996	1.52e-07
φ_8	0.99996	2.03e-07

Table 5.9. Left-hand side: tuners selected through a biological analysis in [148] and employed as input in first two experiments for ANN regression reported aside. Center: Statistical tests for the experiment (with generalization, and normalization of only light data, by dividing them by 1000) in which the NPQ dynamics has been better reproduced. Right-hand side: results for an experiment performed along the same conditions, apart that all datasets have been normalized to the interval $[0, 1]$, and not reproducing the observed dynamics [32]. Neural network weights can be downloaded from [153].

We have performed several other experiments, with generalization and by introducing the normalization of every dataset to the interval $[0, 1]$ in order to give the same importance to every input of the neural networks. All of them, as well as the results in Table 5.9, show that, there are cases where regression has a better performance but the functions generated are not able anymore to reproduce the observed dynamics.

In Table 5.10 we finally report the results of three experiments performed with three different sets of input variables. The analytical forms of synthesized regulation functions are reported in [153].

	$\overline{R}^2$	RMSE
φ_1	1.0	2.93e-05
φ_2	0.99996	2.89e-03
φ_3	0,962	70.05e00
φ_4	1.0	6.61e-05
φ_5	1.0	2.11e-04
φ_6	1.0	2.40e-05
φ_7	0.99993	1.86e-07
φ_8	0.99998	7.44e-08

	$\overline{R}^2$	RMSE
φ_1	1.0	1.56e-05
φ_2	1.0	2.82e-03
φ_3	0.96	72.27e00
φ_4	0.99999	5.95e-05
φ_5	1.0	1.96e-04
φ_6	1.0	3.54e-05
φ_7	0.99996	1.46e-07
φ_8	0.99987	1.96e-07

	$\overline{R}^2$	RMSE
φ_1	0.99987	2.96e-04
φ_2	0.99986	5.91e-03
φ_3	0.94748	82.87e00
φ_4	0.99987	3.02e-04
φ_5	0.99999	1.07e-03
φ_6	0.99985	3.15e-04
φ_7	0.99984	3.03e-07
φ_8	0.99987	1.96e-07

Table 5.10. Statistical indexes for neural networks trained with generalization and normalization by using three different sets of variables [32]. Left-hand side: variables in $\mathcal{N}_1$, reported in Table 5.5. Center: variables in $\mathcal{N}_4$ reported in Table 5.7). Right-hand side: small set of variables $\mathcal{N}_5$ reported in Table 5.8. Only this last test has given a faithful dynamics, for all substances but hydrogen. Neural network weights can be downloaded from [153].

5.6.4 Discussions

A first analysis on the performance of linear and ANN regression, applied to an MP model of a biological phenomenon such as NPQ, may be carried on by comparing the results of Tables 5.6, 5.7, 5.8, with the three tables, respectively, reported (from the left to the right) in Table 5.10. Indeed, in the first case, regression has been performed by starting from the same set of input variables ($\mathcal{N}_1$), while in the other two cases, regulators with same sets of tuners (respectively, $\mathcal{N}_4$ and $\mathcal{N}_5$) have been obtained. Let us recall that, stepwise regression has a possibly different set of input variables and set of tuners (because a selection of variables regulating the functions is performed by the method itself), while in ANN they coincide.

Comparing Tables 5.6 and 5.7 with the first two tables in Table 5.10, neural networks outperformed stepwise regression on functions $\varphi_1, \varphi_3, \varphi_4, \varphi_5, \varphi_6$, while for functions $\varphi_2, \varphi_7, \varphi_8$, stepwise regression scored better statistical indexes. We remark, however, that regulation functions synthesized by stepwise regression perfectly reproduce the dynamics, and those by neural networks do not. Comparing the stepwise regression parsimonious model of Table 5.8 with the neural networks working on $\mathcal{N}_5$, statistical indexes are still better for neural networks, but while stepwise functions can reproduce the entire observed dynamics, neural networks faithfully reproduce the time evolution of all substances but hydrogen.

As a conclusion, to model NPQ, stepwise regression seems to give a better performance than regression by neural networks. On the other hand, in Section 5.4 we have presented a neural approach for modeling the mitotic cycle in early amphibian embryos, namely, to infer flux regulation maps associated with reactions [33]. In that case, simple neural networks with one hidden layer having three neurons have been able to learn good regulators from a dataset of about 4000 observations. Interestingly enough, functions synthesized by means of stepwise regression for the same MP model were not able to reproduce the dynamics of the mitotic cycle, even if scoring low residuals. We performed several experiments in this context, by applying stepwise regression on different sets of polynomial basis functions: *i)* all substances/parameters involved in the mitotic cycle network [33], *ii)* variables resulting from the correlation analysis described in section 5.6.2, and *iii)* variables resulting from a combined correlation and the scatter plot analyses among fluxes and substance/parameter time-series (as we have done, in case of NPQ, for $\mathcal{N}_2$). In all these experiments we have obtained models able to fit flux values but unable to reproduce the dynamics of mitotic cycle.

In the following, we discuss a few issues which arose in the attempt to understand the inadequacy of stepwise regression (for mitotic cycle model) and of ANN (for NPQ).

In case of mitotic cycle, we noticed that the estimation of a specific couple of regulators is crucial to control the dynamics (they are φ_5 and φ_7 by referring to the table of reactions reported at page 198 of [33]), indeed, in our stepwise regression experiments, to have simulations in accordance with the observed dynamics it is enough to replace such regulators with the original flux time series (computed by log-gain theory). On the other hand, high degree polynomials of many basis functions are needed to satisfy the performance specifications of these two specific regulators, revealing a complexity which is difficult to tackle by linear regression. Residual analyses exhibited non-random structures, a clear sign that the inferred

maps fit the data poorly, and residual plots revealed a nonlinear relationship among the regulators φ_5 and φ_7 and some substances.

When a set of analysis reveal nonlinear relationships among variables of a dataset, it is often possible to transform some of these elements in order to obtain linear relationships [1]. In our case, we tried to apply only nonlinear transformations, which increase linear relationships among substances and fluxes and, thus, change the correlations among variables. Namely, we applied a multi-step trial-and-error process to transform some of our data, according to the following steps: *i)* choose a transformation method, *ii)* transform some of the independent variables, *iii)* find a model of φ_r by stepwise regression including the transformed variables, *iv)* reduce some basis functions of φ_r by statistical inference (to have a more parsimonious model), *v)* analyze the adjusted coefficient of determination and create a residual plot. If the residual plot shows a random pattern and the adjusted coefficient of determination exceeds a given threshold, the transformation was successful, otherwise return to step *i)* to start with a different transformation method. In this way, we obtained models of φ_5 and φ_7 able to fit flux time-series and to reproduce, together the other regulators inferred by using stepwise regression, the dynamics of mitotic cycle. Unfortunately though, there are many ways to transform variables to achieve linearity for regression analysis, and the above technique can require a lot of computation time.

Therefore, as a final successful approach, we sampled the time-series of the mitotic cycle with $\tau = 1$ minute (instead of 0.06 secs), in order to have 100 macro-observations, and we considered corresponding data of macro-fluxes[2]. In this case, we have been able to learn good regulators by simply applying these steps: *i)* a *principal component analysis* [104] to analyze the relationships among fluxes and substances, *ii)* stepwise regression to infer the regulators, and *iii)* statistical inference, based on t-statistics for the regression coefficient and on F-test for the regression model, to retain only the relevant basis functions for parsimonious models. This procedure points out the additional value of MP modeling approach, dealing with all data sampled at observational (macro) intervals of time, and being able to deduce new knowledge on the regulative mechanism underlying the modeled biological system.

In case of NPQ, we noticed that the MP graph may be partitioned in a few independent functional subunits, while two of them being crucial to keep on the entire dynamics - in the following, let us denote with R the rules and with F the flux regulators. Some subgraphs, such as those producing Q_{cum} (cumulated heat) by reaction R_2, or F_{cum} (cumulated fluorescence) by reaction R_3, are important to have the final products of our system, but they do not affect any other subunit of the graph. An analogous consideration may be done for the subgraph producing v (violaxanthin) and z (zeaxanthin) by R_7 (which by the way produced and consumed a same amount of x), or that including substance p (NADPH) and reactions R_1 and R_6 (which have identical fluxes since p is constant).

There are then mainly two dynamically significant subgraphs (apart of any biological consideration), pointed out in Figure 5.27 as subgraph 1 and 2. The first involves open and closed receptors (c and o) with reactions R_1 and R_4, whose rates difference, initially positive, needs to converge to zero as the system evolves.

[2] The author wish to gratefully thank Vincenzo Manca, who kindly suggested this idea.

This synchronization seems quite difficult to reach by neural networks, since on the fluxes F_1 and F_4 they are trained independently and even a small error in their predictions may generate an error in the time evolution of c and o. In general, if residuals of synchronized functions have opposite effects then they tend to disappear, otherwise they cumulate the error. This error is often amplified as the simulation goes on, because when the model reaches unobserved states, neural networks may produce perturbed fluxes which bring the system some more far from the correct dynamics. From our analyses in fact, it seems that the nonlinear form of neural network increases their ability to fit observed data, but it enhances the risk to do overfitting, and thus to have wrong flux values around observed states. Moreover, since neural networks training algorithms are stochastic, they may produce functions having sometimes opposite residuals and sometimes residuals with common effects. This is why, by using neural networks, we sometimes generate functions able to reproduce the observed dynamics and sometimes not.

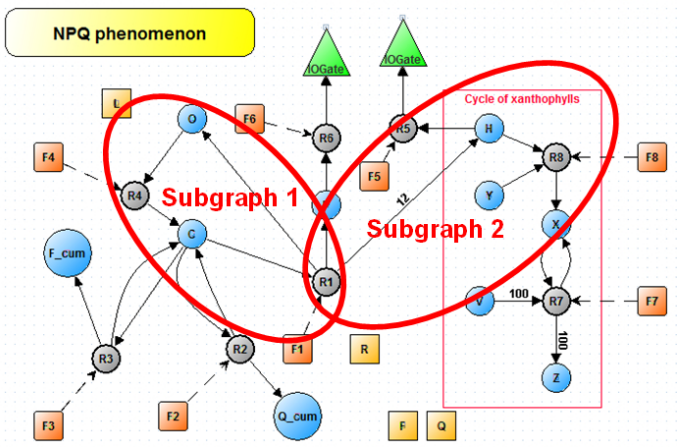


Fig. 5.27. Dynamically significant subgraphs identified in MP graph of Figure 5.22 modeling NPQ (figure from [32] with permission).

Subgraph 2 involves substances h (hydrogen ions), y (inactive violaxanthin de-epoxidase), x (active violaxanthin de-epoxidase) and reactions: R_5, and R_1, very correlated each other (F_5 is approximatively 12 times F_1), and which flux difference keep the amount of hydrogen ions within observed (relatively very small) values, and R_8, which affects the amount of hydrogen as well but has a flux not correlated with those of R_5, and R_1. The observed dynamics of h has values between $6 \cdot 10^{-4}$ and $122 \cdot 10^{-4}$ μmol, which are obtained by adding, at each step, a flux $12 \cdot F_1$ (of the order of 1 mol), and then subtracting the flux F_5 (of the order of 1 mol) and the flux F_8 (of the order of 10^{-5} mol). Since fluxes $12 \cdot F_1$ and F_5 are much greater than the average values of h, small relative error on one of these two fluxes (e.g., an error of 0.1% in F_5 is about $0.001mol$) yields a relatively high error in h ($0.001mol$ corresponds to an error of about the 20% if the amount of H is 0.005). Accordingly, small relative residuals in F_1 and F_5 correspond to big relative errors in updating h, so that the dynamics of h is brought away from the observed dynamics. Since h

is employed as argument in many flux regulation functions (except for the set $\mathcal{N}_5$ of variables), the error can quickly propagate to the rest of the system, and this would explain the results in Table 5.10.

The case studies here investigated prove that complex interactions taking place in real biological networks may need different regression techniques in order to be modeled. Furthermore, along this trend, further investigation needs to be carried out, both to simultaneously synthesize a group of correlated systemic functions, rather than a single function, and to analyze how the form of flux functions around the given fluxes can affect the dynamics.

Author's publications for this chapter

- A. Castellini, G. Franco, and R. Pagliarini. Data analysis pipeline from laboratory to MP models. *Natural Computing*.
- A. Castellini and V. Manca. Learning regulation functions of metabolic systems by artificial neural networks. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO '09*, pages 193-200, Montreal, Québec, Canada, 2009. ACM Publisher.
- A. Castellini, V. Manca, and Y. Suzuki. Metabolic P system flux regulation by artificial neural networks. In G. Păun et al., editors, *10th Workshop on Membrane Computing, Lecture Notes in Computer Science 5957*, pages 196-209. Springer-Verlag Berlin Heidelberg, 2010.

6

MetaPlab virtual laboratory

The huge amount of experimental data available for metabolic pathways requires suitable computational tools in order to be stored, retrieved, visualized and analyzed, as well as ad-hoc mathematical models to be correctly interpreted [84, 233]. In this last chapter we survey some of the main software for bioinformatics and systems biology (Section 6.1), and we present the MetaPlab virtual laboratory [34, 146, 241], a software platform entirely developed to generate and analyze discrete dynamics of biochemical systems in terms of MP systems (Section 6.2). This software, entirely developed by the MNC[1] group at the University of Verona in collaboration with the Center for BioMedical Computing (CBMC) [70], involves a main graphical interface, presented in Subsection 6.2.1 and an extensible Java plugin based architecture, described in Subsection 6.2.2. As a virtual laboratory, MetaPlab is equipped with several virtual tools, in the form of plugins, performing various processing jobs, such as, *dynamics computation*, *flux discovery*, and *regulation function synthesis*, which are detailed through Subsections 6.2.3-6.2.9.

6.1 Software for bioinformatics and systems biology: a brief overview

The last decade has seen the development of several high-throughput techniques able to perform biochemical, genetic or pharmacological tests on hundreds of thousands of samples per day [26, 133]. This huge amount of data needs suitable tools in order to be stored, retrieved, visualized and analyzed, and, moreover, it requires ad-hoc mathematical models to be correctly interpreted [233]. The simultaneous increasing of computational power availability has brought to a substantial deployment of several software tools able to satisfy some of these requirements. As for the storing process, database tools, such as, *GenBank* [11], *EMBL Nucleotide Sequence Database* [127], *DDBJ* (DNA Data Bank of Japan) [215], *KEGG* (Kyoto Encyclopedia of Genes and Genomes) [112, 113] and *PDB* (Protein Data Bank) [12], provide several interconnected information about genes, proteins and biological pathways. Some of these tools are also equipped with retrieval and analysis facilities, such as the well known *BLAST* (Basic Local Alignment Search Tool) [6] and

[1] MNC stands for Models of Natural Computing.

FASTA [132,169], that enable quick similarity searches through DNA and protein alignment.

Computational modeling [66] is becoming a key approach to understand the biochemical processes underlying observed data [119], and many software tools have been developed for this purpose (see the list in [243]). There exist *simulators* based on: *i)* discrete-stochastic approaches [50, 51, 99, 129, 167], often based on the Gillespie's algorithm, *ii)* continuous-deterministic approaches [99, 164, 224, 226], mainly involving ordinary differential equations (ODE), and *iii)* hybrid approaches [50, 99, 117]. Some tools provide also graphical facilities for improving *model design and visualization*, and enhancing the readability of biological features through the model representation. An example in this direction is represented by Cell Illustrator^{TM} [164], a software tool which enables to graphically model metabolic, signal transduction and gene regulatory pathways by means of hybrid functional Petri nets [154] and to simulate deterministically their dynamics [31]. Other tools for biological modeling aim to *analyze fluxes* in metabolic networks [121] and to manage data and components *integration* [49,204], but only a few software suites originate as integrated tools coping with all the phases of biochemical modeling.

COPASI (COmplex PAthway SImulator) [99] is a first example of integrated suite. It combines graphical facilities for editing and visualizing model parameters, with simulation, plotting, analysis, exportation and parameter estimation tools. The software, implemented in C++, can be used through graphical user interfaces or command line, and it enables both a mathematical view of the model (by means of variables and differential equations) and a biochemical perspective employing concepts of reaction, compartments, metabolites, and so on.

Another well known suite aiming at the whole-cell simulation is E-CELL [224, 226]. It is an object-oriented tool implemented in C++ attempting to model cellular processes at a various level, such as protein-protein interactions, protein-DNA interactions, regulation of gene expression and other cellular interactions, with the final aim to construct a cell model for *in silico* experiments. An E-CELL model involves three types of objects: *substances*, *genes* and *reaction rules*, where substances represent molecular species, genes represent DNA sequences (in which coding sequences, protein binding sites and intergenic spacers are reported) and reaction rules represent typical reactions in metabolic pathways (complex reactions, such as transcription or translation are modeled by series of reactions). Notice that information about the objects involved in these models are recovered from knowledgebases such as KEGG. Graphical user interfaces enable the user to access to all the functionalities provided by the software. Simulations of cell behaviors are performed by numerically integrating differential equations describing the system under investigation.

The multiplicity of software and viewpoints for understanding the logic and functioning of biochemical systems have required a standardization of model representation, which has been achieved by the *Systems Biology Markup Language* (SBML) [243]. It is a tool-neutral, computer readable format for representing models of biochemical reaction networks, and it is applicable to metabolic networks, cell signaling pathways, genomic regulatory networks and other systems investigated in systems biology [41, 101, 102]. SBML is based on XML (the eXtensible

Markup Language) [238], a standard medium for representing and transporting data, which is widely supported on the Internet as well as by the systems biology and bioinformatics communities. The main aim of SBML is model portability, indeed, by encoding a biochemical model into an SBML file we enable:

- to read the model in a lingua franca whatever is its specific format (differential equations, algebraic equations, reactions, pathway diagrams, event rules, etc.);
- to simulate the model by using multiple SBML-compatible tools (e.g., discrete-stochastic and continuous-deterministic simulators) without rewriting the model file;
- to disseminate the model in peer-reviewed literature, where reviewers and readers can independently test the model and reproduce the proposed results;
- to ensure model survivability beyond its visualization/simulation engine.

The first version of SBML, called SBML Level 1, was released on 2nd March 2001, thanks to an international community of software developers and users. The most recent version of the standard, which is the Level 2, Version 4, has the structure reported in Table 6.1 [243]:

beginning of model definition
list of function definitions
list of unit definitions
list of compartment types
list of species types
list of compartments
list of species
list of parameters
list of initial assignments
list of rules
list of constraints
list of reactions
list of events
end of model definition

Table 6.1. Structure of SBML files

Each list encodes, in a specific XML format, some elements of the biochemical model under investigation. The user can choose the elements he/she needs for encoding the model, since each list is optional.

Another international project related to the SBML, aims to define a standard graphical notation for visually describing biological networks and processes. The *Systems Biology Graphical Notation* (SBGN) [242] is including a comprehensive set of symbols for process diagrams [120] (see Figure 6.1), entity relationship diagrams and activity flow diagrams. For such a graphical notation to be accepted by the community and employed as a standard, software tools and data resources are being made available.

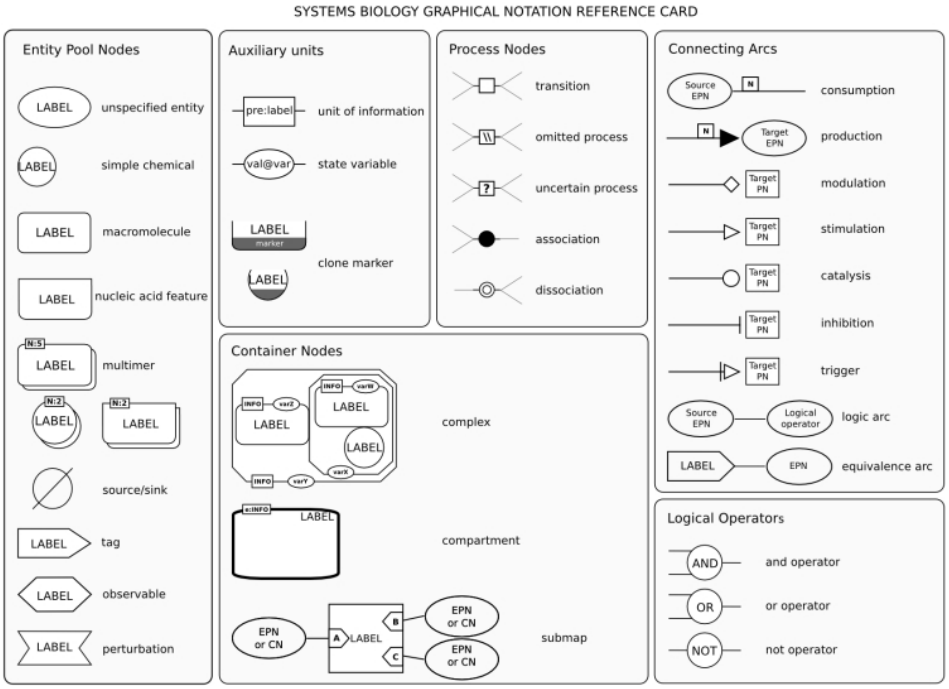


Fig. 6.1. SBGN symbols for representing biological networks by process diagrams [242].

The current proliferation of software packages for processing cellular networks has resulted in a duplication of capabilities. Since developing such tools is time-expensive, their reuse is wished in order to enable developers to concentrate on novel functionalities rather than to reinvent existing packages. An effort in this direction is represented by the *Systems Biology Workbench* (SBW), a software framework that allows heterogeneous application components, written in diverse programming languages and running on different platforms, to communicate and use each others' capabilities via a fast binary-encoded message system [204, 244]. Each SBW-compatible application, potentially running on separate distributed computers, can communicate with other SBW-compatible applications via a simple network protocol. The interfaces to the SBW system are encapsulated in client-side libraries that are provided for different programming languages. In this way, developers can build on previous work without having to understand in detail the often intricate internal workings of other tools. What a developer needs to know is only the interface that the tool exposes.

As shown in Figure 6.2, SBW consists of two components, a broker for routing messages (rounded-corner rectangles) and modules which send and receive messages (stars). All connections between modules and a broker take place via standard TCP/IP sockets. All messages are transmitted in a binary format for maximum performance. If a message needs to be sent between two different computers, then messages are sent first to the broker on the remote machine, this in

turn routes the message to the correct remote module. Modules may be written in a variety of languages, including, Java, C/C++, Delphi, Perl, Python and Matlab.

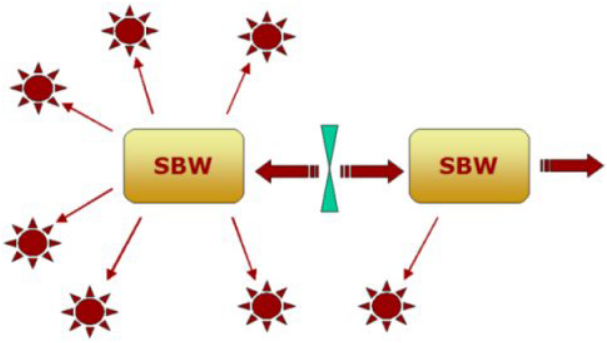


Fig. 6.2. Communication among SBW brokers and modules [244].

6.1.1 Software based on P systems

At the end of this brief survey we want to mention three interesting tools based on P systems for biological modeling and systems biology. The first is *P lingua* [131], a programming language for membrane computing which aims to be a standard to define P systems. This tool, developed by the research group on natural computing, at the University of Seville (Spain), is equipped with *i)* a command-line compiler which checks both syntactic and semantic programming errors on P lingua files and translates them to XML files, *ii)* a command-line simulator which computes P system dynamics according to stochastic strategies. Another valuable tool is the Infobiotics Workbench developed at the ASAP group of the University of Nottingham (UK) [240]. It is a computational framework written in C++ which provides a user-friendly front-end allowing modelers to design in-silico experiments, analyze and visualize results by means of four components: *i)* a modeling language which allows to define multicellular models including geometrical informations, *ii)* a multi-compartmental stochastic simulator based on Gillespie's SSA (see Section 3.2), *iii)* a module for formal model analysis using stochastic model checkers PRISM and MC2, *iv)* a module for structure and parameter model optimization based on evolutionary algorithms. Finally, the third tool we mention is a simulator based on dynamical probabilistic P systems (see Section 3.3) and developed in 2006 at the University of Milano-Bicocca (Italy). Written in C++ this software is freely available from the P system web site [223] together with some experiments about Vibrio Fischeri. From the same web site many other simulators based on several P system variants can be downloaded. In the next section we introduce MetaPlab, a modeling tool based on MP systems and developed in the last four years at the computer science department of the University of Verona.

6.2 MetaPlab

The process of modeling biochemical processes by MP systems involves various steps, which can be summarized in the following three phases: *i)* data collection, *ii)* computation of the unknown elements of the MP model, and *iii)* model validation and analysis. What we aim to do now is to automatize some of these phases by means of suitable software tools. Let us start with a simple example. Suppose to know, from experimental observations, the set of substances involved in a biological process (X, item 1 of Definition 6 in Chapter 4), the chemo-physical parameters with their evolution functions (V, item 3), the reactions (R, item 2), and the mathematical laws which regulate reaction's fluxes (Φ, item 5). Given these elements, the dynamics (δ, item 10) of a related MPF system can be computed for a certain number of steps by solving Equations (4.2) and (4.3).

This task, we will call *dynamics computation*, is only one of several biologically inspired mathematical problems which can be tackled by MP systems. Table 6.2 collects a few of these tasks focusing on the known and the unknown elements of related MPF systems. The second problem proposed in Table 6.2 is called *flux discovery*, and entails the computation of flux time-series $U[1], \ldots, U[t-1]$ which yield an observed dynamics $\delta_{<t}$ of substances and parameters (where $\delta_{<t}$ denotes a dynamics having t steps). A mathematical theory for solving this problem, called log-gain theory, has been already presented in Section 4.6. The subsequent task concerns *regulation function discovery*. It is a regression problem which aims at generating flux regulation functions Φ which fit a (known) set of flux time-series $U[t], \ldots, U[t-1]$ obtained for a dynamic $\delta_{<t}$. As explained in Chapter 5, these functions can be generated by traditional regression methods [32, 148, 149] as well as by evolutionary and bio-inspired techniques, such as genetic programming [124] and neural networks [24, 32, 33].

Problem	**Known elements**	**Unknown elements**
Dynamics computation	X, R, V, Φ, q_0	δ
Fluxes discovery	$X, R, V, U[0], \delta_{<t}$	$U[1], \ldots, U[t-1]$
Regulation discovery	$R, U[0], \ldots, U[t-1], \delta_{<t}$	Φ
Dynamics analysis	$X, R, V, \delta_{<t}$	Statistical params, etc.

Table 6.2. Some biologically inspired problems which can be tackled within the MPF systems framework. Unknown elements should be computed from known elements by means of suitable mathematical techniques and computational tools.

The last problem listed in Table 6.2 concerns the *analysis* of MP system *dynamics*, a data-mining task which involves the discovery of new biological information from observed dynamics. It is related to: *i)* the discovery of dynamics patterns and statistical parameters (e.g., dynamics and flux correlations), *ii)* the clustering

of observed time-series, and *iii)* the analysis of the dynamical behaviors occurring from different (environmental and structural) conditions.

Of course, it would be very useful to systematically attack these and further problems by means of a set of suitably developed computational tools. The software *MetaPlab* [34,146,241], presented in the following, meets this target by supporting an extensible set of plugins, each dedicated to a specific task. MetaPlab is a *virtual laboratory* which *assists biologists to understand internal mechanisms of biological systems and to forecast, in silico, their response to external stimuli, environmental condition alterations and structural changes.*

6.2.1 General features and input GUI

MetaPlab is a stand-alone program which can be used through some graphical user interfaces (GUIs) by which MP graphs can be easily generated, processed, visualized, analyzed and related to experimental data. This software extends the capabilities of the former *Psim* [18,22] package, which enables only to generate and visualize MP dynamics. The peculiarity of MetaPlab is that each model-processing job, such as the model dynamics computation, the flux discovery, and the regulation function synthesis, is performed by a specific plugin tool. In fact, the extensible plugin based architecture, described in Subsection 6.2.2, makes MetaPlab a proper virtual laboratory wherein MP plugins represent virtual tools for processing MP models.

MetaPlab is a free software distributed, under the GNU General Public License (GPL) [239], at the MetaPlab official website [241]. The Java implementation of this software ensures its complete portability across every platform supporting a Java virtual machine. The official user-guide [146] of the software is available online at [241].

The MetaPlab's graphical user interface, displayed in Figure 6.3, is called *input GUI* since it enables the user to input MP models by means of MP graphs, to import experimental data and to visualize them in a "network-oriented" way. Three main areas, can be identified in this GUI: *i)* a menu and toolbar section, on top, whereby the user can manage (i.e., open, close, save, print, export, etc.) model files, edit (i.e., undo, redo, delete, etc.) MP graphs elements, adjust MP graph visualization (i.e., zoom, show grids and rulers, etc.), check model correctness and start new model-processing sessions; *ii)* a central drawing area, where MP graphs are visualized; *iii)* a side-bar, on the right, containing tools for generating new MP graphs from scratch, navigating them and checking their correctness. Specifically, in this area modelers can find drag-and-drop buttons for creating substance, parameter, reaction and flux nodes, a pointer for drawing arches between nodes, a navigator tool for quickly moving across large MP graphs and a real-time checker detecting model inconsistencies.

Figure 6.4 shows some steps of the process of MP graph generation performed by means of the MetaPlab input GUI. Substance nodes S_0, S_1 and S_2 are created, in the top-left side picture, by selecting the substance button in the side-bar and then clicking on the drawing area, where blue-circular nodes automatically appear. Subsequently, the reaction button and the input/output button are selected to generate, respectively, reaction nodes (gray circles R_0, R_1, R_2, R_3, R_4)

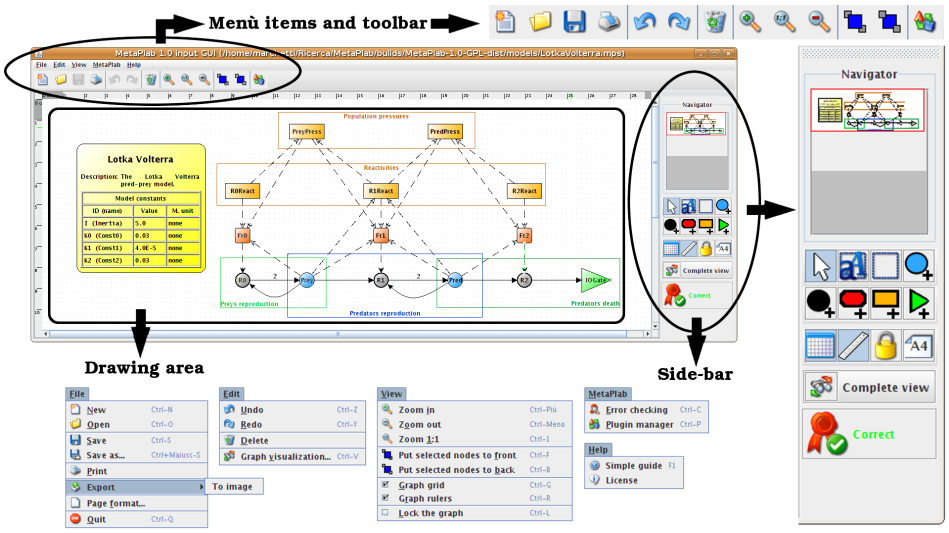


Fig. 6.3. MetaPlab input GUI [146].

and input/output gates (i.e., green triangles). The stoichiometry of the system is completely described at the third step, where arches are drawn between substance and reaction nodes by using the specific pointer tool. The fourth step concerns the creation of flux nodes F_0, F_1, F_2, F_3, F_4, which are suitably connected, at the successive step, to reaction nodes through dashed arches. In the last picture, node labels have been set to the name of the elements they represent, namely, substances A, B, C, reactions R_1, R_2, R_3, R_4, R_5, and so on. Moreover, the model name and constants have been set into the organism node.

Once an MP graph has been built, the user can include specific information in each node by simply clicking on the node and filling in some fields of the window which pops up. For instance, by clicking on a substance node one accesses to a window (showed in Figure 6.5) whereby the name, the initial quantity and the molar weight of the substance can be defined and visualized. In order to set and import from (export to) file substance time-series coming from experiments or simulations, the user can click the button "Edit time series", which opens the window displayed in Figure 6.6. It contains a table listing substance quantities over time and a chart visualizing the same data.

The procedure described so far for accessing substance node properties, is employed also to access the properties of parameter, flux and reaction nodes. For the specific case of flux nodes, a field called "evolution" enables the user to define a flux regulation function. The substances and parameters nodes corresponding to the arguments of a flux regulation function are automatically linked, by dashed arches, to the flux node containing the regulation function itself, as showed in Figure 6.7. This "network-oriented" access to model data is very easy-to-use even for people more unfamiliar with biological modeling. A more technical description on how to employ the input GUI to generate MP models and how to analyze MP model data can be found in [146].

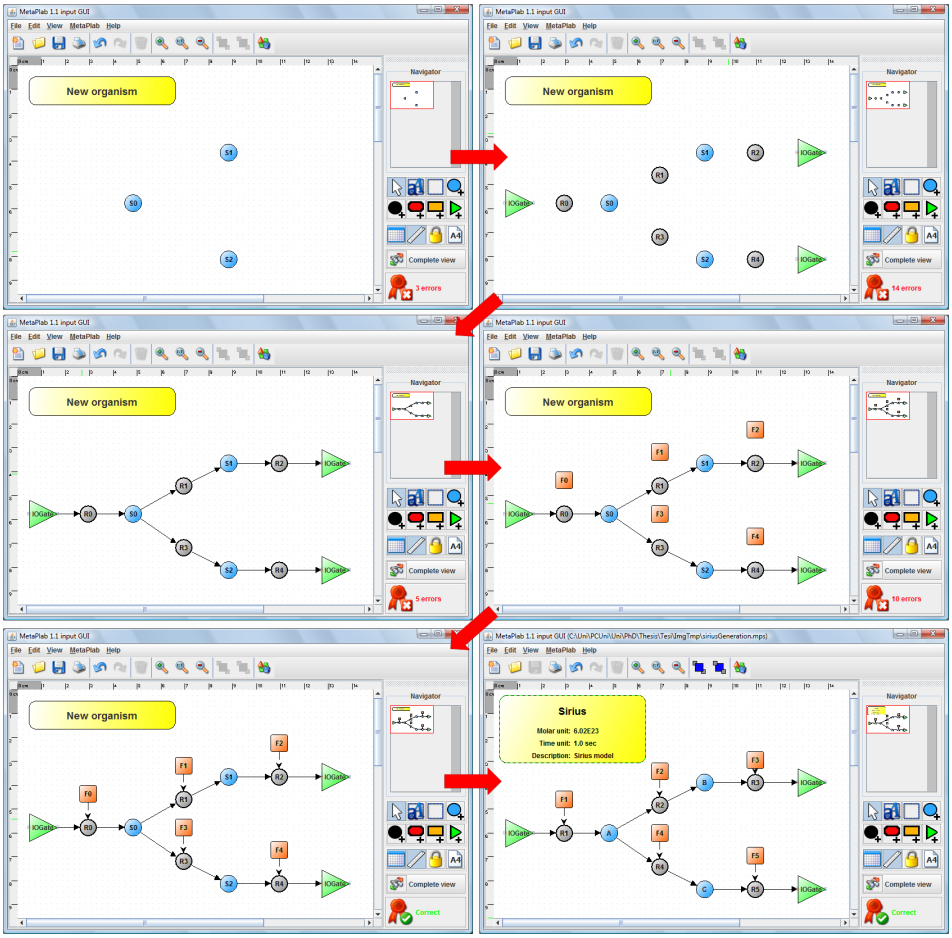


Fig. 6.4. MP graph generation through the MetaPlab input GUI.

When an MP graph has been generated, the modeler usually wants to simulate it, to analyze its dynamics and, in general, to infer new information about the biological system under investigation. Each of these processing tasks is accomplished, within the MetaPlab framework, by a specific plugin. The MetaPlab input GUI provides the user with a button which starts up the *plugin manager*, a simple tool by which plugins can be selected and launched. In the next section we introduce the plugin architecture, while subsequent sections describe specific plugin tools.

6.2.2 Plugin-based architecture

The software architecture of MetaPlab is showed in Figure 6.8. It is an original architecture, devised and developed by the author of this thesis, which involves four layers: the first one, displayed on the left side and identified by the label "*MP graphs*", copes with the definition and visualization of MP graphs, and it is represented by the MetaPlab input GUI; the second layer, concerning the *MP*

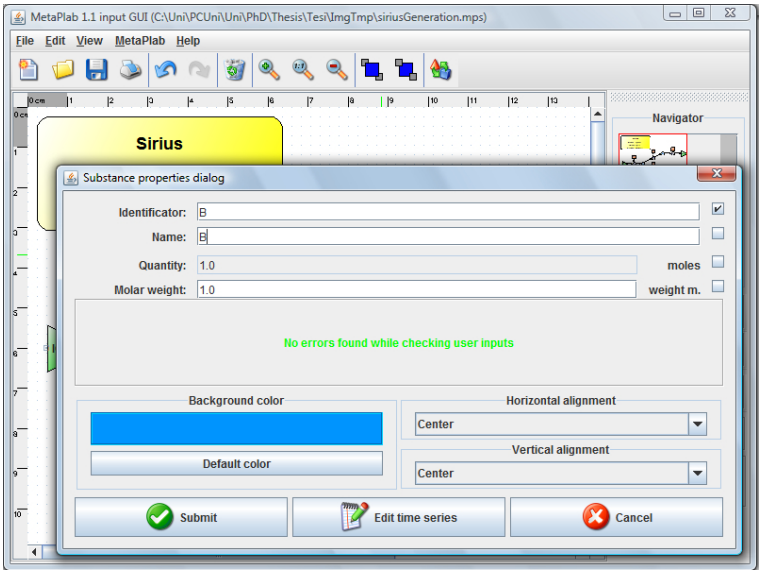


Fig. 6.5. A window for setting up and visualizing substance properties.

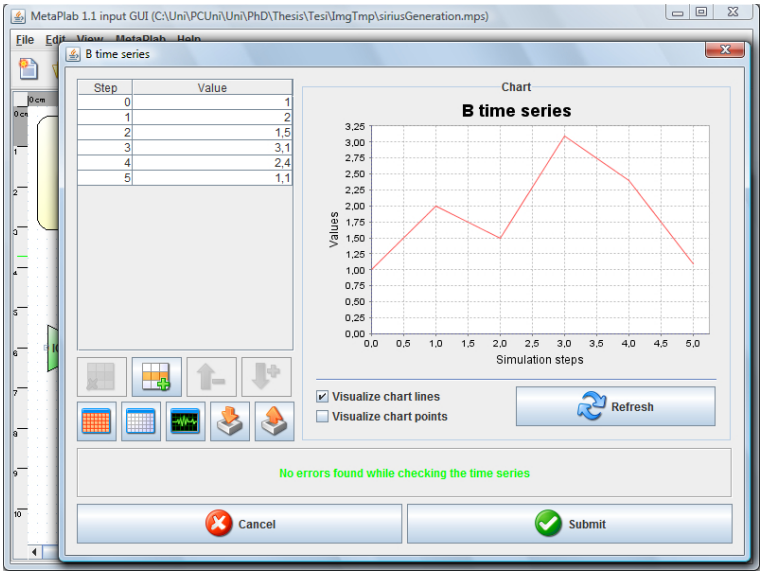


Fig. 6.6. A window for setting up and visualizing substance time-series.

store data structure, is dedicated to MP model storing; the third layer, called *data processing*, is the core of this architecture since it involves an extensible set of plugins for processing MP models and a *plugin manager* by which the user can easily select and launch plugins depending on the specific task he/she has to deal with; finally, the forth layer, showed on the right side of Figure 6.8, can be seen as an appendix of the third layer, since it collects all the MP model visualizations

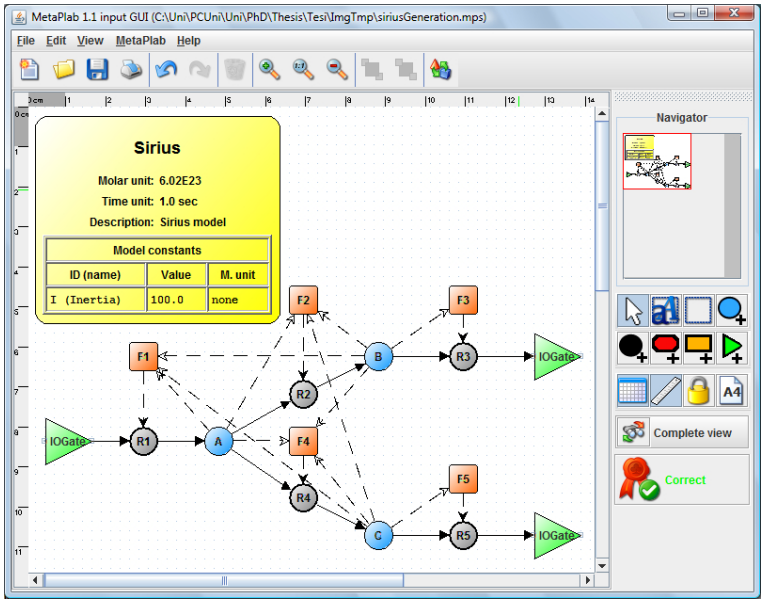


Fig. 6.7. Final MP graph obtained through the MetaPlab input GUI.

provided by plugins. These visualizations are called *MP vistas* and they can involve mathematical views (i.e., the ODE translation of an MP system), dynamics views (i.e., time evolution charts or phase charts), and any other kind of view chosen by plugin developers. In the following we report some technical details of each layer.

MP graphs. The starting point of any job in MetaPlab is represented by the definition of an MP model in the form of MP graph. The MetaPlab input GUI, described above, enables the user to easily accomplish this task in a graphical way. The reader may refer to Subsection 6.2.1 for a description of this graphical interface.

MP store data structure. The second layer of Figure 6.8 consists of an object-oriented data structure, called *MP store*, which has been designed to store all the elements of an MP system by suitable Java objects.

As shown in Figure 6.9 and, with more technical details in Figure 6.10, the first object we find in an MP store is a Membrane object. It includes all the elements of an MP system, i.e., substances X, parameters V, reactions R and fluxes Φ. Each *substance* $x \in X$ is represented by an object which stores *i)* the substance name x, *ii)* its molar weight $\mu(x)$ and *iii)* the time-series of its dynamics $((\delta(i))(x) \mid i \in \mathbb{N})$. Each *parameter* $v \in V$ is implemented by an object having two main fields, the first one stores the parameter regulation function h_v as a string, while the second holds the time-series of its dynamics $((\delta(i))(v) \mid i \in \mathbb{N})$ as a vector (called "values") of real numbers. *Flux* objects are very similar to parameters, indeed each flux stores, into a string, the regulation function φ_r of a reaction r, and it contains the related flux time-series $(\varphi_r(\delta(i)) \mid i \in \mathbb{N})$ into a vector. Finally, each *reaction* object

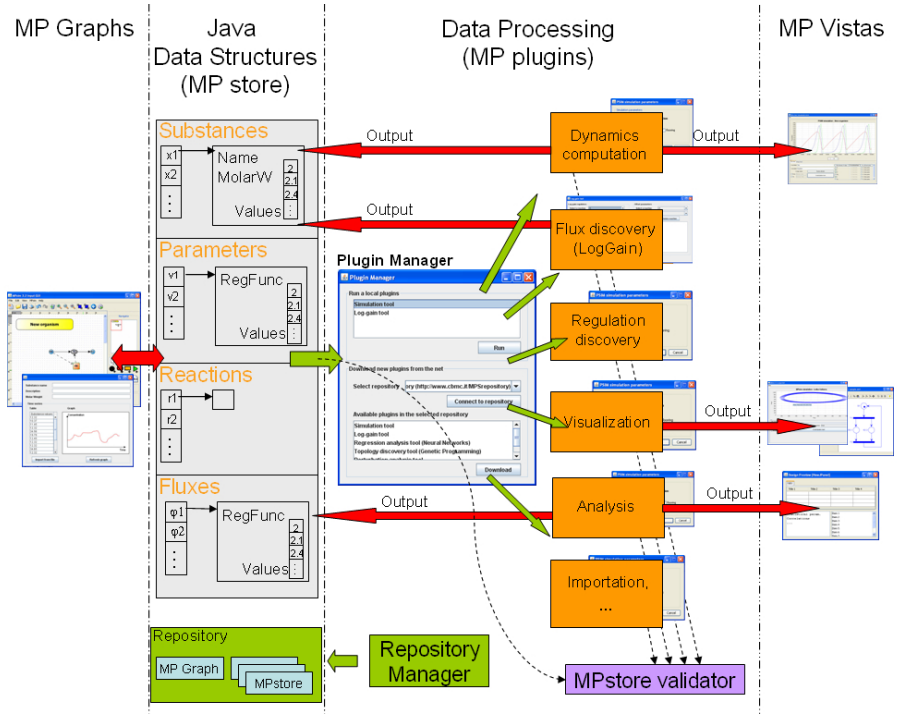


Fig. 6.8. MetaPlab architecture (figure from [34] with permission).

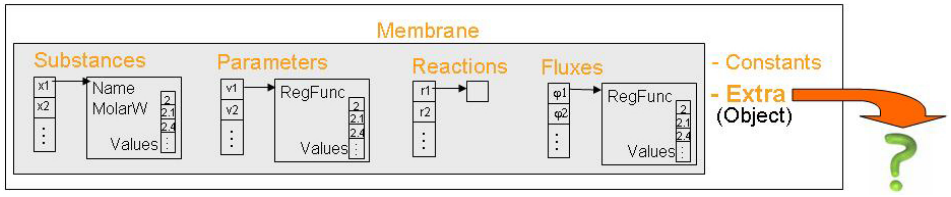


Fig. 6.9. MP Store data structure.

implements a reaction rule $r \in R$ by means of a vector of pointers addressing to the substances involved in r. A multiplicity field related to each pointer states the stoichiometric contribution of the substance to the reaction.

The MP store data structure is a key element of the MetaPlab architecture since its standard structure ensures a complete compatibility among the entire set of plugin tools, and between the plugins and the input GUI. In fact, each MP model developed through the input GUI is automatically stored into an MP store instance representing a kind of "working board" for plugins during the processing stage.

Figure 6.10 reports, in a schematic way, the headers of all the Java classes involved in the MP store data structure. Each of the eight boxes, called respectively, *MPStoreExt*, *ConstantMPExt*, *MembraneMPExt*, *SubstanceExt*, *FluxExt*, *ParameterExt*, *ReactionExt* and *MultiplicityExt*, represents a Java object of the homony-

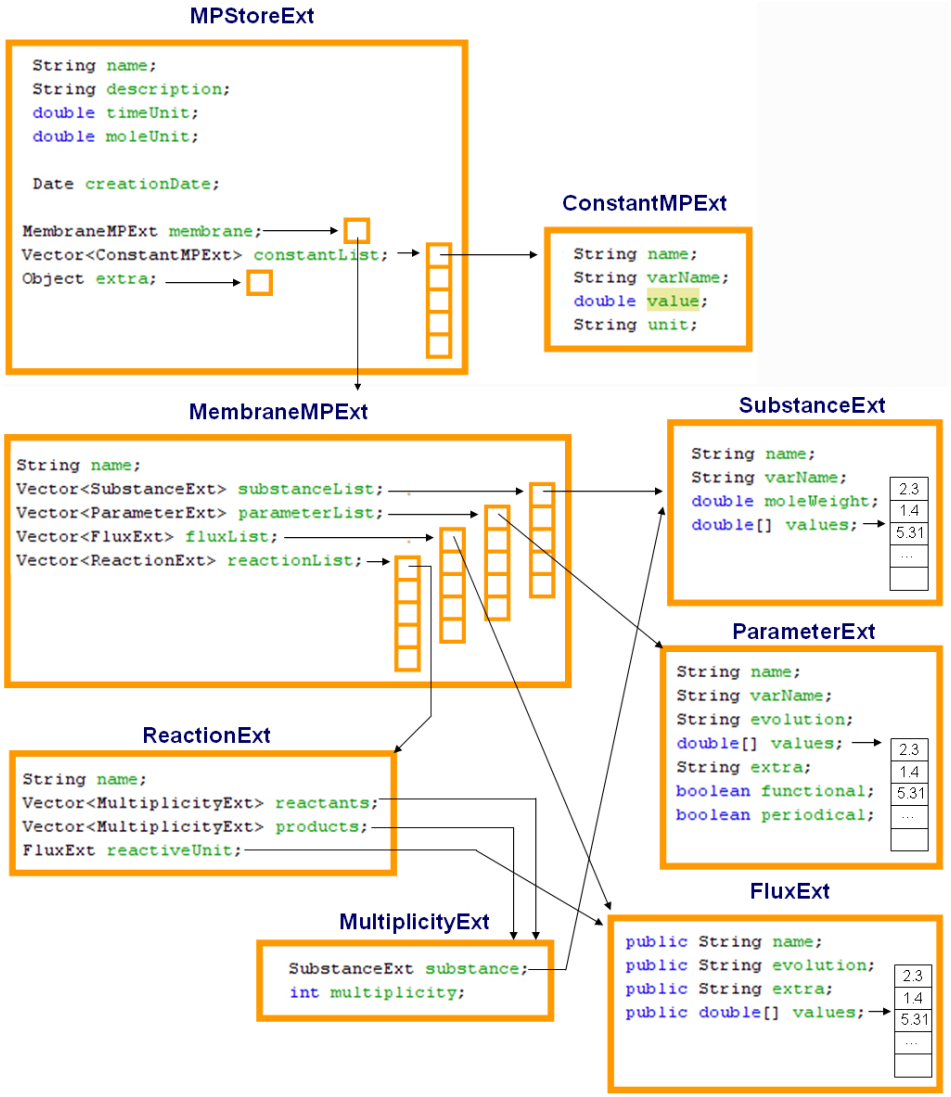


Fig. 6.10. MP store data structure details. Each big box contains the header of a Java class involved in the MP store data structure. Green strings represents Java fields. Each of them is preceded by a string indicating the field type. Small squares represents Java object references, while array of small boxes represent Java vectors containing sets of objects.

mous type. Each textual line inside these boxes is a Java field, where field names (green strings) are preceded by field types. Small squares represents object references, while arrays of small boxes represent Java vectors containing sets of objects.

Data processing. The third layer of Figure 6.8 represents the core of the new architecture. It is a plugin based module coping with MP systems data processing.

This layer is composed by *i)* an extensible set of Java *plugins*, listed on the right of the third layer, each equipped with specific input and (auxiliary) output GUIs, and *ii)* a *plugin manager*, depicted on the left of the third layer, which automatically loads MP plugins when MetaPlab is started up and it makes them available to be launched by the user.

MP plugins are the MetaPlab's processing units. Each of them is involved in a specific computational task, such as the dynamics computation of an MP system, the estimation of its regulation functions, the analysis of its dynamics, or the importation of metabolic models from databases. To accomplish one of these (or further) tasks, a plugin gets two possible **inputs**: one or more MP store instances of the model under investigation (loaded into the *inMPStoreExt* vector of Figure 6.11), and, possibly, a set of auxiliary data coming from the plugin input GUIs (if the plugin provides any graphical interface). Plugin **outputs** are saved into a specific vector (called *outMPStoreExt* in Figure 6.11) and automatically returned to the plugin manager, so that other plugins can subsequently employ them for further processing. Another way to save plugin outputs, which however does not always ensures format compatibility with other plugins, is file exportation. Moreover, some kinds of plugin results can also be visualized by suitable *MP vistas*, namely, specific views aiming to better explain the information they convey.

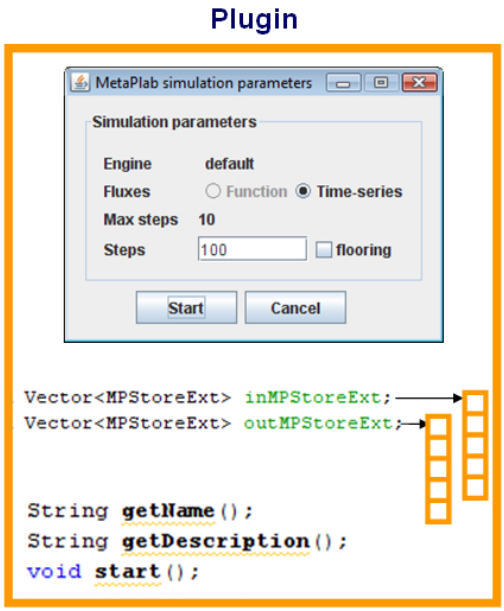


Fig. 6.11. A plugin example. In the bottom, vectors *inMPStoreExt* and *outMPStoreExt* are represented, which hold, respectively, the input MP store objects coming from the plugin manager and the output MP store object to return to the plugin manager at the end of the computation.

Figure 6.12 depicts the *plugin manager* GUI which can be launched from the MetaPlab input GUI menu, and it enables the user to choose plugins from a list

and to run them. When MetaPlab is started up, the plugin manager automatically loads plugin files from a specific directory contained in the package. The *upper side* of the plugin manager window displays a list of the available plugins and a brief description of their features. Each plugin can be launched by selecting the related entry in the list and clicking the *run* button. Plugins can automatically load MP store objects from the *mpStoreExtList* vector of the plugin manager. Moreover, if a plugin saves its output into MP store objects and returns them to the plugin manager (which enqueues them in the *mpStoreExtList* vector), then further plugins can subsequently load them as an input and process it again, as graphically showed in Figure 6.13. Whenever a plugin computation stops, the plugin manager is displayed, in order to give the user the chance to launch another plugin.

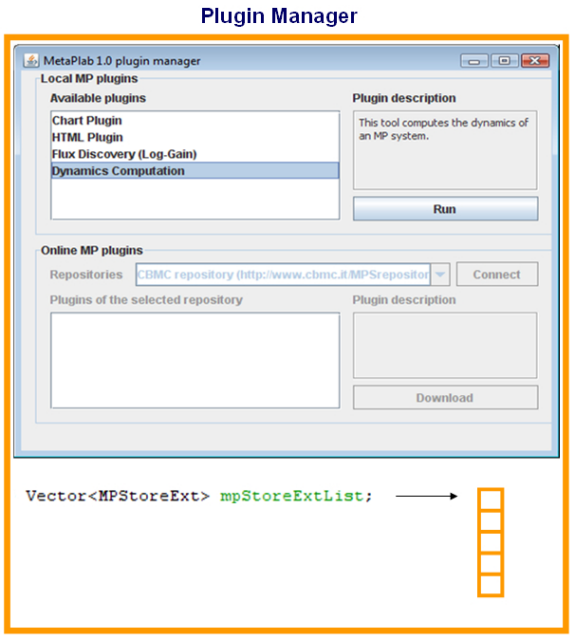


Fig. 6.12. MetaPlab Plugin Manager. In the upper side of the GUI the user can run plugins by choosing them from a list. The lower side of the GUI allows the user to download new plugins from forthcoming on-line repositories. In the bottom, vector *mpStoreExtList* is reported, which enable the plugin manager to store MP store objects coming both from the MetaPlab input GUI and from plugins.

The *lower side* of the plugin manager GUI will enable to upload new plugins from suitable repositories. Because of their intrinsic open and easy structure, MP plugins can be implemented (by following a few simple rules described below) by whoever wants to attack a specific modeling problem through MP systems. From this perspective, the forthcoming on-line repositories will enable the exchange of these computational tools among the MetaPlab users, in order to encourage their reuse. The MetaPlab website [241] already supports the manual download of new

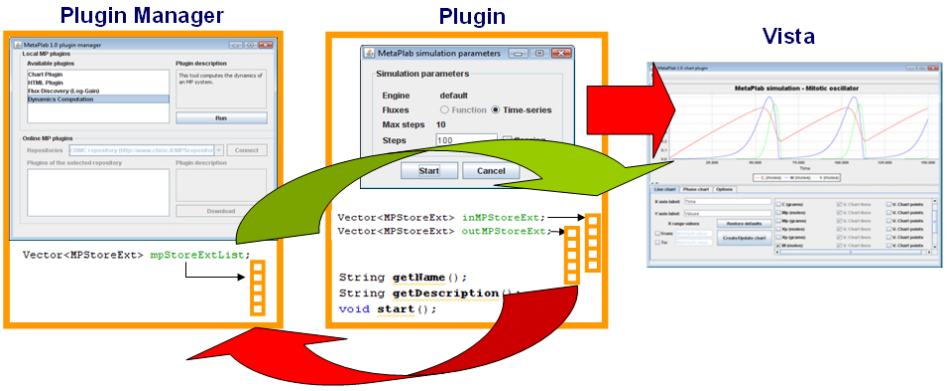


Fig. 6.13. Data computed by a plugin can be processed again by further plugins, as in a pipeline.

plugins and it provides a complete software documentation for each of these tools.

MP vistas. The fourth level of the architecture deals with the graphical representation of MP structures and MP dynamics. These views support the analysis of specific features of an MP system. A vista can, for instance, show substance and parameter charts together, plot phase diagrams or visualize statistical indexes about the entire dynamics.

Auxiliary modules. Two further modules are displayed at the bottom of Figure 6.8: the *MP store validator* and the *repository manager*. The first one is a Java library which assists plugin designers to check the MP store consistency. The second, which is currently under construction, will assist the user to systematically store and retrieve the results of many virtual experiments related the same MP model.

How to implement a plugin. Whoever wants to attack a new modeling problem by means of MetaPlab can develop new plugins by simply extending the abstract class *PluginExtAbs* (see Figure 6.14) released within the MetaPlab source package. Indeed, a plugin can involve one or more Java classes but it must have a startup class (extending *PluginExtAbs*) whose name corresponds to the plugin name. In this class three methods have to be implemented, namely, method *getName*, which returns a string of the plugin name, method *getDescription*, which returns a string describing the plugin main features, and method *start*, which executes the plugin computation. These three methods are called by the plugin manager, respectively, to show the plugin name and description in the plugin list, and to execute the plugin computation when the user clicks on the *run* button. The plugin startup class automatically inherits three fields from the abstract class *PluginExtAbs*: a vector of input MP stores (called *inMPStoreExt*) representing the input models and data, a vector of output MP stores (called *outMPStoreExt*) representing the output models and data, and a *caller* referring to the Java object which manages the plugin. Moreover, the plugin startup class also inherits from *PluginExtAbs* some methods

for managing communication and synchronization between the plugin and the rest of the application (i.e., methods *setCaller*, *notifyCaller*, *setInputMPStoreExt* and *getOutputMPStoreExt*). In this way, the developer can focus his/her attention only on the specific task the plugin has to accomplish (which has to be written in the *start* method), without dealing with secondary issues.

```
public abstract class PluginExtAbs implements PluginExt {
    protected Vector<MPStoreExt> inMPStoreExt;
    protected Vector<MPStoreExt> outMPStoreExt;
    protected MPSimPluginExt caller;

    public abstract String getName();
    public abstract String getDescription();
    public abstract void start();
    public String getType() {
        return ("Other");
    }
    public void setCaller(Object o) {
        caller=(MPSimPluginExt)o;
    }
    public void notifyCaller() {
        ((Restorable)caller).restoreCaller();
    }
    public String toString() {
        return (this.getName() + " - " + " - " + this.getDescription());
    }
    public void setInputMPStoreExt(Vector<MPStoreExt> mpse) {
        inMPStoreExt = mpse;
    }
    public Vector<MPStoreExt> getOutputMPStoreExt() {
        return outMPStoreExt;
    }
}
```

Fig. 6.14. Java abstract class PluginExtAbs. It implements some methods of the Java interface PluginExt (see Figure 6.15) while leaving the implementation of three plugin-dependent methods to the user.

Let us consider, for instance, the Java class *EmptyPlugin* displayed in Figure 6.16, which is the startup class of an homonymous (toy) plugin. It extends the abstract class *PluginExtAbs* thus inheriting (input, output and caller) fields and (communication/synchronization) methods as described above. Moreover, it implements four methods: a constructor, and the three abstract methods (*getName*, *getDescription* and *start*) of *PluginExtAbs*. In particular, method *getName* returns the string *"Empty plugin"*, which will be displayed in the plugin list of the plugin manager, method *getDescription* returns the string *"This plugin only prints a string"*, which will be displayed in the description text field of the plugin manager, and finally, method *start* prints the string *"This string is printed"* in the standard output and notifies its caller of the accomplished computation. Of course, real plu-

```
public interface PluginExt {
    public String getName();          /** Returns the plugin name. */
    public String getType();          /** Returns the plugin type */
    public String getDescription(); /** Returns the Plugin's use description. */
    public void start();              /** Gives the control to the Plugin. */
    public void setCaller(Object o);/** Sets the plugin's caller to o. */
        /** Notifies to the plugin's caller the end of the execution */
    public void notifyCaller();
        /** Returns a plug-in title for the visualization in the PluginManger.*/
    public String toString();
        /** Acquires the MPStoreExt currently loaded by the PluginManager */
    public void setInputMPStoreExt(Vector<MPStoreExt> mpse);
        /** Returns an array of output MPstoreExt */
    public Vector<MPStoreExt> getOutputMPStoreExt();
}
```

Fig. 6.15. Java interface PluginExt.

gins have much more complicated codes inside the *start* method, since they can access input MP stores data structure, process their data, launch graphical user interfaces for acquiring new data or displaying vistas, generate output MP stores, and so on.

Once the required methods have been implemented, the plugin must be compiled, obtaining a *.jar* file having the same name of the startup class (i.e., *EmptyPlugin.jar* in our toy example). This file has then to be copied in the *pluginsExt* directory of the MetaPlab binary distribution, so that it is automatically loaded by the plugin manager when the software will be relaunched. In order to launch this new plugin, the user has simply to load an MP graph by the MetaPlab input GUI, to run the plugin manager (by the specific button in the input GUI), to select the plugin name in the plugin list and to click the *run* button. As for the *EmptyPlugin*, the string *"This string is printed"* will be showed in the standard output.

Launching plugins in succession: the MetaPlab data flow. To complete this section we explain in more detail the mechanism for launching plugins in succession. This mechanism enables the user to work with virtual tools, namely plugins, in a similar way as biologists do with experimental tools in laboratory. Let us consider, for instance, a simulation plugin which computes the dynamics of MP models from initial conditions, and a chart plotting plugin able to plot MP dynamics. In order to use these two plugins in succession the user has to follow these steps. First of all, he/she creates or loads an MP model by the MetaPlab input GUI. This model, containing the stoichiometry, the regulation and the initial conditions of the system, is automatically stored in an MP store object. Then, the user launches the plugin manager, which acquires the MP store and it puts this object in an internal vector (see vector *mpStoreExtList* in Figures 6.12 and 6.13). Subsequently, the simulation plugin is launched and the MP store object is automatically passed from the plugin manager to the the input vector (called *inMPStoreExt* in Figures 6.11 and 6.14) of the simulation plugin, so that data can be employed by the plugin to compute the dynamics of the system. This dynamics is stored in a new MP store object and put in the output vector (called *outMPStoreExt* in Figures 6.11 and 6.14) of the simulation plugin. When the simulation

```
public class EmptyPlugin extends PluginExtAbs {
    /** Creates a new instance of EmptyPlugin */
    public EmptyPlugin() {
        System.out.println("[EmptyPlugin]: instance built");
    }
    /** Returns the plug-in name */
    public String getName() {
        return("Empty Plugin");
    }
    /** Returns a description of what the plug-in do */
    public String getDescription() {
        return ("This plugin only prints a string.");
    }
    /** Computes the plug-in's results */
    public void start() {
        System.out.println("This string is printed.");
        this.notifyCaller();
    }
}
```

Fig. 6.16. A toy example of MP plugin. When it is launched from the plugin manager it only prints a string on the standard output.

plugin finishes (and it calls the method *notifyCaller*), the plugin manager automatically loads the MP store objects in the output vector of the simulation plugin and enqueues them into its vector *mpStoreExtList*.

Now, the computation of the first plugin is finished and its output MP store is stored in the plugin manager, thus the chart plotting plugin has to be launched in order to graphically visualize these new data. When the user selects the chart plotting plugin, *the last MP store object loaded in vector mpStoreExtList* is passed to the plugin. In this way, the dynamics computed by the simulation plugin is plotted by the chart plotting plugin. Of course, this process can be iterated if more than two plugins has to be launched.

Notice that, the vector *mpStoreExtList*, is a kind of queue (contained in the plugin manager) which stores the history of plugin results obtained during a virtual experiment. In the current version of the architecture, plugins receive in input only the last MP store object put in the vector *mpStoreExtList*. Future extensions may provide a direct access to the vector *mpStoreExtList*, in order to enable the user to pass to plugins one or more MP stores selected from the intermediate results of a virtual experiment. In this way, it would be possible also to employ plugins which returns more than a single MP store, since the user could select which of them to process again and which to discard.

The architecture described so far enables to systematically tackle many problems about biological systems modeling by means of a set of MP plugins. In the next sections we review some of these problems and we present the plugin tool we implemented to automatically solve them.

6.2.3 Dynamics computation

Given an MP system in which the stoichiometry, the initial state and the regulation functions are known, one usually wants to compute the dynamics of the system for a certain number of steps. The standard way to generate this time evolution is to apply, at each computation step, the difference equations 4.2 and 4.3 which deterministically computes the next state of the system, i.e., substance concentrations and parameters values, from the current state.

The plugin tool we have developed to automatically generate the MP dynamics is called *simulation plugin* and its graphical interface is showed in Figure 6.17. It asks the user to set a few simulation parameters, such as, the number of steps to perform (i.e., field "*steps*"), an optional "*guard*", i.e., a proposition about some dynamics properties, which stops the computation when it becomes false, the name of the (optional) file where to save the achieved dynamics (i.e., field *save in*). The input of this plugin is an MP store data structure containing all the information about an MP model, the output is an MP store of the same MP model but containing also the computed dynamics of each substance and parameter. If the input MP store already contains t steps of a previously computed dynamics, the field *initial step* should be set to $t + 1$ in order to keep this evolution and to compute some more steps. Moreover, if one wants to store into the output MP store only the last t steps of the computed dynamics he/she has to set the *store last* field to t. MP flux objects sometimes contain regulation functions, while in other circumstances they hold only flux time-series. In the first case the *fluxes* radio button must be set to "*by function*", while in the second case it must be set to "*by time-series*", so that the right flux source is selected.

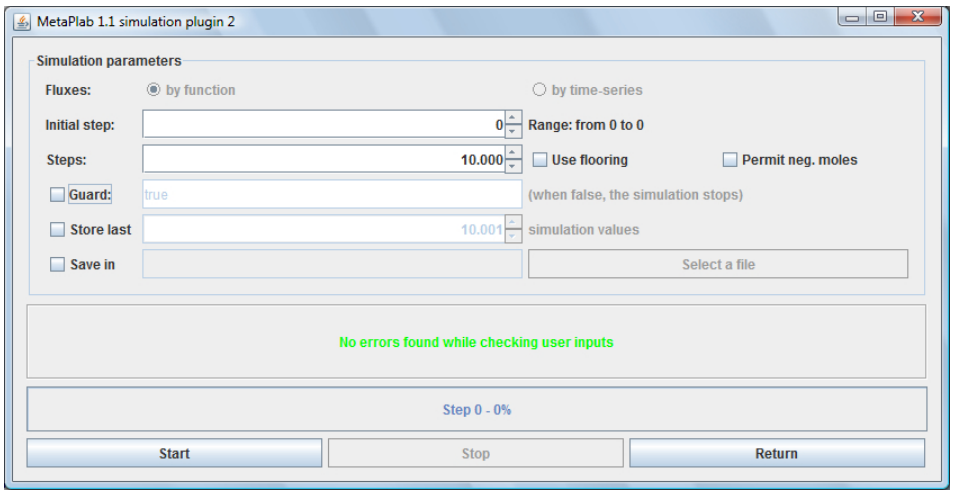


Fig. 6.17. The graphical user interface of the simulation plugin.

When all the simulation parameters have been set, the simulation can be launched by clicking on the *start* button. In this way, time-series of substances, parameters and fluxes are generated and stored into the output MP store, while

the progress bar at the bottom of the interface informs the user about the processing state. The user can then choose to perform another simulation or to return the result to the plugin manager by clicking the *return* button. A first analysis of the simulation results can be performed by closing the plugin manager and thus updating the MP graph in the input GUI with the newly generated dynamics. By clicking on each node of the MP graph, dynamics data can be browsed and visualized (by means of both tables and charts) in a network-oriented way. More accurate analyzes can be performed by the plugin presented in the next section.

6.2.4 Chart plotting

MetaPlab provides the *chart plotting* plugin for plotting together substance, parameter and flux time evolutions of an MP model. It takes in input an MP store object from the plugin manager and displays an MP vista of its dynamics. The graphical interface of this tool, showed in Figure 6.18, is split in two main areas: a 2D chart, on top, and a control area, at the bottom, where three tabbed panes enable the user to choose a chart type between *line chart* and *phase chart*, and to set the main *options* of the chart.

If the *line chart* is selected, the x axis represents the time instant and the y axis represents the values of the plotted entities, which can be substance concentrations, parameter values or flux values. In the left side of the bottom panel, the user can set the x-axis and the y-axis labels, and select the plotting time-interval. In the right side, the user can select which time-series to plot among all the substances, the parameters and the fluxes of the input MP model, and he/she can choose if to visualize only data points or also lines connecting data point.

On the other hand, if the *phase chart* tab is selected (at the bottom of Figure 6.18), the user has to choose the time-series to plot in the x and y axes and the visualization features (points chart or line chart) before generating the chart. Finally, the *option* tab opens a panel where the user can choose to sample the dynamics data with a certain step. Notice that, zooming facilities are available by selecting specific areas of the chart panel.

6.2.5 Flux discovery

The problem of flux discovery, already introduced in Section 4.6, represents a crucial issue of MP modeling, since it is the first step towards the synthesis of MP regulation functions from experimental data. This plugin computes flux time-series from observed substance and parameter time-series by means of the log-gain theory. In particular, this tool requires an MP model wherein *i)* the stoichiometry is completely known, and *ii)* each substance (and parameter) has an associated time series of observed concentrations (values). The plugin output is a copy of the input MP store containing also suitable flux time-series computed using the log-gain theory.

The user is asked to define, through the graphical interface of Figure 6.19, a set of *log-gain regulators* (also called tuners) for each reaction (on the left) and a set of reactions (called *offset parameters*) satisfying the "covering log-gain property" (on the right). The theoretical definition of these concepts is reported in

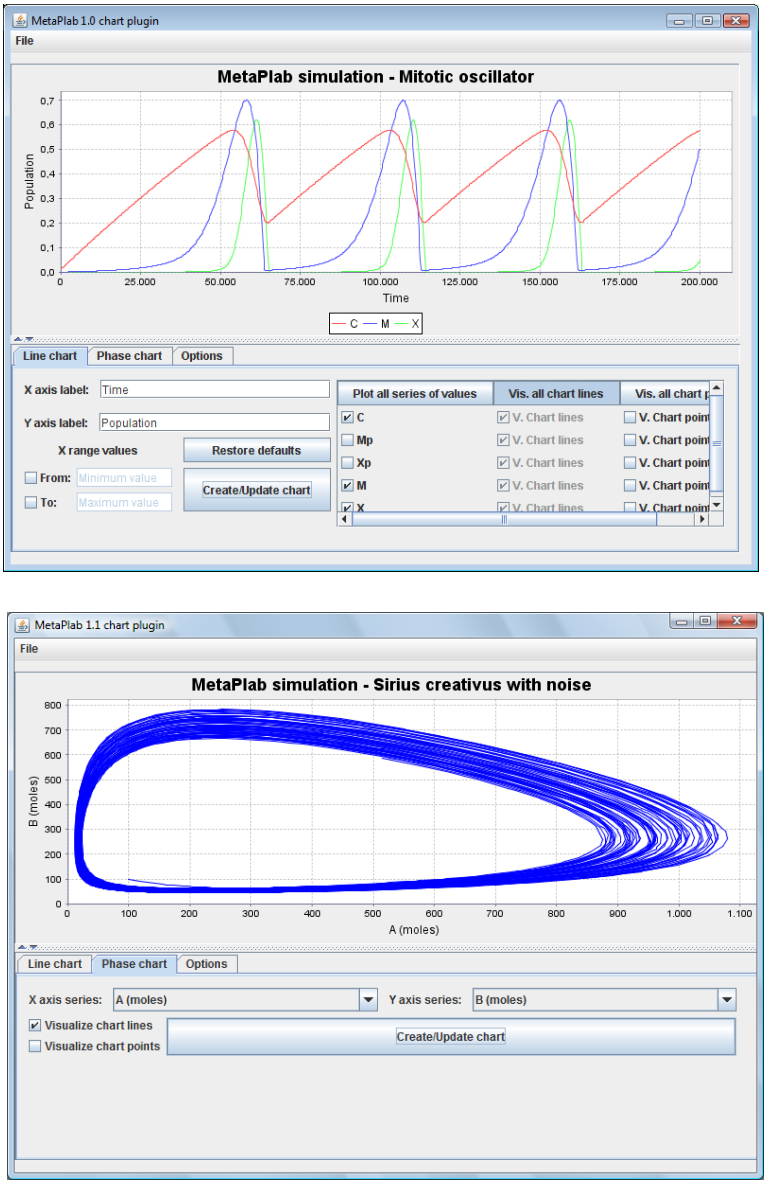


Fig. 6.18. On top: a line chart of the oscillations of the mitotic cycle in early amphibian embryos achieved by the chart plotting plugin. At the bottom: a phase chart of Sirio's dynamics.

Section 4.6. In order to select a regulator x for a reaction r, namely, a substance or a parameter which is directly involved in the regulation of flux φ_r, the user has to select the reaction r and the substance/parameter x from the combo-boxes on the top-left side of the interface and to click on the *add regulator* button. The *delete regulator* button enables to remove a regulator entry once it has been selected in

the regulator list (on the left-bottom side). On the right side, the user can "cover" a substance x with a reaction r by selecting the substance x and the reaction r from the combo-boxes and then clicking on the *add reaction* button. Also in this case, an offset parameter can be removed by selecting the related entry in the offset parameter list and clicking on the *delete reaction* button.

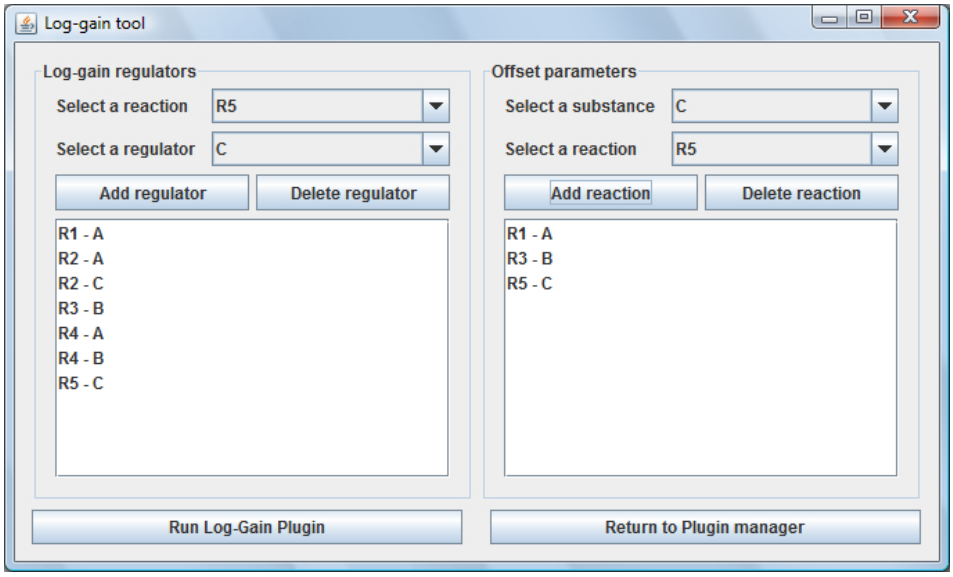


Fig. 6.19. Graphical interface of the flux discovery plugin.

Once regulators and offset parameters have been suitably defined, the *run log-gain plugin* button can be clicked, so that the plugin computes a set of flux time-series which makes the model evolve according to the observed dynamics. In particular, the tool automatically arranges and solves an equation system involving a substance difference module $SD[i+1]$ (see system 4.27 in Section 4.6) and an offset log-gain module $OLG[i]$ (see system 4.29 in Section 4.6) for each step i of the observed dynamics. The resulting flux time-series are stored in the appropriate fields of an output MP store which is returned to the plugin manager when the *return to the plugin manager* button is clicked. In this way they can be visualized through the MetaPlab input GUI and employed by other plugins, such as those presented in the next section, which need flux time-series to synthesize flux regulation functions. A complete description of the functionalities of the flux discovery plugin can be found in [146].

6.2.6 Regulation function synthesis by linear regression

The synthesis of MP regulation functions from datasets of observed substance and flux time-series is a regression problem which has been deeply investigated in Chapter 5. The choice of a good regression technique for solving this problem deeply depends on the knowledge one has about the form of the expected function.

In general, when fluxes are known to be a linear combination of substances and parameters, then linear regression analysis is employed, while if the relationship between substances/parameters and fluxes is not linear, then nonlinear regression is used. In Chapter 5 we have also introduced two main regression techniques for generating regulation functions, namely, linear regression [1] (see Section 5.2) and neural networks [24, 33] (see Section 5.3). In this and in the next subsection we, respectively, present two plugins with the aim to automate the application of these two regression methods within the MetaPlab environment.

The *Linear Regression tool* is an MP plugin which receives in input, from the plugin manager, an MP store object containing the stoichiometry of the system under investigation, and a time-series for each substance, parameter and flux. It generates, for each reaction r, a flux regulation function fitting the flux data and having the following form:

$$\varphi_r(q) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_z X_p \tag{6.1}$$

where $\beta_i,\ i = 0, \ldots, p$, are real coefficients computed by the least squares method (see Section 5.2 for more details), and $X_i,\ i = 0, \ldots, p$, are monomials obtained by raising substance and parameter variables to some real power and then possibly multiplying them together. For instance, a regulation function generated by the plugin for a reaction r of an MP model having three substances, i.e., A, B and C, and one flux, i.e., P, could be $\varphi_r(q) = -0.023 + 1.112A^2B - 8.432P$. The plugin output consists of a copy of the input MP store, in which flux objects contain also the newly computed regulation functions.

Linear regression plugin employs two graphical user interfaces. The first one appears as soon as the tool is launched from the plugin manager and it enables the user to generate parametric polynomials of the form of Equation 6.1 (they are said parametric since coefficients $\beta_i,\ i = 0, \ldots, z$ are unknown). The second interface is displayed when the generation of parametric polynomials is finished, and it enables to compute the polynomial coefficients by means of the least squares method. Now, let us start explaining the first GUI, which is displayed in Figure 6.20.

First interface: Polynomials generation. The first GUI of the linear regression plugin is composed of the four areas highlighted in Figure 6.20. By clicking on the topmost button, in the *workspace selection* area, we firstly select a workspace directory where the plugin automatically stores experiment files related to the MP model under investigation. Then, we select a reaction of the MP model from the combo-box in the *reaction selection* area (e.g., reaction $R1$ is selected in Figure 6.20). Afterward, we start to generate one or more parametric polynomials for the selected reaction. Every parametric polynomial related to a reaction r is stored in a text file called *r.txt* in the workspace directory. Each polynomial is a linear combination of monomials involving substance/parameter variables.

To insert a new substance (parameter) in a polynomial we select the substance (parameter) variable from the list called *substances and parameters*, in the *polynomial generator* area, we raise it to a proper power and we possibly multiply or add it to other substance and parameter variables by clicking, respectively, on the "*" button or on the the "+" button. While a polynomial is being written its elements are visualized step-by-step in a specific text box (see the text box containing the

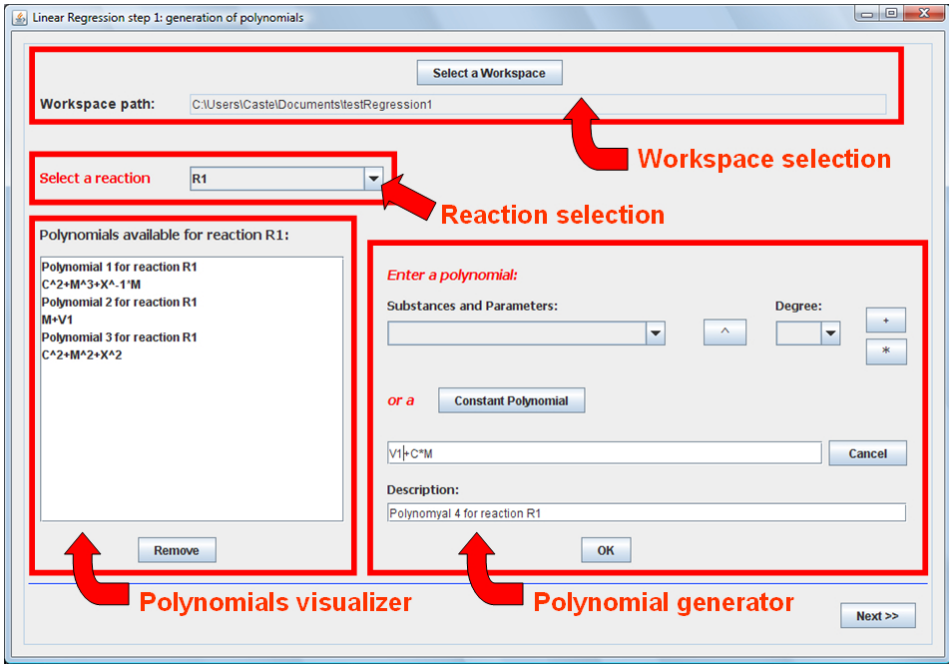


Fig. 6.20. First GUI of the linear regression plugin: generation of parametric polynomials.

polynomial *V1+C*M* in Figure 6.20). When every monomial has been inserted we write a polynomial description in the bottom text-box and we click on the *ok* button to add the new polynomial to the list of *polynomials available for reaction* R_i, where R_i depends on the reaction selected above. Polynomials can be deleted from this list by clicking the *remove* button in the *polynomials visualizer* area. The overall process has to be repeated for each reaction of the model, until each reaction is associated with at least one parametric polynomial. At that point, we click on the *next* button to open the second GUI.

Second interface: computation of polynomial coefficients. The second GUI is composed of three main areas, as displayed in Figure 6.21. In the first one (on top) we can easily select a parametric polynomial for each reaction, among the polynomials generated with the first GUI. We firstly select a reaction from the combo-box and then we use the *add* and *remove* buttons to select or unselect parametric polynomials. Once a specific set of polynomials has been selected, the *do regression* button has to be clicked to compute the coefficients which make those polynomials fit flux time-series in the MP store.

Results, namely optimized polynomials, are showed in the central area of the interface. The user can analyze these results and choose if to save them in a file together with a suitable description or to discard them performing a new regression process. Saved experiments are stored into a text file called *exp.txt* inside of the workspace directory, and they are indicated by a suitable entry in the list placed

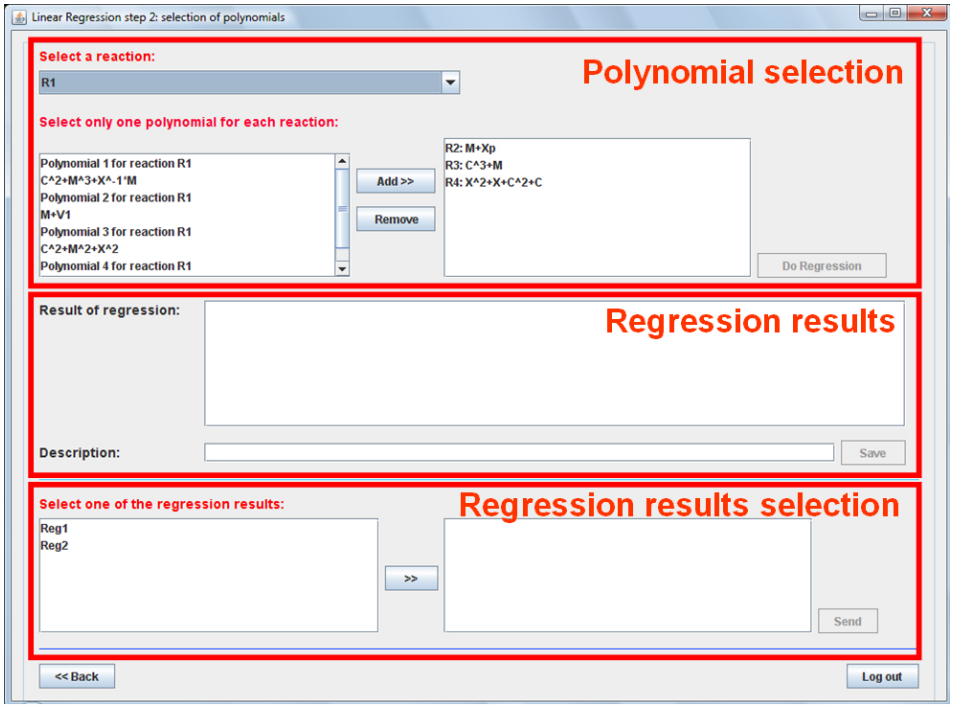


Fig. 6.21. Second GUI of the linear regression plugin: estimation of polynomial coefficients.

on the left-hand side of the *regression results selection* area. The user can select these entries, visualize the related polynomials (by clicking the ">>" button) and send them to the plugin manager.

Workspace management. The workspace directory selected from the first GUI contains two kinds of text files employed by the plugin: *i)* a file called *r.txt* for each reaction r, and *ii)* a file called *exp.txt*. Each file *r.txt* stores a list of parametric polynomials which can be used for computing regulation function φ_r by the least square method. File *exp.txt* stores all the results (i.e., sets of polynomials) obtained by the plugin for a specific MP model. In order to avoid improper functioning of the plugin, the user is asked not to manipulate files *r.txt* and *exp.txt* by text editors. Moreover, when a preexisting workspace is selected, the user has to make sure that it refers to the current MP model, otherwise some warning or error messages appear depending on the type of the anomaly.

6.2.7 Regulation function synthesis by optimized neural networks

The second plugin we have implemented for synthesizing flux regulation functions from observed time-series is called *NeuralSynth* [33]. Even if this plugin has the same aim of the previous one, the methodology employed to generate regulation functions is completely different. While the linear regression tool generates (linear)

regression functions having a polynomial form, NeuralSynth creates (nonlinear) regression functions represented by neural networks. The possibility to employ neural networks having just one hidden layer of sigmoid neurons to approximate any continuous functional mapping is proved in [76] thus, a regression technique based on this tool can, in principle, be employed even when the form of the required function is completely unknown (see Section 5.3 for more theoretical details).

The plugin input is an MP store object containing the stoichiometry of the system under investigation, and a time-series for each substance, parameter and flux. To generate flux regulation functions, substance and parameter time-series are provided to each neural network as inputs, while flux time-series are employed as target outputs. The plugin output is a copy of the input MP store, but containing also all the flux regulation functions (in the form of neural networks) which fit the flux time-series. The plugin enables the user to generate feed-forward neural networks, and then to automatically train them to fit flux data.

The plugin makes use of six graphical interfaces. The *main interface*, showed in Figure 6.22, appears as soon as the plugin is launched from the plugin manager. It simply enables the user to access the *neural network builder*, a tool for generating neural network topologies, and to choose one among four algorithms for learning these networks. By the neural network builder, displayed in Figure 6.23, the user can set up a single feed-forward neural network representing all the regulation functions of the MP model, or a feed-forward neural network for each regulation function. In the first case, a single neural network having m output neurons is generated (where m is the number of reactions in the MP system), in the latter case, the NN builder creates m networks having a single output neuron. To switch between the two modes the radio button on top of Figure 6.23 has to be used. Of course, the second mode requires a specific set up of every single network topology according to the type of the regulation function it has to represent. To do that, the user has to choose, from the proper combo-box, the reaction to whom the network refers, before starting to set up the network topology.

The current topology of the network under construction is displayed in the bottom side of the interface, by means of a table. Each row of the table represents a neuron layer, while the four columns of the table show, respectively, *i)* the sequential position of the layer, *ii)* the number of neurons in the layer, *iii)* the type of activation functions of the neurons in the layer, *iv)* the number of bias neurons in the layer. For instance, the neural network represented in the NN builder of Figure 6.23 has three layers: the input layer, in position 0, has three neurons with a linear activation function and one bias neuron; the hidden layer, in position 1, has three sigmoid layers and one hidden layer; the output layer, in position 2, has five sigmoid neurons and no bias neurons. Notice that, this network represents all the regulation functions of the MP model under investigation, indeed the radio button on top of Figure 6.23 is set to the left-hand side item. This is confirmed also by the fact that the network has more than one output neurons.

New neuron layers can be added to the network just by selecting the row of the table where the layer has to be inserted, by setting up the number of normal neurons, their type and the number of bias neurons from the combo-boxes placed under the network table, and clicking on the *add* button on the right. Neuron layers can be deleted by selecting a row of the table and clicking on the *delete*

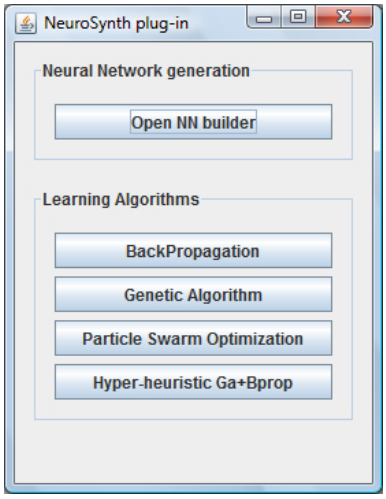


Fig. 6.22. NeuralSynth main GUI.

button. Moreover, neuron layer features can be updated by selecting a specific row, setting up the new features in the combo-box under the network table and by clicking on the *update* button.

The input of each neural network can be selected among the substances and the parameters of the MP system under investigation listed in the *tuners* section, situated in the central part of the NN builder. Initially, all the substances and parameters are selected and visualized in the list of *current tuners*, but tuners can be deleted by selecting them in the list and clicking on the *delete tuner* button. In order to add new tuners, the user has to select an additional tuner from the homonymous combo-box and to click on the *add tuner* button. Once the networks has been built, the user closes the NN builder and selects, in the main interface (see Figure 6.22), a learning algorithm. Depending on the selected strategy, a specific GUI opens, which enables to tune all the parameters of the specific learning algorithm and to start the training stage. As an example, we report in Figure 6.24 and 6.25 the GUIs employed for setting up the learning parameters of, respectively, backpropagation and genetic algorithm.

As for backpropagation, the user can choose to put all the time series data points in the training set (by selecting the *memorization* strategy) or to split them into a training set and a validation set (*generalization* strategy). Subsequently, he/she can set a training error threshold and a max number of epochs to be performed before stopping the training. The learning rate and the momentum can be set to single values or to ranges of values. In the second case all the combinations of parameters within the defined range are tested (with a grain chosen by the user). If needed, the weight elimination techniques can be employed by ticking the proper check box (see Chapter 5 for more theoretical details about parameter setting).

When all the learning parameters have been set, the user can choose to re-train every network more than one time, by typing the number of *restarts* in the homonymous text box placed on top of the *results* section. Then, a log file can be selected where the plugin will write all the results of the computation, the

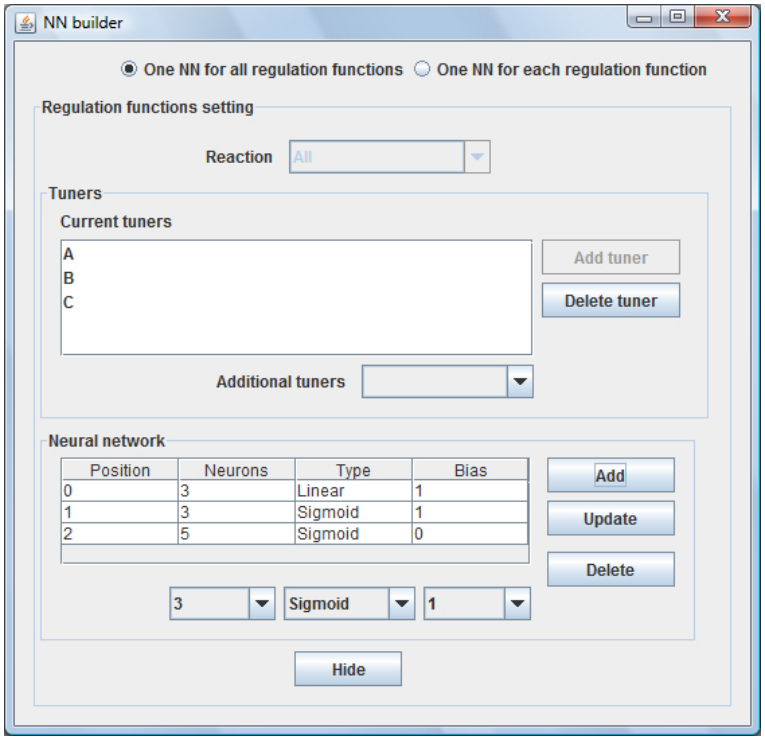


Fig. 6.23. NeuralSynth neural network builder.

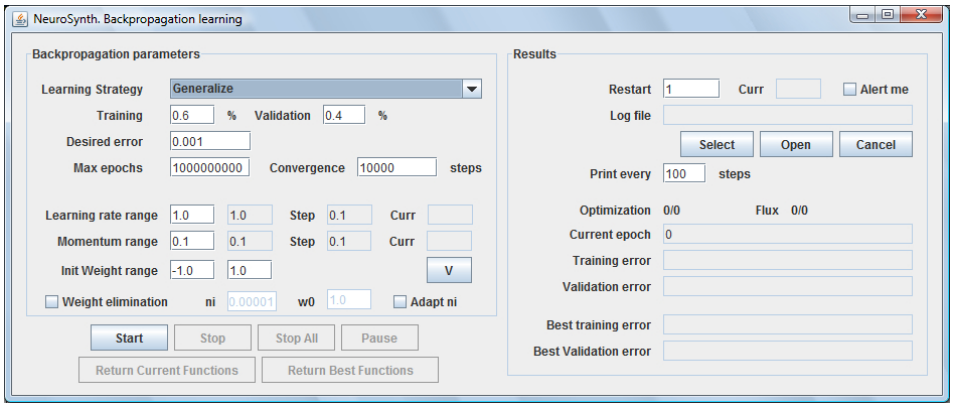


Fig. 6.24. NeuralSynth backpropagation learning GUI.

achieved regulation functions, the used parameters and the time elapsed during every computation. The training begins when the *start* button is clicked. The process evolution can be monitored from the text fields placed in the bottom part of the *results* section.

At the end of the computation the user can analyze the results of the training stored in the log file. It contains a header, storing the parameters employed by

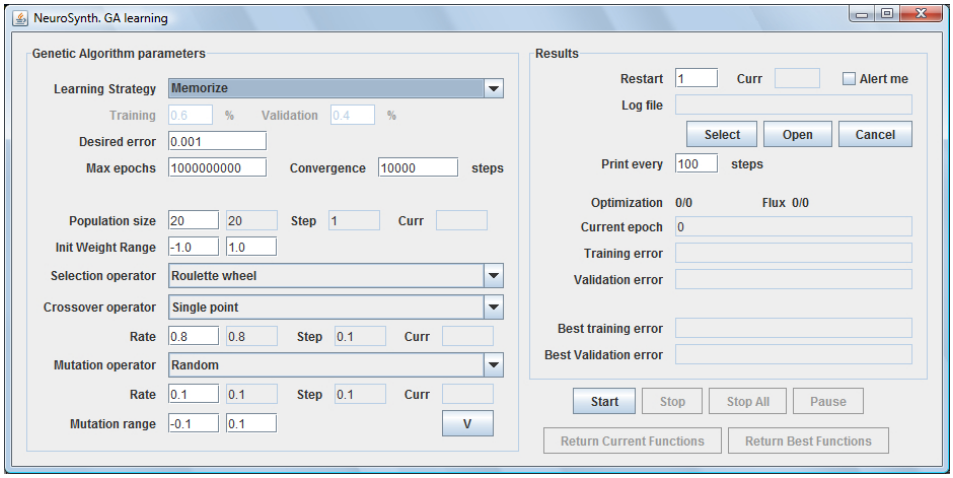


Fig. 6.25. NeuralSynth genetic algorithm learning GUI.

the learning algorithms and a list of all the regulation functions computed during the training stage, the time elapsed for computing them and some details about the topologies of the employed neural networks. The best regulation functions generated during the training process can be returned to the plugin manager within the output MP store, by clicking the *return best functions* button at the bottom of the *result* area.

In the following we report some tests performed to validate the learning algorithms implemented by NeuralSynth. The positive results obtained in these tests have been also confirmed by the successful application of the plugin to the case study of mitotic oscillator in early amphibian embryos, reported in Section 5.4 and presented for the first time in [33].

Validation of learning algorithms

The learning algorithms implemented by the NeuralSynth plugin, namely, back-propagation, GA, PSO and a memetic algorithm, have been validated by means of three test functions: *sine*, *sine with gaussian noise* (on the left-hand side of Figure 6.26) and two-dimension Michalewicz's function [158] (on the right-hand side of Figure 6.26). Each function has been sampled in order to generate a related training set. Then, these data sets have been submitted to every learning algorithm to assess its function synthesis capabilities and its performance over a large range of optimization parameters.

For each learning algorithm, Table 6.3 collects the results achieved from the synthesis of every test function. The best validation error (e) and the related number of steps (s) show, respectively, the accuracy of the best function found by a learning algorithm and the convergence speed of the algorithm. Given a specific algorithm (a column of Table 6.3) and a test function (a row of Table 6.3), the mean validation error (m) and the standard deviation (σ) have been computed over all the learning tests performed by different combinations of optimization parameters.

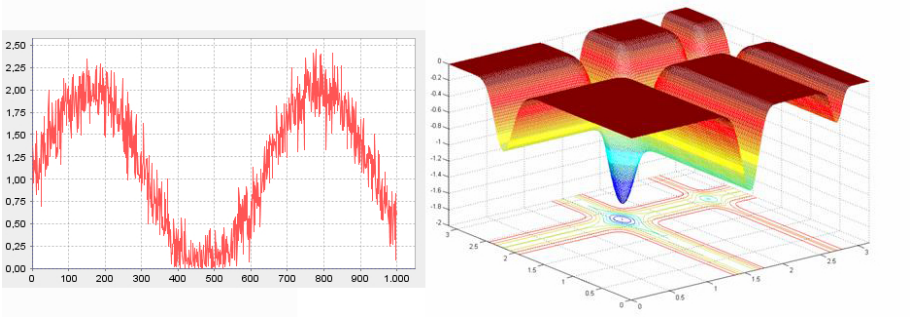


Fig. 6.26. On the left: sine function with gaussian noise ($\mu = 0$, $\sigma = 0.2$). On the right: two dimensional Michalewicz's function. Related functions synthesized by the NeuralSynth plugin are available in [153]

These values display the algorithm sensitiveness to learning parameter variations and the mean performance of the algorithm in term of accuracy.

The parameters employed by each algorithm in order to generate its best sine function are listed in the following:

- Backpropagation: learning rate $\eta = 1.0$ and momentum rate $\alpha = 0.2$.
- GA: rank selection, single point crossover (recombination rate=0.8), random addition mutation (muatation rate=0.1), elitist replacement and population of 20 chromosomes.
- PSO: inertia $w = 0.6$, cognitive coefficient $c_1 = 1.3$, social coefficient $c_2 = 1.7$ and swarm of 20 particles.
- Memetic algorithm. Ga: rank selection, double point crossover (recombination rate=0.8), random mutation (muatation rate=0.05), elitist replacement, population of 11 chromosomes, GA steps $n_1 = 100$. Backpropagation: learning rate $\eta = 0.8$, momentum rate $\alpha = 0.1$, backpropagation steps $n_2 = 1000$.

The best function achieved from the training set of sine with gaussian noise have been synthesized by the following learning parameters:

- Backpropagation: learning rate $\eta = 0.8$ and momentum rate $\alpha = 0.1$
- GA: rank selection, double point crossover (recombination rate=0.5), random addition mutation (mutation rate=0.15), elitist replacement and population of 20 chromosomes.
- PSO: inertia $w = 0.6$, cognitive coefficient $c_1 = 1.5$, social coefficient $c_2 = 1.7$ and a swarm of 20 particles.
- Memetic algorithm. Ga: rank selection, double point crossover (recombination rate=0.5), random mutation (mutation rate=0.1), elitist replacement, population of 11 chromosomes, GA steps $n_1 = 100$. Backpropagation: learning rate $\eta = 0.8$ and momentum rate $\alpha = 0.1$, backpropagation steps $n_2 = 1000$.

Finally, the best Michalewicz's approximation functions have been generated by the following learning parameters:

- Backpropagation: learning rate $\eta = 0.8$ and momentum rate $\alpha = 0.0$.

	Bprop	GA	PSO	Memetic
Sine	e: 2.105E-5 s: 10969 m: 2.210E-4 σ: 1.609E-4	e: 3.315E-4 s: 60000 m: 1.131E-3 σ: 3.802E-4	e: 1.252E-4 s: 60000 m: 1.215E-3 σ: 2.733E-3	e: 2.0634E-5 s: 22000 m: 9.204E-5 σ: 5.104E-5
Sine+noise	e: 5.432E-3 s: 10369 m: 6.581E-3 σ: 5.442E-4	e: 5.690E-3 s: 38214 m: 6.263E-3 σ: 3.851E-4	e: 5.291E-3 s: 60000 m: 7.155E-3 σ: 2.368E-3	e: 5.368E-3 s: 22000 m: 5.962E-3 σ: 3.306E-4
Michalewicz	e: 1.574E-4 s: 25694 m: 1.345E-3 σ: 9.228E-4	e: 1.539E-3 s: 60000 m: 3.097E-3 σ: 1.681E-3	e: 1.359E-3 s: 60000 m: 4.170E-3 σ: 4.072E-3	e: 1.347E-4 s: 22000 m: 1.826E-4 σ: 1.088E-4

Table 6.3. Tests: best validation errors (e) and related number of steps (s), mean validation error (m) and error standard deviation (sd).

- GA: tournament selection, double point crossover (recombination rate 0.7), random addition mutation (mutation rate 0.05), elitist replacement and population of 20 chromosomes.
- PSO: inertia $w = 0.6$, cognitive coefficient $c_1 = 1.5$, social coefficient $c_2 = 1.5$ and swarm of 20 particles.
- Memetic algorithm: Ga: rank selection, double point crossover (recombination rate=0.8), random mutation (mutation rate=0.1), elitist replacement, population of 11 chromosomes, GA steps $n_1 = 100$. Backpropagation: learning rate $\eta = 0.8$ and momentum rate $\alpha = 0.0$, backpropagation steps $n_2 = 1000$.

For every test, the pattern set T has been randomly split in two subsets: a training set T_1 including 60% of the samples and a validation set T_2 containing the remaining 40%. A neural network having two hidden layers of five neurons has been always employed. All the tests have been performed on a Intel(R) Core(TM)2 Duo CPU T7250, 2.00GHz processor having 2038 MB of memory.

Tests show interesting results for every learning technique. Synthesized functions (available at the web page [153]) approximate in a satisfying way the functions in question. Error seems to concentrate on minimum and maximum points. The memetic algorithm scored the best accuracy while sometimes it requires a greater number of steps than backpropagation. However, the joint employment of GA and backpropagation seems to give to the memetic technique both the ability to look for good solutions in a spread search space, and the speed of a gradient-based approach. Backpropagation achieved the second best results, while GA and PSO often could not perform a very fine tuning of their solutions. The number of steps required by the non-gradient-based algorithms is higher than backpropagation's and memetic algorithm's. The advantages of GA and PSO over backpropagation emerge when further constraints are considered by the fitness function making it nondifferentiable and multiobjective.

An example: Sirius

In the following a typical application of regulation functions synthesis by neural networks is analyzed by means of a very simple MP system, namely *Sirius* [140], which is displayed in Figure 6.27. This model has already been defined in Sections 4.6 and 5.5.2. Here we have sampled the dynamics of Figure 6.27 to obtain three substance time-series and we have computed the related five flux time-series by the log-gain theory. Subsequently, substance values have been used as training set inputs and flux values as outputs to train a neural network. NeuralSynth has been launched in order to automatically synthesize a set of regulation functions that give to Sirius the required oscillatory dynamics. A neural network having just one hidden layer with six hidden neurons has been employed.

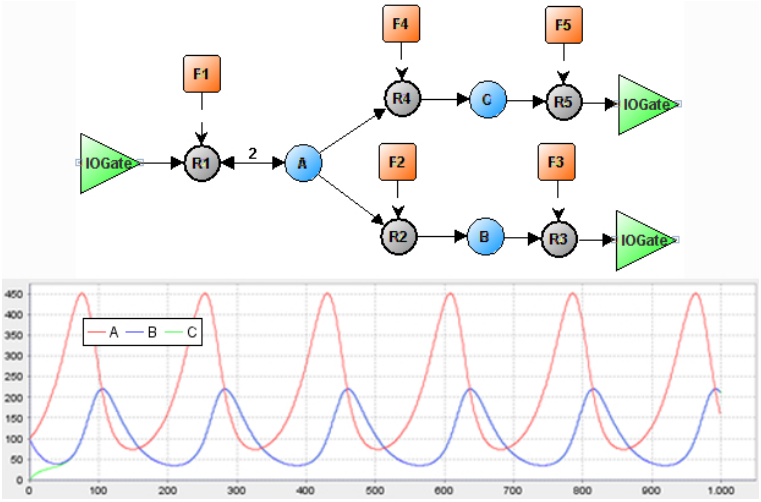


Fig. 6.27. On top: Sirius model. At the bottom: Sirius dynamics

Notice that, the results displayed in Figure 6.28 do not represent the synthesized regulation functions but rather the dynamics generated by employing those functions inside the Sirius MP model. Table 6.4 collects the errors and the number of steps required by each learning algorithm to compute the best regulation functions, while the employed learning parameters are listed below:

- Backpropagation: learning rate $\eta = 0.8$ and momentum rate $\alpha = 0.2$.
- GA: rank selection, single point crossover (recombination rate 0.7), random addition mutation (mutation rate 0.1), elitist replacement and population of 20 chromosomes.
- PSO: inertia $w = 0.6$, cognitive coefficient $c_1 = 1.3$, social coefficient $c_2 = 1.7$ and swarm of 20 particles.
- Memetic algorithm: Ga: rank selection, double point crossover (recombination rate=0.5), random mutation (mutation rate=0.05), elitist replacement, population of 7 chromosomes, GA steps $n_1 = 100$. Backpropagation: learning rate $\eta = 0.6$ and momentum rate $\alpha = 0.1$, backpropagation steps $n_2 = 3000$.

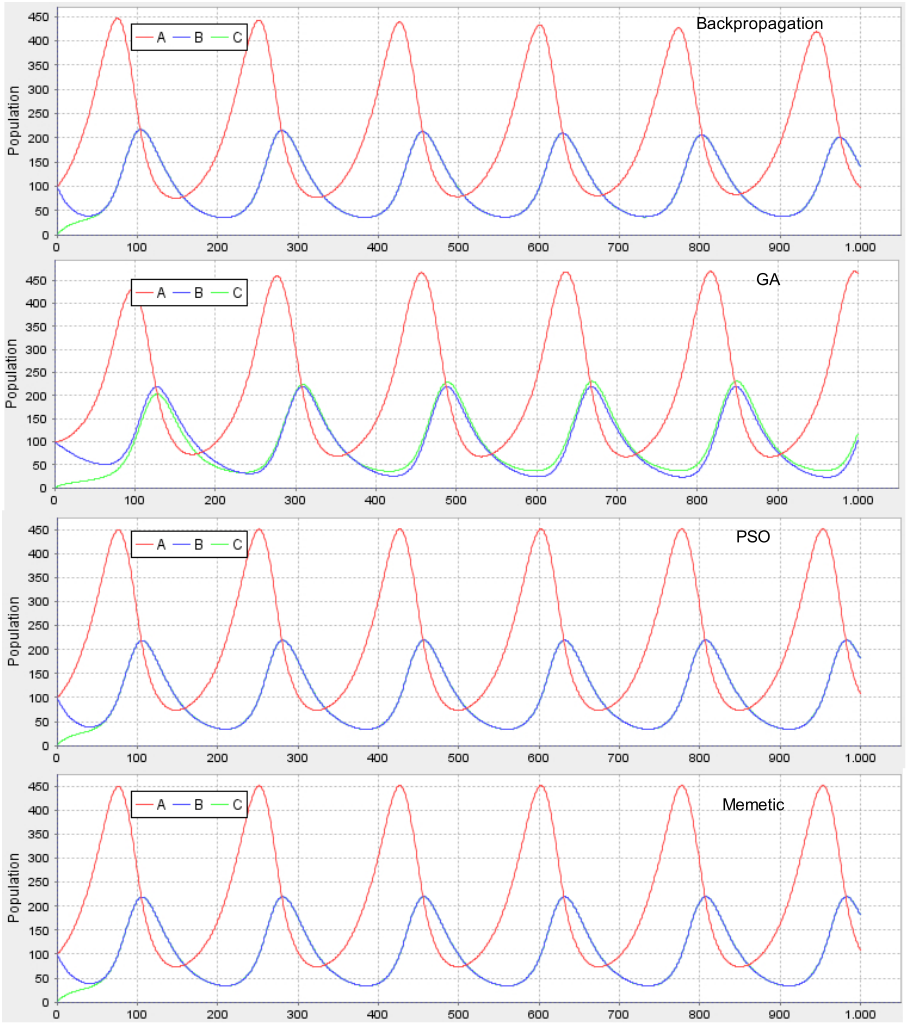


Fig. 6.28. Sirius dynamics generated by regulation functions synthesised, respectively, by backpropagation, GA, PSO and a memetic algorithm.

Bprop	GA	PSO	Memetic
e: 1.683E-5	e: 7.608E-4	e: 1.117E-4	e: 1.321E-5
s: 13800	s: 60000	s: 60000	s: 93000

Table 6.4. Sirius best errors (e) and number of steps (s).

Even in this case the memetic algorithm has synthesized functions having a better accuracy than backpropagation, PSO and GA, but it has performed a higher number of learning steps. All the algorithms however succeeded in synthesizing a set of regulation functions that generate an oscillatory dynamics. We remark that

no information about the function form has been employed but the maps have been completely synthesized from observations.

6.2.8 Integration with SBML

In order to have a textual rather than a graphical representation of MP systems, in [147] a way to export MP systems into suitable XML documents is defined, and a way to validate these representations through the definition of an XML Schema Document is studied. In that work, the attention has been focused on the definition of an XML document which permits the exportation of MP models as they are, without important modifications. However, a main format used in biological contexts is the *Systems Biology Markup Language* [243], which is an XML-based language designed for representing biological models in widely distributed simulation/analysis tools.

MetaPlab is equipped by a plugin which permits to automatically map an MP model to its SBML representation, giving an SBML document which can be imported by any software supporting SBML models importation (notice that, the author of the thesis is not directly involved in this research topic). Each type of component in a model is described in SBML by using a specific type of data object that organizes the relevant information. A high level SBML model definition consists of the list of elements reported in Table 6.1.

Since an MP system has an elementary membrane structure, the resulting SBML model, after an exportation procedure, gives a model with only one compartment (the skin membrane) where *i)* constants are mapped into SBML parameters, *ii)* substances are mapped into SBML species; *iii)* parameters are mapped into SBML parameters whose evolution in time is specified by an SBML assignment rule (if the MP parameter is defined by an evolution formula) or by a list of SBML events (if the MP parameter is defined by a time series of values); and *iv)* reactions are mapped into SBML reactions with kinetic laws expressed by the corresponding formulae of the flux regulation maps.

SBML models have components which seem to suggest a natural mapping into an MP model, but MP systems have been developed for discrete simulations while SBML have been designed to represent ODE systems. Hence, in order to guarantee an adequate mapping of the dynamics, MetaPlab SBML plugin assumes that the resulting SBML model will be simulated with a numerical integration time step equal to the MP model interval time.

MP parameters given by time series are mapped into an appropriate list of SBML events without delay. Each event must have a trigger condition which permits to fire at the sound simulation time. To implement this feature, the SBML plugin adds to the MP model a reaction increasing a time "counter" substance T of one unit at each step. More technically, each value $v_j[i]$ of a parameter time series is mapped by the SBML plugin into an event which assigns the value $v_j[i]$ to the parameter whenever the trigger condition $T = i$ holds. This method might be also employed to map substance time series, but it does not work for flux time series, because SBML events cannot modify kinetic laws. In this case, the SBML plugin first creates a new SBML parameter which will be used to define the kinetic law and then modifies the parameter by means of the above method.

The inverse process, consisting, for instance, in the importation in MetaPlab of a differential model from an SBML specification, assumes a greater importance. With this aim MetaPlab has been recently extended to compute the model dynamics not only according to EMA system (4.2), but also as a numerical integrator, with respect to some standard algorithms (such as Euler and Runge-Kutta), in order to study the dynamics of a differential model imported by an appropriate plugin (under development). For this reason it is essential to set a parameter called "time resolution" which specifies the discretization level of the numerical integration. This feature increases the interest for the MP-SBML combination. In fact, let we consider the case study in which we use a numerical integrator that computes the dynamics with a resolution of $\frac{1}{100}$ seconds. By using MetaPlab, in 100 steps we can easily get the unitary fluxes of each MP reaction. After this, it is possible to generate a temporal series of (unitary) flux arrays which are the input for a regression plugin (based on least squares method or neural networks) that gives us the definition of the flux regulation maps, and then a new MP model deduced from the differential one. These models will have the same behavior in each instant separated by a time interval of one second, but, the MP model might give more knowledge due to its different scale of systemic logic.

6.2.9 Other plugins

The process of plugin development, started straight after the implementation of the plugin architecture in 2008, has involved three Ph.D. students and two M.D. students at the University of Verona, and our hope is to expand this activity in the next future. Besides the plugins described through Subsections 6.2.3-6.2.9 also the following plugins have been implemented:

- HTML plugin,
- flux integrator plugin.

These plugins are available from version 1.2 of MetaPlab. A complete description of their aims and some operating instructions can be found at the MetaPlab website [241].

6.2.10 Future developments of MetaPlab

As described so far, MetaPlab provides many interesting tools and graphical functionalities for supporting MP modeling. Although the graphical user interface makes very simple to define and visualize models, it has also some drawbacks. For instance, it does not allow users to run processes on servers through the command line. In order to overcome this problem, the author is pondering the possibility to develop a new architecture in which plugins are released as web services. Service oriented architecture (SOA) would make plugins available to the entire scientific community which could freely access them via web by means of standard interfaces defined by the Web Service Description Language (WSDL). Several kinds of clients can connect to web services, such as tools implemented in Java, C, Visual Basic or Matlab, thus the plugin accessibility could be increased by means of this new technology. Moreover, web services would be run on a high-performance server

recently bought by the Center for BioMedical Computing in Verona, enabling the remote execution of complex tasks also from very small laptops.

Another important part of MetaPlab which should be improved is the model repository. Currently, each model generated by the user is stored in a local file which can contain also one dynamics (observed or simulated). Models are shared only by the MetaPlab website, where users can download MP models. In the future a public database of MP models and experiments could be released on our web server, which would enable a complete sharing of MP models and related experiments. In this database, models and experiments should be managed as separated entities, where each experiment should be linked to at least one (coherent) model. Such a model repository could enable to perform data-mining analysis when the database will reach large dimensions.

Author's publications for this chapter

- L. Bianco and A. Castellini. Psim: a computational platform for Metabolic P systems. In G. Eleftherakis, P. Kefalas, G. Păun, G. Rozenberg, and A. Salomaa, editors, *8th International Workshop on Membrane Computing, Lecture Notes in Computer Science 4860*, pages 1-20. Springer, 2007.
- A. Castellini and V. Manca. MetaPlab: A computational framework for metabolic P systems. In D. W. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *9th International Workshop on Membrane Computing, Lecture Notes in Computer Science 5391*, pages 157-168. Springer-Verlag, 2009.
- V. Manca, A. Castellini, G. Franco, L. Marchetti, and R. Pagliarini. Metaplab 1.1 user guide. Url: http://mplab.scienze.univr.it. 2009.

7

Conclusions

Understanding the mechanisms of life is an exciting challenge of our century. Biological systems are very complex networks involving intricate interconnections among a huge number of elements. The investigation of these systems requires advanced tools for representing, describing, visualizing and analyzing their main components, which may be substances and reactions of chemical systems, elements of ecosystems, or many other kinds of biological entities. Much work has been done in the last centuries for discovering and putting together the tiles of "life mosaic": in 1665 Robert Hooke coined the term "cell" to describe the basic unit of life, in 1953 Watson and Crick made their first announcement on the double-helix structure of DNA, and in 2001 a first draft of the human genome was published, just to mention a few stages of this amazing story. In the last decades many powerful tools have been developed as well, able to provide huge amounts of experimental data in shorter and shorter time.

Nowadays biology, medicine and pharmaceutics may take advantage of recently conceived mathematical and computational tools for analyzing these data, in the search for new treatments, drugs and knowledge about biological processes. The same tools are also used in synthetic biology for engineering new bacteria which accomplish specific functions, such as, producing useful substances or surviving in unusual conditions. This is why the development of mathematical and computational tools has recently became increasingly crucial.

One of the most important features for handling the high complexity of biological processes seems to be the possibility to observe these systems from an adequate abstraction level. Metabolic P systems are a new modeling framework, derived from P systems, which provides a macroscopic, global and time-discrete perspective on metabolic processes and related dynamics. Advantages of this approach are a natural mapping between real elements (i.e., substances, parameters and reactions) and model's elements, the possibility to adapt the model perspective to the temporal grain of observed data, and an original theory, called log-gain theory, for computing reaction fluxes from datasets of observed dynamics, without using kinetic constants usually hard to estimate.

In this thesis we have reported three main results: *i)* an equivalence between MP systems and hybrid functional Petri nets, *ii)* a pipeline for the generation of flux regulation maps from dynamics data by means of statistical techniques, classi-

cal regression and optimized neural networks, and *iii)* a software, called MetaPlab, which supports many stages of the MP modeling process.

The equivalence of MP systems with a formalism related to Petri nets results specially relevant in the framework of biological modeling since it highlights the representational and computational potentialities of MP systems. The mapping procedure has been tested on a fundamental biological process, i.e. the *glycolytic pathway controlled by the lac operon gene*, showing how a real process can be represented, simulated and analyzed by MP systems.

The second result, which represents the core of this work, provides modelers with a clear procedure consisting of five steps, and a set of statistical and optimization tools for generating flux regulation maps. Tests performed on the case studies of the *mitotic cycle in amphibian embryos* and the *non photochemical quenching phenomenon* have shown that neural networks are useful tools for representing regulation maps when the non-linearity of observed fluxes is significant, so that multiple linear regression would produce polynomials with many high-degree monomials. On the other hand, a limit of neural networks is the difficult interpretation of their (complex) structure from a biochemical point of view. Among the four optimization techniques implemented for training neural networks (i.e., backproagation, genetic algorithms, particle swarm optimization and a memetic algorithm), the memetic algorithm resulted to reach the best performance in terms of training error.

The new plugin-based architecture for the software MetaPlab turns out to be very useful for ensuring a continuous update and extension of this virtual laboratory. So far eight plugins have been implemented by three Ph.D. students and one M.D. student at the University of Verona, and our purpose is to expand this activity in the next future. The tools currently available allow to graphically generate MP models, to compute their dynamics, to plot time evolutions by charts, to discover reaction fluxes by the log-gain theory, to synthesize flux regulation functions by linear regression and neural networks, and to export MP model files to SBML and HTML.

Many research lines may be developed in the next future. The most crucial of them are:

- the comprehension of the effect of time grain on the complexity and readability of MP models. The peculiarity of MP systems is to model biological systems in a discrete timescale, which is different from the microscopic (continuous) timescale of ODE systems. It has been recently noticed that changing the time scale by which the system is observed, different aspects of the system logic can be highlighted. For instance, an MP model generated from a dataset having a time interval of 1 second may show different biological features with respect to an MP model having a time interval of 1 minute;
- the application of the MP modeling pipeline presented in Chapter 5 to new areas of biological and non-biological modeling. Some recent research interests are addressed to the insulin pathway, but also economical, financial or business systems could be analyzed with the aim to develop new kinds of models for economical or business intelligence applications [159];
- from the experience gained by working at this thesis, has emerged the crucial importance of statistical and optimization tools for the synthesis and the anal-

ysis of MP models. Our aim is to develop further techniques based on classical statistical methods but also on unconventional inferential methods, such as neural networks and evolutionary computing, for the analysis, prediction and classification of complex biological behaviors.

References

1. A. D. Aczel and J. Sounderpandian. *Complete business statistics*. McGraw-Hill/Irwin, 2006.
2. L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.
3. T. K. Ahn, T. J. Avenson, M. Ballottari, Y. C. Cheng, K. K. Niyogi, R. Bassi, and G. R. Fleming. Architecture of a charge-transfer state regulating light harvesting in a plant antenna protein. *Science*, 320(5877):794–797, 2008.
4. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Science, 2002.
5. A. Alhazov, M. Margenstern, V. Rogozhin, Y. Rogozhin, and S. Verlan. Communicative P systems with minimal cooperation. In G. Mauri, G. Păun, M. J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *5th International Workshop on Membrane Computing, Lecture Notes in Computer Science 3365*, pages 161–177. Springer-Verlag Berlin Heidelberg, 2005.
6. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.
7. K. Atkinson. *An Introduction to Numerical Analysis*. Wiley, 2nd edition, 1989.
8. R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.
9. P. B. Belousov. A periodic reaction and its mechanism. In *Sbornik Referatov po Radiatsionni Meditsine*, pages 145–147, 1959. In russian.
10. S. A. Benner and M. A. Sismour. Synthetic biology. *Nature Reviews Genetics*, 6(7):533–543, 2005.
11. D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler. GenBank. *Nucleic Acids Research*, 36, Database issue:D25–D30, 2008.
12. H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, 2000.
13. F. Bernardini and M. Gheorghe. Population P systems. *Journal of Universal Computer Science*, 10(5):509–539, 2004.
14. F. Bernardini, M. Gheorghe, and N. Krasnogor. Quorum sensing P systems. *Theoretical Computer Science*, 371(1-2):20–33, 2007.
15. D. Besozzi. *Computational and Modelling Power of P Systems*. PhD thesis, University of Milan, Milan, Italy, 2004.
16. D. Besozzi, P. Cazzaniga, G. Mauri, D. Pescini, and L. Vanneschi. A comparison of genetic algorithms and particle swarm optimization for parameter estimation in

stochastic biochemical systems. In C. Pizzuti, M. D. Ritchie, and M. Giacobini, editors, *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, 7th European Conference, EvoBIO 2009, Lecture Notes in Computer Science 5483*, pages 116–127. Springer, 2009.

17. L. Bianco. *Membrane Models of Biological Systems.* PhD thesis, University of Verona, Verona, Italy, 2007.
18. L. Bianco and A. Castellini. Psim: a computational platform for Metabolic P systems. In G. Eleftherakis, P. Kefalas, G. Păun, G. Rozenberg, and A. Salomaa, editors, *8th International Workshop on Membrane Computing, Lecture Notes in Computer Science 4860*, pages 1–20. Springer, 2007.
19. L. Bianco, F. Fontana, G. Franco, and V. Manca. P systems for biological dynamics. In Ciobanu et al. [44], pages 81–126.
20. L. Bianco, F. Fontana, and V. Manca. Reaction-driven membrane systems. In L. Wang, K. Chen, and Y. S. Ong, editors, *Advances in Natural Computation, First International Conference, ICNC 2005, Changsha, China, August 27-29, 2005, Part II*, volume 3611 of *Lecture Notes in Computer Science*, pages 1155–1158. Springer, 2005.
21. L. Bianco, F. Fontana, and V. Manca. P systems with reaction maps. *International Journal of Foundations of Computer Science*, 17(1):27–48, 2006.
22. L. Bianco, V. Manca, L. Marchetti, and M. Petterlini. Psim: a simulator for biomolecular dynamics based on P systems. In *IEEE Congress on Evolutionary Computation*, pages 883–887, Singapore, 2007.
23. L. Bianco, D. Pescini, P. Siepmann, N. Krasnogor, F. J. Romero-Campero, and M. Gheorghe. Towards a P systems pseudomonas quorum sensing model. In Hoogeboom et al. [98], pages 197–214.
24. C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.
25. U. Brandes and T. Erlebach, editors. *Network Analysis: Methodological Foundations (outcome of a Dagstuhl seminar, 13-16 April 2004)*, Lecture Notes in Computer Science 3418. Springer, 2005.
26. J. J. Burbaum and N. H. Sigal. New technologies for high-throughput screening. *Current Opinion in Chemical Biology*, 1(1):72 – 78, 1997.
27. E. K. Burke and G. Kendall. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques.* Springer, 2005.
28. K. P. Burnham and D. R. Anderson. *Model selection and multimodel inference: a practical information-theoretic approach.* Springer, 2nd edition, 2002.
29. A. Castellini and G. Franco. On modeling signal transduction networks. In *Sixth Brainstorming Week on Membrane Computing, BWMC6*, pages 67–77, Sevilla, Spain, 2008. Fenix Editora.
30. A. Castellini, G. Franco, and V. Manca. Toward a representation of hybrid functional Petri nets by MP systems. In Y. Suzuki, M. Hagiyra, H. Umeo, and A. Adamatzky, editors, *Natural computing*, volume 1 of *Proceedings in Information and Communications Technology*, pages 28–37. Springer Japan, 2009.
31. A. Castellini, G. Franco, and V. Manca. Hybrid functional Petri nets as MP systems. *Natural Computing*, 9(1):61–81, 2010.
32. A. Castellini, G. Franco, and R. Pagliarini. Data analysis pipeline from laboratory to MP models. *Natural Computing*, 2010. DOI: 10.1007/s11047-010-9200-6. In print.
33. A. Castellini and V. Manca. Learning regulation functions of metabolic systems by artificial neural networks. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO '09*, pages 193–200, Montreal, Québec, Canada, 2009. ACM Publisher.

34. A. Castellini and V. Manca. MetaPlab: A computational framework for metabolic P systems. In D. W. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *9th International Workshop on Membrane Computing, Lecture Notes in Computer Science 5391*, pages 157–168. Springer-Verlag, 2009.
35. A. Castellini, V. Manca, and L. Marchetti. MP systems and Hybrid Petri Nets. In *Studies in Computational Intelligence 129 (NICSO 2007)*, pages 53–62. Springer, 2008.
36. A. Castellini, V. Manca, and Y. Suzuki. Metabolic P system flux regulation by artificial neural networks. In G. Păun, M. J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, and A. Salomaa, editors, *10th Workshop on Membrane Computing, Lecture Notes in Computer Science 5957*, pages 196–209. Springer-Verlag Berlin Heidelberg, 2010.
37. M. Cavaliere. Evolution-communication P systems. In G. Păun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Workshop on Membrane Computing, Curtea de Arges, Romania, Lecture Notes in Computer Science 2597*, pages 134–145. Springer, 2003.
38. P. Cazzaniga, D. Pescini, D. Besozzi, and G. Mauri. Tau leaping stochastic simulation method in P systems. In Hoogeboom et al. [98], pages 298–313.
39. S. Cheruku, A. Păun, F. J. Romero-Campero, M. J. Pérez-Jiménez, and O. H. Ibarra. Simulating FAS-induced apoptosis by using P systems. *Progress in Natural Science*, 17(4):424–431, 2007.
40. D. Y. Cho, K. H. Cho, and B. T. Zhang. Identification of biochemical networks by S-tree based genetic programming. *Bioinformatics*, 22(13):1631–1640, 2006.
41. S. Choi, editor. *Introduction to systems biology.* Humana press, 2007.
42. T. Cibas, F. Fogelman Soulié, P. Gallinari, and S. Raudys. Variable selection with optimal cell damage. In *Proceedings of ICANN'94*, pages 727–730. Springer-Verlag, 1994.
43. T. Cibas, F. Fogelman Soulié, P. Gallinari, and S. Raudys. Variable selection with neural networks. *Neurocomputing*, 12(2–3):223–248, 1996.
44. G. Ciobanu, G. Păun, and M. J. Pérez-Jiménez, editors. *Applications of Membrane Computing.* Natural Computing Series. Springer-Verlag Berlin, 2006.
45. R. Cohen, S. Havlin, and D. Ben-Avraham. Structural properties of scale free networks. In S. Bornholdt and H. G. Schuster, editors, *Handbook of graphs and networks*, chapter 4. Wiley-VCH, 2002.
46. E. J. Crampin, S. Schnell, and P. E. McSharry. Mathematical and computational techniques to deduce complex biochemical reaction mechanisms. *Progress in Biophysics and Molecular Biology*, 86(1):77–112, 2004.
47. Y. Le Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605. Morgan Kaufmann, 1990.
48. R. David and H. Alla. *Discrete, Continuous, and Hybrid Petri Nets.* Springer, 1st edition, 2004.
49. E. Demir, O. Babur, U. Dogrusoz, A. Gursoy, G. Nisanci, R. Cetin-Atalay, and M. Ozturk. PATIKA: an integrated visual environment for collaborative construction and analysis of cellular pathways. *Bioinformatics*, 18(7):996–1003, 2002.
50. K. Dhar, M. T. Chee, S. Sandeep, Y. Li, S. Kishore, K. Arun, B. M. Ridwan, W. S. H. Wah, C. Mandar, and H. Zhu. Grid cellware: the first grid-enabled tool for modelling and simulating cellular processes. *Bioinformatics*, 21(7):1284–1287, 2005.
51. P. Dhar, T. C. Meng, S. Somani, L. Ye, A. Sairam, M. Chitre, Z. Hao, and K. Sakharkar. Cellware–a multi-algorithmic software for computational systems biology. *Bioinformatics*, 20(8):1319–1321, 2004.

52. P. Dittrich, J. Ziegler, and W. Banzhaf. Artificial chemistries - a review. *Artificial Life*, 7:225–275, 2001.
53. A. Doi, S. Fujita, H. Matsuno, M. Nagasaki, and S. Miyano. Constructing biological pathway models with hybrid functional Petri nets. *In Silico Biology*, 4(3):271–291, 2004.
54. M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 26(1):29–41, 1996.
55. N. R. Draper and H. Smith. *Applied Regression Analysis*. Wiley Series in Probability and Statistics. John Wiley & Sons Inc, 3rd edition, 1998.
56. P. du Jardin. Bankruptcy prediction and neural networks: the contribution of variable selection methods. In *Proceedings of European Symposium on Time Series Prediction '08*, pages 271–284, 2008.
57. L. Edelstein-Keshet. *Mathematical Models in Biology*. Society for Industrial & Applied, 2005.
58. M. A. Efroymson. Multiple regression analysis. *Mathematical Methods for Digital Computers*, 1:191–203, 1960.
59. M. A. Efroymson. Stepwise regression - a backward and forward look. *Easter Regional Meetings of the Inst. of Meth. Statist., Florham Park, New Jersey*, 1966.
60. D. Endy. Foundations for engineering biology. *Nature*, 438:449–453, 2005.
61. Y. Evron and R.E. McCarty. Simultaneous measurement of deltapH and electron transport in chloroplast thylakoids by 9-aminoacridine fluorescence. *Plant Physiology*, 124:407–414, 2000.
62. C. Ferretti, G. Mauri, G. Păun, and C. Zandron. On three variants of rewriting P systems. *Theoretical Computer Science*, 301(1-3):201–215, 2003.
63. R. P. Feynman. There's plenty of room at the bottom. In D. H. Gilbert, editor, *Miniaturisation*, pages 282–296. Reinhold, 1961.
64. R. J. Field, E. Koros, and R.M. Noyes. Oscillations in chemical systems. II. Through analysis of temporal oscillation in the bromate-cerium-malonic acid system. *Journal of the American Chemical Society*, 94:8649–8664, 1972.
65. R. J. Field and R. M. Noyes. Oscillations in chemical systems. IV. Limit cycle behavior in a model of a real chemical reaction. *The Journal of Chemical Physics*, 60(5):1877–1884, 1974.
66. J. Fisher and T. A. Henzinger. Executable cell biology. *Nature Biotechnology*, 25(11):1239–1249, 2007.
67. F. Fontana, L. Bianco, and V. Manca. P systems and the modeling of biochemical oscillations. In R. Freund, G. Păun, G. Rozenberg, and A. Salomaa, editors, *6th International Workshop on Membrane Computing, Lecture Notes in Computer Science 3850*, pages 199–208. Springer, 2006.
68. F. Fontana and V. Manca. Discrete solutions to differential equations by metabolic P systems. *Theoretical Computer Science*, 372(2-3):165–182, 2007.
69. F. Fontana and V. Manca. Predator-prey dynamics in P systems ruled by metabolic algorithm. *BioSystems*, 91(3):545–557, 2008.
70. Center for BioMedical Computing web site. Url: http://www.cbmc.it.
71. G. Franco, P. H. Guzzi, V. Manca, and T. Mazza. Mitotic oscillators as MP graphs. In Hoogeboom et al. [98], pages 382–394.
72. G. Franco, V. Manca, and R. Pagliarini. Regulation and covering problems in MP systems. In G. Păun, M. J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, and A. Salomaa, editors, *10th Workshop on Membrane Computing, Lecture Notes in Computer Science 5957*. Springer-Verlag, 2010.
73. R. Freund, L. Kari, M. Oswald, and P. Sosík. Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science*, 330(2):251–266, 2005.

74. R. Freund, G. Păun, and M. J. Pérez-Jiménez. Tissue P systems with channel states. *Theoretical Computer Science*, 330(1):101–116, 2005.
75. P. Frisco. *Computing with Cells: Advances in Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2009.
76. K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989.
77. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and A. Riscos-Núñez. An overview of P-lingua 2.0. In G. Păun, M. J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, and A. Salomaa, editors, *10th Workshop on Membrane Computing, Lecture Notes in Computer Science 5957*, pages 264–288. Springer-Verlag Berlin Heidelberg, 2010.
78. M. Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123, 1970.
79. D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22:403–434, 1976.
80. D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
81. D. T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics*, 115(4):1716–1733, 2001.
82. D. T. Gillespie. Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, 58:35–55, 2007.
83. A. Gisselsson, A. Szilagyi, and H. Akerlund. Role of histidines in the binding of violaxanthin de-epoxidase to the thylakoid membrane as studied by site-directed mutagenesis. *Physiologia Plantarum*, 122:337–343, 2004.
84. G. Goel, I. Chou, and E. O. Voit. System estimation from metabolic time-series data. *Bioinformatics*, 24(21):2505–2511, 2008.
85. A. Goldbeter. A minimal cascade model for the mitotic oscillator involving cyclin and cdc2 kinase. *PNAS*, 88(20):9107–9111, 1991.
86. T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, 1999.
87. O. R. Gonzalez, C. Küper, K. Jung, P. C. Naval, and E. Mendoza. Parameter estimation using simulated annealing for S-system models of biochemical networks. *Bioinformatics*, 23(4):480–486, 2007.
88. M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and D. Ramírez-Martínez. A software tool for verification of spiking neural P systems. *Natural Computing*, 7(4):485–497, 2008.
89. I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.
90. G. Hardin. The tragedy of the commons. *Science*, 162(3859):1243–1248, 1968.
91. W. Härdle. *Applied Nonparametric Regression*. E-book, 1994.
92. S. Hashem. Sensitivity analysis for feedforward artificial neural networks with differentiable activation functions. In *Proceedings of the 1992 International Joint Conferences on Neural Networks (IJCNN92)*, pages 419–424. IEEE Press, 1992.
93. R. Heinrich and S. Schuster. *The Regulation Of Cellular Systems*. Chapman & Hall, New York, 1996.
94. R. R. Hocking. The analysis and selection of variables in linear regression. *Biometrics*, 32(1):1–49, 1976.
95. R. Hofestädt. A Petri net application to model metabolic processes. *Journal of System Analysis, Modeling and Simulation*, 16(2):113–122, 1994.

96. R. Hofestädt and S. Thelen. Quantitative modeling of biochemical networks. *In Silico Biology*, 1:39–53, 1998.
97. J. H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, MI, 1975.
98. H. J. Hoogeboom, G. Păun, G. Rozenberg, and A. Salomaa, editors. *7th International Workshop on Membrane Computing, Lecture Notes in Computer Science 4361.* Springer, 2006.
99. S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074, 2006.
100. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 2nd edition, 2000.
101. M. Hucka, A. Finney, B. J. Bornstein, S. M. Keating, B. E. Shapiro, J. Matthews, B. L. Kovitz, M. J. Schilstra, A. Funahashi, J. C. Doyle, and H. Kitano. Evolving a lingua franca and associated software infrastructure for computational systems biology: the Systems Biology Markup Language (SBML) project. *Systems Biology*, 1(1):41–53, 2004.
102. M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J. H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novère, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. and Wang. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.
103. M. Ionescu, G. Păun, and Takashi Yokomori. Spiking neural P systems. *Fundamenta Informaticae*, 71(2-3):279–308, 2006.
104. A.J. Izenman. *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning.* Springer Publishing Company, Incorporated, 2008.
105. J. Jack and A. Păun. Discrete modeling of biochemical signaling with memory enhancement. In *Transactions on Computational Systems Biology XI, Lecture Notes in Computer Science 5750*, pages 200–215. Springer-Verlag, 2009.
106. J. Jack, A. Păun, and A. Rodriguez-Paton. Effects of HIV-1 proteins on the Fas-mediated apoptotic signaling cascade: A computational study of latent CD4+ T cell activation. In D. W. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *9th International Workshop on Membrane Computing, Lecture Notes in Computer Science 5391*, pages 227–246. Springer-Verlag, 2009.
107. J. Jack, A. Rodriguez-Paton, O. H. Ibarra, and A. Păun. Discrete nondeterministic modeling of the Fas pathway. *International Journal of Foundations of Computer Science*, 19(5):1147–1162, 2008.
108. J. Jack, F. J. Romero-Campero, M. J. Pérez-Jiménez, O. H. Ibarra, and A. Păun. Simulating apoptosis using discrete methods: a membrane system and a stochastic approach. In *Proceedings of International Conference of Unconventional Computation, UC'07, Kingston, Ontario, Canada*, pages 50–63, 2007.
109. D. S. Jones and B. D. Sleeman. *Differential Equations and Mathematical Biology.* Chapman & Hall/CRC Mathematical Biology and Medicine, 2003.
110. K. Kakimoto and T. Taura. A framework for analyzing sustainability by using the rewriting system. In *Proceedings of EcoDesign '03: the 3rd International Symposium on Environmentally Conscious Design and Inverse Manufacturing*, pages 69–74. IEEE, 2003.

111. A. Kanazawa and D. M. Kramer. In vivo modulation of nonphotochemical exciton quenching (NPQ) by regulation of the chloroplast ATP synthase. *PNAS*, 99(20):12789–12794, 2002.
112. M. Kanehisa, M. Araki, S. Goto, M. Hattori, M. Hirakawa, M. Itoh, T. Katayama, S. Kawashima, S. Okuda, T. Tokimatsu, and Y. Yamanishi. KEGG for linking genomes to life and the environment. *Nucleic Acids Research*, 36, Database issue:D480–D484, 2007.
113. M. Kanehisa and S. Goto. KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 28(1):27–30, 2000.
114. L. Kari and G. Rozenberg. The many facets of natural computing. *Commun. ACM*, 51(10):72–83, 2008.
115. J. Keizer. *Statistical Thermodynamics of Nonequilibrium Processes*. Springer, New York, 1987.
116. J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.
117. T. R. Kiehl, R. M. Mattheyses, and M. K. Simmons. Hybrid simulation of cellular behavior. *Bioinformatics*, 20(3), 2004.
118. S. Kikuchi, D. Tominaga, M. Arita, K. Takahashi, and M. Tomita. Dynamic modeling of genetic networks using genetic algorithm and S-system. *Bioinformatics*, 19(5):643–50, 2003.
119. H. Kitano. Computational systems biology. *Nature*, 420:206–210, 2002.
120. H. Kitano, A. Funahashi, Y. Matsuoka, and K. Oda. Using process diagrams for the graphical representation of biological networks. *Nature Biotechnology*, 23(8):961–966, 2005.
121. S. Klamt, J. Stelling, M. Ginkel, and E. D. Gilles. FluxAnalyzer: exploring structure, pathways, and flux distributions in metabolic networks on interactive flux maps. *Bioinformatics*, 19(2):261–269, 2003.
122. R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.
123. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, 1992.
124. J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic Programming IV : Routine Human-Competitive Machine Intelligence (Genetic Programming)*. Springer, 2003.
125. N. Krasnogor and J. E. Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.
126. S. N. Krishna, R. Rama, and H. Ramesh. Further results on contextual and rewriting P systems. *Fundamenta Informaticae*, 64(1-4):241–253, 2005.
127. T. Kulikova, R. Akhtar, P. Aldebert, N. Althorpe, M. Andersson, A. Baldwin, K. Bates, S. Bhattacharyya, L. Bower, P. Browne, M. Castro, G. Cochrane, K. Duggan, R. Eberhardt, N. Faruque, G. Hoad, C. Kanz, C. Lee, R. Leinonen, Q. Lin, V. Lombard, R. Lopez, D. Lorenc, H. Mcwilliam, G. Mukherjee, F. Nardone, M. P. Pastor, S. Plaister, S. Sobhany, P. Stoehr, R. Vaughan, D. Wu, W. Zhu, and R. Apweiler. EMBL nucleotide sequence database in 2006. *Nucleic Acids Research*, 35, Database issue:D1–D5, 2007.
128. P. Leray and P. Gallinari. Feature selection with neural networks. *Behaviormetrika*, 26:16–6, 1998.
129. H. Li, Y. Cao, L. R. Petzold, and D. T. Gillespie. Algorithms and software for stochastic simulation of biochemical reacting systems. *Biotechnology Progress*, 2007.
130. A. Lindenmayer. Mathematical models for cellular interactions in development i. filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280–299, 1968.

131. P lingua web site. Url: http://www.p-lingua.org.
132. D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985.
133. B. Liu, S. Li, and J. Hu. Technological advances in high-throughput screening. *American Journal of Pharmacogenomics*, 4(4):263–276, 2004.
134. J. A. Lotka. *Elements of Physical Biology*. Williams and Wilkins Company, 1924.
135. D. J. C. MacKay. Bayesian non-linear modelling for the prediction competition. In *ASHRAE Transactions*, volume 100, pages 1053–1062. ASHRAE, 1994.
136. V. Manca. MP systems approaches to biochemical dynamics: Biological rhythms and oscillations. In Hoogeboom et al. [98], pages 86–99.
137. V. Manca. Topics and problems in metabolic P systems. In G. Păun and M. J. Pérez-Jiménez, editors, *Fourth Brainstorming Week on Membrane Computing, BWMC4*, pages 173–184, Fenix Editora, Sevilla, Spain, 2006.
138. V. Manca. Metabolic P systems for biochemical dynamics. *Progress in Natural Sciences,* Invited Paper, 17(4):384–391, 2007.
139. V. Manca. Discrete simulations of biochemical dynamics. In M. H. Garzon and H. Yan, editors, *13th International Meeting on DNA Computing, DNA13, Lecture Notes in Computer Science 4848*, pages 231–235. Springer, 2008.
140. V. Manca. The metabolic algorithm: Principles and applications. *Theoretical Computer Science*, 404:142–157, 2008.
141. V. Manca. Fundamentals of metabolic P systems. In G. Păun, G. Rozenberg, and A. Salomaa, editors, *Handbook of Membrane Computing*, chapter 19. Oxford University Press, 2009.
142. V. Manca. Log-gain principles for metabolic P systems. In A. Condon, D. Harel, J.N. Kok, A. Salomaa, and E. Winfree, editors, *Algorithmic Bioprocesses*, Natural Computing Series, chapter 28. Springer, 2009.
143. V. Manca. Metabolic P dynamics. In G. Păun, G. Rozenberg, and A. Salomaa, editors, *Handbook of Membrane Computing*, chapter 20. Oxford University Press, 2009.
144. V. Manca and L. Bianco. Biological networks in metabolic P systems. *BioSystems*, 91(3):489–498, 2008.
145. V. Manca, L. Bianco, and F. Fontana. Evolutions and oscillations of P systems: Applications to biochemical phenomena. In G. Mauri, G. Păun, M. J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *5th International Workshop on Membrane Computing, Lecture Notes in Computer Science 3365*, pages 63–84. Springer-Verlag Berlin Heidelberg, 2005.
146. V. Manca, A. Castellini, G. Franco, L. Marchetti, and R. Pagliarini. Metaplab 1.1 user guide. Url: http://mplab.scienze.univr.it. 2009.
147. V. Manca and L. Marchetti. XML representation of MP systems. In *IEEE Congress on Evolutionary Computation, CEC 2009*, pages 3103–3110. Trondheim, Norway, 18-21 May 2009, 2009.
148. V. Manca, R. Pagliarini, and S. Zorzan. A photosynthetic process modelled by a metabolic P system. *Natural Computing*, 8(4):847–864, 2009.
149. V. Manca, R. Pagliarini, and S. Zorzan. Toward an MP model of non-photochemical quenching. In D. W. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *9th International Workshop on Membrane Computing, Lecture Notes in Computer Science 5391*, pages 299–310. Springer-Verlag, 2009.
150. C. Martín-Vide and G. Păun. String-objects in P systems. In *Proceedings of Algebraic Systems, Formal Languages and Computations Workshop*, pages 161–169, Kyoto, 2000. RIMS Kokyuroku, Kyoto University.
151. C. Martín-Vide, G. Păun, and G. Rozenberg. Membrane systems with carriers. *Theoretical Computer Science*, 270(1-2):779–796, 2002.

152. T. F. Massoud, G. J. Hademenos, W. L. Young, E. Gao, J. Pile-Spellman, and F. Viñuela. Principles and philosophy of modeling in biomedical research. *FASEB Journal*, 12(3):275–285, 1998.
153. External material for this Ph.D. thesis.
Url: http://mplab.scienze.univr.it/external/thesiscastellini/index.html.
154. H. Matsuno, Y. Tanaka, H. Aoshima, A. Doi, M. Matsui, and S. Miyano. Biopathways representation and simulation on Hybrid Functional Petri Nets. *In Silico Biology*, 3(3):389–404, 2003.
155. K. Maxwell and G.N. Johnson. Chlorophyll fluorescence - a practical guide. *Journal of Experimental Botany*, 51(345):659–668, 2000.
156. D. A. McQuarrie. Stochastic approach to chemical kinetics. *Journal of Applied Probability*, 4(3):413–478, 1967.
157. P. Mendes and D. Kell. Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation. *Bioinformatics*, 14(10):869–883, 1998.
158. Z. Michalewicz. *Genetic algorithms + data structures = evolution programs, 3rd ed.* Springer-Verlag, 1996.
159. Z. Michalewicz, M. Schmidt, M. Michalewicz, and C. Chiriac. *Adaptive Business Intelligence*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
160. T. Mitchell. *Machine Learning*. McGraw-Hill Education, ISE Editions, 1997.
161. J. E. Moody and J. Utans. Principled architecture selection for neural networks: Application to corporate bond rating prediction. In *Neural Information Processing Systems 4*, pages 683–690. Morgan Kaufmann, 1992.
162. J. Murray. *Mathematical biology: I. An introduction*. Springer, 2005.
163. M. Mutyam and K. Krithivasan. Improved results about universality of P systems. *Bulletin of the EATCS*, 76:162–168, 2002.
164. M. Nagasaki, A. Doi, H. Matsuno, and S. Miyano. Genomic object net: I. A platform for modelling and simulating biopathways. *Applied Bioinformatics*, 2(3):181–184, 2004.
165. A. Navot, L. Shpigelman, N. Tishby, and E. Vaadia. Nearest neighbor based feature selection for regression and its application to neural activity. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18 (NIPS 2005)*, pages 995–1002. MIT Press, Cambridge, MA, 2006.
166. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 1st edition, 2000.
167. N. Le Novere and S. T. Shimizu. Stochsim: modelling of stochastic biomolecular processes. *Bioinformatics*, 17(6):575–576, 2001.
168. R. Pagliarini, G. Franco, and V. Manca. An algorithm for initial fluxes of metabolic P systems. *International Journal of Computers, Communications & Control*, IV(3):263–272, 2009.
169. W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Science USA*, 85(8):2444–2448, 1988.
170. M. J. Pérez-Jiménez and F. J. Romero-Campero. A study of the robustness of the EGFR signalling cascade using continuous membrane systems. In J. Mira and J. R. Álvarez, editors, *First International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2005, Lecture Notes in Computer Science 3561*, pages 268–278, 2005.
171. M. J. Pérez-Jiménez and F. J. Romero-Campero. P systems: a new computational modelling tool for systems biology. *Transactions on Computational Systems Biology VI, Lecture Notes in Bioinformatics, 4220*, pages 176–197, 2006.
172. D. Pescini, D. Besozzi, G. Mauri, and C. Zandron. Dynamical probabilistic P systems. *International Journal of Foundations of Computer Science*, 17(1):183–204, 2006.

173. D. Pescini, D. Besozzi, C. Zandron, and G. Mauri. Analysis and simulation of dynamics in probabilistic P systems. In A. Carbone and N. A. Pierce, editors, *11th International Meeting on DNA Computing, DNA11, Lecture Notes in Computer Science 3892*, pages 236–247. Springer-Verlag Berlin Heidelberg, 2006.
174. C. A. Petri. *Kommunikation mit automaten.* Bonn: Institut fur Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.
175. R. D. Present. *Kinetic Theory of Gases.* McGraw-Hill, New York, 1958.
176. W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C.* Cambridge University Press, Cambridge, UK, 2nd edition, 1992.
177. I. Prigogine. *From Being to Becoming: Time and Complexity in the Physical Sciences.* Freeman and Company, San Francisco, CA, 1980.
178. A. Păun. On P systems with active membranes. In I. Antoniou, C. Calude, and M.J. Dinneen, editors, *Unconventional Models of Computation*, pages 187–201, London, 2000. Springer-Verlag. Contributed paper.
179. A. Păun. P systems with string-objects: Universality results. In *Pre-Proceedings of Workshop on Membrane Computing*, Curtea de Arges, Romania, 2001.
180. A. Păun, M. J. Pérez-Jiménez, and F. J. Romero-Campero. Modeling signal transduction using P systems. In Hoogeboom et al. [98], pages 100–122.
181. A. Păun and G. Păun. The power of communication: P systems with symport/antiport. *New Generation Computing*, 20(3):295–306, 2002.
182. G. Păun. Computing with membranes. Technical Report 208, Turku Centre for Computer Science, 1998.
183. G. Păun. P systems with active membranes: Attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–90, 1999.
184. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
185. G. Păun. *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
186. G. Păun, M. J. Pérez-Jiménez, J. Pazos, and A. Rodríguez-Patón. Symport/antiport P systems with three objects are universal. *Fundamenta Informaticae*, 64(1-4):353–367, 2004.
187. G. Păun, Y. Suzuki, and H. Tanaka. P systems with energy accounting. *International Journal of Computer Mathematics*, 78(3):343–364, 2001.
188. D. Pyle. *Data Preparation for Data Mining (The Morgan Kaufmann Series in Data Management Systems).* Morgan Kaufmann, 1999.
189. V. N. Reddy, M. L. Mavrovouniotis, and M. N. Liebman. Petri net representations in metabolic pathways. In J. W. Shavlik L. Hunter, D. B. Searls, editor, *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology*, pages 328–336. AAAI Press, 1993.
190. P.N. Refenes, A.D. Zapranis, and J. Utans. Neural model identification, variable selection and model adequacy. In *Neural Networks in Financial Engineering, Proceedings of NnCM-96*, 1996.
191. W. Reisig. *Petri nets: an introduction.* EATCS, Monographs on Theoretical Computer Science. Springer-Verlag New York, 1985.
192. I. Rivals and L. Personnaz. MLPs (mono layer polynomials and multi layer perceptrons) for nonlinear modeling. *Journal of Machine Learning Research*, 3:1383–1398, 2003.
193. T. M. Robinson. *Heraclitus: Fragments.* University of Toronto Press, Toronto, Canada, 1987.
194. F. J. Romero-Campero. *P systems, a computational modelling framework for systems biology.* PhD thesis, University of Sevilla, Sevilla, Spain, 2008.
195. F. J. Romero-Campero, H. Cao, M. Camera, and N. Krasnogor. Structure and parameter estimation for cell systems biology models. In *Proceedings of the 10th*

Annual Conference on Genetic and Evolutionary Computation, GECCO '08, pages 331–338, Atlanta, GA, USA, 2008. ACM Publisher.
196. F. J. Romero-Campero and M. J. Pérez-Jiménez. A model of the quorum sensing system in vibrio fischeri using P systems. *Artificial Life*, 14(1):95–109, 2008.
197. F. J. Romero-Campero and M. J. Pérez-Jiménez. Modelling gene expression control using P systems: Lac operon, a case study. *BioSystems*, 91(3):438–457, 2008.
198. F. J. Romero-Campero, J. Twycross, H. Cao, J. Blakes, and N. Krasnogor. A multi-scale modeling framework based on P systems. In D. W. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *9th International Workshop on Membrane Computing, Lecture Notes in Computer Science 5391*, pages 63–77. Springer-Verlag, 2009.
199. G. Rozenberg. Computer science, informatics and natural computing personal reflections. In *New Computational Paradigms: Changing Conceptions of What Is Computable*, pages 373–379. Springer, 2008.
200. G. Rozenberg, A. E. Eiben, and J. N. Kok. Preface. *Theoretical Computer Science*, 287:1–2, 2002.
201. G. Rozenberg and A. Salomaa, editors. *Handbook of formal languages, vol. 1: word, language, grammar*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.
202. D. W. Ruck, S. K. Rogers, and M. Kabrisky. Feature selection using a multilayer perceptron. *Neural Network Computing*, 2(2):40–48, 1990.
203. T. P. Ryan. *Modern Engineering Statistics*. Wiley-Interscience, 2007.
204. H. M. Sauro, M. Hucka, A. Finney, C. Wellock, H. Bolouri, J. Doyle, and H. Kitano. Next generation simulation tools: the systems biology workbench and biospice integration. *OMICS*, 7(4):355–372, 2003.
205. M. A. Savageau. Biochemical systems analysis, I. Some mathematical properties of the rate law for the component enzymatic reactions. *Journal of Theoretical Biology*, 25(3):365–369, 1969.
206. M. A. Savageau. Biochemical systems analysis, II. The steady-state solutions for an n-pool system using a power-law approximation. *Journal of Theoretical Biology*, 25(3):370–379, 1969.
207. M. A. Savageau. Biochemical systems analysis, III. Dynamic solutions using a power-law approximation. *Journal of Theoretical Biology*, 26(2):215–226, 1970.
208. M. A. Savageau. *Biochemical Systems Analysis: A Study of Function and Design in Molecular Biology*. Addison Wesley Publishing Company, 1976.
209. M. A. Savageau and E. O. Voit. Recasting nonlinear differential equations as S-systems: a canonical nonlinear form. *Mathematical Biosciences*, 87:83–115, 1987.
210. E. Schrödinger. *What is Life? : With Mind and Matter and Autobiographical Sketches (Canto)*. Cambridge University Press, Cambridge, UK, 1992.
211. G. A. F. Seber and C. J. Wild. *Nonlinear Regression*. Wiley, 2003.
212. N. M. Shnerb, Y. Louzoun, E. Bettelheim, and S. Solomon. The importance of being discrete: Life always wins on the surface. *PNAS*, 97(19):10322–10324, 2000.
213. N. Soranzo, G. Bianconi, and C. Altafini. Comparing association network algorithms for reverse engineering of large-scale gene regulatory networks. *Bioinformatics*, 23(13):1640–1647, 2007.
214. J. B. Stanbury, J. B. Wyngaarden, D. S. Fredrickson, J. L. Goldstein, and M. S. Brown, editors. *The Metabolic Basis of Inherited Disease*. McGraw-Hill, New York, 5th edition, 1983.
215. H. Sugawara, O. Ogasawara, K. Okubo, T. Gojobori, and Y. Tateno. DDBJ with new system and face. *Nucleic Acids Research*, 36, Database issue:D22–D24, 2008.
216. Y. Suzuki. An investigation of the brusselator on the mesoscopic scale. *International Journal of Parallel, Emergent and Distributed Systems*, 22(2):91–102, 2007.

217. Y. Suzuki, J. Takabayashi, and H. Tanaka. Investigation of tritrophic interactions in an ecosystem using abstract chemistry. *Journal of Artificial Life and Robotics*, 6(3):129–132, 2002.
218. Y. Suzuki and H. Tanaka. Order parameter for a symbolic chemical system. In *ALIFE: Proceedings of the sixth international conference on Artificial life*, pages 130–139, Cambridge, MA, USA, 1998. MIT Press.
219. Y. Suzuki and H. Tanaka. Chemical evolution among artificial proto-cells. In *Artificial Life VII*, pages 54–64. MIT Press, 2000.
220. Y. Suzuki and H. Tanaka. Modeling p53 signaling pathways by using multiset processing. In Ciobanu et al. [44], pages 203–214.
221. Y. Suzuki, S. Tsumoto, and H. Tanaka. Analysis of cycles in symbolic chemical system based on abstract rewriting system on multisets. In *Proceedings of the International Conference on Artificial Life V*, pages 482–489. MIT press, 1996.
222. MP systems on Scholarpedia.
Url: http://www.scholarpedia.org/articles/Metabolic_P_systems.
223. P systems web site. Url: http://ppage.psystems.eu.
224. K. Takahashi, N. Ishikawa, Y. Sadamoto, H. Sasamoto, S. Ohta, A. Shiozawa, F. Miyoshi, Y. Naito, Y. Nakayama, and M. Tomita. E-Cell 2: multi-platform E-CELL simulation system. *Bioinformatics*, 19(13):1727–1729, 2003.
225. C. Talcott and D. L. Dill. The pathway logic assistant. In *Proceedings of the Workshop Computational Methods in Systems Biology (CMSB)*, 2005.
226. M. Tomita, K. Hashimoto, K. Takahashi, T. S. Shimizu, Y. Matsuzaki, F. Miyoshi, K. Saito, S. Tanida, K. Yugi, J. C. Venter, and C. A. Hutchison. E-CELL: software environment for whole-cell simulation. *Bioinformatics*, 15(1):72–84, 1999.
227. K. Torkkola. Feature extraction by non-parametric mutual information maximization. *Journal of Machine Learning Research*, 3:1415–1438, 2003.
228. V. Tresp, R. Neuneier, and G. Zimmermann. Early brain damage. In *Advances in Neural Information Processing Systems 9, NIPS 1996*, pages 669–675. MIT Press, 1997.
229. B. V. Trubitsin and A. N. Tikhonov. Determination of a transmembrane pH difference in chloroplasts with a spin label tempamine. *Journal of Magnetic Resonance*, 163:257–269, 2003.
230. M. Umeki and Y. Suzuki. A simple membrane computing method for simulating bio-chemical reactions. *Computing and Informatics*, 27(3):529–550, 2008.
231. M. Umeki and Y. Suzuki. Simulation of the oregonator model by using deterministic arms. *Journal of Artificial Life and Robotics*, 13(2):551–554, 2009.
232. L. Vanneschi and S. Silva. Using operator equalisation for prediction of drug toxicity with genetic programming. In L. S. Lopes, N. Lau, P. Mariano, and L. M. Rocha, editors, *Progress in Artificial Intelligence, 14th Portuguese Conference on Artificial Intelligence, EPIA 2009, Lecture Notes in Computer Science 5816*, pages 65–76. Springer, 2009.
233. E. Voit, A. R. Neves, and H. Santos. The intricate side of systems biology. *PNAS*, 103(25):9452–9457, 2006.
234. E. O. Voit. *Computational Analysis of Biochemical Systems: A Practical Guide for Biochemists and Molecular Biologists*. Cambridge University Press, 2000.
235. V. Volterra. Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. memorie della classe di scienze fisiche, matematiche e naturali. *Memorie Academia dei Lincei*, 2:31–113, 1926.
236. L. von Bertalanffy. *General Systems Theory: Foundations, Developments, Applications*. George Braziller Inc., New York, NY, 1967.
237. Cell Illustrator Project web site. Url: http://www.genomicobject.net.
238. Extensible Markup Language web site. Url: http://www.w3.org/xml/.

239. GNU General Public Licence web site.
Url: http://www.gnu.org/copyleft/gpl.html.
240. Infobiotics Workbench web site.
Url: http://www.infobiotic.org/infobiotics-workbench/index.html.
241. MetaPlab web site. Url: http://mplab.scienze.univr.it.
242. Systems Biology Graphical Notation web site. Url: http://www.sbgn.org.
243. Systems Biology Markup Language web site. Url: http://sbml.org.
244. Systems Biology Workbench web site. Url: http://sbw.sourceforge.net.
245. A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In R. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *NIPS*, pages 875–882. Morgan Kaufmann, 1990.
246. D. J. Wilkinson. *Stochastic Modelling for Systems Biology.* Chapman & Hall/CRC, 1st edition, 2006.
247. A. T. Winfree. The prehistory of the belousov-zhabotinsky oscillator. *Journal of Chemical Education*, 61(8):661–663, 1984.
248. S. Wolfram. *A New Kind of Science.* Wolfram Media, 1st edition, 2002.
249. M. Yacoub and Y. Bennani. HVS: A heuristic for variable selection in multilayer artificial neural network classifier. In *Proceedings of Artificial Neural Networks in Engineering (ANNIE'97)*, pages 527–532, 1997.
250. X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.
251. C. Zandron, C. Ferretti, and G. Mauri. Solving NP-complete problems using P systems with active membranes. In *Proceedings of the Second International Conference on Unconventional Models of Computation, UMC'00, London, UK*, pages 289–301. Springer-Verlag, 2001.
252. A. M. Zhabotinsky. Periodic processes of malonic acid oxidation in a liquid phase. *Biofizika*, 9:306–311, 1964. In russian.

8

Acknowledgments

Here I am, at the end of this very intense as well as very exciting experience. When I started this adventure my aim was to learn to do research and I did not imagine my future reserved to me so many kind people to meet, so many fascinating places to visit, so many intriguing things to discover. This is the time to gratefully thank all those people who have enabled this dream to be realized.

First of all, I want to express my sincere gratitude to Prof. Vincenzo Manca, who gave me the chance to start working in his group three years ago and who has supported my research and visits to UK, Japan, Canada, Spain and many other beautiful places. I am very thankful to him for the time he has spent teaching me the fundamentals of scientific research, for the inspiration he has provided me during our long discussions, and for having "contaminated" me with his passion for science. The intellectual and human honesty he has shown me during these years is one of the most valuable teachings I will bring with me for the rest of my professional life.

I am also very grateful to Giuditta Franco who has always cared for my scientific growth, and who has spent a lot of time reviewing my writings and providing me with many precious feedbacks, remarks, and advices. Furthermore, I would like to thank her because she has always believed in me and she has supported my research projects. It has been a pleasure to work with her and I hope our collaboration will bring new interesting results in the future.

My thanks goes to Fausto Spoto and Matteo Cristani who have followed, with Prof. Manca, the developments of my thesis since the beginning of my PhD. Their questions and suggestions have given an important contribution to this thesis. Another significant support has been given by the referees of this thesis, Prof. Mario Pérez-Jiménez, Andrei Păun, Claudio Zandron and Leonardo Vanneschi. I wish to express them my gratitude for the valuable reviews provided. I thank Prof. Laudanna, for interesting discussions about signal transduction.

At the University of Verona many people have worked hard for enabling me and other PhD students to do valuable research. I would like to mention Prof. Bonacina, the former chair of the PhD School of Science Engineer Medicine, Prof. Segala and Prof. Viganò, respectively, the former and the current coordinator of the PhD course in computer science. My thanks goes also to secretaries, Giacomina, Aurora, Laura, Martina, Manuela, Caterina, for their kindness and patience.

Doing research and writing papers would not be so funny if I did not share my office with so many nice guys: Marco, Francesca, Enrico, Giulia, Samuele, Elisa, Amy, Nicoletta, Alessandro, Gabriele, Sara, Michele, Stefano, Anna and, last but not least, the most "energetic", Eleonora. A special thank is dedicated to the people who have collaborated with me in the research group of Prof. Manca, namely, Luca Bianco, Michele Petterlini, Roberto Pagliarini, Luca Marchetti and Rosario Lombardo, and to the students I have supervised during my PhD, Federica Agosta and Marika Rodegher.

My PhD started during a training period at the Italian Space Agency in Rome. I have met many brilliant people there: Danilo Rubini (the supervisor that every trainee would hope to have), Cesare Albanesi, Valeria Guarnieri, Gabriele Mascetti, Germana Galoforo, Luca Castaldi, Antonella Morese, and also Emilio Sanchez, Padre Melchor, Giovanni, Marco and Teodora. It is always a pleasure to remember them.

Back to Verona I started to attend the first workshops, brainstormings and conferences. I'm very grateful to the community of membrane computing which has made me feel at home from the beginning. In particular my thanks goes to Prof. Gheorge Păun for his kindness.

The second year of my PhD was also full of great experiences. I had the opportunity to visit for five months the Automated Scheduling Optimization and Planning (ASAP) group at the University of Nottingham (UK), where I specialized in optimization techniques and neural networks. I would like to gratefully thank Prof. Natalio Krasnogor, for his warm welcome, the valuable support and the helpful discussions (I still owe him many lunches and coffees...). In Nottingham I also had the chance to meet several wonderful people, among them, Duncan and Dave, my housemates, and Enrico, German, Jonathan, James, Elkin, Jaume, Pawel, Pedro, Rupa, Abdullah, Amr, Maria my colleagues. Special thanks go to my "coffee-mates": Francisco Romero-Campero, who gave me an invaluable help during the house hunting and who taught me many things about stochastic modeling, Juan Pedro Castro Gutierrez, who kindly gave me hospitality in his house for some days, and Daniel Soria.

Straight after Nottingham I went to Japan for a one-month visit to the Science and Complex Interaction Lab (SCI-LAB) at the Nagoya University. My stay there would not be so pleasant if Prof. Yasuhiro Suzuki did not provide me with every kind of help and assistance. I am grateful to him and to his group for the beautiful hospitality they gave me during my stay. In particular I would like to mention Teito Nakagawa, who brought me to Kyoto and to excellent sushi restaurants, Tsunoda Kenji, Fujie Masahiro, Suzuki Junji, Uesugi Shunsuke, Onishi Shunsuke, Setsuyuki Aoki, for the warm welcome, and Mai Kuroshima, who made me try the tea ceremony.

We are coming to the end of this long section of acknowledgments. I would like to dedicate these last words to the most important people in my life. Without their support, their trust and their love I would have never reached this achievement. Thank you very much Mum and Dad for all what you have done for me. And thank you very much Silvia, for having been by my side during all these years of hard work and long travels, and for willing to become my wife. You have given a sense to everything I have done.