Evolving Chemical Reaction Networks

Elisabeth Degrand

DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2019

Evolving Chemical Reaction Networks

ELISABETH DEGRAND

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Evolving Chemical Reaction Networks

ELISABETH DEGRAND

Master in Computer Science
Date: May 23, 2019
Supervisor: Danupon Na Nongkai
Examiner: Johan Håstad
Principal: Inria Saclay - Lifeware team - François Fages, Mathieu Hemery
Swedish title: Utveckling av kemiska reaktionsnätverk
School of Electrical Engineering and Computer Science

Abstract

One goal of synthetic biology is to implement useful functions with biochemical reactions, either by reprogramming living cells or programming artificial vesicles. In this perspective, we consider Chemical Reaction Networks (CRNs) as a programming language. Recent work has shown that continuous CRNs with their dynamics described by ordinary differential equations are Turing complete. That means that any function over the reals that is computable by a Turing machine in arbitrary precision, can be computed by a CRN over a finite set of molecular species. The proof uses an algorithm which, given a computable function presented as the solution of a PIVP (Polynomial Initial Values Problem), generates a finite CRN to implement it. In the generated CRNs, the molecular concentrations play the role of information carriers, similarly to proteins in cells. In this Master's Thesis, we investigate an approach based on an evolutionary algorithm to build a continuous CRN that approximates a real function given a finite set of the values of the function. The idea is to use a two-level parallel genetic algorithm. A first algorithm is used to evolve the structure of the network, while the other one enables us to optimize the parameters of the CRNs at each step. We compare the CRNs generated by our method on different functions. The CRNs found by evolution often give good results with quite unexpected solutions.

Sammanfattning

Ett mål med syntetisk biologi är att genomföra användbara funktioner med biokemiska reaktioner, antingen genom omprogrammering av levande celler eller programmering av artificiella vesiklar. I detta perspektiv anser vi Chemical Reaction Networks (CRNs) som ett programmeringsspråk. Det senaste arbetet har visat att kontinuerliga CRNs med dynamik som beskrivs av vanliga differentialekvationer är Turingkompletta. Det betyder att en funktion över de realla talen som kan beräknas av en Turing-maskin i godtycklig precision, kan beräknas av en CRN över en ändlig uppsättning molekylära arter. Beviset använder en algoritm som, givet en beräkningsbar funktion som presenteras som lösningen av ett PIVP (Polynomial Initial Values Problem), genererar en ändlig CRN för att implementera den. I de genererade CRN:erna spelar molekylkoncentrationerna rollen som informationsbärare, på samma sätt som proteiner i celler. I detta examensarbete undersöker vi ett tillvägagångssätt baserat på en evolutionär algoritm för att bygga en kontinuerlig CRN som approximerar en verklig funktion med en ändlig uppsättning av värden för funktionen. Tanken är att använda parallell genetisk algoritm i två nivåer. En första algoritm används för att utveckla nätets struktur, medan den andra möjliggör att optimera parametrarna för CRN:erna vid varje steg. Vi jämför de CRN som genereras av vår metod på olika funktioner. De CRN som hittas av evolutionen ger ofta bra resultat med ganska oväntade lösningar.

Contents

Chapter 1

Introduction

Synthetic biology is a discipline that uses engineering to build biological systems. One goal of this field is to implement useful functions using biochemical reactions, either by reprogramming living cells or programming artificial vesicles. For example, Courbet et al. [8] designed and built a set of enzymatic reactions encapsulated in a vesicle that diagnoses different forms of diabetes by fluorescence. Chemical Reaction Networks (CRNs) are used as a model to describe systems of chemical reactions. In this thesis, we consider CRN as a programming language and propose an algorithm for CRN program synthesis.

The dynamics of a CRN can be described by ordinary differential equations. With this dynamics, it is possible, by simulation and given initial values for the concentrations of every species in the system, to obtain an execution *trace*, i.e. the concentrations of the species over time. This trace is a function of time. In cases where the trace of a species converges to a real value, an input-output function can be defined corresponding to the *dose-response* diagrams studied by biologists. The inputs are the initial concentrations of a set of species and the outputs are the final concentrations of another set of species that stabilize.

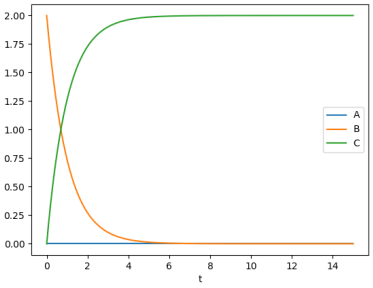
Example 1. A very simple example of CRN that computes the sum of two concentrations is given by the following reactions :

$$A \xrightarrow{k_1} C$$
$$B \xrightarrow{k_2} C$$

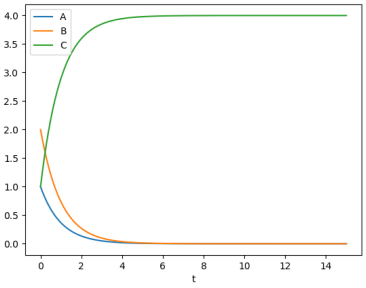
where $k_1 = k_2 = 1$ are the rate constants, quantifying the rate of the chemical reactions. The system of ordinary differential equations that describes the dynamics of the system is the following. How to obtain this system will be described later in the thesis.

$$\begin{cases} a' = -a \\ b' = -b \\ c' = a + b \end{cases} \qquad \begin{cases} a(0) = a_0 \\ b(0) = b_0 \\ c(0) = c_0 \end{cases}$$

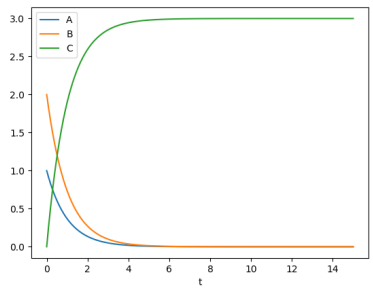
Figure 1.1 gives the traces of the CRN for different initial values. The concentration of A is in blue, the concentration of B in orange and the concentration of C in green. As the final concentrations of the species



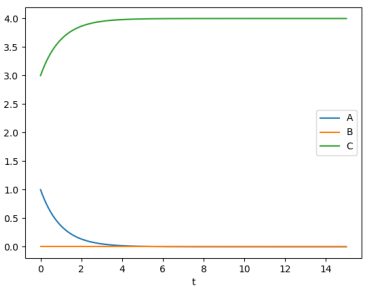
(a) Trace with initials values $a_0 = 0, b_0 = 2, c_0 = 0$



(b) Trace with initials values $a_0 = 1, b_0 = 2, c_0 = 1$



(c) Trace with initials values $a_0 = 1, b_0 = 2, c_0 = 0$



(d) Trace with initials values $a_0 = 1, b_0 = 0, c_0 = 3$

Figure 1.1: Trace of the CRN of example 1 for several initial concentrations

are stable, it is possible to draw a dose-response curve. Here we consider the final concentration of C over the initial concentration of A. The initial concentrations of B and C are fixed to $b_0 = 2, c_0 = 0$. Figure 1.2 gives the dose-response diagram. It can be observed that the

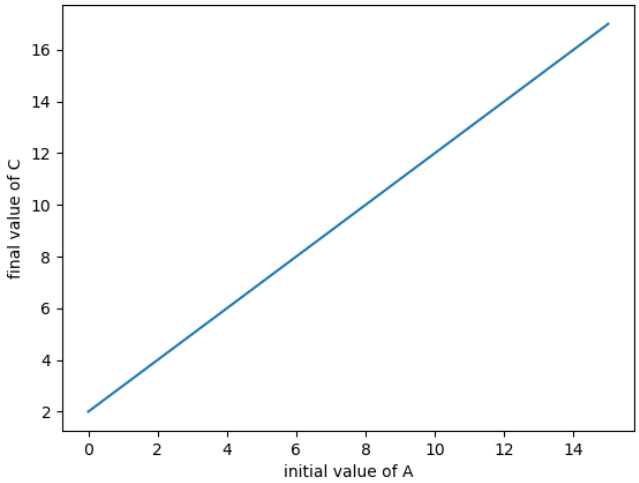


Figure 1.2: Final concentration of C given the initial concentration of A for the CRN of example 1

final concentration of C is the initial concentration of A plus the initial concentration of B.

A recent result has shown that CRNs with their dynamics described by ordinary differential equations are Turing complete. More precisely, any function over the reals that is computable by a Turing machine in arbitrary precision can be computed by a CRN over a finite set of molecular species [17]. Here a function computed by a CRN means that the function can be approximated at any arbitrary precision by a dose-response diagram of a CRN. The proof presents an algorithm that, given a computable function presented as the solution of a Polynomial Initial Value Problem (PIVP), generates a finite CRN to implement it. However, the CRN generated for usual mathematical functions (e.g. cosine, sigmoids, etc.) seem to have a quite different structure from the natural CRNs for similar functions (e.g. oscillators, switches, etc.) [17].

In this thesis, we address the problem of computing a CRN which approximates a function given a finite set of its values, the *data*, either as a function of time or a function of some input variable. Here *approximate* means find a function that closely matches the data given. Indeed the description of the dynamics of a CRN with ordinary differential equations leads to two different types of functions that can

be used to implement functions in the frame of synthetic biology: a function of time that represents the trace of the CRN, and a function of input that represents the dose-response relationship. A first problem is then to approximate *functions of time* with the trace of CRN. The second problem is to approximate *functions of input* with the dose-response diagram of CRN. For this second problem, the Turing completeness of CRN gives the existence of such a CRN for any arbitrary precision. The question is: Is it possible to use an evolutionary algorithm to approximate functions with the trace or the dose-response diagram of a CRN? Will the found CRN be comparable to the hidden CRN used to generate the data provided to the algorithm?

The approximation here is the same as the goal of neural networks or linear regression. The input is a set of points (x_i, y_i) called *data* and the goal is to find a function f from a certain class that minimizes the difference between the y_i and $f(x_i)$. The output of the approximation is the best function f. Here, instead of using a neural network to simulate the result function f, we use a CRN, that can simulate f in two different ways, as the trace of the CRN to approximate functions of time or as its dose-response diagram to approximate functions of input. Since the function f is described by a CRN which can be interpreted, the output here is a CRN. In this thesis, we refer to the functions we wish to approximate as *functions of time* and *functions of input* and to the functions simulated by a CRN as *trace* and *dose-response diagrams*.

The approach adopted in this thesis to handle the problems is to use an evolutionary algorithm to build a CRN from its dynamics that approximates the function. This approach should enable us to address both problems of approximation of functions of time and functions of input.

In the first part of this thesis, we study the background behind the work of this thesis: using the definition and the properties of a CRN, we make the link between CRN and computational analysis to finally give the result of Turing completeness that enables us to know that a CRN computing a given computable function exists. We then recall the literature with a focus on model learning and evolutionary algorithms. In the second part, we present and discuss the proposed algorithm, as well as its implementation. We also show how parallelism can be exploited. We then give evaluation results on some interesting functions for both our objectives. In the last part, we conclude with some perspectives.

Chapter 2

Background

In this section, we study the theoretical background of this thesis, as well as related works that shed light on the resolution to our problem.

2.1 Continuous Chemical Reaction Networks

First, we study Chemical Reaction Networks (CRNs), which can be seen as a mathematical formalism that enables us to study chemical reactions. We shall restrict ourselves to mass-action law kinetics, which will be described in this section.

2.1.1 Chemical Reaction Networks

To represent the reactions that occur chemically, we use the following definitions, from [15].

Definition 1. *Let $\mathcal{M}$ be a* finite set *of n molecular species $\{y_1, \ldots, y_n\}$.*
A reaction *is a triple $R \xrightarrow{k} P$, where*

- *$R : \mathcal{M} \to \mathbb{N}$ is a multiset of reactants*
- *$P : \mathcal{M} \to \mathbb{N}$ is a multiset of products*
- *$k \in \mathbb{R}^+$ is the rate constant*

A multiset is a set where the elements are allowed to be present several times. Here $R(y)$ is the number of times y is present in R and $P(y)$ the number of times y is present in P. These numbers represent the *stoichiometry* of the species. The *stoichiometric coefficient* of a species,

$P(y) - R(y)$, represents how the species contributes to the reaction. k represents the rate of the reaction, i.e. the speed at which the reactants are transformed in products.

We call a *reaction system* $\mathscr{R}$ a finite set of reactions.

We define the *rate function* for the mass-action law kinetics as $f(y) = k * \prod_{y \in \mathcal{M}} y^{R(y)}$. f is defined over the state of the concentrations of the species. When we write $f(y)$, y is $(y_1, \ldots, y_n)$ where y_i is the concentration of the *i*-th species and f the product of the concentration of the reactants multiplied by the *rate constant* k.

Finally, we call an *elementary reaction* a reaction with at most two reactants and with the mass-action law kinetics.

Example 2. As an example, described in [14], we can consider the prey-predator model of Lotka-Volterra. The reactions in this model are :

$$A \xrightarrow{k_1} 2 * A \tag{2.1}$$

$$A + B \xrightarrow{k_2} 2 * B \tag{2.2}$$

$$B \xrightarrow{k_3} \varnothing \tag{2.3}$$

Here, the set of species is $\mathcal{M} = \{A, B\}$, where A is the prey and B the predator. The rate function for the second reaction (2.2) is $f_2(A, B) = k_2 * A * B$. For the first reaction (2.1), it is $f_1(A, B) = k_1 * A$, and for (2.3) $f_3(A, B) = k_3 * B$.

2.1.2 Differential Dynamics

To represent the dynamics of a system $\mathscr{R}$ of reactions, several formalisms can be used, according to the hypotheses that are made.

The *differential semantics* describes the dynamics of the system with a system of Ordinary Differential Equations (ODE). In this semantics, we consider a high level of molecules, and y designates the concentration of the species, which depends on *time*. The corresponding ODE for $\mathscr{R}$ are defined as follow : *the derivative of the concentration of the species is the sum of the rate function multiplied by the stoichiometric coefficient of this species for each reaction where this species is involved*.

$$\frac{dx_j}{dt} = \sum_{(R_i, P_i, f_i) \in \mathscr{R}} (P_i(j) - R_i(j)) \times f_i$$

Example 3. We can now describe the model of Lotka-Volterra with the differential semantic. A is present in reactions (2.1) and (2.2). In the first reaction (2.1), A is present once as a reactant and twice as a product, thus its stoichiometric coefficient is $2 - 1 = 1$. In the second reaction (2.2), the stoichiometric coefficient of A is -1 as it is only a reactant. Then the corresponding ODE is $\frac{dA}{dt} = 1 * f_1 - 1 * f_2$ i.e. $\frac{dA}{dt} = 1 * k_1 * A - 1 * k_2 * A * B$. Doing the same for B, we have the following system of ODEs:

$$\begin{cases} \dfrac{dA}{dt} = k_1 A - k_2 AB \\ \dfrac{dB}{dt} = k_2 AB - k_3 B \end{cases} \tag{2.4}$$

With this semantics, a chemical reaction network is called a *continuous* CRN.

2.2 Computational Analysis and Analog Computability

2.2.1 Computational Analysis

In Computational analysis, a real number is computable if it can be approached with a sequence of rational numbers with arbitrary precision.

Definition 2 ([30]). *A real number $r \in \mathbb{R}$ is computable in the sense of computational analysis if there exists an effective approximation program of r in arbitrary precision, i.e. a Turing machine which takes as input a precision $p \in \mathbb{N}$ and outputs a rational number $r_p \in \mathbb{Q}$ s.t. $|r - r_p| \leq 2^{-p}$.*

Given this definition, it is now possible to define a computable function as a program that maps an approximation of a real x to an approximation of $f(x)$. A Turing machine with oracle is a Turing machine that has access to an oracle, a function that returns a result in constant time. Here, to compute $f(x)$, we consider that the Turing machine has access to x and does not need to compute again.

Definition 3 ([30]). *A function $f : \mathbb{R} \to \mathbb{R}$ is computable if there exists a Turing machine with oracle which computes an approximation of $f(x)$ given x as oracle.*

2.2.2 General Purpose Analog Computer

General Purpose Analog Computer (GPAC) is a model of analog computation introduced by Shannon [27] to formalize the Differential Analyser of Bush [4] using circuit units for sum, product and integrals. Graça and Costa [18] proved that functions of time *generated* by the GPAC are the solutions of polynomials Ordinary Differential Equations (ODEs) with initial values :

Definition 4 ([24]). *A function* $f : [a, b] \to \mathbb{R}$ *where* a, b *are computable reals is* GPAC generable *if there exists a function* $y : \mathbb{R}^+ \to \mathbb{R}^n$ *solution of*

$$\begin{cases} y'(t) = P(y(t)) \\ y(0) = y_0 \end{cases} \tag{2.5}$$

such that $f = y_1$ *(*f *is the first component of* y*) where* P *is a polynomial vector and* $y_0 \in \mathbb{R}$

Example 4. For example, cos is GPAC generable as it is solution of the following system, where $a(t) = cos(t),\ b(t) = sin(t)$:

$$\begin{cases} a' = -b \\ b' = a \end{cases} \qquad \begin{cases} a(0) = 1 \\ b(0) = 0 \end{cases}$$

A polynomial system of Ordinary Differential Equations as used in Definition 4 is called a *Polynomial Initial Value Problem (PIVP)*

Definition 5. *A* PIVP *is an ordinary differential equation*

$$\begin{cases} y'(t) = P(y(t)) \\ y(0) = y_0 \end{cases} \tag{2.6}$$

where $P \in \mathbb{R}^n[\mathbb{R}^n]$ *is a polynomial vector and* $y_0 \in \mathbb{R}^n$ *is the initial values. It is denoted* (P, y_0)

Example 5. An example of PIVP is the following :

$$\begin{cases} x'(t) = ax^2(t) + bx(t) \times y(t) + cy(t) & x(0) = 1 \\ y'(t) = dx(t) & y(0) = 0 \end{cases} \tag{2.7}$$

The system of ODEs for the Lotka-Voltera model (2.4) with given initial values, for example $A(0) = 50$ and $B(0) = 100$, is a PIVP.

2.2.3 GPAC Computable Functions of Input Variable

However, it turned out that this concept of computability as a solution of a system of ODE was not powerful enough to reach Turing completeness and Silva Graça [28] defined a new concept which was reinforced and proved to be Turing complete by Bournez et al. [1]. Instead of considering $y(t)$ the solution of the system, we consider the final state $y(+\infty)$ of the system where the initial values depend on the input x.

Definition 6 ([1]). *A function $f : [a, b] \to \mathbb{R}$ is* GPAC-computable *if there are polynomial vectors $p \in \mathbb{R}^n[\mathbb{R}^n]$, a polynomial $q \in \mathbb{R}^n[\mathbb{R}]$ where p and q have* computable *coefficients such that for all x there exists some (necessarily unique) function $y : \mathbb{R} \to \mathbb{R}^n$ such that*

$$y(0) = q(x), y'(t) = p(y(t))$$

and $|y_1(t) - f(x)| \leq y_2(t)$, with $y_2(t) \geq 0$ decreasing and $\lim_{t\to\infty} y_2(t) = 0$

A GPAC-computable function f is associated to a polynomial dynamical system of n variables. This system is defined by two polynomial vectors. One, q, is used to compute the initial values of the system. And the other one, P defines the dynamic of the system. For each $x \in \mathbb{R}$, the initial values are computed and then we take the solution y of the system. y_1, the first component of y, will converge to $f(x)$, whereas y_2 is a control on the error between y_1 and $f(x)$. f can be considered as a mapping between x and the final value of y_1. Every GPAC-computable function is GPAC-generable, however not every GPAC-generable function is GPAC-computable. In the rest of the thesis, the control of error will no longer be taken into account. The constraint will be $y(t) \to f(x)$.

Example 6. For example, the cosine function is a GPAC-computable function. We consider the following system, for $x \in \mathbb{R}$:

$$\begin{cases} a' = -bc \\ b' = ac \\ c' = -c \end{cases} \quad \begin{cases} a(0) = 1 \\ b(0) = 0 \\ c(0) = x \end{cases}$$

The solution of the system is $c(t) = x\exp(-t)$, $a(t) = cos(x - c(t))$, $b(t) = sin(x - c(t))$. We have $\lim_{t\to+\infty} c(t) = 0$, then $\lim_{t\to+\infty} a(t) = cos(x)$. Figure 2.1 shows the computation of cos(π). The blue curve corresponds to a and converges to -1.

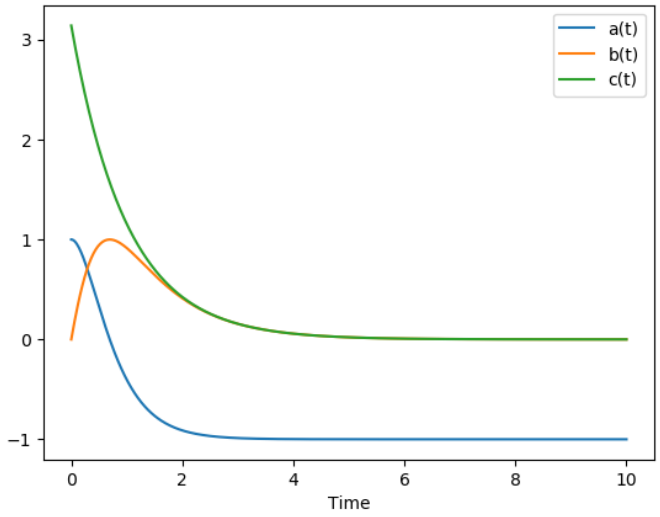


Figure 2.1: Computation of $\cos(\pi)$

Remark 1. For the sake of simplicity, we restrict the definitions of generable 4 and computable 6 to functions $f : [a, b] \to \mathbb{R}$ instead of $f : [a, b]^k \to \mathbb{R}^m$

Moreover, an equivalence between GPAC-computable function and computable functions (as defined in Definition 3) was shown.

Theorem 1 ([1], [2]). *A function is computable (in the sense of computational analysis) if and only if it is GPAC-computable.*

It is important that the coefficients of p and q in Definition 6 are computable, otherwise every constant function $f(x) = a$ would be computable, even if a is not computable, which is not coherent. GPAC is then a computational system that enables us to compute any computable function and, as a consequence, it is then Turing complete.

2.3 Turing Completeness of Continuous CRN

As seen in the previous section, the dynamics of a system of chemical reactions can be described by ODEs. These ODEs, for mass-action-law systems, are polynomial. We want to find an equivalence between ODEs describing a CRN and PIVP (which define GPAC-computable functions).

However, CRNs are specific. In particular, the dynamics of a CRN describe only positive variables. The reason for this is that the variables represent concentrations. Moreover, each monomial in the ODE correspond to a reaction, and some monomials cannot correspond to any reaction. In particular, we restrict to reactions with at most two reactants, that is of monomials of degree at most two. We ask also, to preserve the positivity of the system, that a monomial in p_i with negative coefficient should depend on x_i [16]. For example, $x' = -y$ is not allowed while $x' = -xy$ is valid.

In this section, we start first with a lemma that explains how the equivalence between CRN and PIVP is possible, even with the restrictions due to the specificity of CRN. This part is important to understand that using only positive variables in CRN is not a restriction comparing to CRN. Then, another restriction is introduced, that is to only consider a certain type of reactions. These two restrictions are important to know as they are used in the thesis. And finally, we give the result of Turing completeness.

Encoding With Positive Variables

As we consider here biological reactions where the variables are given by concentrations, only positive reals can be used. Therefore, we need to encode every variable to use only *positive* variables. It is possible to rewrite a PIVP with encoding to have only positive variables. The idea is to write $y = y^+ - y^-$ where y is the difference of its positive and negative part.

Lemma 1 ([17]). *Let f a GPAC-computable function by the PIVP (P, y_0). It is then possible to find $(\hat{P}, \hat{y_0})$ that computes (f^+, f^-), where $f = f^+ - f^-$ where $f^+, f^-\mathbb{R}^+ \to \mathbb{R}^+$.*

Proof. First, we encode each variable $y_i = y_i^+ - y_i^-$ where $y_i^+, y_i^- \in \mathbb{R}$. We can then define $\hat{p_i}(y_1^+, y_1^-, \ldots, y_n^+, y_n^-) = p_i[y^+ - y^-]$, and separate this polynomial into positive and negative parts $\hat{p}_i = \hat{p}_i^+ - \hat{p}_i^-$, where each part has only positive coefficients.
Finally, we consider the following *positive* system:

$$\forall i \leq n, \begin{cases} y_i^{+\prime} = \hat{p}_i^+ - \alpha_i y_i^+ y_i^- \\ y_i^{-\prime} = \hat{p}_i^- - \alpha_i y_i^+ y_i^- \\ y_i^+(0) = \max(0,\ y_i(0)) \\ y_i^-(0) = \max(0, -y_i(0)) \end{cases}$$

which computes (f^+, f^-). The α_i are polynomials with positive coefficients that verifies $\alpha_i \geq \max(\hat{p}_i^+, \hat{p}_i^-)$, for example $\alpha_i = \hat{p}_i^+ + \hat{p}_i^-$. □

This lemma enables us to have only positive variables and ensure that all the monomials are allowed (the only negative coefficients are related to $y_i^+ y_i^-$).

Elementary Reactions

We consider only elementary reactions with at most two reactants. The lemma below shows that any PIVP is equivalent to a PIVP of degree at most two. And PIVP of degree at most two represents the elementary reactions.

Lemma 2 ([7]). *Any solution of a PIVP is the solution of a PIVP of degree at most 2.*

As any PIVP can be transformed to be of degree at most 2, the Turing completeness of PIVP is not modified with a restriction to monomials of degree at most 2.

Turing Completeness

Using the two precedent lemmas, any PIVP can be rewritten as the dynamic of a CRN and it leads to this fundamental theorem in [17]:

Theorem 2 ([17]). *Any computable real function can be computed by an elementary reaction network over a finite set of molecular species.*

This theorem gives an equivalence between GPAC-computable functions and the results of an elementary reaction network. i.e. the final state of this reaction network. As Theorem 1 gives the equivalence between GPAC-computable functions and functions computable by a Turing machine, the CRNs are Turing complete.

2.4 Initialization

Besides, in Definition 6 of GPAC computable, the initial values are given as a polynomial of x, $q(x)$. But no constraint is given on this polynomial, which can have any degree, and when using CRN, these initial values need to be computed first. To avoid this computation that

can be time-consuming, we would like to start with given constant initial values, which correspond to constant initial concentrations in the biological system, except for the species that are the input. In [24], an equivalent definition to Definition 6 is given. It gives then the following property:

Theorem 3. *A function* $f : \mathbb{R}^k \to \mathbb{R}^m$ *is* GPAC-computable *if there is a polynomial vector* $p \in \mathbb{R}^n[\mathbb{R}^n]$ *such that for all* $x = (x_1, \ldots, x_n)$ *there exists some (necessarily unique) function* $y : \mathbb{R} \to \mathbb{R}^n$ *such that*

$$y'(t) = p(y(t)) \quad \begin{cases} y_i(0) = x_i \text{ if } i \leq k \\ y_i(0) = c_i \text{ if } i > k \end{cases}$$

and $lim_{t \to +\infty} \|y_{k+1,\ldots,m+k}(t) - f(x)\| = 0$, *where* $c_i \in \mathbb{R}^+$ *are constants.*

Here the k first components are the input and the m following the output. The initial values of the inputs are the components of x, and the other initial values are constant.

2.5 Learning Models of Dynamical Systems

In this section, we see what related works can offer us to understand the problem studied in this thesis and find a good method to resolve it. As seen before, the dynamics of a CRN can be represented with ODEs, so a part of this section is dedicated to learning of ODEs in a dynamical problem. Then we take a look at some of the related works that are more focused on biological systems.

Brunton, Proctor, and Kutz [3] use a sparse regression to discover the equations of a dynamical problem. However, this method is based on knowing the values of each variable for some time points, so it seems difficult to extend it to our question. Moreover, this article uses analytical derivatives of the data (coming from the physical law they try to discover) and there is no result given when using approximating derivatives.

Cao et al. [5] have a relevant approach given this thesis. They want to discover governing equations for a dynamical system too. They use a two-level genetic algorithm, one for the structure and one for the parameters. During the evolution, the comparison between the true data and the model found is done with numerical integration and ,thus, it does not use the fact that all the variables are known. As a

consequence it is easily transposable to our problem. However, their comparison is based on time series, and they do a simulation for each time point and compare the result of simulation at the next time to the next point in the time series. Their fitness function is then very specific to time series (and thus functions of time), but it can be used for our problem with a modification of the fitness function.

Hsiao and Lee [21] are more focused on biology, since they study gene regulatory networks. Their solution couples two approaches which are optimization and parameter identification. However, this approach is too specific for us, as we do not know prior knowledge of the structure of the network.

Noman, Palafox, and Iba [23] present a review of the use of evolutionary methods for genetic networks. A majority of these are still too specific. Cao et al. [6] use the same approach as in [5] and use their two-level genetic algorithm to evolve a cell model, with structure optimization and parameter optimization. They do not use stochastic semantic (P model) but it is very similar to our problem. As future work, they suggest to use a better GA for the parameter optimization and to parallel the genetic algorithms.

Dinh et al. [13] present a method to evolve reaction networks. They use a DNA toolbox and they evolve the circuit directly. They have one genetic algorithm for both the structure and the parameters.

Finally, we can see that some methods have been given to learn a system of ODEs. Only some of them handle the fact that some variables of the system are unknown. A popular method to deal with such systems seems to be genetic algorithms.

2.6 Evolutionary Algorithms

In this section, we study evolutionary algorithms, as they are popular for this kind of problem, where the search space is particularly complex.

2.6.1 General Genetic Algorithms

In [22] an overview of genetic algorithms is given. A general genetic algorithm (GA) follows the following architecture:

- **Initialization** A population of possible solutions is created, often randomly, from the search space.
- **Evaluation/ Selection** Individual solutions from the population are evaluated through a *fitness function*. Then, a portion of them is *selected*, accordingly to their score at the fitness function.
- **Genetic operators** From the selected population, a new population is generated. It is done mainly through *cross-overs* and *mutations*. It is a way to obtain new possible solutions.
- **Termination** After some cycles of Evaluation - Selection - Genetic changes, the process is ended, based on a termination criterion. A cycle of Evaluation - Selection - Genetic changes is call a *generation*. In addition to this, we call the whole evolutionary procedure a *run*.

To use a genetic algorithm in practice, there are many choices to make. First, you have to chose the data structure, i.e. the encoding. This choice is crucial for the algorithm to succeed. Then, the fitness function has to be chosen. It has to be chosen carefully, to encompass every aspect of the problem. The process of selection is not unique, since several possible processes exist such as *tournament*, *elite* or *rank selection*. The choice of genetic mutators depends to a large extent on the encoding. Depending on the encoding, some mutations are easier than others to implement. Genetic Algorithm is really a frame that should be personalized according to the problem.

2.6.2 Covariance Matrix Adaptation Evolution Strategy

Evolution Strategy (ES) is a class of genetic algorithms. ES uses evolution to optimize black-box numerical functions. CMA-ES (Covariance matrix adaptation evolution strategy) [19] is one of these optimization algorithms. It is a derivative-free optimization algorithm and it performs well on non-convex problems. It is considered as the state-of-the-art algorithm for this type of problems [20]. In an ES, the possible solutions are sampled according to a distribution. This distribution can be updated at each step through a self-adaptation or a covariance matrix adaptation as CMA-ES does.

This algorithm is used since 2007 in Biocham modeling software, the software developed by the Lifeware team to find parameter values from numerical traces of molecular concentrations [12].

Chapter 3

CRN evolution method

With the definition of all the concepts, we can now define the objectives of this thesis more precisely. As said before, the goal is to find CRNs to approximate functions, given only a finite set of values. Here we have two problems of approximation. The first problem is to approximate a time function by the trace of a CRN, i.e. the values of the species over time. It is equivalent to approximate the function with a function generable (as in Definition 4).

The second problem is to approximate an input-output function by the dose-response diagram of a CRN. This is equivalent to approximate the function with a function computable (as in Definition 6). In that case, with the result of Turing completeness, every computable function is computed by a CRN, or equivalently by a PIVP, and then approached as close as wanted by a CRN.

When trying to approximate a function known only on a finite set of values (x_i, y_i), the goal is to minimize the error i.e. the distance between the evaluations of a function f on the x_i and the y_i. This error will be described later. In this section, we describe an algorithm that enables us to approximate the functions of both problems. It would be more convenient if the algorithm could differ as less as possible for the two problems.

3.1 Choice of Algorithm

We make the choice here to represent a CRN by the corresponding PIVP. We only consider elementary reactions, which means that the monomials in the PIVP are of degree at most 2. As seen in Lemma 2, it

is not a restriction to consider only elementary reactions. This enables to have simpler CRN. As seen before, studying the PIVP with restrictions is equivalent to studying the CRN. The PIVP is easier to represent because it is determined by at most two polynomials, one for the dynamic and if needed, the other one for the initial values. Comparing to other models of dynamical systems, we have an advantage because we know that our model should be polynomial. We choose to represent the PIVP as the list of which monomials are present or absent, and their corresponding coefficients as well as the initial values. However, it is not possible to predict how many and which monomials will be needed. Even worse, it is not possible to know how many variables will be needed. Our goal here is to minimize the error between the function given by the model and the data. When the space-state is of unknown dimension, a genetic algorithm is a good way to apprehend optimization problem.

There are two ways to modify the model: with the presence or absence of a monomial (the structure) or with the values of the coefficients associated with the monomials (the parameters). The values of the coefficients are very important, and a model is at risk to be rejected only because of bad coefficients. A satisfactory method to handle this, like in [5], is with a two-level genetic algorithm. One level optimizes the structure, and the other optimizes the parameters.

To understand the situation from a biological perspective, we can consider groups that are very far from each other. In a given group, individuals are close and share important common characteristics (the structure) but are slightly different (the parameters). Two levels of evolution/competition appear. One is between individuals of the same group and has a short characteristic time. The other one is between the different groups, it takes more time as the groups are distant.

Example 7. We can consider as an example, that the groups are different animal species living in the same environment. Each species has its own characteristics that we can call the *structure*, with some differences between the members of a species, the *parameters*. Inside a species, an evolutionary mechanism happens, and the better individuals are selected. However, another competition happens, the competition between the different species, where some species are more suitable for the environment than others.

To perform this dual evolution, during the evaluation part of the

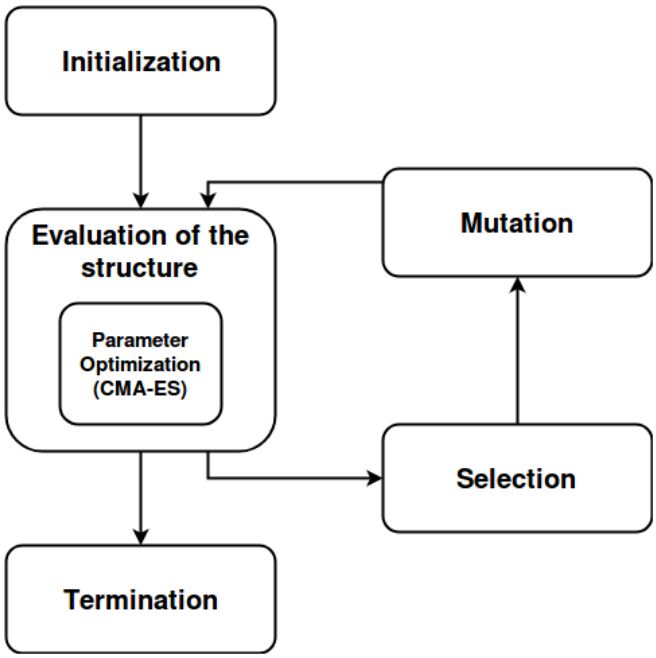


Figure 3.1: Synopsis of the algorithm

first level (structure), its parameters are optimized. It has to be noticed that for a given structure, the number of parameters is fixed. This property is a real advantage. Our optimization problem is non-convex, we have no idea of the derivatives of the objective function and the objective function can be high-dimensional. Consequently we need an optimization algorithm that does not use the derivatives. CMA-ES, described above, is the state-of-the-art for this situation.

With this algorithm, the difference between the approximation of functions of time and functions of input appears in only two places. The initial values are defined differently for the two methods, as they are constant for functions of time, and polynomial in the input for functions of input. The other appears during the evaluation of the solutions, as the CRN is used differently to approximate the function. This will be further discussed below.

Figure 3.1 shows a synopsis of the algorithm, with the optimization of the parameters during the evaluation of the structure.

3.2 Structure Optimization

In this section, we describe the top level of our algorithm, namely the optimization of the structure. Here, the algorithm is the same for the approximation of both functions of time and functions of input. First, we give Algorithm 1 for the genetic algorithm that optimizes the structure. It follows the classic genetic algorithm described in section 2.6. The following paragraphs describe each part of the algorithm.

Algorithm 1 Structure Optimization

```
function EVOLUTION(f)
    population ← INITIALIZEPOPULATION(f)
    for p in population do
        STRUCTUREFITNESS(p)
    end for
    pop_new ← SELECT(population)
    population ← pop_new + MUTATE(pop_new) ▷ Concatenate the
two lists
    return SELECT_BEST(population)
end function
```

Data Structure

In a genetic algorithm, the data structure is important. Here, we are looking for PIVPs, i.e. polynomial ODEs. A PIVP is represented by the number of variables it has and the monomials for the derivatives of each of its variables.

The parameters are searched according to a logarithm scale to respect the biochemical behavior. Hence, they are positive. To be able to have negative coefficients, each monomial has a sign, + or -. This is called the *signature* of the monomial. To handle the positivity constraint, a monomial cannot have a minus sign if its variable is not present in it. For example, the derivative of x can depend on $-xy$ but not on $-y$.

The maximum number of variables is fixed.

Initialization

The population of possible solutions is initialized by random PIVP. Their number of variables is in a given range, with a uniform distribution. They have one or two monomials per variable (with the same probability). We keep the number of monomials low to have economic CRNs, which are easier to interpret.

Evaluation

In this algorithm, the structure is evaluated through the optimization of its parameters. This optimization process will be described more precisely in the next section. The fitness value here is defined as an error, so it has to be minimized. The fitness value of the structure is the fitness value of the structure associated with its best parameters. As the optimization process is non-deterministic, the fitness value is updated at each generation only if the fitness value found during this generation is better than the current one. The parameters that gave the best value are kept.

Selection

An *elitist* selection has been chosen. This means that only the 50% best polynomials are kept and allowed to mutate. This selection is based only on the ranking of the individual solutions according to their evaluation and thus is very robust with respect to the fitness definition.

Mutation

Several mechanisms of mutation are used :

- Adding or removing a monomial
- Adding or removing a variable
- Replace the PIVP by a new random one having the same number of variables and one or two monomials per variable (to enable the exploration in the search space)

At each generation, only one of the mechanisms above can happen. The probability of adding and removing a monomial is set to 0.35.

These other mutations have their probability set to 0.1. Moreover, at each generation, the sign of a monomial is changed (if it handles the positivity constraint).

During the mutation, the good parameters found during evaluation are kept. When a monomial is added, a coefficient is sampled, sampling its logarithm according to the normal law $\mathcal{N}(0, 1)$. If a variable is added, the monomials are added with coefficients as just explained, and new parameters for the initial values are sampled too, following the same mechanism.

When deleting a variable, all the monomials using this variable are deleted. If a variable has no more monomials, a random monomial is added to this variable (to avoid having an empty ODE).

We made the choice to only use genetic mutation, and no cross-over. With our data structure, it would be hard to find a way to have cross-overs and it does not have real meaning.

Termination

At the end (after a fixed number of generations), CMA-ES is applied with more iterations to the best PIVP found. The PIVP with the best coefficients given by CMA-ES is our solution.

Biological Constraints

As seen in the Turing completeness proof in Section 2.3, some constraints have to be handled by a PIVP if this PIVP has to be implemented by a CRN. The first constraint to ensure the positivity of the system is that some monomials are not allowed in a PIVP representing a CRN. During the choice of data structure, this constraint was verified, with only allowing valid monomials. The second constraint is that we can only consider *positive* variables. Therefore, during the parameter optimization, every PIVP that gives non-positive solutions is discarded.

3.3 Parameter Optimization

In this section, we describe the optimization of the parameters, which occurs during the evaluation of the structure. The parameters here are associated with a given structure, which stays the same during the

whole process. The algorithm used is CMA-ES. In this part, the fitness function designate the function given to CMA-ES to be minimized. This function is indeed the fitness function of the coefficients.

As said before, we want the coefficients to be chosen through their logarithm, to have more insight into the order of magnitude. So, instead of evolving the coefficients, CMA-ES evolves the logarithm (in base 10) of the coefficients. The conversion from the logarithm to the coefficients is handled by the fitness function.

Algorithm 2 gives the procedure to optimize the parameters. The algorithm is described in detail below. The evolutionary strategy CMA-ES gives the best value and the state that gives the best value (fit_param here).

Algorithm 2 Parameter Optimization

```
procedure StructureFitness(f, pivp)
    u ← Random(0, 1)
    if u == 1 then
        x0 ← RandomInitialState(pivp)
    else
        x0 ← InitialStateFromBest(pivp)
    end if
    fit_value, fit_param ← CmaEs(FitnessFunction(pivp),x0)
    UpdateBestParameters(pivp, fit_value, fit_param)
end procedure
```

3.3.1 Parameters to Evolve

The parameters to evolve are of two natures: the coefficients associated with a monomial, and those for the initial values. The coefficients to the monomials are as many as the monomials of the PIVP. The parameters for the initial values depend on the problem.

For the function of time, as in Definition 4, the initial values of the PIVP are constants. For the components that are compared to the objective function, the initial values are known; they are the initial values of the function we try to approximate. The other initial values are unknown, they are, thus, given as parameters to CMA-ES.

For the general functions, we use the initial values from Theorem 3. For the first components, the initial values are known, since they are

the input. The other initial values are unknown constants, they are, thus, given as parameters to CMA-ES.

3.3.2 Starting Point

CMA-ES uses very few parameters, but a starting point is needed. First, the starting point is chosen randomly, but then the starting point provided is either the coefficients found previously that gave the best score ever for that PIVP, or a random starting point, each with the same probability. The random starting point is chosen according to a normal law $\mathcal{N}(0, 1)$ of mean 0 and variance 1. This starting point corresponds to the logarithm of the coefficients. When the point is chosen randomly, the starting variance for CMA-ES is 2, otherwise, it is 0.5. There are thus two phases, one of exploration and one of exploitation.

3.3.3 The Fitness Function

As the CRN does not approximate the functions of time and the functions of input the same way, the fitness function is different for the two problems. However, as the goal is to approximate functions, in both cases we want to minimize the error between the function simulated from the CRN and the points given to the algorithm.

To compute the fitness value of a structure associated with some parameters, the solution of the PIVP is numerically integrated. If the solution has non-positive value, it is discarded through a very high fitness value. The result is compared to the objective (this part depends on the problem) through a *loss* function.

To the loss function a constraint is added on the parameters, to avoid to have too big or too small coefficients. Moreover, having very big or small coefficients can be a source of numerical errors in the numerical integration. It is, therefore, more robust to have coefficients around 1 as much as possible. This constraint is $\lambda * \sum |\log(p)|$, which is the sum of the logarithms of the parameters multiplied by a constant. λ is chosen to be smaller than the typical loss for the function. The idea is that this constraint should be less important than having a good result and should preserve the orders of magnitude.

We give here the pseudo-code with the mechanism for each type of function. The loss function will be described next. Algorithm 3 gives the fitness function for functions of time, and Algorithm 4 for functions

of input.

Algorithm 3 Fitness function for functions of time

```
function FitnessFunction(f, pivp, param)
    coeff, y0 ← ParamToCoeff(param)
    sol ← Integrate(pivp, coeff, y0)
    return Loss(sol, f)
end function
```

Algorithm 4 Fitness function for functions of input

```
function FitnessFunction(f, pivp, param)
    coeff, y0 ← ParamToCoeff(param)
    value ← 0
    for x_i in x do
        sol ← Integrate(pivp, coeff, y0(x_i))
        value ← value + Loss(sol, f_i)
    end for
    return value / len(x)
end function
```

Functions of Time

As a recall, the objective here is to approximate the trace of a function of time $f : \mathbb{R}^+ \to \mathbb{R}^m$ with the trace of some species of a CRN. We have S points from the function $\{(t_s, f_s), s \in [1, \dots, S]\}$. We assume that the t_s are sorted in ascending order.

Here the points are compared to the solution of a PIVP. The solution is obtained thanks to a numerical integration on the interval $[t_1, t_S]$.

We call y the solution of the PIVP. We want to compare the point (t_s, f_s) with the point $(t_i, y(t_i))$. The loss function is then

$$\mathcal{L}(f, y) = \frac{1}{S} \sum_{s=1}^{S} \left[\sqrt{\sum_{i=1}^{m} (f_s^i - y(t_s)^i)^2} \right]$$

At each time step, the norm (the error) between f and y is computed. Then the empirical mean of these errors is computed and this becomes the loss value.

Functions of Input Variable

As a recall, the objective here is to approximate a function $f : \mathbb{R}^k \to \mathbb{R}^m$ known only on a finite set of points with the final states of some species of a CRN for different initial concentrations, which depend on the input. We have S points from the function $\{(x_s, y_s), , s \in [1, \ldots, S]\}$. Here the points are compared to the final states of a CRN, i.e. to the final values of a PIVP. For each input x_s, the solution is obtained thanks to a numerical integration on the interval $[0, 1]$. The initial values of the PIVP are defined as in Theorem 3.

The PIVP has to be numerically integrated for each input. We call h_s the solution of the PIVP for the input x_s. We want to compare (x_s, y_s) with $(x_s, h_s(1))$. In Theorem 3, the first k components are for the inputs and the m following for the output. The solution is to be read on these m components. The loss function is then

$$\mathcal{L}(h, f) = \frac{1}{S} \sum_s \|h^s_{k+1,\ldots,m+k}(1) - f_s\| + \|\nabla_{k+1,\ldots,m+k} h^s(1)\|$$

where $\|\cdot\|$ is the norm, computed as above for the functions of time and $\nabla h = (h'_1, \ldots, h'_m)$.

The derivative is asked to be small to ensure the convergence of the components that are needed.

The loss is the empirical mean of the errors for each input.

3.3.4 Termination

CMA-ES already has default values for its termination. The termination can depend on the total number of iterations, the number of iterations without any improvement or the structure of the internal covariance matrix. The algorithm stops when one of the critera is reached. These criteria can depend on the dimension of the problem. We did not modify the default values of the termination criteria of CMA-ES, except for the time limit. We fixed a maximum time for CMA-ES to avoid that it takes too much time. The time limit depends on the problem, but it is 200 s in general.

3.4 Parallel Implementation

To implement our solution, the language chosen is Python (3.6.3). To implement CMA-ES, the package `cma` was used. The numerical inte-

grator used is LSODA, it is part of the `scipy` library.

This work was granted access to the HPC resources of CINES under the allocation 2018-AP011010715 made by GENCI. Thanks to that, parallelization could be evaluated. The package used for that is `mpi4py` ([10], [11], [9]). In a genetic algorithm, the most consuming time is the evaluation part. However, the evaluations of the different individuals in the population are independent, so it is easy and natural to parallelize this part to save time.

`mpi4py` works exactly like MPI, except that it uses Python formalism. MPI (Message Passing Interface) is a formalism that enables us to coordinate several processes. With MPI, the program is run by every process, but some parts are executed only by some processes.

Our algorithm uses two different genetic algorithms, and, therefore, parallelism can be used twice. One layer of parallelism is easy to implement, as there is a function `Scatter` that takes a list from a root process and scatter it to all processes. The function `Gather` enables us to perform the inverse operation. Adding the second layer is more difficult, especially in our case, where it is not possible to transform the two layers in one layer.

To overcome this challenge, we use the function `Split`, which creates new communicators that consists of 10 processes. There are as many new communicators as individuals in the population. At each generation, the population is scattered between the different groups of processes. Each group is dedicated to one individual. Each group performs the evaluation of the individual. The population of CMA-ES is set to 10. In a group, during the evaluation part of CMA-ES, each solution of CMA-ES is evaluated by a different process, through a scatter/gather mechanism.

Chapter 4

Evaluation Results

4.1 Choice of the Functions

In this thesis, we want to address the problem of approximation of functions either of time or of input with CRN. In this chapter, the algorithm described in the last chapter is evaluated on a selection of functions. Concerning the approximation of functions, the measure of success is the error, which should be as small as possible. Also if it is possible, we would like to compare the CRN found by evolution to the hidden CRN used to generate the data provided to the algorithm.

Here, the evaluation of our algorithm is based on the error between the data and the function given by the algorithm. Moreover, to have a basis for comparison, the CRN found by evolution are compared if possible to the CRN used to generate the data. More general conclusions will come in the last chapter of this thesis.

The objective here is not to retrieve a specific CRN. A question studied by the Lifeware team is to compare the biological CRNs, the CRN obtained from the proof of Turing completeness and the CRN obtained by evolution. This whole issue is far beyond the scope of this thesis, but we will try to give some insights on this topic.

4.1.1 Functions of Time

For the functions of time, we first choose cosine as a sanity check. The PIVP corresponding to cosine has been described in Example 4. It is a quite simple PIVP as it has only two variables and two monomials. For this function, the biological constraints are not taken into account,

i.e. every monomial is allowed and the solutions of the PIVP can be negative. The purpose of testing our algorithm on cosine is first to be sure that this algorithm can retrieve a simple function. Moreover, it would help us have more insight into the output of our genetic algorithm.

The next example chosen is the Heaviside function. This function is interesting in the frame of synthetic biology. One of its goals is to program biochemically as we do with a computer. Programming implies a notion of *sequentiality*, each action should be performed one after the other. This implies then to know when the previous action has been completed. The output of Heaviside switches from one state to another instantaneously. Having an approximation of Heaviside would enable to implement the sequentiality when programming biochemically. This function is not continuous, so it cannot be the solution of a PIVP. However, it can be approximated, usually with a sigmoid function. So the expectation would be that the trace of the best CRN found is a sigmoid, but probably different from the standard mathematical sigmoids.

Finally, we choose to study a biological example, a model of cell division given by Tyson in [29]. The question is to compare the CRN found by evolution to the CRN of the model. This model is studied by other members of Lifeware team so it can be an object of comparison in the future.

4.1.2 Functions of Input

For the functions of input, we start with a sanity check with the cosine function. The PIVP is described in Example 6. It is already more complex than for functions of time, as it has three variables and six monomials. The expectation is that the output of the genetic algorithm is a PIVP that succeeds to closely approximate cosine. The PIVP found by evolution is compared to the one given in Example 6. Once again, for this only example, the biological constraints are not taken into account.

Another sanity check example is studied, but with the biological constraints this time. The function we test on is the sum function. It is a simple example in the sense of a corresponding CRN can be the

following:

$$A \xrightarrow{k_1} C$$
$$B \xrightarrow{k_2} C$$

where $k_1 = k_2$. But there are two input species, which make the model more complex. This function is then interesting to check the biological constraints and the result of the algorithm with the presence of two inputs. As for cosine, the CRN resulting from evolution is compared with the CRN just above.

To evaluate the problem of the functions of input, Heaviside is studied again, but as an input-output function now. It is interesting because in biology some step functions can be observed to filter the input, for example in the MAPK signaling cascade [25] that acts as an analog-digital converter. Moreover, Heaviside is not computable, so the CRN resulting from evolution is not foreseeable.

And finally, we choose to study MAPK signaling cascade [25], as it implements a step function with three levels, each one being a sharper sigmoid. Here the goal is only to approximate the first two levels of MAPK. We would like to compare the CRN found by evolution to the CRN used to generate the data.

4.2 Functions of Time

In this section, we approximate functions of time, given their values on a finite set of points, with the trace of a CRN which corresponds to the solution of a PIVP.

4.2.1 Cosine Function

The first function of time that we try to approximate is the cosine function. As this function is non-positive, we do not ask the PIVP to have biological behavior. In particular, every monomial is allowed and the function can be negative. For this function, we do not consider any CRN. We want to approximate cosine with the first component of the solution of a PIVP. The expectation is to have a very small error. The parameters for the search are given in the Table 4.1.

The best PIVP found is the following :

$$\begin{cases} x'=3.44 \times 10^{-6}x + y \\ y'=-x - 3.47 \times 10^{-6}y \end{cases} \quad \begin{cases} a(0)=1 \\ b(0)=3.83 \times 10^{-13} \end{cases}$$

That can be compared to the PIVP derived mathematically and given in Example 4.

$$\begin{cases} a'=-b \\ b'= a \end{cases} \quad \begin{cases} a(0)=1 \\ b(0)=0 \end{cases}$$

Table 4.1: Parameters for cos

Parameter	Value
Size of population	96
Number of generations	10
Maximum number of variables	10
Maximum time for CMA-ES (s)	180
Maximum time for CMA-ES for the best(s)	600
Points given	100 between 0 and 10
Constraint λ	10^{-8}

The results are presented on figures 4.1 and 4.2. Figure 4.1 represents the cosine function and the simulation of the best PIVP found by evolution. From this table we can see that the true and the found function are indistinguishable. It is a good result because it shows that our algorithm can evolve basic functions. The second figure, Figure 4.2, is the loss value of the best PIVP at each iteration. We can see that from the first iteration that a good approximation was found and an approximation equivalent to the final one was found in at iteration 3. It means that the structure of a PIVP that can approximate cosine was present in the initial population of this run.

A remark is that $\cos$ is solution of any PIVP of the following form, where $ab = -1$:

$$\begin{cases} x'=ay \\ y'=by \end{cases} \quad \begin{cases} a(0)=1 \\ b(0)=0 \end{cases}$$

Then there is an infinity number of possible IVP that generates $\cos$. If we do not take into account the terms in 10^{-6}, the PIVP found correspond to this, with $a = 1$ and $b = -1$. The PIVP found by evolution is then quite similar to the *mathematical* one given above.

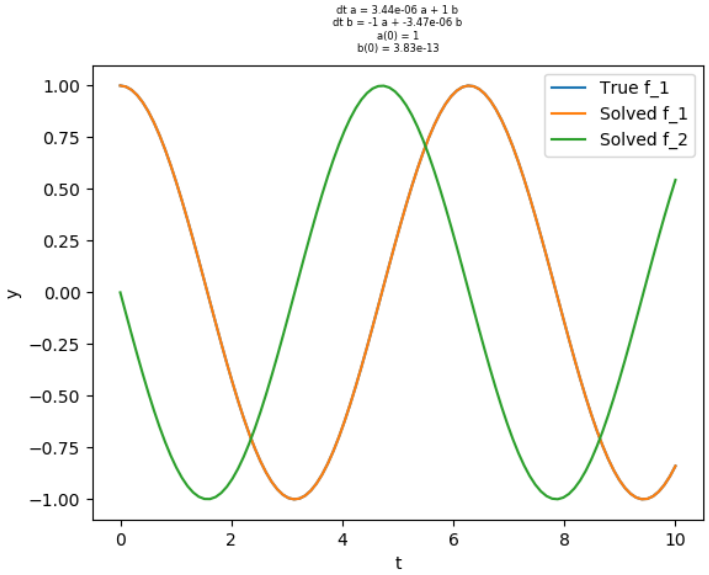


Figure 4.1: Simulation of the best PIVP found by evolution (in orange and green) against cosine (in blue)

4.2.2 Heaviside Function

The function that we want to approximate here is the Heaviside function. Its value is 0 until it reaches a threshold, here 0.5 and it is then equal to 1. This function is not computable in the computational analysis and therefore not the solution of a PIVP. It is interesting to see how the evolution mechanism approximates this non-computable function. The best PIVP found by evolution is the following :

Table 4.2: Parameters for heaviside

Parameter	Value
Size of population	96
Number of generations	80
Maximum number of variables	10
Maximum time for CMA-ES (s)	200
Maximum time for CMA-ES for the best(s)	600
Points given	500 between 0 and 1
Constraint λ	10^{-4}

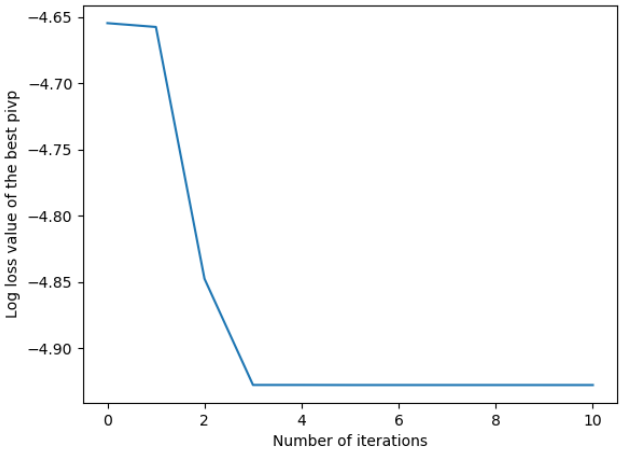


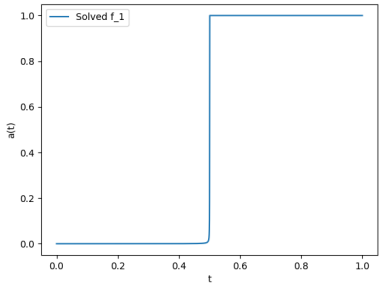
Figure 4.2: Best fitness value over the iterations for cosine

$$\begin{cases} a'{=}0.000111c^2 \\ b'{=}1.79c^2 \\ c'{=}1ab - 1.38bc + 3.07c^2 \end{cases} \qquad \begin{cases} a(0){=}0 \\ b(0){=}0.993 \\ c(0){=}1.14 \end{cases}$$

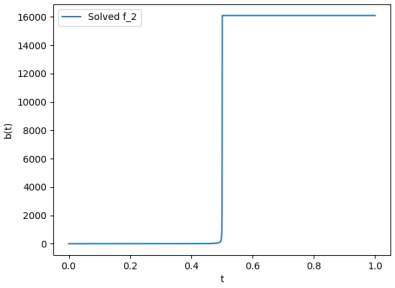
Its fitness function is of 0.0007535 and it was found with the parameters in Table 4.2. It can be noticed that the derivatives of a and b are proportional. So $b = \lambda a + b(0)$. As the different components have a very large difference of order of magnitude (around 1 for a and around 15000 for b), we present each component on a different figure.

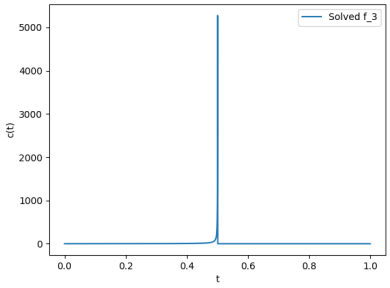
Figures 4.3a, 4.3b and 4.3c are the graphs of the three components of the solution of the PIVP. We can see that b (on Figure 4.3b) is an affine transformation of a (on Figure 4.3a). The component a, which is the component asked to approximate the function, has the behavior of Heaviside for $t \in [0, 1]$.

Figure 4.4 shows the best fitness value at each iteration. There is an alternation of plateau and improvement. This alternation is quite characteristic of genetic algorithms.



(a) First component of the result





(b) Second component of the result (c) Third component of the result

Figure 4.3: Simulation results of the best CRN for Heaviside function of time

The CRN corresponding to the PIVP is:

$$
\begin{aligned}
2\ c &\xrightarrow{0.000111} a + 2\ c \\
2\ c &\xrightarrow{1.79} b + 2\ c \\
a + b &\xrightarrow{1} a + b + c \\
c + b &\xrightarrow{1.38} b \\
2c &\xrightarrow{3.07} 3\ c
\end{aligned}
$$

It is not easy to interpret this CRN, but it seems that the behavior of a comes from the fact that in a first phase, the production of c corresponding to the third and the last equations is more important than its annihilation corresponding to the fourth equation. After $t = 0.5$, the annihilation of c is more important than its production, which slows significantly the production of a. The function found is remarkably

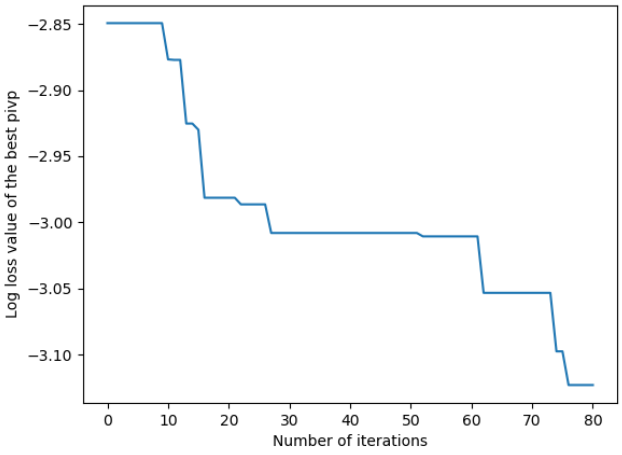


Figure 4.4: Best fitness value over the iterations for Heaviside

stiff and the corresponding CRN is simple with only five reactions. Then the genetic algorithm has given us an unexpected CRN whose solution approximates Heaviside remarkably well.

4.2.3 Cell Cycle

In [29], a model for the cell division is given. The system given can have three different modes: a steady state, an oscillator mode and an excitable switch mode. Here we study the oscillator mode. From simulations of this model, we want to find the corresponding CRN. The parameters for the search are given in Table 4.3. The best PIVP found is the following. It can be noticed that the initial value is not in 0, as there was a first transitory phase between 0 and 3, and we were only interested in the stable phase.

$$\begin{cases} a' = -0.13\ ae + 0.0157\ bg \\ b' = -25\ bc + 1169\ dg \\ c' = 0.748\ df \\ d' = 0.986\ c^2 \\ e' = -19.6\ de - 8.12\ eg + 0.00502\ e + 179\ f \\ f' = -0.104\ ef + 0.000611\ g^2 \\ g' = 8306\ cg - 70202\ fg - 1.52\ g^2 \end{cases} \qquad \begin{cases} a(3.41) = 0.0094 \\ b(3.41) = 0.94 \\ c(3.41) = 0.000546 \\ d(3.41) = 0.000646 \\ e(3.41) = 0.0496 \\ f(3.41) = 0.000079 \\ g(3.41) = 0.000816 \end{cases}$$

It has to be compared to the PIVP given by Tyson :

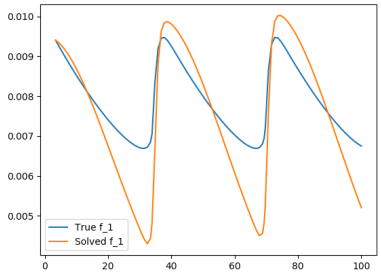
$$\begin{cases} a' = -k8 * a + k9 * b + k6 * d \\ b' = -k3 * b * f + k8 * a - k9 * b \\ c' = -k7 * c + k6 * d \\ d' = k4p * e + k4 * d^2 * e - k5 * d - k6 * d \\ e' = k3 * f * b - k4p * e - k4 * d^2 * e + k5 * d \\ f' = k1 - k2 * f - k3 * f * b \end{cases}$$

Table 4.3: Parameters for the cell division

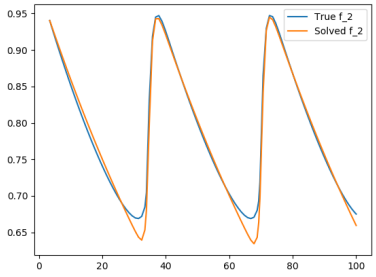
Parameter	Value
Size of population	48
Number of generation	80
Maximum number of variables	10
Maximum time for CMA-ES (s)	200
Maximum time for CMA-ES for the best (s)	600
Points given	100 between 3.41 and 100
Constraint λ	10^{-4}

It can be noticed that there is no monomial in common between the two PIVP. However Figure 4.5 gives the simulation of both the best CRN found by evolution (in orange) and the model (in blue). It shows that two variables were well retrieved, b on Figure 4.5b and e on Figure 4.5e. For the variables a (Figure 4.5a) and f (Figure 4.5f), the results are not the same but the oscillations are present with the right period. The additional component found during evolution, g on Figure 4.5g looks like c and d, with peaks on the same time points. Evolution used a variable with the same behavior than c and d to compute the others. Figure 4.6 gives the best fitness value over the iterations. After only 30 iterations over 80, an approximation close to the final one was found, with only small improvements over the 50 last iterations.

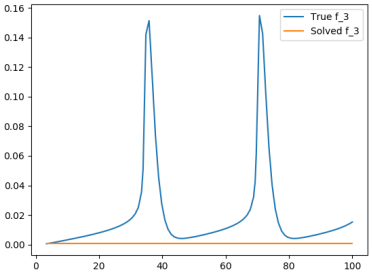
The CRN found by evolution has one more variable than the CRN used to generate the data, but it uses fewer monomials: 15 instead of 19. However, with 19 monomials and six variables, the initial CRN was quite complex.



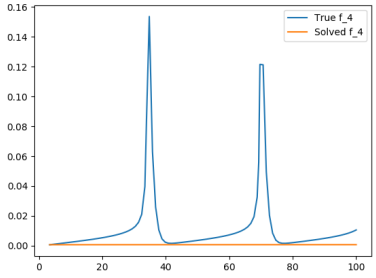
(a) First component of the result: CdC2



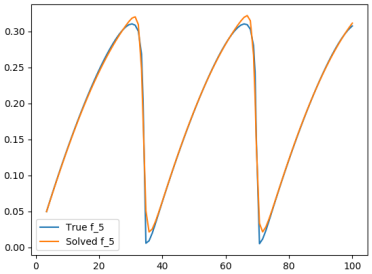
(b) Second component of the result: Cdc2 p1



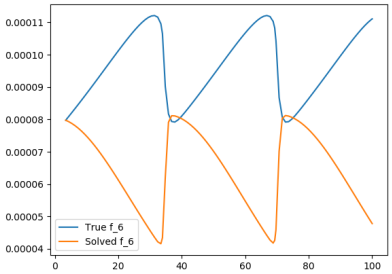
(c) Third component of the result: Cyclin p1



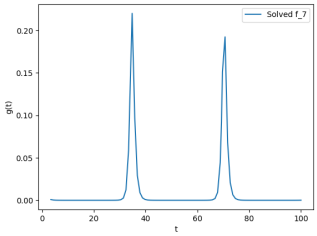
(d) Fourth component of the result: Cdc2-Cyclin p1



(e) Fifth component of the result: Cdc2-Cyclin p1,p2



(f) Sixth component of the result: Cyclin



(g) Seventh component of the result: additional variable

Figure 4.5: Simulation results of the best CRN by evolution for cell cycle in orange and simulations of the model for cell cycle in blue

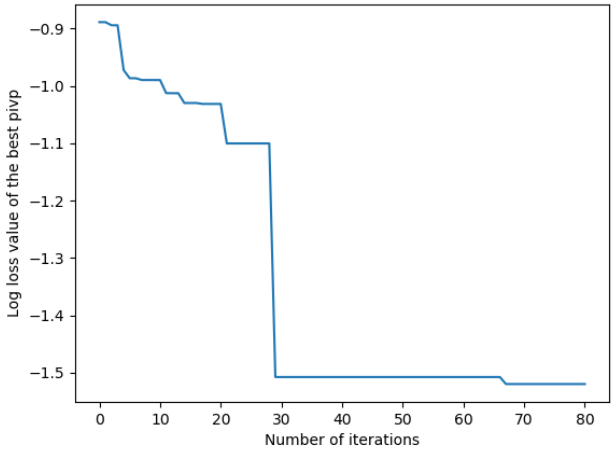


Figure 4.6: Best fitness value over the iterations for cell cycle

4.3 Functions of Input

The goal now is to approximate functions of input, given their values on a finite set of points, with functions computed by a CRN. Whereas we were interested before in the value of the species of the CRN over time, we now only keep their final values. Here the first components of the PIVP are for the input and the following ones for the output.

4.3.1 Cosine Function

In this section we are interested in the cosine function and, therefore, do not require the PIVP to have a biological meaning. We want to evolve the cosine function and compare the PIVP result of evolution to the PIVP mathematically derived (see Example 6). The parameters for the evolution can be found in Table 4.4.

The best PIVP found by evolution, with a fitness function of 5.53×10^{-7} is the following:

$$\begin{cases} a'{=}-292a - 0.182c \\ b'{=}-2.24 \times 10^{9}ac + 4.15 \times 10^{-10}c \\ c'{=}3.8 \times 10^{-5}ab - 1c^2 \end{cases} \qquad \begin{cases} a(0){=}x \\ b(0){=}1 \\ c(0){=}3.82 \times 10^{-17} \end{cases}$$

and the mathematically derived PIVP is

$$\begin{cases} a' = -a \\ b'{=}-ac \\ c' = ab \end{cases} \qquad \begin{cases} a(0){=}x \\ b(0){=}1 \\ c(0){=}0 \end{cases}$$

Here the input is the component a and the result has to be read on the component b.

Figure 4.7 shows the function found by evolution (in green) and the points given to the algorithm (the blue crosses). There is also an orange line, but it can not be seen in the graph, which is the true cosine function between -10 and 20. The result found by evolution is very good because it approximates the function on the point given to the algorithm, but it is good even outside of this range of points. The best fitness function over the iterations on Figure 4.8 is interesting. We can see a very good improvement at iteration 11 and less important improvements afterward. The important improvement laid when the general structure is selected, and after only small changes happen, as

Table 4.4: Parameters for cos

Parameter	Value
Size of population	96
Number of generations	35
Maximum number of variables	6
Maximum time for CMA-ES (s)	300
Maximum time for CMA-ES for the best(s)	600
Points given	12 between 0 and 6.5
Constraint λ	10^{-8}

adding or removing a useless monomial or slightly improving the coefficients.

We now compare this PIVP to the one derived mathematically (the PIVP from Example 6 has been rewritten to match the variables). All the monomials in the mathematical equation are in the evolved PIVP. The evolved PIVP has other terms too. Some of them might not be useful at this point of evolution. It can be another version of a PIVP that computes $\cos$. This example shows that the genetic algorithm can find PIVP that approximates really well cosine whereas this PIVP is slightly different from the mathematical one.

4.3.2 Sum

In the idea of programming biochemically, we now want to study the sum function. The sum function is the really basic function $f(x, y) = x + y$. Since we are in a biological context, all the values are positives here. As for the other functions, the parameters of the search are given in Table 4.5. The best PIVP found had a fitness value of 1.37×10^{-7} and is given here:

$$\begin{cases} a'{=}-2.03 \times 10^{-5}\ a^2 \\ b'{=}2.03 \times 10^{-5}\ a^2 \\ c'{=}8.72\ a^2 + 17.4\ ab + 8.72\ b^2 - 8.72\ c^2 \end{cases} \qquad \begin{cases} a(0){=}x \\ b(0){=}y \\ c(0){=}1.44 \end{cases}$$

For this problem, the input corresponds to variables a and b and the ouput has to be read on variable c. First, we can notice that $a' = -b'$ so the quantity $a + b$ remains constant because its derivative is null. The derivative of c can be rewritten as $c' = 8.72\,[a^2 + 2\ ab + b^2 - c^2]$ which

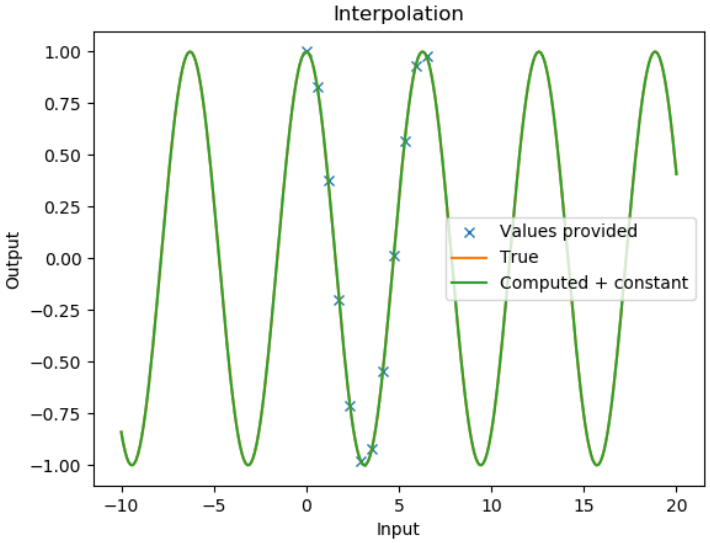


Figure 4.7: Input-output function from the best PIVP in green against the cosine function in orange, with the points provided to the algorithm as blue crosses

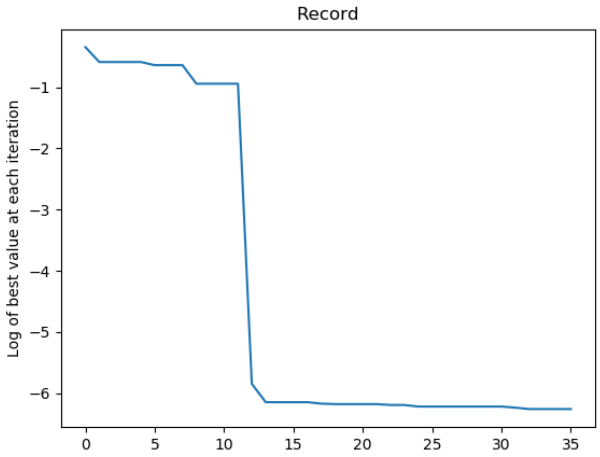
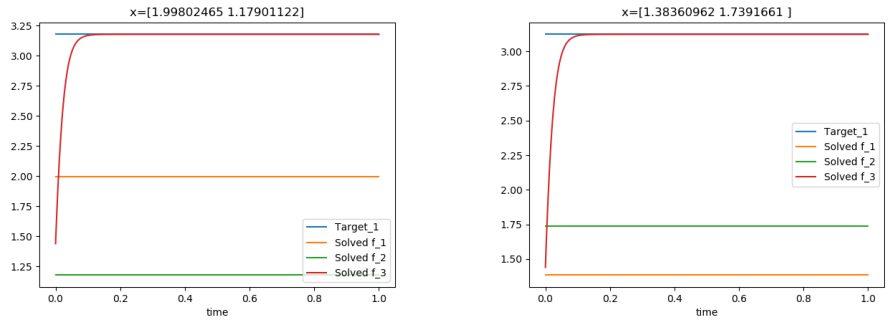


Figure 4.8: Best fitness value over the iterations for cosine

Table 4.5: Parameters for the sum

Parameter	Value
Size of population	48
Number of generations	80
Maximum number of variables	5
Maximum time for CMA-ES (s)	100
Maximum time for CMA-ES for the best(s)	600
Points given	10 in $[1,3] \times [1,3]$
Constraint λ	10^{-8}

gives $c' = 8.72\left[(a+b)^2 - c^2\right]$. This means that the derivative of c is null only if $c = a + b$. As $a + b$ stays constant, the stable value of c is $x + y$. This differential equation has no easy analytical solution, but we can see that if $c < a + b$, then c will grow and if $c > a + b$, c will decrease.



(a) Example of computation for $x = 1.998$ and $y = 1.179$

(b) Example of computation for $x = 1.384$ and $y = 1.739$

Figure 4.9: Examples of computation for the best CRN found by evolution for the sum, the value $f(x, y)$ is in blue

In Figure 4.9 some examples of computation are shown. The output c is the red curve. It can be observed that c grows until it reaches a stable value of $x + y$.

Figure 4.10 shows the best fitness value over the iterations. A good approximation is found after 10 iterations, and the final 70 iterations are only small improvements.

This PIVP was tested on a grid of $[1,3] \times [1,3]$ with a step of 0.1. The mean loss value (without the constraint) on this grid was of $3 \cdot$

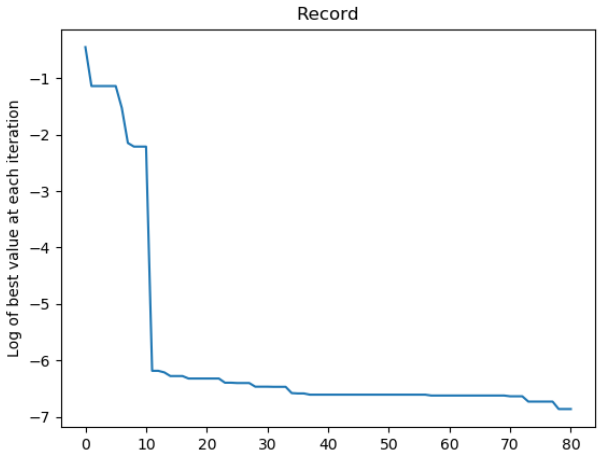


Figure 4.10: Best fitness value over the iterations for the sum

10^{-11}, which means that the PIVP retrieves the sum on the grid tested, although only 10 points were given to the algorithm.

This PIVP corresponds to a CRN with five reactions. The four last reactions have the same rate constant. The last reaction is the annihilation of c which is against its creation with the three reactions of the middle that products c, where a, b and $a + b$ are catalysts, i.e. there are needed for the reaction, but there are not modified.

$$
\begin{aligned}
2\ a &\xrightarrow{0.00002} 2\ a + b \\
2\ a &\xrightarrow{8.72} 2\ a + c \\
2\ b &\xrightarrow{8.72} 2\ b + c \\
a + b &\xrightarrow{8.72} a + b + 2\ c \\
2\ c &\xrightarrow{8.72} c
\end{aligned}
$$

This CRN can be compared to the one usually used to generate the sum function. It can be noticed that it is the CRN used as an example in the introduction.

$$
\begin{aligned}
A &\xrightarrow{k} C \\
B &\xrightarrow{k} C
\end{aligned}
$$

The CRN found by evolution is more complex in term of reactions and in understanding that the one we had. However, if the evolution algorithm finds a CRN even more complex that gives a good approximation, the algorithm has no reason to reject it for a less complex solution. In the algorithm, there is no constraint on the number of monomials, so any good approximation can be kept. In the CRN given in the introduction, the species A and B are destroyed to product C, whereas they only act as catalysts in the reaction of the CRN from evolution. This example shows that the genetic algorithm can approximate a function with an unexpected CRN.

4.3.3 Heaviside Function

We previously studied the Heaviside function as a function of time. Now we study it as a function of input. A Heaviside function of input can be used as an analog-digital converter. The parameters for the evolution are given in the Table 4.6. The best PIVP found by the process of evolution is the following. Its fitness function is 0.000755.

$$\begin{cases} a'{=}-0.186\ a^2 - 82.6\ ac + 0.998\ bc + 34.8\ a \\ b'{=}-1\ ab - 0.999\ b^2 + 1\ bc + 1.09\ a - 15.2\ b \\ c'{=}0.984\ ab - 77.3\ ac - 0.998\ c^2 \end{cases} \quad \begin{cases} a(0){=}x \\ b(0){=}0.999 \\ c(0){=}1.53 \end{cases}$$

The input is given by the first component, a and the output has to be read on the component b.

Table 4.6: Parameters for heaviside

Parameter	Value
Size of population	48
Number of generations	80
Maximum number of variables	10
Maximum time for CMA-ES (s)	200
Maximum time for CMA-ES for the best(s)	600
Points given	12 between 0 and 1
Constraint λ	10^{-4}

Figure 4.11 shows the function computed by the CRN in green and the Heaviside function in orange. They are very close, almost indistinguishable. This means that the corresponding CRN gives an impressively good approximation of the Heaviside function. The blue crosses

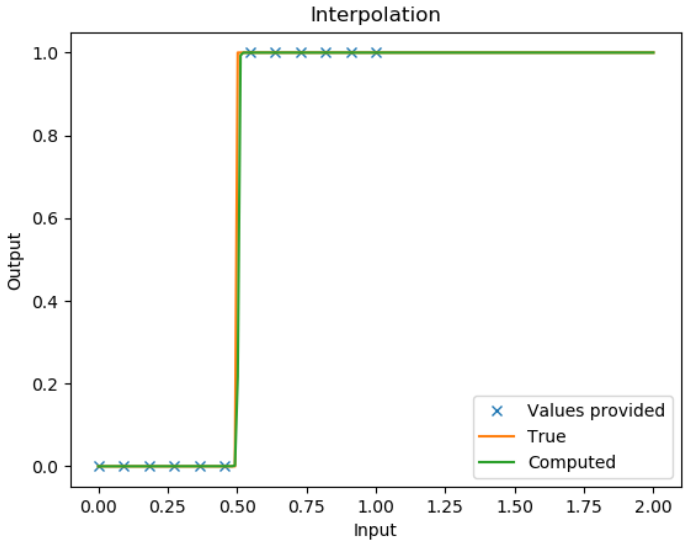


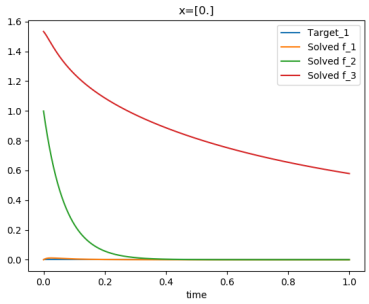
Figure 4.11: Dose-response diagram of the CRN found by evolution in green against the Heaviside function in orange, with the blue crosses being the data provided to the algorithm

are the points given to the algorithm. Figures 4.12a and 4.12b shows the computation of the result for some inputs. The blue line shows the true value $f(x)$, the orange line is the input and the blue one the output.

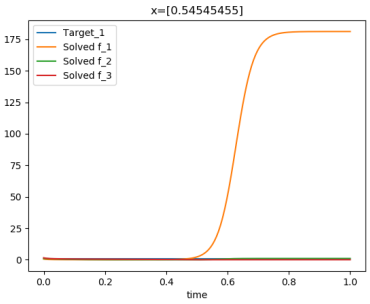
It has to be remembered that only 12 points of data were given to the algorithm. The genetic algorithm gives good results even with a small dataset.

Figure 4.13 shows the best fitness value over the iterations. After 10 iterations, a rather good approximation is found, which is slightly improved over the remaining iterations.

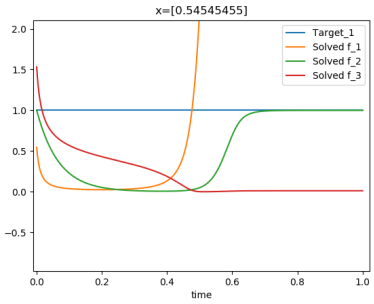
The CRN corresponding to the PIVP is quite complex as it has 12 reactions, therefore we do not present it here. However, having a CRN that approximates the Heaviside function so well is very promising as the function computed by the CRN has the same behavior as Heaviside. As a consequence, it enables us to have an analog-digital converter given by a CRN.



(a) Example of computation for $x = 0$



(b) Example of computation for $x = 0.54$



(c) Example of computation for $x = 0.54$ (zoom)

Figure 4.12: Examples of computation for the best CRN found by evolution for Heaviside

4.3.4 MAPK Cascade

Here, we study another biological model, the mitogen-activated protein kinase (MAPK) cascade simplified to its first two levels. The model comes from [25]. Here we study the response of two proteins according to the concentration of another one.

As the concentrations of the species are very small at the beginning (around 10^{-16}) and grow very fast (to 1), and since we would like to catch the growth rather than the values themselves, it is more interesting to compare the logarithms of the function. For this function only, the loss is replaced by the loss of the logarithms. The inputs given to the algorithm are such that their logarithms are linearly distributed. The parameters used for searching are presented in Table 4.7. The best

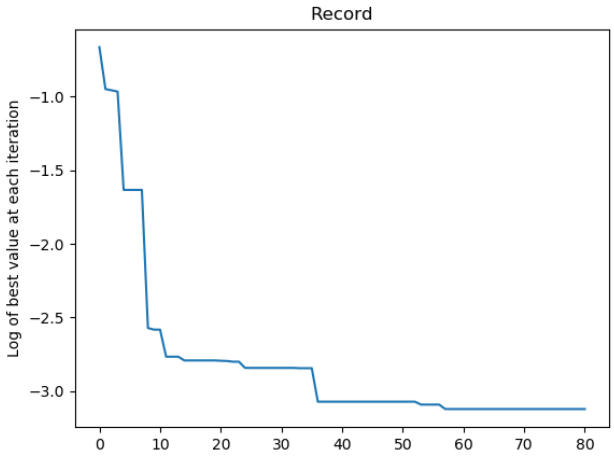


Figure 4.13: Best fitness value over the iterations for Heaviside

PIVP found by evolution is the following:

$$\begin{cases} a'=1.01 \times 10^{5}\ ag + 0.268\ dg + 1.7 \times 10^{-6}\ ef \\ b'=1.27 \times 10^{-10}\ a^2 + 3.51 \times 10^{7}\ dg \\ c'=8.38 \times 10^{5}\ bd + 0.0127\ bg - 4.24 \times 10^{7}\ cf \\ d'=-542\ ad + 3.42\ ae - 0.0298\ de \\ e'=0.0015\ c^2 \\ f'=0.000545\ bd + 4.51 \times 10^{-6}\ c^2 \\ g'=0.00221\ eg \end{cases} \qquad \begin{cases} a(0)=x \\ b(0)=7.41 \times 10^{-15} \\ c(0)=9.22 \times 10^{-20} \\ d(0)=5.39 \times 10^{-18} \\ e(0)=21.3 \\ f(0)=1.61 \times 10^{-7} \\ g(0)=6.36 \times 10^{-10} \end{cases}$$

It can be compared to the PIVP of the model, where p1, p2, p3 are constants of reaction, and E2 and E3 catalysts, which means that their concentrations stay constant.

$$\begin{cases} a'=0 \\ b'=p1\ am - p3\ E2 * b - p2\ bj + p3\ k + p1\ k - p2\ bh + p3\ i + p1\ i \\ c'=p1\ i - p3\ E3 * c \\ h'=p1\ k - p3\ E3 * h - p2\ bh + p3\ i + p3\ E3 * c \\ i'=p2\ bh - p3\ i - p1\ i \\ j'=p3\ k - p2\ bj + p3\ E3 * h \\ k'=p2\ bj - p3\ k - p1\ k \\ m'=p3\ E2 * b - p1\ am \end{cases}$$

Here the input is the component a, which corresponds to the protein E1, and the result has to be read on the components b (KKKp) and c (KKpp).

Table 4.7: Parameters for mapk

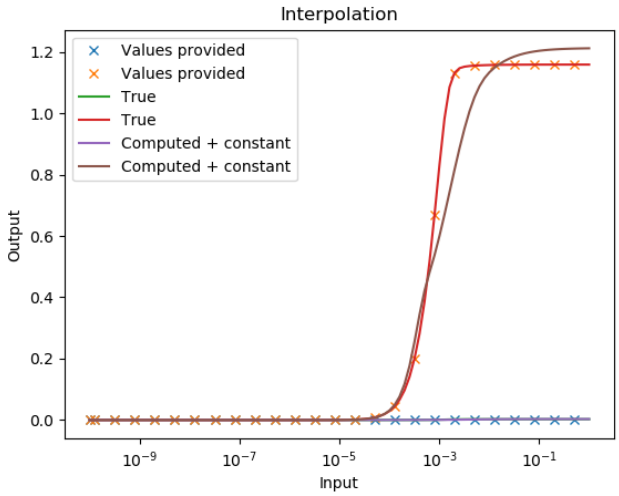
Parameter	Value
Size of population	96
Number of generations	35
Maximum number of variables	10
Maximum time for CMA-ES (s)	300
Maximum time for CMA-ES for the best(s)	600
Points given	26 between 10^{-10} and 0.5
Constraint λ	10^{-8}

The two PIVP seem totally different and the artificially evolved PIVP is much smaller. Our PIVP has 7 variables and 15 monomials whereas the PIVP of the model has 8 variables and 27 monomials. it is difficult to make any match between the non-output variables of the two PIVP.

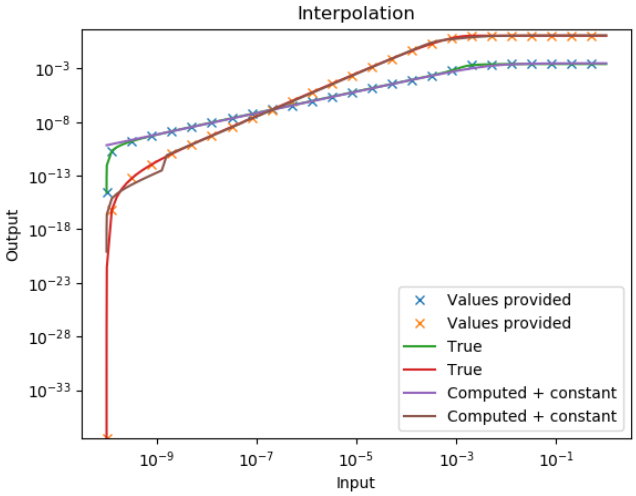
On the figure with the comparison of the logarithms, Figure 4.14b, we can see that except for the small values of x, the logarithm of the function is well captured by the result of evolution. However, on Figure 4.14a, we can see that the values for big x are not perfect.

Figure 4.15 shows the best fitness value over the iterations. Contrary to the other curves of fitness value, there are small improvements over the 70 first iterations and important improvements over the last 10 iterations.

The genetic algorithm was able to approximate the general behavior of the logarithms of the first two levels of MAPK, but with a different CRN.



(a) Function obtained by evolution against the true function



(b) Function obtained by evolution against the true function (both axes are logarithmic)

Figure 4.14: Dose-response diagrams of the best CRN found by evolution and of the original model for MAPK

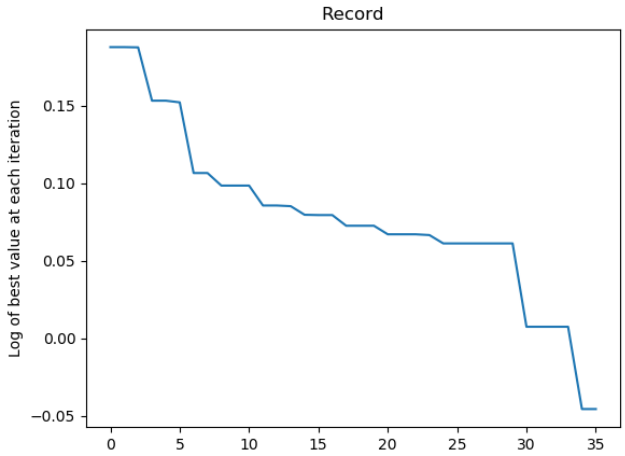


Figure 4.15: Best fitness value over the iterations for MAPK

4.4 Parallelization

To get a better understanding of how parallelism affects the time needed for evolution, we have compared three versions of our program. The population is the same for the three programs at each generation, which means that the population mutates the same way. The seed for CMA-ES is fixed too and no time limitation is given to CMA-ES. The population size is 3, there is only onegeneration. The PIVP studied are the following, with the logarithms of the coefficients given being the starting point of CMA-ES:

$$\begin{cases} a'{=}{-}10\ ac \\ b'{=}10\ f \\ c'{=}{-}10\ e \\ d'{=}{-}10\ bf \\ e'{=}10\ a^2 + 10\ f \\ f'{=}{-}10\ be + 10\ c^2 - 10\ e \end{cases} \quad \begin{cases} a(0){=}x \\ b(0){=}10 \\ c(0){=}10 \\ d(0){=}10 \\ e(0){=}10 \\ f(0){=}10 \end{cases}$$

$$\begin{cases} a'{=}{-}10\ c^2 \\ b'{=}10\ bc + 10\ a - 10\ b \\ c'{=}10\ bc - 10\ c \end{cases} \quad \begin{cases} a(0){=}x \\ b(0){=}10 \\ c(0){=}10 \end{cases}$$

$$\begin{cases} a'{=}{-}10\ a \\ b'{=}10\ b - 10 \end{cases} \quad \begin{cases} a(0){=}x \\ b(0){=}10 \end{cases}$$

The population size into CMA-ES is set to 10 and the initial standard deviation to 2. The function that is approximated is Heaviside function of input as defined in 4.3.3. The three versions are the following : one version with no parallelism at all (called *0 level of parallelism*), one version where only the evaluation of the structure is in parallel and CMA-ES is sequential (called *1-level parallelism*) with three nodes and finally the version with both structure and parameter evaluation paralleled, described above and called *2 level of parallelism* with 30 nodes.

First, the time needed from 0 to 1 cannot be expected to be divided by 3 and the time needed from 1 to 2 cannot be expected to be divided by ten. The first reason is that the communication between the nodes needs some time. Moreover, the evaluation of the parameters and even more of the structure do not need the same amount of time. The time of a paralleled evaluation is at least the time needed for the evaluation the most time-consuming.

Table 4.8: Running time for the two levels of parallelization on the Heaviside function on input example

Levels of parallelism	Running time (s)	Number of cores
None	2380	1
1-level (Genetic)	1955	3
2-level (Genetic, CMA-ES)	571	30

However, we can see that having two layers of parallelism is very cost saving. On this example, the time needed is divided by four with this mechanism.

In our implementation, there is a time limitation for CMA-ES, which means the second layer of parallelism does not change the time needed for a whole run, but more evaluations can be performed during the same time limit.

Chapter 5

Discussion and Conclusion

In this section, we draw conclusions about the results and the method, and we suggest future work that can be performed.

5.1 Evaluation of the Results

In this thesis, we have proposed an genetic algorithm to find CRN that approximate functions, either by the trace of a CRN or a function of input corresponding to the dose-response diagram of a CRN.

Results on functions of time

Concerning the functions of time, the sanity check of cosine was good. The best PIVP found by evolution is very similar to the PIVP known. The evolutionary algorithm can , therefore, be used to approximate at least simple functions.

The results on Heaviside are very interesting because we expected something like a usual sigmoid function, and the function generated by the best CRN is much sharper. The CRN found is quite complex to interpret but the simulations show that it approximates very well Heaviside.

For the cell cycle, the approximation is less good. The CRN found has no reaction in common with the initial CRN. However, the traces of some of the variables were retrieved. Furthermore, another variable was added, which behaves like some other species of the system.

Results on functions of input

For the functions of input, we had two approximations of quite simple functions cosine and sum. The conclusions for both are quite equivalent. The best result of evolution gives a very good approximation which seems mathematically correct, however, it is not the same PIVP than the hidden expected. For the cosine function, the result and the mathematically derived PIVP are quite close, but not exactly the same. For the sum function, the two PIVP/CRN are totally different. It shows that alternate CRN can be found by evolution.

Concerning Heaviside, once again the function found is a very sharp sigmoid that approximates Heaviside very well. The CRN found is quite complex to interpret as it has eight reactions and three variables. The results are way better than expected, because the function is well approximated by a sharp function, and, even if the data points given were between 0 and 1, the approximation still holds between 1 and 2.

Finally, on the MAPK cascade, the function found by evolution approximates well the original function, even if the CRN found by evolution is totally different to the hidden one.

On almost all the examples for both problems, the CRN found approximates well the original function, even if the evolved CRN is quite different from the original CRN. It seems reasonable as there can be several CRN that approximates the same function, and these equivalent CRN cannot be discriminated a priori. It is then not an issue that the evolved CRNs are not the same as the original ones. It is even more interesting, as the new CRN give us new ways to implement these functions.

5.2 Evaluation of the Method

Our method uses two nested genetic algorithms to evolve the PIVP that describes the dynamics of the CRN. A first layer evolves the structure of the PIVP and another one optimizes its parameters. Compared to the method from the proof of Turing completeness 2.3, this method enables us to approximate any function by a CRN, with no prior knowledge of it. The choice of evolving the PIVP corresponding to the dynamics rather than the CRN itself seems reasonable. The structure of a PIVP is easy to modify, and the evaluation of the PIVP is

immediate while for CRN, it requires to derive the PIVP first to evaluate the CRN.

Another choice is to have only a few monomials in the initial population, one or two monomials per derivatives. When a PIVP has a lot of monomials, it can become quite complex. With more monomials, there are more parameters to optimize, which can lead to more evaluations of the PIVP in CMA-ES, then requiring more time.

Moreover, with a large number of monomials, the corresponding CRN have a large number of reactions compared to the number of species and then is very difficult to interpret.

CRN often lead to stiff systems that are difficult to integrate numerically. In Python, there are not many integrators available. We used the one from `scipy.integrate`. The Heaviside function was the function where the numerical integration causes the most difficulties. For the approximation of functions of time, it happened that the numerical integration for the best PIVP was unstable, depending on the points for which to solve the function. For the approximation of functions of input, the solver returned 0 when it failed, causing a good final value that was not true. Part of this problem could be solved with knowing beforehand if at least one of the component will diverge, causing a failure of the numerical integrator.

The last point of discussion is how to improve the performances on a fitting point of view of this method. It is well known that the best way to improve the performances of fitting is to provide more points of learning to the algorithm. In our case, as our data is generated, it is really easy to access data. The more important question is the impact of adding more data on time consumption. Since in our algorithm, CMA-ES is used with a time limit, needing more time means fewer iterations of CMA-ES and then poorer performances. As said before, the most expensive part is numerical integration.

For the problem of approximation of functions of input, adding one more data point means having one more numerical integration. Therefore, it is very expensive to add data points. And, with the time limit, it can be better to have very few data points to have more iterations of CMA-ES. It is sometimes better to have a few data points than a lot for performance. And as seen before, very good results were found for the cosine function with only 12 data points.

Concerning the problem of the approximation of functions of time,

the impact of adding data is less obvious. We consider here that the numerical integrator does not rely on the points of evaluation. If the data points added are in the same range that the already collected data points, then the numerical integration is done anyway, so the time-consuming part will only be in the loss function with more points to compare. This part has already been explored with having 500 data points of comparison. If the new data points are outside of this range, running the algorithm with these new points added requires more numerical integration. Consequently, the numerical integrations will be more time consuming, depending on the range of the new points. Another mean of having more data points is to generate new data points from the same dynamics, but with different initial values, when the dynamics are known. In that case, adding new data provides more knowledge on the function, but it requires a new numerical integration, which requires more time.

With this method, the trade-off between adding new data points and time is very important, and having too many data points can imply poor optimization and then poor performances.

5.3 Future Work

A suggestion for future work is to gain more knowledge about the evolution of the CRN. Due to time cost, we were not able to run each evolution problem many times, but with a more efficient program, it could be interesting to understand the path of evolution, if evolution gives often the same result or very different ones, how it builds the good PIVP.

Another work is to take into account the robustness of the CRNs, i.e. their sensitivity to a small variation of input or coefficients. These CRNs would be more reliable and it could diminish the effect of numerical errors. This could be done at no extra cost in `Biocham` [26] using the Temporal Logic framework for fitness function.

5.4 Conclusion

To conclude, in this Master Thesis we discussed two problems. One was how to approximate functions of time with the trace of CRN with an evolutionary algorithm, and the other how to approximate func-

tions of input with functions corresponding to dose-response diagrams of CRN with an evolutionary algorithm. Both problems were handled with a two-level genetic algorithm. A first genetic algorithm evolved the structure of the PIVP, which represented the dynamics of a CRN, and the other one evolved the parameters of the PIVP. This method gave interesting results. The CRN resulting from evolution gave very good approximations that were often different from the hidden ones. Evolution enables us to consider alternate CRNs to implement functions.

Bibliography

[1] Olivier Bournez et al. “Polynomial differential equations compute all real computable functions on computable compact intervals”. In: *Journal of Complexity* 23.3 (2007), pp. 317–335.

[2] Olivier Bournez et al. “The general purpose analog computer and computable analysis are two equivalent paradigms of analog computation”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 3959 LNCS. Springer, Berlin, Heidelberg, 2006, pp. 631–643.

[3] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. “Discovering governing equations from data by sparse identification of nonlinear dynamical systems.” In: *Proceedings of the National Academy of Sciences of the United States of America* 113.15 (Apr. 2016), pp. 3932–7.

[4] Vannevar Bush. “The Differential Analyzer, A New Machine Foe Solving Differential Equations”. In: *Journal Franklin Institute* 212.4 (1931).

[5] Hongqing Cao et al. “Evolutionary modeling of systems of ordinary differential equations with genetic programming”. In: *Genetic Programming and Evolvable Machines* 1.4 (2000), pp. 309–337.

[6] Hongqing Cao et al. “Evolving cell models for systems and synthetic biology”. In: *Systems and Synthetic Biology* 4.1 (Mar. 2010), pp. 55–84.

[7] David C. Carothers et al. “Some Properties of Solutions to Polynomial Systems of Differential Equations”. In: *Electronic Journal of Differential Equations* 2005.40 (2005), pp. 1–17.

[8] Alexis Courbet et al. "Computer-aided biochemical programming of synthetic microreactors as diagnostic devices". In: *Molecular Systems Biology* 14.4 (2018).

[9] Lisandro D. Dalcin et al. "Parallel distributed computing using Python". In: *Advances in Water Resources* 34.9 (Sept. 2011), pp. 1124–1139.

[10] Lisandro Dalcín, Rodrigo Paz, and Mario Storti. "MPI for Python". In: *Journal of Parallel and Distributed Computing* 65.9 (Sept. 2005), pp. 1108–1115.

[11] Lisandro Dalcín et al. "MPI for Python: Performance improvements and MPI-2 extensions". In: *Journal of Parallel and Distributed Computing* 68.5 (May 2008), pp. 655–662.

[12] Elisabetta De Maria et al. "Design, Optimization, and Predictions of a Coupled Model of the Cell Cycle, Circadian Clock, DNA Repair System, Irinotecan Metabolism and Exposure Control under Temporal Logic Constraints". In: *Theoretical Computer Science* 412.21 (May 2011), pp. 2108–2127.

[13] Huy Q. Dinh et al. "An Effective Method for Evolving Reaction Networks in Synthetic Biochemical Systems". In: *IEEE Transactions on Evolutionary Computation* 19.3 (June 2015), pp. 374–386.

[14] François Fages. "Artificial Intelligence in Biological Modeling". In: *A Guided Tour of Artificial Intelligence Research*. Ed. by Pierre Marquis, Odile Papini, and Henri Prade. Springer-Verlag, In press 2017.

[15] François Fages, Steven Gay, and Sylvain Soliman. "Inferring Reaction Models from ODEs". In: *CMSB'12: Proceedings of the tenth international conference on Computational Methods in Systems Biology*. Vol. 7605. Lecture Notes in BioInformatics. Springer-Verlag, Sept. 2012, pp. 370–373.

[16] François Fages and Sylvain Soliman. "Abstract Interpretation and Types for Systems Biology". In: *Theoretical Computer Science* 403.1 (2008), pp. 52–70.

[17] François Fages et al. "Strong Turing Completeness of Continuous Chemical Reaction Networks and Compilation of Mixed Analog-Digital Programs". In: *CMSB'17: Proceedings of the fiveteen international conference on Computational Methods in Systems Biology*. Vol. 10545. Lecture Notes in Computer Science. Springer-Verlag, Sept. 2017, pp. 108–127.

[18] D.S. Graça and J.F. Costa. "Analog computers and recursive functions over the reals". In: *Journal of Complexity* 19.5 (2003), pp. 644–664.

[19] Nikolaus Hansen and Andreas Ostermeier. *Completely derandomized self-adaptation in evolution strategies.* June 2001.

[20] Nikolaus Hansen et al. "Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009". In: *Proceedings of the 12th annual conference comp on Genetic and evolutionary computation - GECCO '10*. New York, New York, USA: ACM Press, 2010, p. 1689.

[21] Yu Ting Hsiao and Wei Po Lee. "Reverse engineering gene regulatory networks: Coupling an optimization algorithm with a parameter identification technique". In: *BMC Bioinformatics* 15.Suppl 15 (Dec. 2014), S8.

[22] Melanie Mitchell. "An introduction to genetic algorithms". In: *Computers & Mathematics with Applications* 32.6 (1996), p. 133.

[23] Nasimul Noman, Leon Palafox, and Hitoshi Iba. "Evolving genetic networks for synthetic biology". In: *New Generation Computing* 31.2 (Jan. 2013), pp. 71–88.

[24] Amaury Pouly. "Continuous models of computation: from computability to complexity". PhD thesis. Ecole Polytechnique and Universidade Do Algarve, July 2015.

[25] Liang Qiao et al. "Bistability and Oscillations in the Huang-Ferrell Model of MAPK Signaling". In: *PLoS Computational Biology* 3.9 (Sept. 2007), pp. 1819–1826.

[26] Aurélien Rizk et al. "A general computational method for robustness analysis with applications to synthetic gene networks". In: *Bioinformatics* 12.25 (June 2009), pp. il69–il78.

[27] C.E. Shannon. "Mathematical theory of the differential analyser". In: *Journal of Mathematics and Physics* 20 (1941), pp. 337–354.

[28] Daniel Silva Graça. "Some recent developments on Shannon's general purpose analog computer". In: *Mathematical Logic Quarterly: Mathematical Logic Quarterly* 50.4-5 (2004), pp. 473–485.

[29] John J. Tyson. "Modeling the cell division cycle: cdc2 and cyclin interactions". In: *Proceedings of the National Academy of Sciences* 88.16 (Aug. 1991), pp. 7328–7332.

[30] K. Weihrauch. *Computable Analysis: an Introduction*. Springer, 2000.

TRITA -EECS-EX-2019:232