# Metabolic P System Flux Regulation by Artificial Neural Networks

Alberto Castellini[1], Vincenzo Manca[1], Yasuhiro Suzuki[2]

[1] Verona University, Dept. of Computer Science
Strada Le Grazie 15, 37134 Verona, Italy
{alberto.castellini, vincenzo.manca}@univr.it
[2] Nagoya University, Dept. of Complex Systems Science
Furo-cho, Chikusa-ku, Nagoya, 464-8601, Japan
*ysuzuki@is.nagoya-u.ac.jp*

**Summary.** Metabolic P systems are an extension of P systems employed for modeling biochemical systems in a discrete and deterministic perspective. The generation of MP models from observed data of biochemical system dynamics is a hard problem which requires to solve several subproblems to be overcome. Among them, the flux tuners discovery aims to identify substances and parameters involved in tuning reaction fluxes. In this paper we propose a new technique for discovering flux tuners by using neural networks. This methodology, based on backpropagation with weight elimination for neural network training and on an heuristic algorithm for computing tuning indexes, has achieved encouraging results in a synthetic case study.

## 1 Introduction

Many kinds of models have been developed in order to provide new insight on chemically reacting systems, among them, ordinary differential equations (ODE) [16, 35] represent a milestone for continuous and deterministic modeling, while models based on the Gillespie's algorithm [13, 14] are widely used for discrete and stochastic modeling. A key point in the development of new modeling frameworks seems to be represented by the choice of the right abstraction level, since complex systems usually show different characteristics when viewed from different "distances". The majority of models now available seem to be either very low level (too detailed), or very high level (too coarse grain), while many biological systems seem to require an intermediate level of abstraction. The *executable biology* approach [10] suggests to employ *computational models*, namely, a new class of models that mimic natural phenomena by executing algorithm instructions, rather than using computer power to analyze mathematical relationships among the elements of biological systems.

Rewriting systems, in their basic form, consist of a set of terms and a set of rewriting rules stating how terms can be transformed. Many extensions of these

systems have been applied to biological modeling, such as the well known *L systems* [19], developed in 1968 by the Hungarian theoretical biologist and botanist Lindenmayer to provide a formal description of the growth patterns of various types of algae. *P systems* [28, 29, 30], from the name of G. Păun who devised them in 1998, represent a novel computational model originated from the combination of multisets rewriting systems and membrane compartmentalization. This approach lends itself to be used as a computational model for biological systems, wherein multisets of objects represent chemical elements, while rewriting rules and rewriting application strategies represent a kind of algorithm to be executed in order to mimic the phenomenon under investigation.

Several extensions of P systems have been developed so far [9, 32], some of them also coping with biological systems modeling [25, 27, 33, 34]. In particular, *metabolic P systems*, or *MP systems*, suggest a deterministic strategy, based on the generalization of chemical laws, for computing the amount of objects moved by rules at each computational step [20, 21, 23, 24, 25]. Equivalences between MP systems and, respectively, autonomous ODE [11] and Hybrid Functional Petri nets [5, 6] have been recently proved, and several biological processes have been modeled by means of MP systems, such as the Lotka-Volterra dynamics [25], the mitotic cycles in early amphibian embryos [24] and the *lac* operon gene regulatory mechanism in glycolytic pathway [5]. These case studies show that, being intrinsically time-discrete and based on multiset rewriting, MP models are able to give a different viewpoint on biological processes respect to traditional ODE models. A software called MetaPlab has been also proposed [8, 26] which enables the user to generate MP models by means of some useful graphical tools, and then to simulate their dynamics, to automatically estimate regulation functions and to perform many other tasks.

An MP system involves *i)* a set of *substances*, *ii)* a set of *parameters* (e.g., temperature, pH, etc.) and *iii)* a set of *reactions* each equipped with a corresponding *flux regulation function*. Such functions compute reaction fluxes, namely the amount of substances transformed by each reaction given a specific state of the system.

A crucial problem of MP model designing concerns the synthesis of flux regulation functions from observed time evolutions. In particular, the question is the following: "Given the time-series of substance concentrations and parameter values of a process observed every time interval $\tau$, and given the stoichiometry of the system under investigation, which are the flux regulation functions that make an MP model evolve with the observed dynamics?" The *log-gain theory* [20, 21, 22] supports the first step of the regulation function synthesis by enabling to deduce the time-series of flux values from the time-series of substances and parameters of an observed dynamics. Once flux time-series have been generated, the discovery of functions that compute these fluxes can be accomplished by techniques of mathematical regression.

In [7] a new approach is proposed to the synthesis of MP regulation functions relying on artificial neural networks (ANNs) as universal function approximators [3],

and employing both traditional and evolutionary algorithms [4] for learning these networks. Moreover, a plug-in tool for MetaPlab has been implemented to automate the learning stage. Here we extend this approach with a technique for weight elimination in ANNs [3, 36] and an algorithm for identifying flux "tuners", namely, the set of substances and parameters actually involved in the regulation of each flux. In the next section we formally introduce MP systems and the problem of flux discovery, while Section 3 presents the usage of ANNs for flux regulation function synthesis. In Section 4 and 5 we report, respectively, the new technique for discovering flux tuners and an application of this technique to a simple case study.

## 2 MP systems and MP graphs

In MP systems *reactions* transform *substances*, *flux regulation maps* establish the amount of matter transformed by each reaction at each step, and *parameters*, which are not directly involved in reactions, affect the flux regulation maps together with substance quantities. We refer to [22] for a formal definition of these systems, where also a detailed motivation of the principles underlying them is given.

The main intuition of MP dynamics is the *mass partition principle*, which expresses a discrete deterministic and molar reading of metabolic transformations, as opposite to the infinitesimal deterministic and local perspective of the mass action principle of classical differential models. For our further discussion it is useful to focus on the following simple example. Let $r_1, r_2, r_3, r_4$ be the following set of reactions:

$$\begin{aligned} r_1 &: 2a + b \to c \\ r_2 &: b \to c \\ r_3 &: b + c \to a \\ r_4 &: a \to 2b \end{aligned} \tag{1}$$

We consider the substances $a, b, c$ along the time instants $i = 0, 1, 2, \ldots$ (for the sake of simplicity here we avoid to consider parameters) and $\Delta a[i], \Delta b[i], \Delta c[i]$ are the variations of $a, b, c$, respectively, at time $i$. The quantities $u_1[i], u_2[i], u_3[i], u_4[i]$, are the number of molar units transformed by reactions $r_1, r_2, r_3, r_4$, respectively, in the step from time $i$ to time $i+1$. According to reactions (1) we get the following linear system at time $i$:

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

which becomes, in vector notation:

$$\Delta X[i] = \mathbb{A} \times U[i], \tag{3}$$

where

$$\begin{aligned} \Delta X[i] &= (\Delta a[i], \Delta b[i], \Delta c[i])', \\ U[i] &= (u_1[i], u_2[i], u_3[i], u_4[i])', \\ \mathbb{A} &= (\mathbb{A}(x,r)|x \in X, r \in R) = \begin{pmatrix} -2 & -1 & 1 \\ 0 & -1 & 1 \\ 1 & -1 & -1 \\ -1 & 2 & 0 \end{pmatrix} \end{aligned}$$

The log-gain theory for MP systems [22] provides algebraic methods which, from a time-series of vectors $\Delta X[i]$, generates the time-series of $U[i]$. When $U[i]$ are known, we face the problem of discovering some functions $\varphi_1, \ldots, \varphi_m$, as many as the dimension of $U[i]$, such that $\varphi(X[i]) = U[i]$. This problem of regulation maps discovery can be split into two subproblems: *i)* discovering the arguments on which each $\varphi_j$ depends, *ii)* defining the right mathematical form of $\varphi_j$. In the following sections we propose some new methodologies, based on ANNs, for solving both these subproblems, while now, a graphical representation of MP systems as bipartite graphs called *MP graphs* [24] is introduced. Substances, parameters, reactions and fluxes (e.g., respectively, $A$, $Pressure$, $R_3$ and $Flux_1$ in Figure 1) are depicted by different kind of nodes; stoichiometric (plain) arches connect reactant to reactions (e.g., $A \to R_3$) or reactions to products (e.g., $R_3 \to C$) and they possibly have labels denoting reaction stoichiometry if it is different from 1 (e.g., label 2 on arch $R_3 \to C$); regulatory (dashed) arches having a black arrow link fluxes to the reaction they regulate (e.g., $Flux_1 \to R_1$); finally, regulatory (dashed) arches having a white arrow connect substances or parameters to the fluxes which they regulate (e.g., $C \to Flux_1$). Notice that environment compartmentalization is not considered in the current version of the model but this feature will be topic of future work.

## 3 Artificial neural networks for flux regulation functions synthesis

The choice a regression technique for synthesizing flux regulation functions from substance, parameter and flux time-series deeply depends on the knowledge one has about the form of the expected functions. In particular, if the function is known to be a linear combination of its numerical parameters then *linear regression* analysis is used [1], such as the least squares method, while if the function is a nonlinear combination of its parameters then *nonlinear regression* analysis is employed [31].

Here we consider the very general case in which the form of regulation functions is completely unknown. Artificial neural networks (ANNs) [3] turn out to be a convenient approach in this situation, since they approximate very general

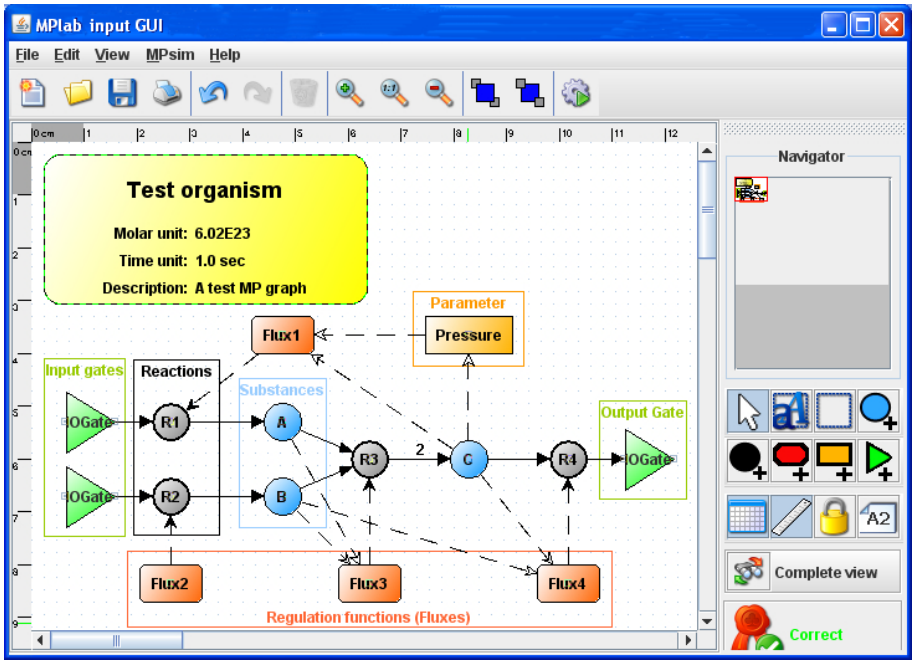


**Fig. 1.** An MP graph visualized by a graphical user interface of MetaPlab. Frame labels point out MP system elements in the MP graph representation. Substances, reactions and parameters describe the stoichiometry of the system, while fluxes regulate the dynamics.

maps just nonlinearly combining simple seed functions. An ANN is a mathematical model that takes its inspiration from the networks of interconnected neurons constituting the central nervous system. It has two key elements: a set of *neurons*, representing processing units, and a set of *synapses*, namely, weighted interconnections conveying information among neurons. A meaningful representation for ANNs employs graphs, where nodes symbolize neurons and edges stand for synapses, as displayed in Figure 2. Every neuron $u_j$ computes its output $y_j$ by the equation $y_j = f(\sum_i w_{ji} y_i)$, where function $f(\cdot)$ 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 representing the weight of the synapse connecting $u_i$ to $u_j$. Activation functions are usually 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. A particular type of ANNs we consider here are *feed-forward* neural networks, which have no feedback loops. In these networks, neurons are usually arranged in layers, where the *input-layer* receives input from the environment, the *output-layer* returns its output to the environment, and *hidden layers* process the information and pass it on through the network.

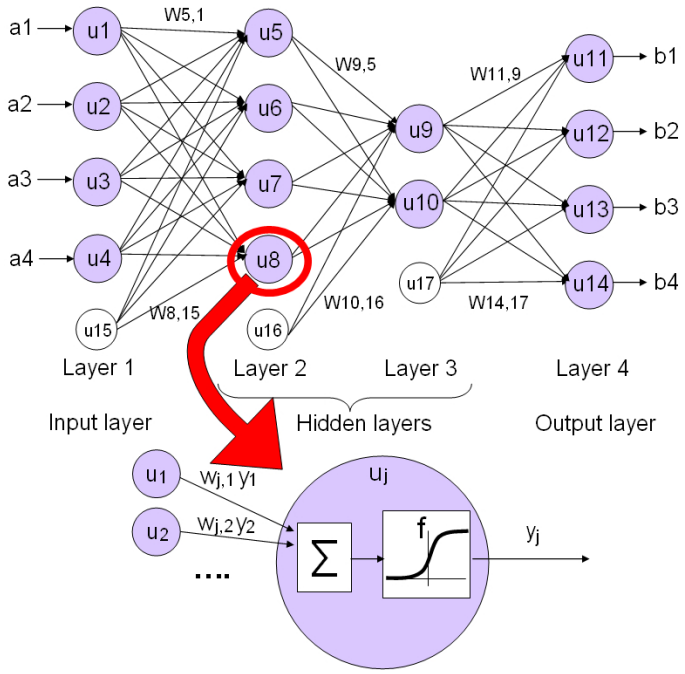


**Fig. 2.** A feed-forward neural network having four layers of neurons, namely, an input layer with four input neurons and one bias neuron; two hidden layers with, respectively, four and two normal neurons, and one bias neuron; an output layer with four output neurons. Every neuron of layer $i$ is connected to every (non-bias) neuron of layer $i+1$. Normal neurons (gray nodes) compute an activation function (usually sigmoid) of a weighted sum of their input. Bias neurons (white nodes) provide a constant unitary input.

We employ ANNs for discovering flux regulation functions since they have a natural ability to represent both linear and nonlinear relationships between a set of input variables (in our case substance and parameters) and a set of output variables (fluxes), and to learn these relationships from data sets. Moreover, it has been proved [12] that ANNs having at least one hidden layer and sigmoid neurons are able to approximate any continuous functional mapping, if no limit is imposed on the number of hidden neurons.

Given an MP system with $n$ substances, $k$ parameters and $m$ reactions we connect to it $m$ neural networks, each having $n+k$ input neurons connected to substance and parameter nodes, and one output neuron linked to a specific flux node, as displayed in Figure 3. The number of hidden layers and hidden neurons should be tuned according to the complexity of the functions under investigation. As a rule of thumb, the more "complex" the regulation function, the higher the number of hidden layers and hidden neurons. If the complexity of the searched

function is unknown, then different topologies should be tested, until a good approximation is found.

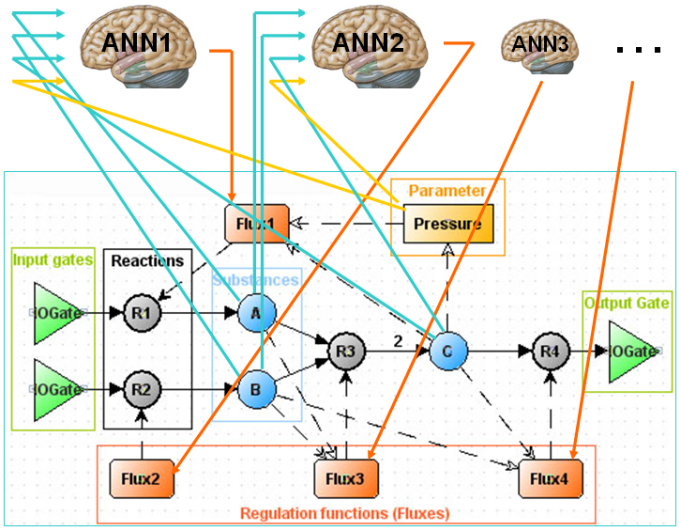


**Fig. 3.** MP system fluxes computed by one neural network for each reaction. Substances and parameters are connected to input neurons while the only output neuron of each network is connected to a specific flux [7].

Once the neural network topologies have been defined the information contained into a training set of observed data, has to be stored within synaptic weights. The process of weight tuning is called *training* and it is performed by the so called *learning algorithms*, namely, optimization techniques able to search for a set of weights which gives to the network a behavior defined by a set of examples, the *training set*. In our case, 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 [20, 21, 22]. During the training stage, 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 target outputs stored in the training set.

In [7] a Java software called *NeuralSynth* has been presented which trains feed-forward neural networks, within the MetaPlab suite, by means of four optimization algorithms, namely, *backpropagation* [3], *genetic algorithms* (GA) [15, 37], *particle swarm optimization* (PSO) [17] and a *memetic algorithm* [18]. In that work the memetic algorithm has been proved to achieved the best performance in discovering the regulation functions of an MP model of the mitotic cycle in early amphibian embryos.

## 4 Flux tuners discovery by artificial neural networks

The problem we tackle in this section concerns the automatic discovery of flux tuners from observed data. In the following we will call *tuners* of flux $\varphi$, the substances and the parameters which are involved in tuning $\varphi$ during the time evolution of the system. 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 of the system itself. Discovering these substances and parameters provides important understanding about the system and can suggest new experiments. Moreover, this information is very important also for generating sound MP systems, since regulation functions employed in these models should have as few independent variables as possible in order to yield reliable predictions [1]. This statement could sound a bit counterintuitive since it seems logical that, if our regulation functions incorporates as many variables as possible, then the flux prediction should be more accurate. Actually, this is true only if the number of data points to be fitted has no limitations (which is not realistic), indeed, 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].

The methodology we present in the following for discovering flux tuners by means of neural networks, consists of two steps: *i)* application of the *weight elimination* technique [3, 36], during the network training, for removing unnecessary synapse weights, *ii)* assignment, to each substance (parameter) of the MP system, of a *tuning index* for each flux, rating the "propensity" of the substance (parameter) to tune the flux itself.

### 4.1 Weight elimination

Weight elimination [3, 36] 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 [3]:

$$E = \sum_{k \in \mathcal{T}} (target_k - output_k)^2 + \lambda \sum_{i \in \mathcal{C}} \frac{w_i^2}{\hat{w}^2 + w_i^2}. \tag{4}$$

and then to minimize this function by means of backpropagation. The first term of Equation (4), called *performance term*, represents the square error between network output and target output 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 $\lambda$) to each weight $w_i \in \mathbb{R}$ such that $|w_i| >> \hat{w}$, 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 modified cost function of Equation (4), 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 $\lambda$, while the performance term keeps far from zero the weights actually needed to fit training data. Notice that, parameter $\lambda$ 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 $\lambda$ usually changes depending on the problem, thus, in [36] some heuristic rules are presented for dynamically tuning the value of $\lambda$ 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 [7], a Java software which can be employed within the MetaPlab virtual laboratory to automatically learn neural networks from experimental data. The first step of our tuner discovery strategy can be performed by this software, which can be downloaded from [2], that is, neural networks are trained on time-series data and, at the same time, their unnecessary weights are removed.

### 4.2 Tuning indexes assignment

The second step of the tuners discovery strategy proposed in this work involves the analysis of the neural networks achieved at the first step, with the aim to evaluate the sensibility 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 3) a *tuning index*:

$$\xi(x) = \sum_{p \in path(x,o)} \prod_{w \in p} |w| \tag{5}$$

where $path(x, o)$ is the set of all paths from the input neuron $x$ to the (only) output neuron $o$ (which is connected to the flux node $\varphi(q)$ according to the schema of Figure 3), 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 propensity 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.

The idea behind this heuristic for computing tuning indexes is informally explained by means of Figure 4. 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 4 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 3.

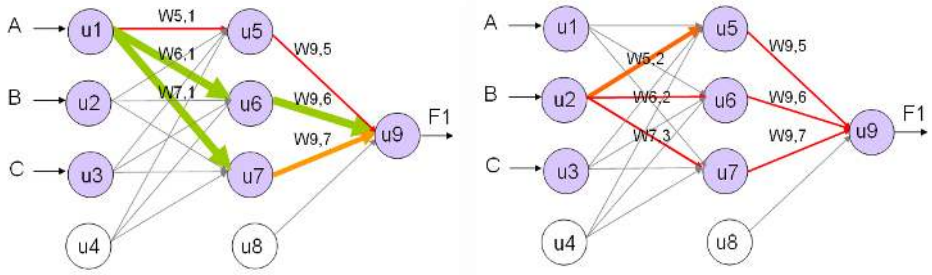


**Fig. 4.** Weight analysis of paths from the input neurons $u_1$ (on the left) and $u_2$ (on the right), to the output neuron $u_9$ for computing the tuning indexes of, respectively, substance $A$ and $B$ in respect of flux $F_1$.

Let us consider a simple example. On the left side of Figure 4, the contribution of path $u_1 \rightarrow u_5 \rightarrow u_9$, that is $|w_{5,1}| \cdot |w_{9,5}|$, is lesser than the contribution of path $u_1 \rightarrow u_6 \rightarrow u_9$, that is, $|w_{6,1}| \cdot |w_{9,6}|$, since $|w_{5,1}|$ and $|w_{9,5}|$ are lesser 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 $B$.

## 5 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, does not have any biological counterpart but its analysis

is however interesting because of the oscillations it generates when specific regulation functions are employed. As displayed in Figure 5, Sirius has three substances, $A$, $B$ and $C$, and five reactions $R_1, \ldots, R_5$. In [20] the following flux regulation functions have been manually generated:

$$\begin{aligned} F1 &= \frac{k_1 a}{k_1 + k_2 c + k_4 b + k_a} \\ F2 &= \frac{k_2 ac}{k_1 + k_2 c + k_4 b + k_a} \\ F3 &= \frac{k_3 b}{k_3 + k_b} \\ F4 &= \frac{k_4 ab}{k_1 + k_2 c + k_4 b + k_a} \\ F5 &= \frac{k_5 c}{k_5 + k_c} \end{aligned} \tag{6}$$

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, is featured by a very similar trend for substances $B$ and $C$, which differ only in the first fifty steps.

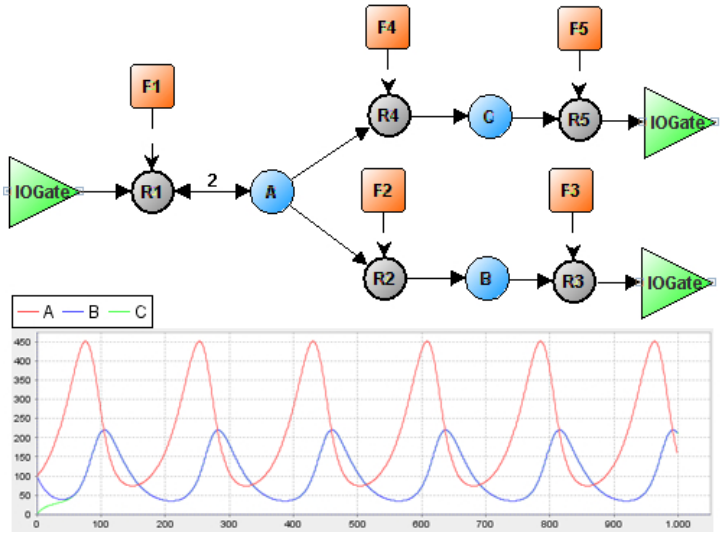


**Fig. 5.** On top: Sirius model. At the bottom: Sirius dynamics

We have sampled the dynamics of Figure 5 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 1, 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 $\lambda$ has been manually tuned for this case study but some heuristics [36] 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.

| | **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 1.** 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.

Let us analyze the results of Table 1. 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 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 1, 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 $C$ among 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 [7], is very similar to the original one, showed in Figure 5.

## 6 Conclusions and future work

In this paper we have presented a new technique, based on artificial neural networks, for discovering flux tuners within the framework of MP systems. This strategy involves a first training stage wherein each neural network learns a flux regulation function from observed time-series by means of backpropagation with weight elimination. Subsequently, for each flux a tuning index is associated to each substance and parameter of the MP system in order to evaluate its propensity to tune the flux. The technique has achieved encouraging results in a synthetic case study wherein data have been generated by known functions. Further work has to be done in order to get a stronger validation for case studies involving real biological systems. Moreover, some heuristic techniques employed in this paper to learn neural networks, i.e., evolutionary and swarm optimization, could be directly applied for discovering flux tuners, without incorporating neural networks.

## References

1. A. D. Aczel and J. Sounderpandian. *Complete Business Statistics*. McGraw-Hill/Irwin, 2006.
2. WMC10 additional material.
   Url: http://mplab.sci.univr.it/external/wmc10/page.html.
3. C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
4. E.K. Burke and G. Kendall. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, 2005.
5. 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.

6. A. Castellini, G. Franco, and V. Manca. Toward a representation of hybrid functional Petri nets by MP systems. In Y. Suzuki et al., editor, *Natural computing*, volume 1 of *PICT*, pages 28–37. Springer Japan, 2009.
7. A. Castellini and V. Manca. Learning regulation functions of metabolic systems by artificial neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2009*. ACM Publisher, 2009. Accepted.
8. A. Castellini and V. Manca. MetaPlab: A computational framework for metabolic P systems. In D. W. Corne et al., editor, *LNCS 5391*, pages 157–168. Springer-Verlag, 2009.
9. G. Ciobanu, G. Păun, and M. J. Pérez-Jiménez, editors. *Applications of Membrane Computing*. Natural Computing Series. Springer-Verlag Berlin, 2006.
10. J. Fisher and T. A. Henzinger. Executable cell biology. *Nature Biotechnology*, 25(11):1239–1249, 2007.
11. F. Fontana and V. Manca. Discrete solutions to differential equations by metabolic P systems. *Theoretical Computer Science*, 372(2-3):165–182, 2007.
12. K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989.
13. 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.
14. D. T. Gillespie. Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, 58:35–55, 2007.
15. J. H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI, 1975.
16. D. S. Jones and B. D. Sleeman. *Differential Equations and Mathematical Biology*. Chapman & Hall/CRC Mathematical Biology and Medicine, 2003.
17. J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proc. IEEE Int. Conf. on Neural Networks*, volume 4, pages 1942–1948, 1995.
18. N. Krasnogor and J.E. Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Trans. Evolutionary Computation*, 9(5):474–488, 2005.
19. A. Lindenmayer. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280–299, 1968.
20. V. Manca. The Metabolic Algorithm: Principles and applications. *Theoretical Computer Science*, 404:142–157, 2008.
21. V. Manca. Fundamentals of metabolic P systems. In G. Păun et al., editor, *Handbook of Membrane Computing*, chapter 16. Oxford University Press, 2009.
22. V. Manca. Log-gain principles for metabolic P systems. In A. Condon et al., editor, *Algorithmic Bioprocesses*, Natural Computing Series, chapter 28. Springer, 2009.
23. V. Manca. Metabolic P dynamics. In G. Păun et al., editor, *Handbook of Membrane Computing*, chapter 17. Oxford University Press, 2009.
24. V. Manca and L. Bianco. Biological networks in metabolic P systems. *BioSystems*, 91(3):489–498, 2008.
25. V. Manca, L. Bianco, and F. Fontana. Evolutions and oscillations of P systems: Applications to biochemical phenomena. In *LNCS 3365*, pages 63–84. Springer, 2005.
26. V. Manca, A. Castellini, G. Franco, L. Marchetti, and R. Pagliarini. Metaplab 1.1 user guide. Url: http://mplab.sci.univr.it. 2009.

27. 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.
28. G. Păun. Computing with membranes. Technical Report 208, Turku Centre for Computer Science, 1998.
29. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
30. G. Păun. *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
31. G. A. F. Seber and C. J. Wild. *Nonlinear Regression*. Wiley, 2003.
32. The P Systems Web Site. Url: http://ppage.psystems.eu/.
33. Y. Suzuki, Y. Fujiwara, J. Takabayashi, and H. Tanaka. Artificial life applications of a class of P systems: Abstract rewriting systems on multisets. In *LNCS 2235*, pages 299–346. Springer, 2000.
34. Y. Suzuki and H. Tanaka. Modeling p53 signaling pathways by using multiset processing. In Ciobanu et al. [9], pages 203–214.
35. E. O. Voit. *Computational Analysis of Biochemical Systems : A Practical Guide for Biochemists and Molecular Biologists*. Cambridge University Press, 2000.
36. A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In R. Lippmann et al., editor, *NIPS*, pages 875–882. Morgan Kaufmann, 1990.
37. X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, September 1999.