Heuristic optimization algorithms in the study of biological networks

Riu Rodríguez Sakamoto

Heuristic optimization algorithms in the study of biological networks

Riu Rodríguez Sakamoto

Memoria presentada como parte de los requisitos para la obtención del título de Doble Grado en Física e Ingeniería de Materiales por la Universidad de Sevilla.

Tutorizada por

Prof. María del Carmen Lemos Fernández

Junio, 2019

Contents

Abstract

This work uses Genetic Algorithms (GA) to study *in silico* Gene Regulatory Models (GRM) and the relation between their structure and the temporal and spatial behavior. We first present a structure estimation algorithm based on the work by Ando and Iba [2] evaluating the networks temporal expression pattern, and in the second part of the thesis we apply a similar GA to find a GRM able to reproduce L. Wolperts French Flag model [28], evaluating its spatial pattern through simulation using CompuCell3D.

1 Introduction and objectives

1.1 Heuristic algorithms

1.1.1 Approachable problems

Problems whose definition can't allow analytical approaches and whose solution space is too broad to allow a exhaustive search can be found across many fields of science, where vast amounts of information are to be treated. Big Data has rightfully found its application in physics, biology, material science, among other domains in the last couple of decades due to technological advancements and an increasing availability of digitized data. Heuristic algorithms are used to trade accuracy, precision or completeness for speed to give an approximate solution to this kind of problems.

1.1.2 Algorithms

The following three heuristic algorithms have all an inspiration in natural phenomena: ant colonies, metal annealing and genetics. A common term in heuristic algorithms is the *fitness* function, that is, the metric that describes how close a given solution is to achieving the desired goal. Depending on the problem, it can be a simple analytical function or as complex as a result of a simulation.

Ant Colony Optimization

Ant Colony Optimization is a family of algorithms inspired by the behavior of ants in a physical environment and aims to solve discrete optimization problems. Many ant species communicate between individuals and with the environment not visually but with the secretion and detection of chemicals called *pheromones*. Ants use these pheromone trails for example to mark the path to food sources. Other ants can then follow this path and reinforce the path leaving their own pheromones. This positive feedback characterizes the system, where the probability of an ant following a certain path increases with the number of ants that chose the path before.

There are multiple implementations and variations of this family of algorithms, e.g. the *Max-Min* Ant System or the Ant Colony System. The first developed algorithm, called *Ant*

Algorithm 1 Ant System pseudocode (applied to the Travelling Salesman Problem).

$m \leftarrow$ number of ants
repeat
  for all $k = 0$ to m **do** ▷ Construct ant solutions
    $i \leftarrow$ random initial point
    for all construction step **do**
      $j \leftarrow$ next point to visit applying a *random proportional rule*
      $i \leftarrow j$
    end for
  end for
  for all $\mathcal{E}(i,j)$ **do** ▷ Update pheromone
    $\tau_{ij} \leftarrow (1-\rho)\tau_{ij}$ ▷ Pheromone evaporation
    for all $k = 0$ to m **do**
      $\Delta\tau_{ij}^{k} \leftarrow \begin{cases} 1/C^k & \mathcal{E}(i,j) \in T^k \\ 0 & \text{otherwise} \end{cases}$
    end for
    $\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=0}^{m} \Delta\tau_{ij}^{k}$
  end for
until termination-condition

Figure 1.1: In the case of the Travelling Salesman Problem (TSP), i and j represent the cities to visit. For each k-th ant from a total of m ants, a route is built step by step. j is set by a random proportional rule (eq. (1.1)). τ_{ij} is the pheromone present between points i and j. $\rho \in (0,1]$ is the pheromone evaporation rate. $\mathcal{E}(i,j)$ represents the edge or path from i to j. C^k is the length of the tour T^k built by the k-th ant, computed as the sum of lengths of the arcs belonging to T^k. Thus shorter tours get greater amounts of pheromone.

System (Algorithm 1), consists of two phases which repeat until an optimal solution is found:

1. Ant's solution construction: Each ant traverses the search space forming a path.
2. Pheromone update: This is done once all the ants have finished their paths, and the amount of pheromone deposited by each ant is a function of the paths fitness.

The *random proportional rule* is a probabilistic action choice rule associated with the following probability of an ant k to go from point i to point j:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in \mathcal{N}_i^k}[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}} \quad \text{if } j \in \mathcal{N}_i^k \tag{1.1}$$

where $\eta_{ij} = 1/d_{ij}$ is a heuristic value available a priori defined as the inverse of the distance d_{ij}, α and β are parameters which determine the relative influence of the pheromone trail τ_{ij} and the heuristic information η_{ij}, and $\mathcal{N}_i^k$ are the neighbor sites that ant k in site i hasn't visited yet. For a more detailed explanation, along with variants of the Ant System algorithm, see the work by Dorigo et al. [7].

Their applications are broad: Many NP-hard problems can be classified into one of these four categories:

- Routing: Like the Travelling Salesman Problem presented in Algorithm 1, these problems consist of visiting a set of locations in an optimal order. AntNet is an Ant Colony Optimization algorithm designed to solve routing problems in telecommunications networks introduced by Di Caro and Dorigo [6].
- Assignment: This task consists of assigning a set of items $\mathcal{I}$ to a number of resources $\mathcal{R}$ under some constraints, which can be seen as a mapping $f : \mathcal{I} \rightarrow \mathcal{R}$. The objective function is a function of the mapping.
- Scheduling: Allocating a limited amount of resources to tasks over time.
- Subset: The solution to these kind of problems consists of a subset of the available components subject to constraints. E.g. the Weight Constrained Graph Tree Partition Problem, where an undirected graph with weighted nodes and arcs has to be split into different trees -group of connected nodes- all with their weight between a minimum and a maximum. This NP-hard problem is found for example in the design of telecommunication networks and is generally not approximable [7].

Simulated Annealing

This algorithm takes its name from the physical process of steal annealing, that is, the controlled cooling of the metal where initially at high temperatures the atoms in the molten state are randomly located, and later when it gradually solidifies, the steel finds an energetic minimum. Ideally, a slow enough annealing leads to the ground state. A rapid quenching might induce some irregularities that are trapped in, resulting in a solid with higher final energy (local minimum).

The different final states of the metal correspond to the feasible solutions found by the algorithm. The energy corresponds in this analogy to the fitness function, and ground state to the optimal solution of the problem. A fast cooling is analogous to a local search, where the solution converges to a local optimum.

The general operation of the simulated annealing algorithm is very similar to that of a local search: it starts at a random point x_c in search space S. It evaluates x_c with the fitness function. Finds a new point x_n in the neighborhood of x_c and evaluates it too. If x_n has a better fitness than x_c, x_n replaces x_c. Else, x_n replaces x_c only with a probability $P(x_n) = \exp(-\delta/T)$, where δ is the difference in fitness of x_c and x_n. The general structure is described in Algorithm 2.

Algorithm 2 Simulated Annealing pseudocode (minimization problem).

```
t ← 0
initialize T > 0
select random x_c ∈ S
evaluate x_c
repeat
    repeat
        select new point x_n ∈ S near x_c
        if eval(x_n) < eval(x_c) then
            x_c ← x_n
        else if U[0, 1) < exp(−δ/T) then
            x_c ← x_n
        end if
    until termination-condition
    T ← g(T, t)
    t ← t + 1
until halting-condition
```

Figure 1.2: x_c and x_n are points of the search space. If the fitness of x_n is lower than x_c (lower thus better, as this is a minimization problem), it replaces it and searches for a new x_n in the vicinity of the newly assigned x_c. If not, it replaces x_c with x_n only if a random value between 0 and 1 is lower than $\exp(-\delta/T)$, where δ is the difference in fitness of x_c and x_n.

Genetic Algorithm

Genetic Algorithms (GA) are based on the mechanics of genetics and natural evolution, and were firstly developed by Holland (1975) in his book *Adaptation in Natural and Artificial Systems* [13] as a method to both understand the natural adaptation processes in living organisms and to design a mathematical framework common to other fields like economics ('optimal planning'), artificial intelligence and psychology ('learning').

GAs search iteratively for a solution of an optimization problem. Each candidate solution (also called an *individual*) is a set of properties coded in a 1D array of data called *chromosome*. The algorithm starts with a random population of these candidate solutions and applies various operations to mutate and alter the individuals in the population each generation, until an individual with an acceptable fitness value has been found or until the maximum number of generations has been reached (Algorithm 3). The basic operators are:

- Mutation: An individual is mutated in a random point of its chromosome, resulting in a slightly different individual with a similar fitness value to its original. This operator introduces some variability that can act as a local search in solution space. In the case of a simple 1D-array of bits, a random position m is chosen and it's value is flipped.
- Crossover: Two individuals exchange parts of their chromosomes. The specific way of crossing two chromosomes depends on the problem design, but in the simple case of a 1D-array of bits, a random position p is chosen and each sub-array is exchanged with

Algorithm 3 Genetic Algorithm pseudocode (minimization problem).

$max\_gen \leftarrow$ number of maximum generations
$goal\_fitness \leftarrow$ desired value of the evaluation function
$population \leftarrow$ population of random individuals
$g \leftarrow 0$
repeat
  for all $individual \in population$ **do**
    $eval(individual)$
  end for
  $population \leftarrow mutation(population)$
  $population \leftarrow crossover(population)$
  $population \leftarrow selection(population)$
  $g \leftarrow g + 1$
until $g = max\_gen$ **or** $\exists\, individual \in population : eval(individual) < goal\_fitness$

Figure 1.3: In each generation g of a maximum of max_gen, each individual in the population is evaluated and the whole population is the mutated, breed and selected to produce the population of the next generation.

the other individual.

- Selection: At the end of each generation, a portion of the existing population is selected to breed a new generation. Individuals with better fitness values are more likely to survive. Out of the many selection operators, the simplest is taking the best n individuals of the current generation.

Analogies

We have seen these three examples of heuristic algorithms and their basic functionality. In order to explain all of them, an analogy was used to give an intuition of their inner mechanism. These analogies are shown in Table 1.1. As it often happens, making these analogies between seemingly unrelated topics gives some insightful intuitions in the process of algorithm design. It is useful to have these similarities in mind as a tool and not as a constraint on what can or cannot be modified.

Optimization problem	Physical system		
	Ant Colony System	Simulated Annealing	Genetic Algorithm
Feasible solution	Ant path	Metal microscopic state	Chromosome
Evaluation function	Pheromone update	Energy	Life
Local search	$\eta_{ij} \gg \tau_{ij}$	Rapid quenching	Genomic uniformity
Optimal solution	Best ant	Ground state	Best individual

Table 1.1: Analogies.

1.2 Gene Regulatory Models

Gene Regulatory Models (*GRMs*) describe the interaction of DNA, RNA and proteins to express transcriptomic and proteomic behavior of a cell, coded in a DNA chain. GRMs can describe for example the *cell fate determination*, or how a particular stem cell develops into the final cell type with specialized structure and function, as proposed by Okawa et al. [22]. The DNA chain is formed by instructions that specify the interaction between different genes that activate or inhibit the production of other genes. Gene Regulatory Networks (GRNs), also called transcription networks, are graph representations where the interactions between genes are represented as edges between nodes. Each gene of the network has an expression level, which is simply the amount or concentration of that particular gene in the network. A GRN modeling about the 20% of the transcription interactions in the bacterium *E. coli* is shown in Figure 1.4. Depending on the nature of the interactions, multiple formulations - not necessarily independent from each other- can be given for *in silico* simulations of these networks:

- Continuous networks: continuously-weighted interactions (edges) between genes (nodes) with continuous expression levels, similar to artificial neural networks in the field of machine learning - in particular, recurrent neural networks [20]-.
  - Directed Cyclic Graph: Markov network with continuous and bounded expression levels of genes, as in [2, 18, 19]. They are Markov networks because genes don't retain any memory, so when modeled as a stochastic process, the conditional probability distribution of the next time step only depends on the state of the previous step [11].
  - Directed Acyclic Graph (DAG): Bayesian network which contains no cyclic relations. The acyclic property allows the definition of node's children and parents. From this relations we can define a Markov Blanket of a node, consisting of its parents, children and any other parents of its children. The analysis of Markov Blankets using Conditioned Independence tests gives an insight into the probability distribution of the expression pattern [23].
- Boolean networks: Genes can only be in two possible states: active or inactive. This simplification accelerates simulation but diminishes the precision of results [18].
- Coupled ordinary differential equations: They specify the rate of change of concentration of each gene X_i as a function of the current concentrations of all genes as

  $$dX_i/dt = f_i(X_1, X_2, \ldots, X_N)$$

  Analyzing the fixed point of the system ($dX_i/dt = 0 \quad \forall i$) one obtains constant expression patterns. However, this cannot guarantee their stability nor if such states can be reached from initial conditions.
  - System of linear differential equations: They are used to model the expression level dynamics by Ando and Iba [2]. The linearity means that a gene is activated based on a linear combination of other genes from which there's a connection (edge). There's no product between concentrations, so this model doesn't take into account the synergistic action of transcription factors. "A simple example of synergy is a

situation in which only the complex of two different transcription factors is able to repress transcription, whereas either of the constituents of the complex have no effect. In that case, repression will only occur under conditions in which both transcription factors are being expressed" [16].

- Stochastic gene networks: The stochastic nature of gene expression, backed up experimentally [8], favors the introduction of fluctuations and correlations (Gillespie algorithm, master equation) in numerical simulations in order to avoid invalid results obtained from purely deterministic solutions. These techniques can be implemented within the framework of coupled differential equations [12].

These models are independent of the posterior study by genetic algorithms, but give different information to the heuristic algorithm for its evaluation and model manipulation.

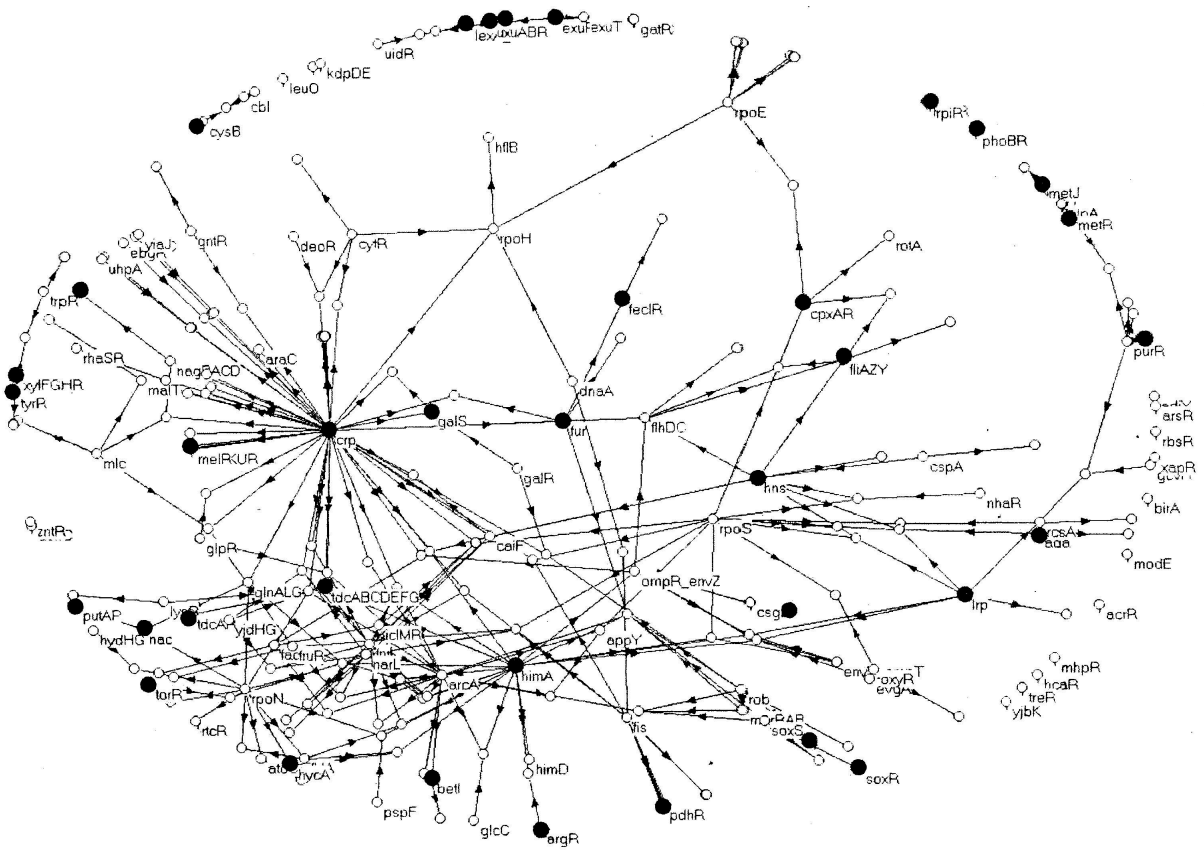


Figure 1.4: Experimental GRN representing about 20% of transcription regulations in the bacterium *Escherichia coli*. Nodes are genes and directed edges indicate regulations. [1]

1.3 Objectives

The main goal is to find a GRN with specific edges and connection weights so that its observable properties -the expression level of the genes across time or space- can match some known pattern. Two different strategies are laid out:

Optimization across time (Figure 1.5a) Search for a GRN comparing the whole evolution of the expression levels of one cell across a period of time with the one for a known target GRN. The experimental network and the target network share the same initial conditions. This method is similar to the ones used by Sakamoto and Iba [24] and by Ando and Iba [2]. Sakamoto and Iba propose a structure of the kinetic equations - including non-linear terms, e.g. the derivative of X_1 can depend on $X_1 \cdot X_2$ like $dX_1/dt = kX_1X_2$ -, from which the coefficients k are the variables that it are searched for. In the first part of this work a system of linear differential equations is proposed, where the elements of the influence matrix W are the variables to be optimized. These values dictate which genes influence which other and the strength of this influence, or in other words, the coefficients k_{ab} for genes a and b in

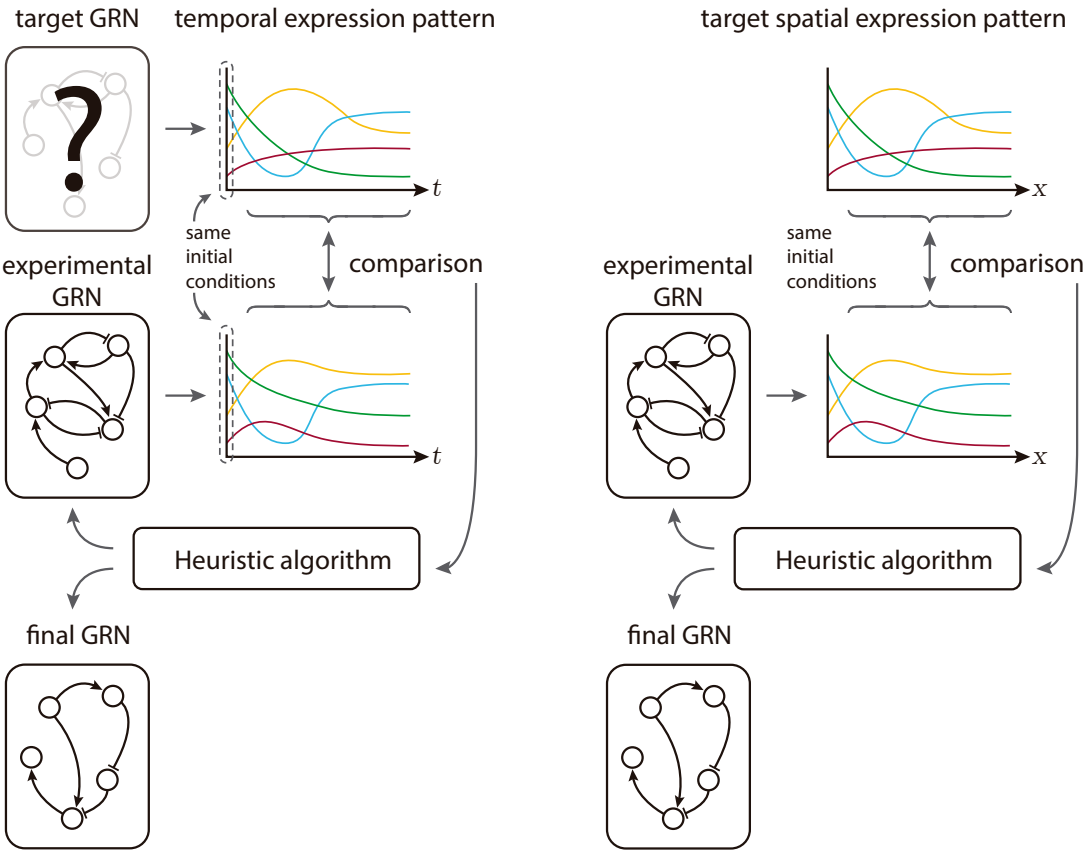
$$dX_b/dt = \cdots + k_{a-1,b}X_{a-1} + k_{ab}X_a + k_{a+1,b}X_{a+1} + \cdots$$

Optimization across space (Figure 1.5b) Here we search for the desired GRN comparing the final stable expression levels across space. This requires the simulation of multiple cells (all with the same GRN, but with different initial conditions) laid out in physical space. This brings the possibility of introducing interactions of cells with the environment and between cells, which complicate the simulation procedure. We therefore resort to a simulation software called CompuCell3D.

1.4 Used tools

- Python: a high-level language focused on rapid application development (RAD).
  - DEAP (*Distributed Evolutionary Algorithms in Python*) [10] is a Python package providing a framework for evolutionary algorithms and a set of tools to log various statistics. It provides sufficient freedom to program custom algorithms, and its codebase is available at `https://github.com/deap/deap`. We use this as a base to program our heuristic algorithms.
- CompuCell3D [26]: This program provides a flexible and extensible simulation environment to evaluate our GRNs across space. According to the web page:

  > "CompuCell3D is a flexible scriptable modeling environment, which allows the rapid construction of sharable Virtual Tissue in-silico simulations of a wide variety of multi-scale, multi-cellular problems including angiogenesis,



(a) Optimization across time

(b) Optimization across space

Figure 1.5: Two different strategies to search for GRNs using heuristic algorithms.

bacterial colonies, cancer, developmental biology, evolution, the immune system, tissue engineering, toxicology and even non-cellular soft materials. CompuCell3D models have been used to solve basic biological problems, to develop medical therapies, to assess modes of action of toxicants and to design engineered tissues [...]. It uses Cellular Potts Model to model cell behavior."

– SBML (*Systems Biology Markup Language*) is a "software-independent language for describing models common to research in many areas of computational biology, including cell signaling pathways, metabolic pathways, gene regulation, and others."[14]. This tool allows us to express the dynamics of our GRNs in a way that CompuCell3D understands.

2 Optimization across time

2.1 Modeling Methodology

The method to implement the GRN is the one used by Ando and Iba [2]. The network is expressed mathematically as a matrix of weights connecting influencing genes or nodes (rows) to influenced genes or nodes (columns). Each element of the matrix corresponds to the weight or regulation strength, which is a continuous value between maximally repressing (-1) to maximally activating ($+1$). The expression levels for each gene are continuous values bounded between complete repression (0.0) to maximal expression (1.0). The state of the network is represented as a vector $\vec{x}(t)$, where each element $x_i(t)$ represents the expression level of gene i at time t. The dynamics are expressed in the following equation:

$$x_i(t + \Delta t) = x_i(t) + r_i \cdot s\left(\sum_{j=0}^{n} \omega_{ji} x_j(t) + h_i\right) - \alpha_i \cdot x_i(t) \tag{2.1}$$

where r_i are scaling coefficients, α_i are decay coefficients, h_i are the *basal activation rate* - that is, the activation rate for gene i when all genes are repressed, $x_j(t) = 0 \quad \forall j$ - , the weight matrix coefficients are represented by ω_{ji}, and $s(x)$ is a sigmoid function:

$$s(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

2.2 Motivation

Figure 2.1 represents the histogram of stable activations $x_i(t \to \infty)$ of the different genes of a GRN with 30 genes following (2.1) from multiple random initial conditions: initial expression levels are drawn from an uniform distribution between the minimum and maximum possible values $[0, 1]$. It can be observed that although the majority of genes stabilize in a narrow range of values independently of initial conditions, some genes have different stabilization levels ($X26$ and $X1$), which suggests that in certain situations the GRN ends up behaving in different modes depending on the history that it went through. We can therefore assert that (2.1) is capable of describing a model for cellular differentiation, given the right coefficients r_i, α_i, h_i and matrix weights ω_{ij}.

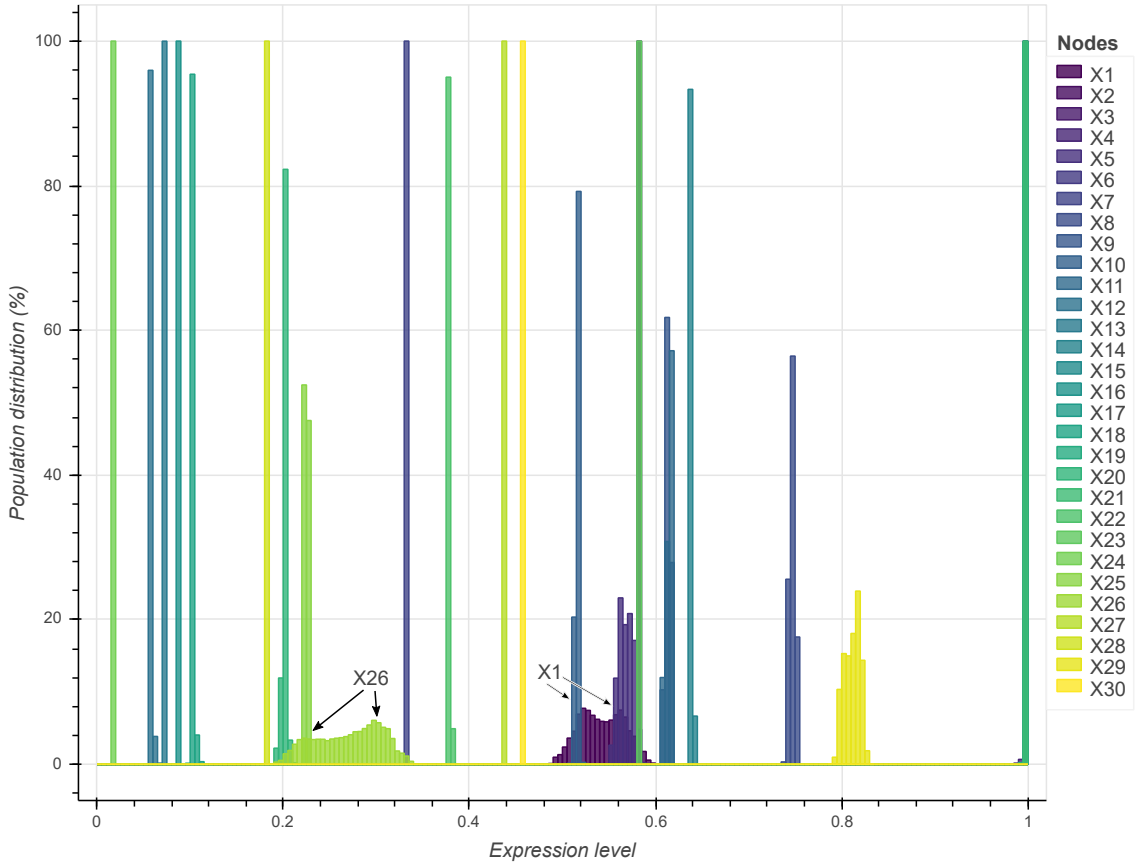


Figure 2.1: Histogram of stable expressions levels from random initial conditions.

2.3 Genetic Algorithm (GA) implementation

We want to obtain the values of the weight matrix w_{ji} that give rise to a certain time expression pattern. To format the matrix in a way that a GA can use, we concatenate the rows of the matrix in a single 1D array of real values, as done in [2]. As we are using a GA, we have to define how we implement the *fitness*, *mutation*, *crossover* and *selection* operators explained in Section 1.1.2.

Fitness

The fitness of an individual is computed from two values: the linear norm δ and the parsimony factor P.

- The linear norm between the generated expression pattern $x_i(t)$ and the target expression pattern $y_i(t)$ through time is given as:

$$\delta = \sum_{i=0}^{N} |y_i(t) - x_i(t)| \tag{2.3}$$

- The parsimony factor P is proportional to the number of pathways through the GRN and the number of non-zero elements m of the influence matrix.

$$P = T \cdot m \tag{2.4}$$

$$T = \frac{0.01 \cdot \#pathways}{N^2} \tag{2.5}$$

The value 0.01 is simply a scaling factor to be combined later with the linear norm into a single fitness value (2.8). It is the same as the one used by Ando and Iba [2].
The number of pathways is strictly speaking infinite, because the GRN is a cyclic graph. We limit the number of pathways calculating them using the adjacency matrix. The adjacency matrix is a square matrix whose element A_{ij} is 1 if there's an edge from node i to node j, and 0 otherwise. $\mathcal{E}$ is the set of all edges of the graph.

$$A_{ij} = \begin{cases} 1 & \text{if } (i \to j) \in \mathcal{E} \\ 0 & \text{if } (i \to j) \notin \mathcal{E} \end{cases} \tag{2.6}$$

If we raise A to the n-th power, each element $(A^n)_{ij}$ has the number of paths from node i to node j consisting on n edges or hops. Therefore, if we sum all elements of A^n for n up to the number of nodes N, we will cover all possible paths through the graph.

$$\#pathways = \sum_{n}^{N} \sum_{i,j} (A^n)_{ij} \tag{2.7}$$

The justification behind the use of this second metric P is based on the Minimum Description Length principle [2]. Using P, the search is biased towards smaller, less interconnected networks, e.g. to the parsimonious model.

The total fitness is a weighted combination of the two values. DEAP allows to give multiple fitnesses individually and their corresponding weights.

$$fitness = w_1 \cdot \delta + w_2 \cdot P \tag{2.8}$$

Two-point crossover

The crossover is the standard two-point crossover: A randomly selected position splits each individual in two. One section of one individual is swapped with the same section of the other individual, maintaining the total length of both individuals. Iterating in pairs over the complete population, the probability of mating two individuals is p_{cx}.

Gaussian mutation

Mutation is done with a set probability p_{mut} for each individual of the population in a given generation. For an individual to be mutated, each of it's elements has a probability p_{ind} to suffer a Gaussian mutation of mean $\mu = 0$ and standard deviation $\sigma = 0,2$. The values set for these *hyperparameters* is justified, as it often happens in many optimization techniques, through trial and error. We also take these values from the ones used by Knabe et al. [16].

Tournament selection

Tournament selection is also a standard procedure: it selects the best individual among three randomly chosen ones and repeats until it obtains enough to make a new generation. The criterion to choose the best individual is their fitness.

2.4 Statistics: Receiver Operating Characteristic

Strictly speaking, a receiver operating characteristic curve is used to evaluate a binary classifier. It plots the true positive rate against the false positive rate. These statistical measures can be defined in our case for comparing graphs following the criteria shown in Table 2.1.

- True positive rate (TPR) is the proportion of actual activating edges in the target graph that are correctly identified as such in the acquired graph. It is also called sensitivity.

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} = 1 - FNP = sensitivity \tag{2.9}$$

  where TP are true positives, P are total positives, FN are false negatives and FNP is false negative rate.
- False positive rate (FPR) is the ratio between the number of repressing edges wrongly categorized as activating and the total number actual repressing edges. It can also be defined as $1 - specificity$, where the specificity or false positive rate is the proportion of actual repressing edges that are correctly identified as such.

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP} = 1 - FPR = 1 - specificity \tag{2.10}$$

  where TN are true negatives, N are total negatives, FP are false positives and FPR is false positive rate.

Measure	Definition
TP	An edge that is activating in both the target and acquired graphs.
TN	An edge that is repressing in both the target and acquired graphs.
FP	An edge that is repressing in the target graph but activating in the acquired graph.
FN	An edge that is activating in the target graph but repressing in the acquired graph.
P	Total number of activating edges in the target graph.
N	Total number of repressing edges in the target graph.

Table 2.1: Statistical measures defined for graph comparison.

2.5 Results

The Genetic Algorithm parameters used are different for each graph. In general, bigger or more complex GRNs require more individuals per generation (the population pop is higher) and the GA should run more generations to find a suitable solution. The parameters for each network are shown in Table 2.2. For the first target graph, the target and acquired networks are shown side by side in Figure 2.2. The evolution of the sensitivity and specificity (Section 2.4) is shown in the Receiver Operating Characteristic in Figure 2.3, and the fitness in Figure 2.4. The comparison of the expression levels evolutions across time is displayed in Figure 2.5.

Similar results were obtained for network #2 (Figures 2.6, 2.7, 2.8 and 2.9). However, for bigger GRNs (network #3: Figures 2.10, 2.11 and 2.12), the algorithm fills the weight matrix ω_{ij}, even against the parsimony factor P introduced in the fitness function, resulting in overregulated GRNs. The results are summarized in Table 2.3. For network #1, the Genetic Algorithm successfully replicated 8 out of the 10 edges in the network, out of the $7^2 = 49$ possible, with a edge weight error of 14.96%. This edge weight error is determined from the average difference:

$$e = \frac{1}{n} \sum_{i=1}^{n} \left| \omega_i^{(t)} - \omega_i^{(f)} \right| \tag{2.11}$$

where $n = \left|\mathcal{E}^{(t)}\right|$ is the number of edges in the target network, i.e. the sets cardinality, and $\omega_i^{(t)}$ and $\omega_i^{(f)}$ are the weights of the i-th edge in the target and final graphs respectively. If the final graph doesn't have the corresponding i-th edge, $\omega_i^{(f)} = 0$.

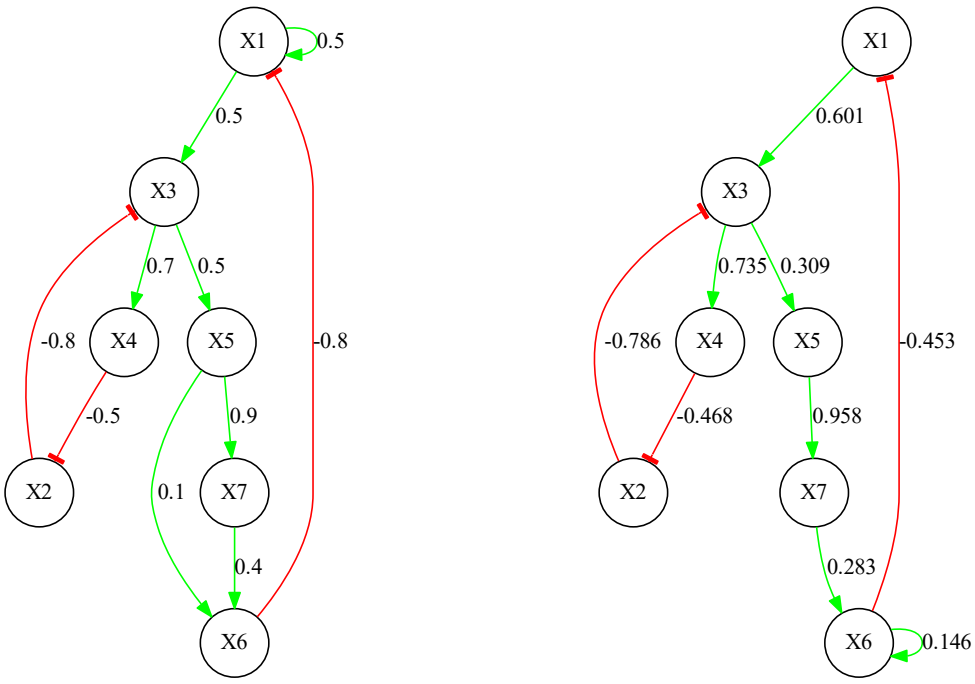
#	N	pop	gen	p_{cx}	p_{mut}	p_{ind}
1	7	1000	150	0.99	0.01	0.01
2	10	7000	150	0.99	0.01	0.01
3	30	8000	600	0.99	0.01	0.01

Table 2.2: GA parameters for the tested networks. N is the number of nodes, pop the number of individuals per generation in the GA, gen is the total number of generations to be computed. p_{cx} is the probability of mating two individuals, p_{mut} the probability of mutating an individual, and p_{ind} the independent probability of each attribute within an individual to be mutated.

#	Fitness	Edges in target	Edges in final	Missing edges	Extra edges	e
1	0.012434	10	9	$X1 \to X1$ $X5 \to X6$	$X6 \to X6$	14.96%
2	0.006762	13	13	$X8 \to X7$	$X7 \to X1$	4.35%
3	0.207424	35	73	see Fig. 2.12 and Fig. 2.11		31.36%

Table 2.3: Properties of the final networks obtained. e is the edge weight error (2.11).

2.5.1 Network #1



(a) Target graph

(b) Acquired graph

Figure 2.2: Comparison between the target and acquired GRNs for network #1. Green edges indicate activating regulations ($\omega_{ij} > 0$) and red edges indicate inhibitory regulations ($\omega_{ij} < 0$). $X1 \rightarrow X1$ and $X5 \rightarrow X6$ are missing regulations, and $X6 \rightarrow X6$ is an extra regulation.

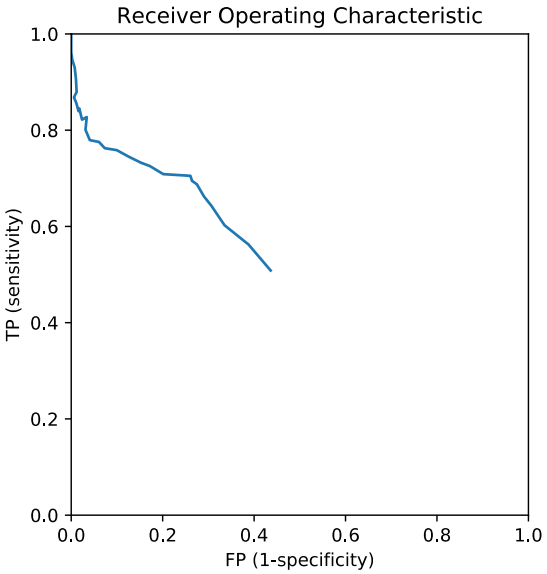


Figure 2.3: Receiver Operating Characteristic evolution for network #1. Center is random, top left is better. The initial population of GRNs starts at the center, and gradually moves towards better individuals as generations evolve.

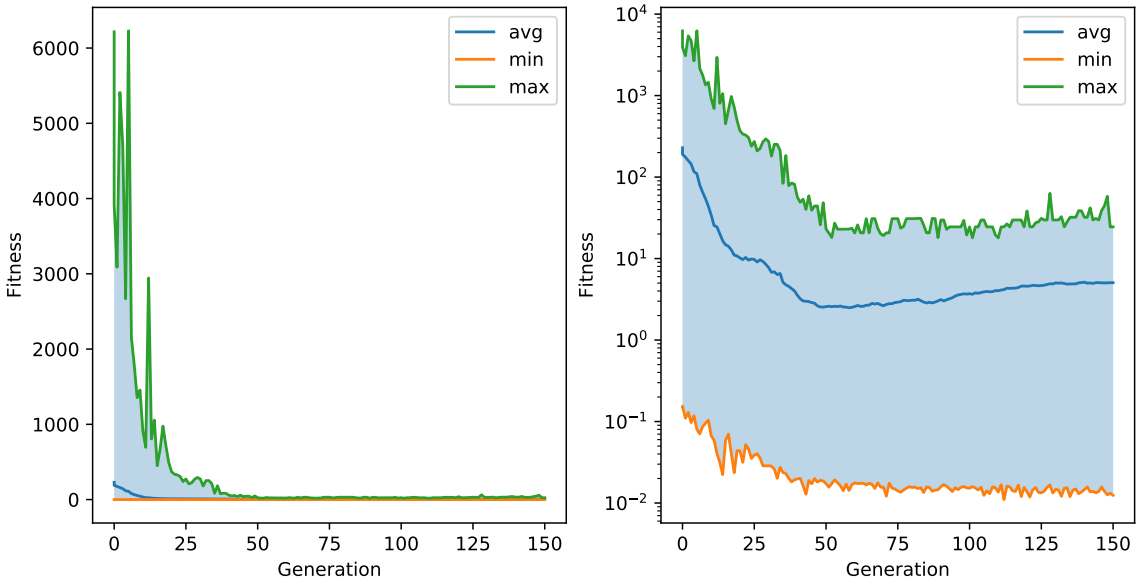


Figure 2.4: Evolution of fitness (objetive function) across generations in scalar (left) and log (right) scales for network #1. Although the best individual (lowest curve, orange line) keeps scoring better fitness values until the end, it's drop rate diminishes significantly, together with the average fitness (blue line) at around generation 50, which suggests a decrease in the marginal efficiency of the algorithm for this network.

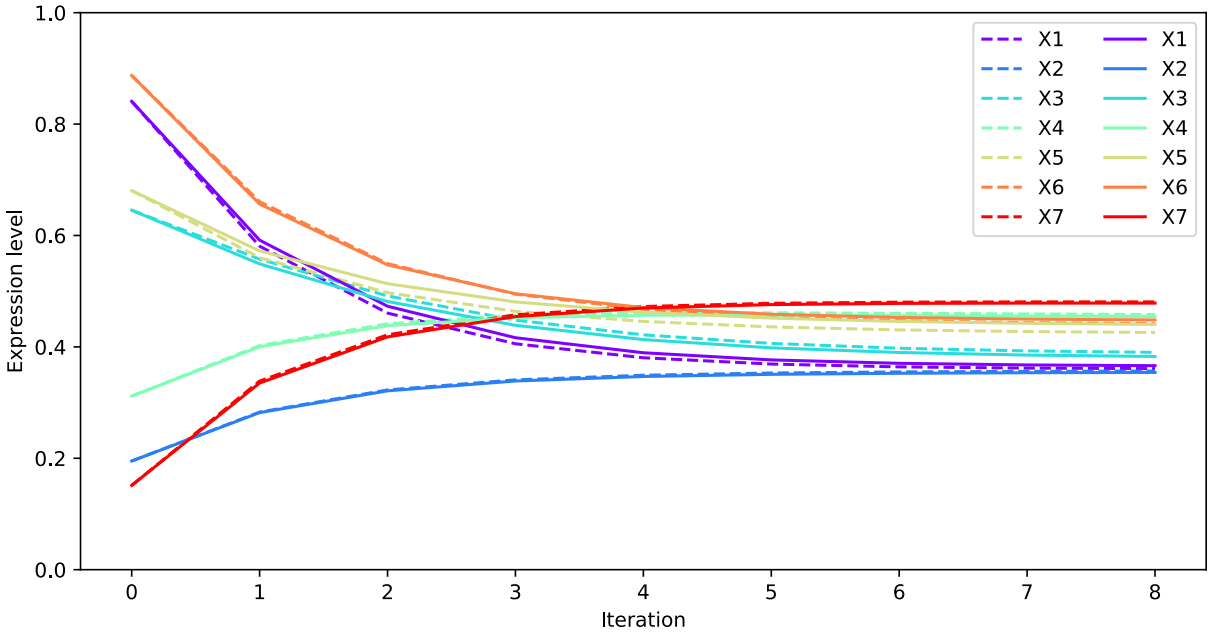
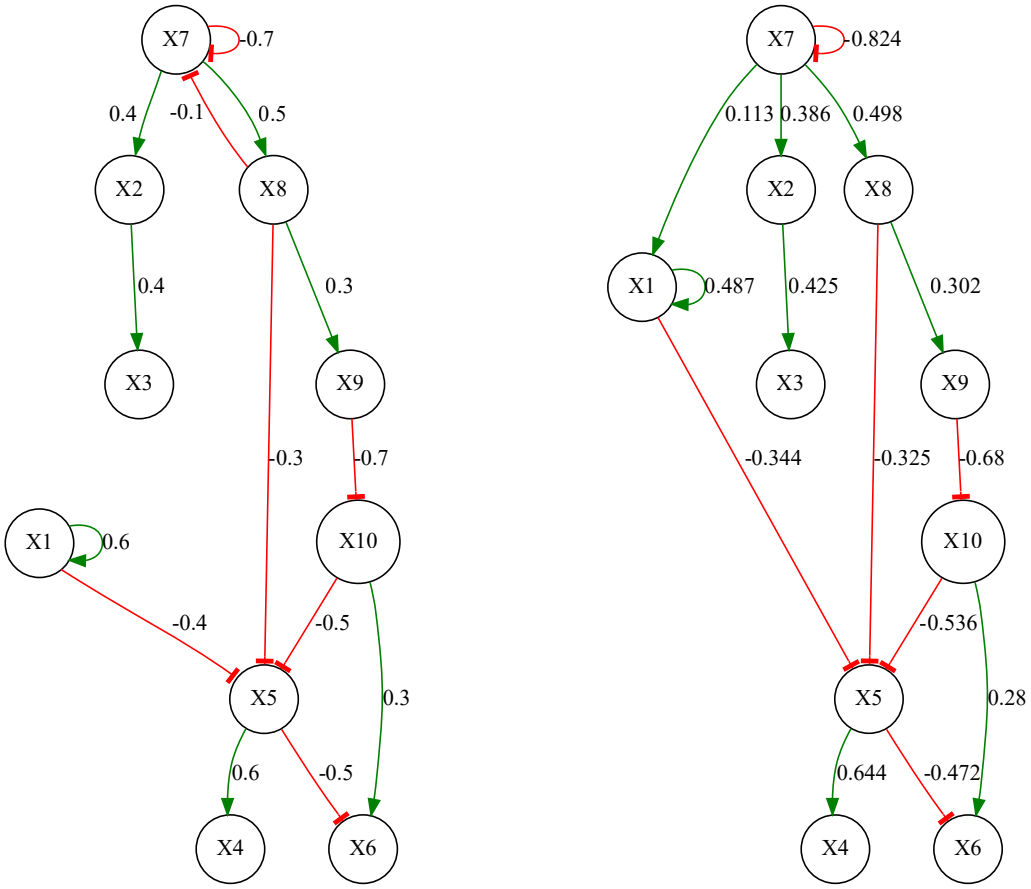


Figure 2.5: Comparison between the target (solid line) and acquired (dotted line) GRNs expression levels across time with identical initial conditions for network #1. A decent overlap of the curves can be observed, which supports the low fitness value obtained for the best individual.

2.5.2 Network #2



(a) Target graph (b) Acquired graph

Figure 2.6: Comparison between the target and acquired GRNs for network #2.

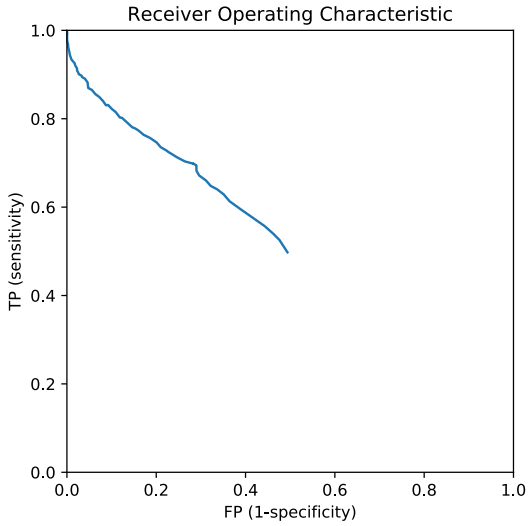


Figure 2.7: Receiver Operating Characteristic evolution for network #2. Center is random (starting point), top left is better.

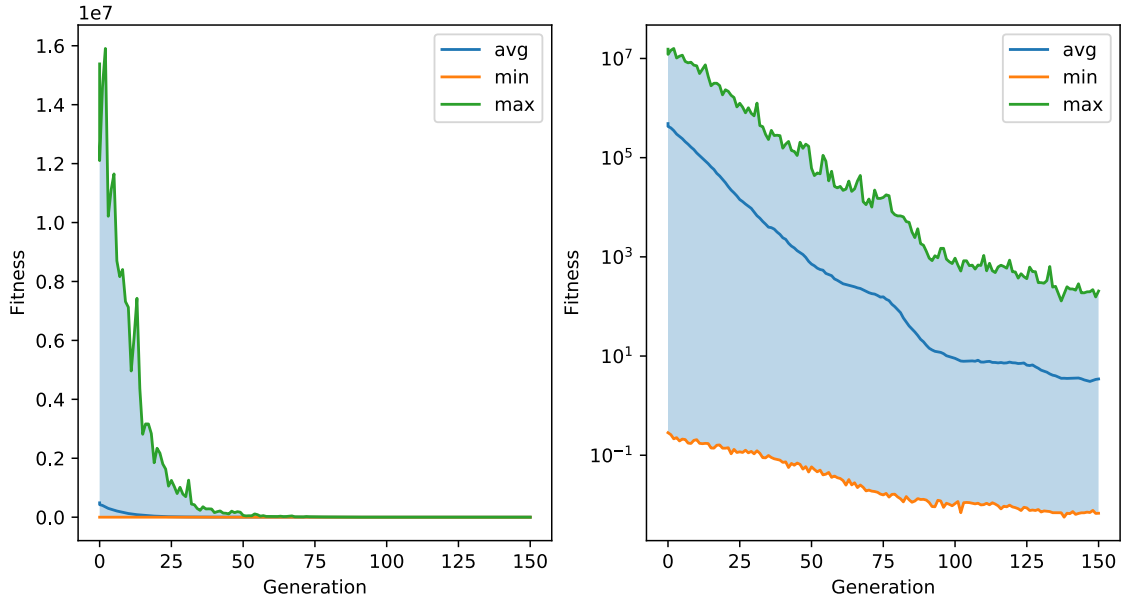


Figure 2.8: Evolution of fitness (objective function) across generations in scalar(left) and log (right) scales for network #2.

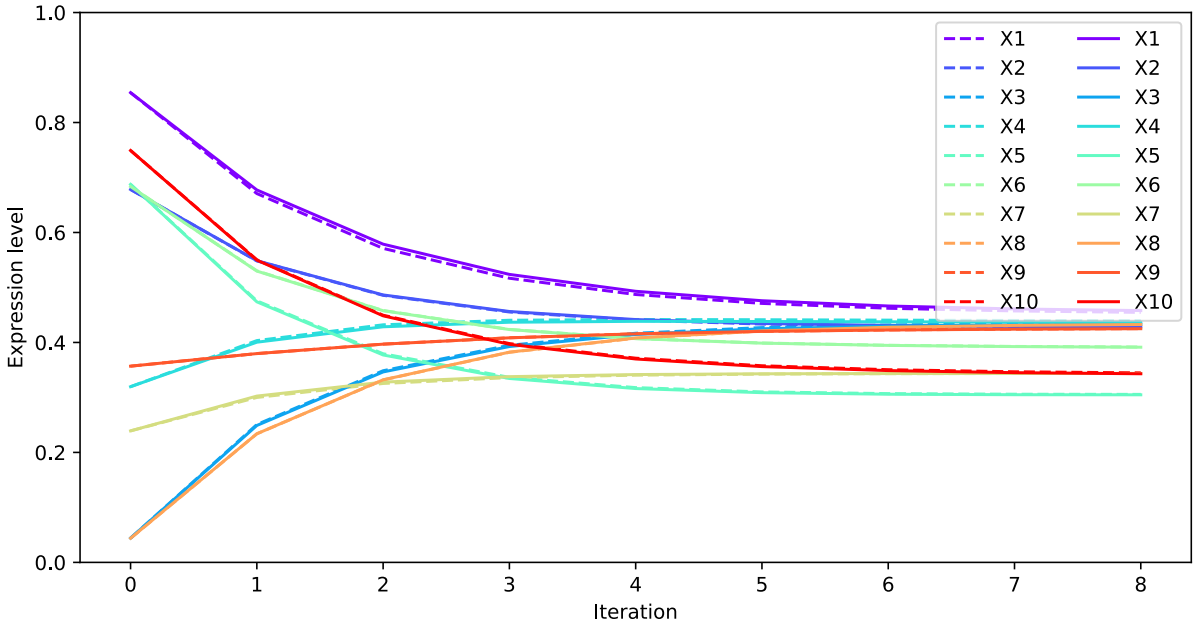
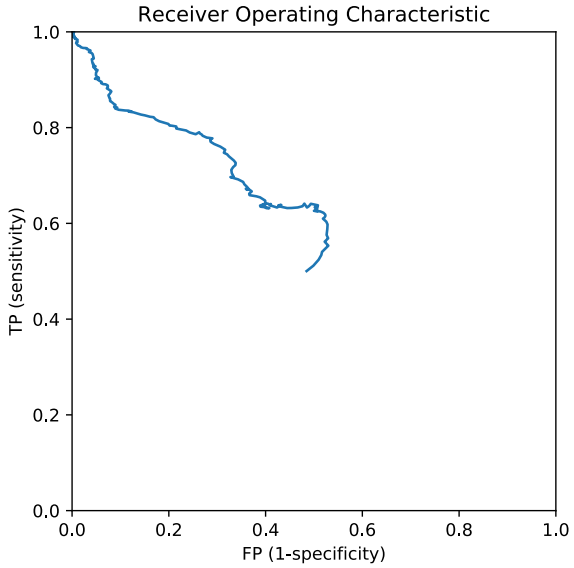
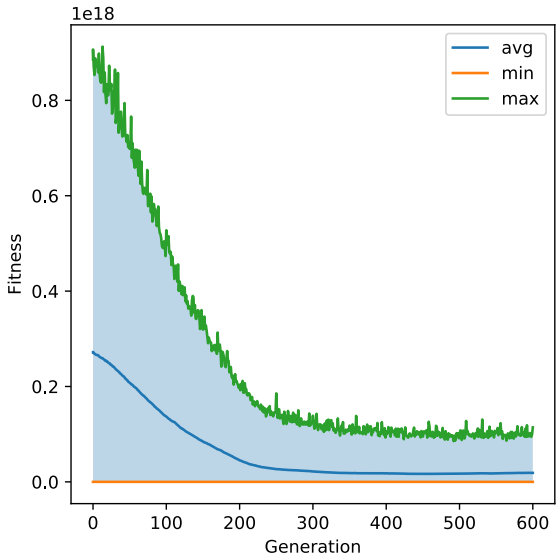


Figure 2.9: Comparison between the target (solid line) and acquired (dotted line) GRNs expression levels across time with identical initial conditions for network #2.

2.5.3 Network #3



(a) Receiver Operating Characteristic evolution for network #3.



(b) Evolution of fitness (objective function) across generations for network #3.

Figure 2.10: ROE and fitness evolution across generations for network #3.

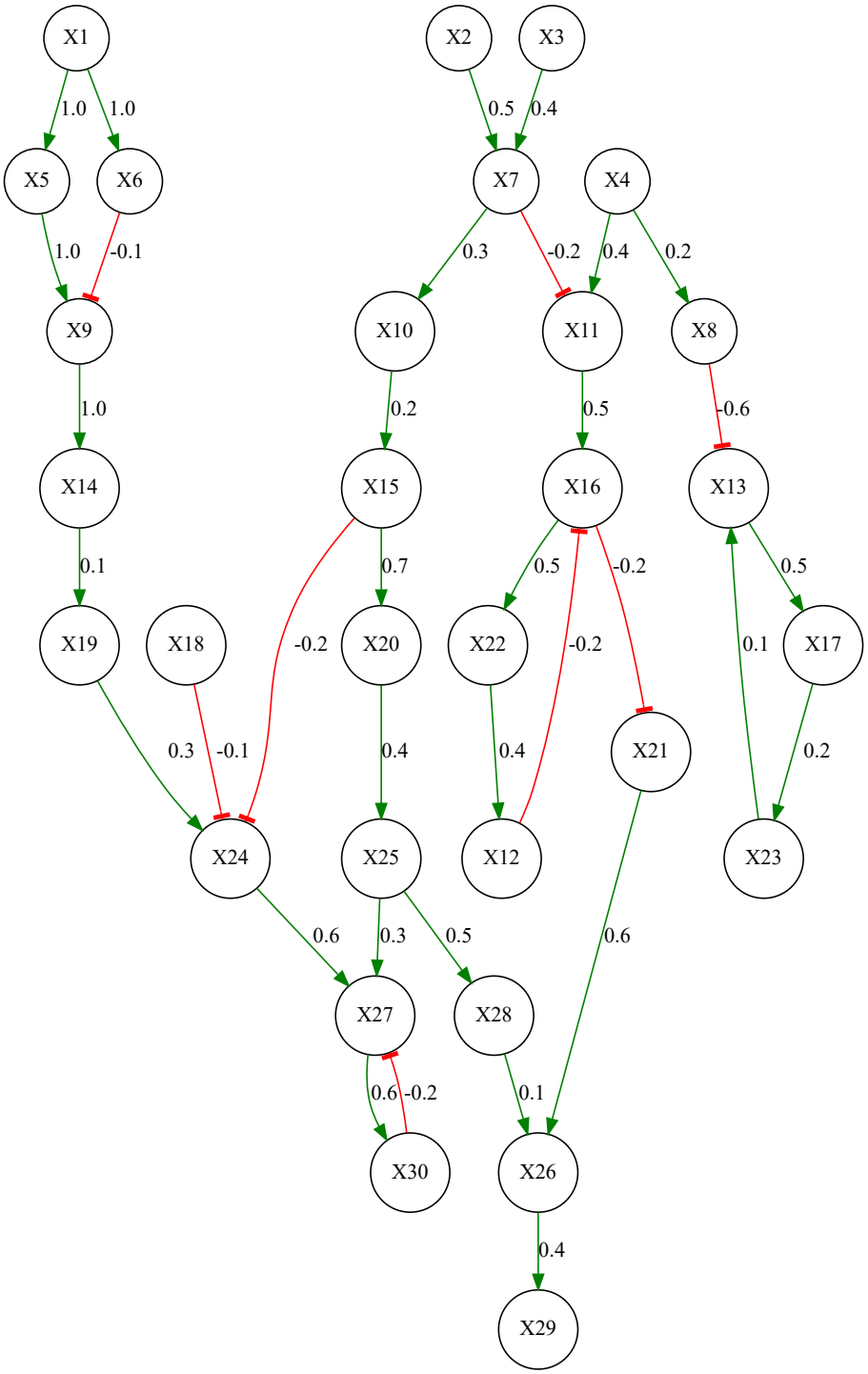


Figure 2.11: Target graph representation of the GRN for network #3.

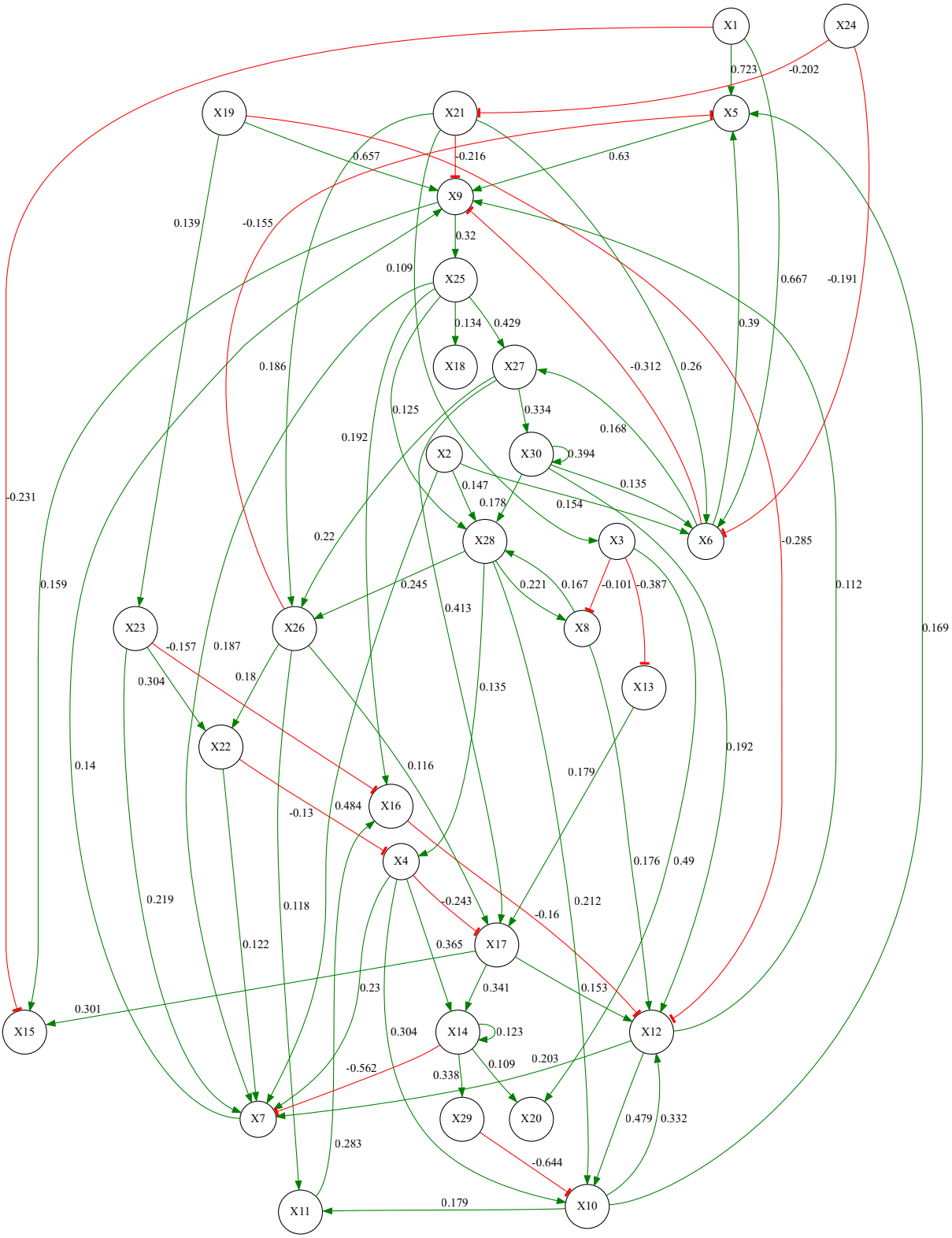


Figure 2.12: Final graph representation of the GRN for network #3. The obtained graph is cluttered with extraneous regulations not present in the target GRN.

3 | Optimization across space

3.1 Motivation

In molecular biology, transcriptomics techniques like RNA-Seq study an organism's RNA transcripts [4]. They are used for gene expression profiling, and because RNA-Seq includes all mRNA transcripts in the cell, it provides information about the active expression of genes at a certain time snapshot. Getting a continuous temporal gene profile has therefore a practical barrier of how short these time intervals between snapshots can be. Single-cell transcriptomics examines the gene expression levels of individual cells and allow an insight into the inference of GRNs [15, 25].

These techniques can give information about the final pattern across space: cells in space end up having different amounts of each RNA (and proteins), effectively differentiating themselves from the rest. This *cellular differentiation* is, together with morphogenesis and cell growth, the basis of evolutionary developmental biology.

The basis of cellular differentiation adopted here is the introduction of *morphogens* in our model. A morphogen can be any substance whose concentration gradient governs the cellular differentiation. Turing, who first coined the term [27], proposed a system of two morphogens: C (catalyst) and I (inhibitor). C catalyses the formation of both itself and I, and I inhibits the formation of C and also diffuses more rapidly than C. This system develops a pattern due to a instabilities introduced by random disturbances to the homogeneous equilibrium.

The French flag model, conceived by developmental biologist Lewis Wolpert [28, 29], defines a similar system, where this time only one certain morphogen forms a gradient across space. Cells in this environment sense the morphogen gradient through specific proteins, and respond with a specific differentiation to match the gradient. This is the morphogen model we are going to adopt in this chapter.

3.2 Implementation

To start the search for the optimal GRN, we will adjust the Genetic Algorithm and the Gene Regulatory Model developed in the later section. A GRN here consists of a fixed set of eight *proteins*[1], which inhibit or activate the production rate of each other. The function of the pro-

teins is shown in Table 3.1. These inhibition/activation relations are modeled as rate equations which are simulated through time until stabilization in a software environment called CompuCell3D. The simulation uses the rate equations to model each cell in a random grid of cells within a 2D plane. Each cell is able to move across the environment following the Cellular Potts Model that CompuCell3D implements. Each cell is internally represented as a collection of pixels of the lattice, and uses an energy formalism to describe cell properties, interactions and behaviors as additive terms of the Hamiltonian. Note that this effective energy does not represent the physical energy of the cells, but a simple way to produce the desired cell behavior [26].

Protein	Address	Function
$pr00$	000	Color determinant #1
$pr01$	001	Color determinant #2
$pr02$	010	None
$pr03$	011	None
$pr11$	100	None
$pr12$	101	Morphogen concentration sensor
$pr13$	110	None
$pr15$	111	None

Table 3.1: Proteins in the GRN, their address in the genome and their predefined functions. $pr12$ senses the morphogen concentration gradient, and can be seen as an *input* protein. $pr00$ and $pr01$ are the proteins that determine the color or type of cell, and can therefore be seen as *output* proteins. The other proteins do not have a predefined function, so they simply act as extra nodes in the GRN.

The other change respect to the optimization across time is the evaluation of the GRNs. Once the simulation is performed, we obtain an expression pattern of the proteins for each gene in a 2D spatial plane. We compare this to the expected result obtained by the French flag model.

Knabe et al. [16] developed an algorithm (including the GRN, the cellular model, mitosis rules, etc) written in `C++` as CompuCell3D plugins. We propose another procedure, written mainly in Python and using CompuCell3D as our simulation environment (Algorithm 4, Fig. 3.1).

All in all, we want to search by means of heuristic algorithms an optimal solution of the system consisting on two parts:

- GRN structure, that is, which protein regulates which other, or the functional form of the rate equations.
- Numerical values of the reaction constants.

[1]Here we use the word 'protein' as the element that has a certain expression level and interacts with other proteins, represented in a GRN as node. It is a synonym of what was called 'gene' in section 2. The term 'gene' is used here as a unit of encoding in the genome.

Algorithm 4 General workflow pseudocode.

```
Start master Python script
Initialize Genetic Algorithm parameters
population ← random population
repeat
    for all genome ∈ population do
        SBML file ← genome
        CompuCell3D non-GUI simulation
        result ← final cell types and positions
        fitness ← evaluate(result)
    end for
    new_population ← Mutation, crossover, selection
    population ← new_population
until halting-condition
```

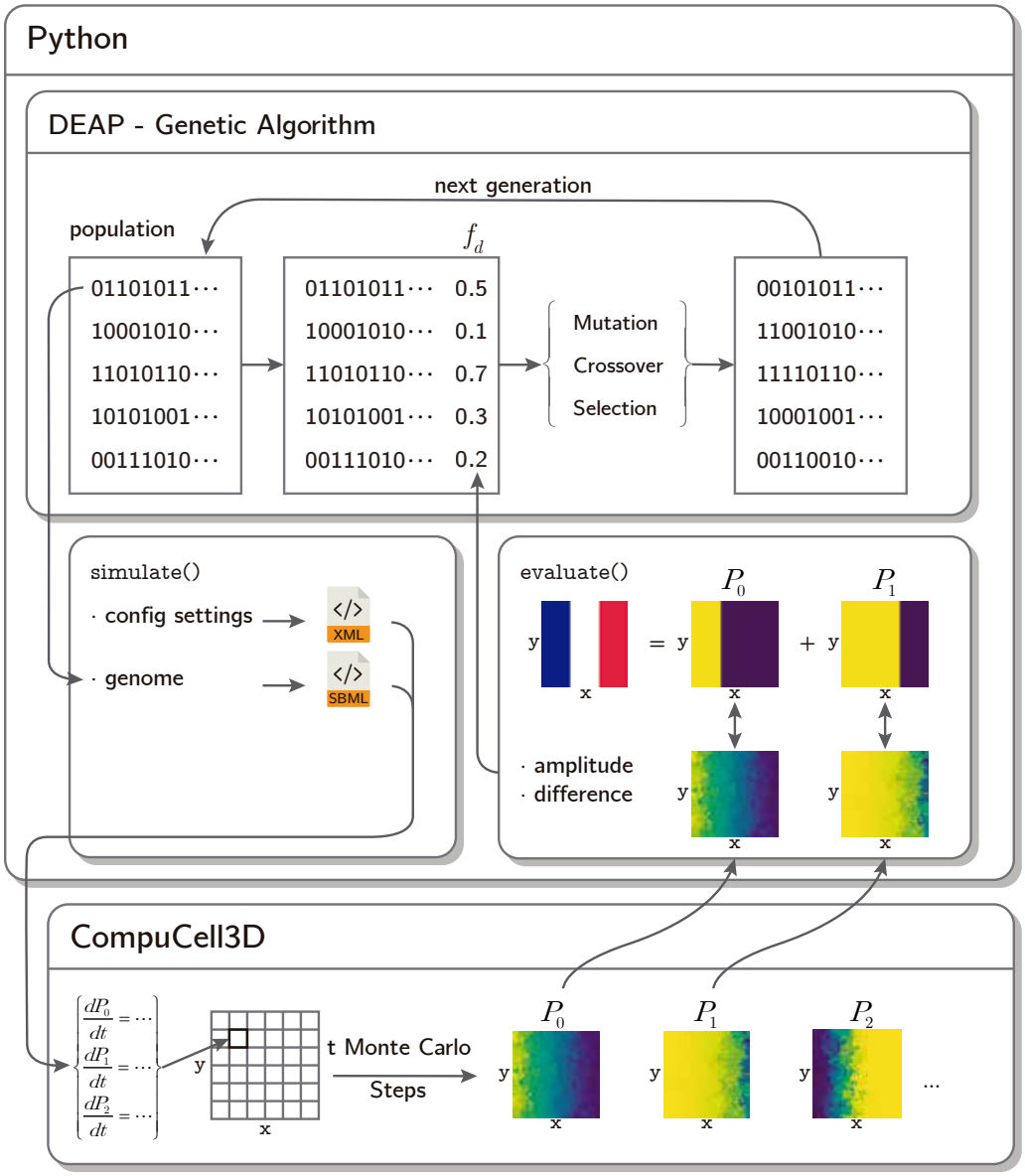


Figure 3.1: General workflow. The program starts with a random population. Each individual of the population is simulated (`simulate()` method), where its genome is translated to a system of differential equations in SBML format and together with the configuration settings is sent to CompuCell3D. Inside this program, the grid of cells is simulated t steps, where each cell has it's own levels of proteins and evolved according to the system of equations notated in the SBML file. At the end of the simulation, all proteins P_0, P_1, ... have a different concentration map across the cell domain, which is extracted and compared with the target maps back in Python (`evaluate()` method). From the difference between the target and acquired protein maps, a fitness value f_d is calculated and attached to the individual. Once all individuals are simulated and evaluated, the genetic algorithm in DEAP performs mutation, crossover and selection and creates the next generation of individuals. It then repeats the cycle for $ngen$ generations, and the best individual at the end is obtained as a result.

3.3 Encoding

To use a genetic algorithm, we need the individuals to have an array-like structure. In order to fulfill this condition, we *encode* the graph-like structure of the GRN, together with the numeric values, to a DNA-like integer array. This imposes a relationship T between the space of 1D genomes and the space of decoded solutions (GRNs). We will discuss both spaces and the decoding function T once we define the structure of the genomes (3.6).

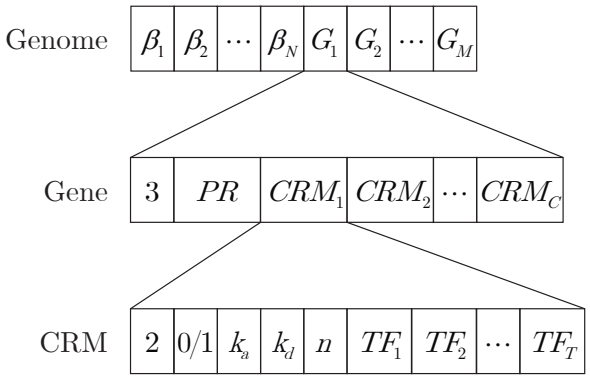


Figure 3.2: Genome array structure. Each genome contains N decay rates corresponding to each of the N proteins, and M genes. Each gene is always preceded by a 3 indicating the start of a gene, followed by the target protein address PR, and C *cis-regulatory modules* (CRMs). The start of each CRM is indicated by a 2, followed by either a 0 or a 1 indicating whether the regulation is activating or repressing. It then encodes the association and dissociation constants k_a and k_d and the Hill coefficient n used in the Hill equation (3.7) (page 29). At the end of each CRM are the T transcription factors TF, that is, the proteins that regulate the target protein PR.

As seen in Figure 3.2, the entire genome is a 1D list of integers that take values between 0 and 3. It starts defining the decay coefficients for N proteins, represented by β_i which is an array of four bits encoding a float in the range $(0, 1]$. It then encodes M genes, all of them starting with a 3. Each gene is structured starting with a 3, followed by the protein address[2] which is going to be affected. The gene ends with C different *cis-regulatory modules* (CRM). Each CRM starts with a 2, followed by a 0 if it's inhibitory or 1 if it's activating, and then its corresponding constants: k_a and k_d are binary encodings of floats in the range $(0, 1]$, and n is the binary encoding of an integer in the range $[0, 2^4 - 1]$. Their meaning will be explained shortly after. Table 3.2 shows the mapping of the binary numbers representable with 4 bits with the floating point numbers in the interval $(0, 1]$ corresponding to k_a and k_d, and the integers corresponding to n.

Finally, at the end of each CRM are T transcription factors, which are the addresses[2] of the proteins that activate or inhibit the production of the affected protein.

[2]The protein address or protein index is an array of 3 bits, allowing to address the $2^3 = 8$ proteins in Table 3.1. For greater number of proteins in the GRN, this index bit length is increased. E.g, for 16 proteins, 4 bits are necessary.

Binary	0000	0001	0010	0011	0100	0101	0110	0111
k_a or k_d	0.0625	0.125	0.1875	0.25	0.3125	0.375	0.4375	0.5
n	0	1	2	3	4	5	6	7

Binary	1000	1001	1010	1011	1100	1101	1110	1111
k_a or k_d	0.5625	0.625	0.6875	0.75	0.8125	0.875	0.9375	1.0
n	8	9	10	11	12	13	14	15

Table 3.2: Binary encoding of all possible values for k_a, k_d and n.

3.4 Decoding

We decode the rate equations from the genome. In order to do this, we interpret that when a protein P_j activates another P_i, this activation is a reaction catalysis in biochemistry: the activator protein P_j binds to a specific location S of the DNA and forms a complex P_i, allowing the transcription of P_i. The location of the DNA can either be free (P_j not present) or bound (with P_j), so the total is $P_{tot} = S + P_i$. The reaction is:

$$S + nP_j \underset{k_d}{\overset{k_a}{\rightleftharpoons}} P_i \tag{3.1}$$

where S is the substrate, P_j is the catalyst, P_i is the product, k_a is the association constant and k_d the dissociation constant. S is the molecule that forms the complex with n molecules of P_j to result in P_i. The three constants k_a, k_d and n are the ones present in each CRM in Figure 3.2.

As it is derived by U. Alon [1], the association rate is proportional to the concentration of S and the concentration of P_j to the power n (the probability of finding n molecules of P_j simultaneously). The proportionality factor is the association constant k_a:

$$association\_rate = k_a S P_j^n \tag{3.2}$$

The dissociation rate is proportional by a factor k_d to the concentration of P_i:

$$dissociation\_rate = k_d P_i \tag{3.3}$$

In equilibrium, the rate of change of the concentration of P_i, that is, the difference between association and dissociation rates, is zero in a steady state approximation:

$$\frac{dP_i}{dt} = k_a S P_j^n - k_d P_i = 0 \tag{3.4}$$

From here, we can obtain the apparent dissociation constant defined by the Law of mass action as:

$$(K_{ij})^n = \frac{k_d}{k_a} = \frac{S \cdot P_j^n}{P_i} \tag{3.5}$$

Solving for the ratio between bound P_i and total $S + P_i$ we get the Hill equation:

$$\frac{P_i}{S + P_i} = \frac{P_j^n}{K_{ij}^n + P_j^n} \tag{3.6}$$

In non-equilibrium, the transcription rate for a protein P_i follows the Hill kinetic equation, expressed as:

$$\frac{dP_i}{dt} = v_{max}\frac{P_j^n}{K_{ij}^n + P_j^n} = v_{max}\frac{(P_j/K_{ij})^n}{1 + (P_j/K_{ij})^n} \tag{3.7}$$

where n is the Hill coefficient and K_{ij} is the apparent dissociation constant (3.5). The Hill kinetic equation (3.7) is also known simply as Hill equation, which is how it is going to be named from here on, but should not be confused with the similar (3.6), which is not a differential equation. For $n = 1$, (3.7) reduces to Michaelis-Menten kinetics, one of the best known models of enzyme kinetics:

$$\frac{dP_i}{dt} = v_{max}\frac{P_j}{K_{ij} + P_j} \tag{3.8}$$

Analogously, the Hill equation for gene P_j repressing P_i has the form:

$$\frac{dP_i}{dt} = v_{max}\left(1 - \frac{P_j^n}{K_{ij}^n + P_j^n}\right) = v_{max}\frac{1}{1 + (P_j/K_{ij})^n} \tag{3.9}$$

The general form of the rate equations is a combination of both and is called the generalized Hill equation, as proposed by Del Vecchio and Murray [5]:

$$\frac{dP_i}{dt} = \frac{v_{max}A(\{P_j\}) + v_b}{1 + A(\{P_j\}) + I(\{P_j\})} - \beta_{P_i}P_i \tag{3.10}$$

where β_{P_i} is the decay rate of the i-th protein, v_b is the basal transcription rate and v_{max} is the maximum transcription rate (saturation). The functions $A(\{P_j\})$ and $I(\{P_j\})$ are the sums of normalized concentrations of the activating and inhibitory proteins of P_i to the power of their respective Hill coefficients n_i, described by the CRMs of P_i. For example, if P_1 and P_2 activate P_3, the activator function has the form:

$$A(P_1, P_2) = \left(\frac{P_1}{K_{31}}\right)^{n_1} + \left(\frac{P_2}{K_{32}}\right)^{n_2} \tag{3.11}$$

We can assert that the transcription rate of P_i doesn't get over v_{max} and that the sigmoidal shape of the Hill equation is conserved for limiting cases ($A \to 0$, $A \to \infty$, $I \to 0$, $I \to \infty$) (Figure 3.3). We can also assert that, by means of the limiting factor of the decay rate, the model doesn't *explode*: The concentration of P_i reaches an equilibrium at a finite positive value:

$$P_i^{eq} = \frac{1}{\beta_{P_i}}\frac{v_{max}A(\{P_j\}) + v_b}{1 + A(\{P_j\}) + I(\{P_j\})} \tag{3.12}$$

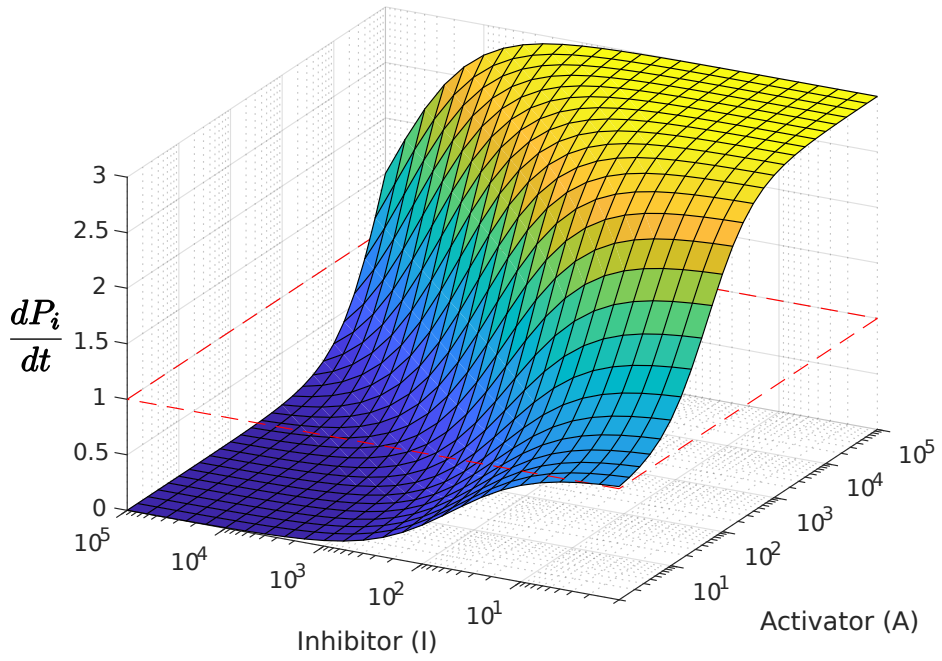


Figure 3.3: Generalized Hill equation without the decay factor ($\beta_{P_i} = 0$) as a function of activator (A) and inhibitor (I) concentrations, with parameters $\nu_{max} = 3$, $\nu_b = 1$, $K_{iA} = 50$, $K_{iI} = 50$, $n_A = 1$, $n_I = 1$. For both low A and I, the transcription rate dP_i/dt is the basal rate ν_b. For high A and low I, it saturates at ν_{max} and for low A and high I the transcription rate tends to zero.

3.5 Logic gates

As illustrated in Figure 3.2, each CRM can have more than one transcription factor (TF) than can act in conjunction as inhibitor or activator of an influenced protein PR. This allows to model logic gates, primarily AND and OR gates. The implementation of logic functions in DNA has been experimentally shown by Okamoto et al. [21]. AND, OR, YES and NOT are the basic logic gates from which all logic functions can be constructed from [9].

We have designed a genome structure (Fig. 3.2) and a set of kinetic equations (3.10) flexible enough that certain combinations give rise to logic functions as an emergent property of the system, rather than hard-coding them explicitly for example as a special 'AND' gene. Remember that *genes* represent regulations or edges between nodes -the *proteins*-. The resulting kinetic equations for logic functions are equivalent to those derived by *thermostatistical modeling* from principles of statistical physics [3]. We can always design a genome to have such behavior (Fig. 3.4):

- AND: Modeled as multiple TFs per CRM. The rate equation will depend on the product

of the concentrations, creating an effective AND gate (extended to continuous values).

$$\frac{dP_3}{dt} = \frac{v_{max}\left(\frac{P_1}{K_{13}}\right)^n\left(\frac{P_2}{K_{23}}\right)^n + v_b}{1+\left(\frac{P_1}{K_{13}}\right)^n\left(\frac{P_2}{K_{23}}\right)^n} - \beta_3 P_3 \tag{3.13}$$

- OR: Modeled as multiple CRMs activating a PR in a gene. If any of the CRM activates, the formation of the corresponding PR will increase.

$$\frac{dP_3}{dt} = \frac{v_{max}\left[\left(\frac{P_1}{K_{13}}\right)^n + \left(\frac{P_2}{K_{23}}\right)^n\right] + v_b}{1+\left[\left(\frac{P_1}{K_{13}}\right)^n + \left(\frac{P_2}{K_{23}}\right)^n\right]} - \beta_3 P_3 \tag{3.14}$$

- YES: Modeled as a PR with a single activating CRM.

$$\frac{dP_2}{dt} = \frac{v_{max}\left(\frac{P_1}{K_{12}}\right)^n + v_b}{1+\left(\frac{P_1}{K_{12}}\right)^n} - \beta_2 P_2 \tag{3.15}$$

- NOT: Modeled as a PR with a single inhibitory CRM.

$$\frac{dP_2}{dt} = \frac{v_b}{1+\left(\frac{P_1}{K_{12}}\right)^n} - \beta_2 P_2 \tag{3.16}$$

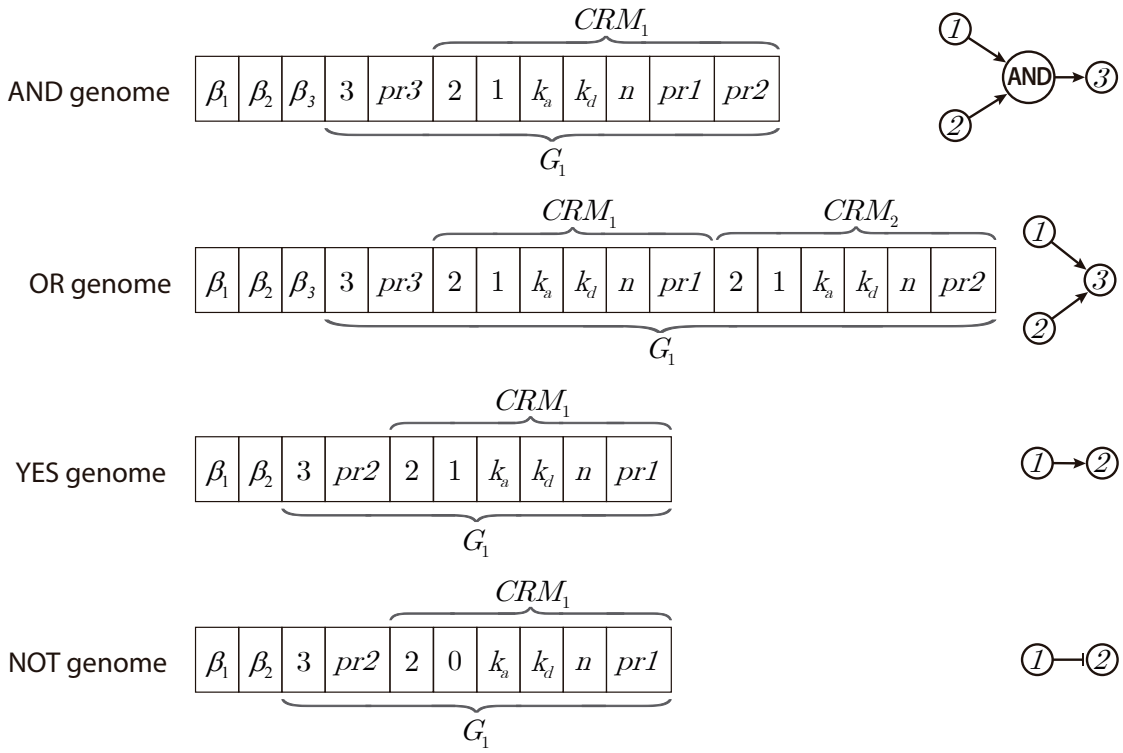


Figure 3.4: Minimal examples of GRNs implementing each logic gate, together with their corresponding graph representation.

3.6 Representations

We have presented three spaces in which an individual or solution can be represented:

- Genome space: Each point in this space represents a 1D-array genome. The genetic algorithm applies operations of crossover, mutation and selection in this space.
- Graph space: Points in graph space represent the decoded GRNs.
- Space of systems of differential equations: Each point in this space represents a system of differential equations governing the dynamics of the protein concentrations and interactions.

There is a bijective correspondence between elements in graph space and space of systems of differential equations. However, the decoding function T (3.4) is not injective nor surjective, because it can transform many genomes to the same GRN - e.g, two genomes consisting of the same genes but in different order -, and because some GRNs can't be encoded in a genome - e.g, a GRN where one protein has a higher decay rate than the bounded $(0, 1]$ range that a genome can express-.

The genome space contains the full hereditary information of individuals, so it can be called *genotype*. On the other hand, the space of systems of differential equations and the graph space, together with environmental information like initial conditions for the simulation, comprise the individuals behavior and observable characteristics, so they represent the *phenotype*.

3.7 GA operations

Mutation and crossover are designed so that every possible outcome is a valid individual. Therefore we avoid having to reject invalid individuals or defining a penalty function.

Mutation

Mutation is simple: It looks for a random position in the genome where there's a 0 or a 1, and flips it's value. We avoid changing the value of positions where there's a 2 or a 3 because these digits mark the structure of the genome (starting positions of a gene and a CRM respectively, as explained in 3.3), so modifying these values would result in a drastic change in phenotype.

Crossover

To mate two genomes, we first choose a random section in each genome. If the chosen section is the first (where the decay rates $\beta_1, \ldots, \beta_N$ are encoded), we choose a random position within the section and exchange each part of the section between the two individuals. If the chosen section is a gene (second section onwards), we choose a random position within the section

but also an offset (from a normal distribution with its width or standard deviation being the address bit length, 3) which we apply to the left of one individual and to the right of the other to locate the final cutting points. We then split the genes in each individual and exchange them. This is similar to the crossover mechanism proposed by Knabe et al. [16].

Selection

We combine two selection algorithms as done in [16]: Tournament selection, where a fixed number of individuals (30 in this case) is randomly chosen from the population and the individual with the highest fitness score is selected, and elitism, where the best individual of the population is always conserved.

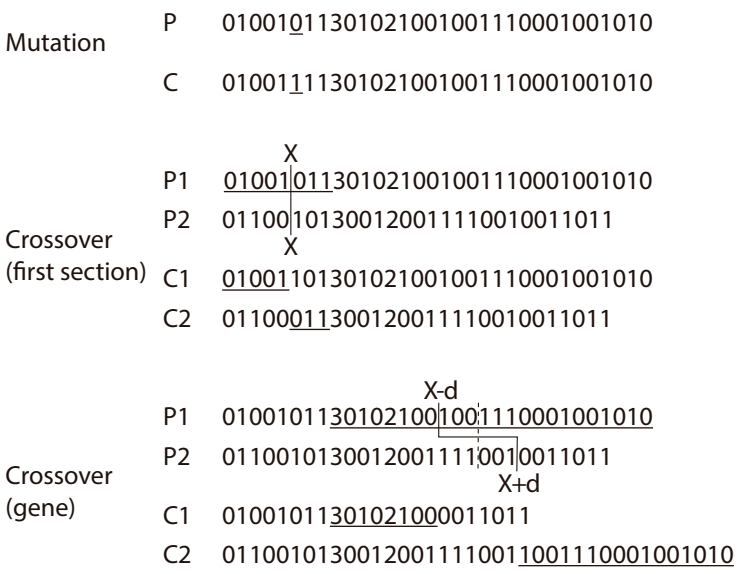


Figure 3.5: Mutation and crossover mechanisms introduce generic variation. *Mutation* requires one parent (P) and produces one child (C) varying one of its bits. Crossover takes two parents (P1 and P2) and produces two children (C1 and C2). *First-section crossover* exchanges the bits on the right of a random position from the first section of both genomes. *Gene crossover* swaps bits to the left of $X - d$ of P1 and the bits to the right of $X + d$ of P2, being d an offset chosen from a normal distribution[3] with width $\sigma = 3$.

pop	gen	p_{cx}	p_{mut}	$tournsize$
500	150	0.90	0.01	30

Table 3.3: GA parameters for GRN optimization across space. pop is the number of individuals per generation in the GA, gen is the total number of generations to be computed. p_{cx} is the probability of mating two individuals, p_{mut} the probability of mutating an individual. $tournsize$ is the number of individuals from which the best is selected in tournament selection.

[3]The standard deviation σ is chosen to be the same as the protein address length. Again, for greater number of proteins in the GRN, this address length is increased so σ will also be greater.

3.8 Results

Figure 3.6 shows the evolution of the evaluated function -fitness- and the genome length across generations. Each generation -along the abscissa- has a population for which there's an individual with a minimum fitness - the best individual, in orange- and an individual with the maximum fitness value - worst individual, in green-. We also calculate the average fitness of the population - blue line-. Similarly for genome lengths, we plot the minimum, maximum and average genome lengths across generations. Figure 3.6a shows that from about generation 120, the best score stabilizes at $fitness \simeq 4.4$.

Figure 3.7 shows the final stable expression profile for the best individual after 150 generations along the x-axis of the 2D plane, averaged from the values in the y-axis. The green curve is the protein sensing the morphogen gradient $pr12$. $pr00$ and $pr01$ are the color determinants. It shows resemblance to the objective French flag model ($fitness = 4.401$).

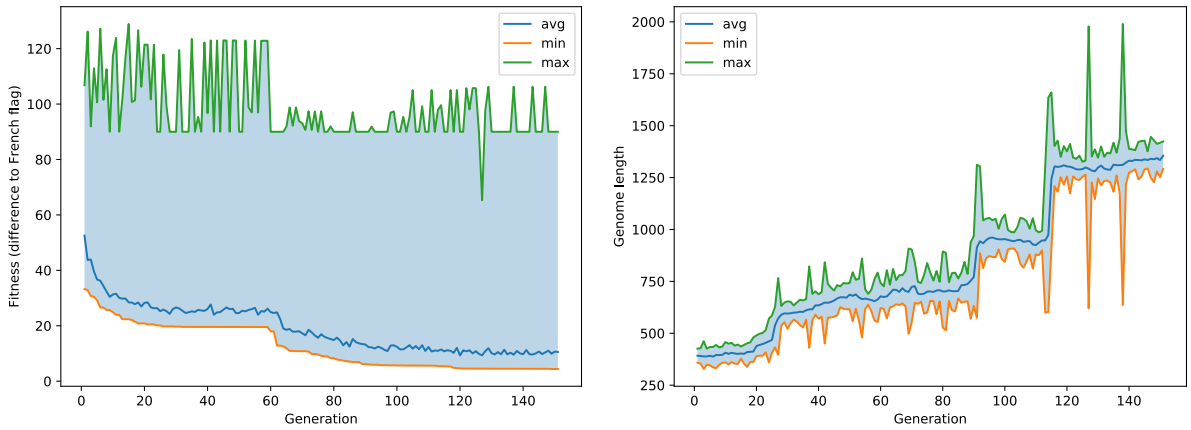
To reconstruct the French flag model, we have compared an ad hoc colormapping based on the spatial fields of $pr00$ and $pr01$ with the theoretical french flag in Figure 3.8. Finally, figure 3.9 shows the complete regulatory network connecting the eight nodes of the system. Proteins $pr02$ and $pr03$ don't have any inputs, which is in accordance with the constant zero-valued field in Figure 3.10. As the fitness does not include any parsimony factor, there are two negative consequences:

- Genomes tend to become longer each generation, adding complexity.
- The final GRN contains some residual elements which do not contribute to the final expression pattern, like $pr02$ regulating an AND gate, effectively suppressing its output as the output of an AND gate depends on the product of input concentrations and $pr02$ has always zero expression level.

However, this information sparsity is desirable because of two reasons:

- Unused sections of the genome can be put into use once again in the next generation after population crossover or mutation.
- It adds diversity to the individuals of the population, observable in the wide range of fitnesses that the evolution encompasses - difference between maximum and minimum fitnesses in Figure 3.6a, and allows the GA to escape local minima.

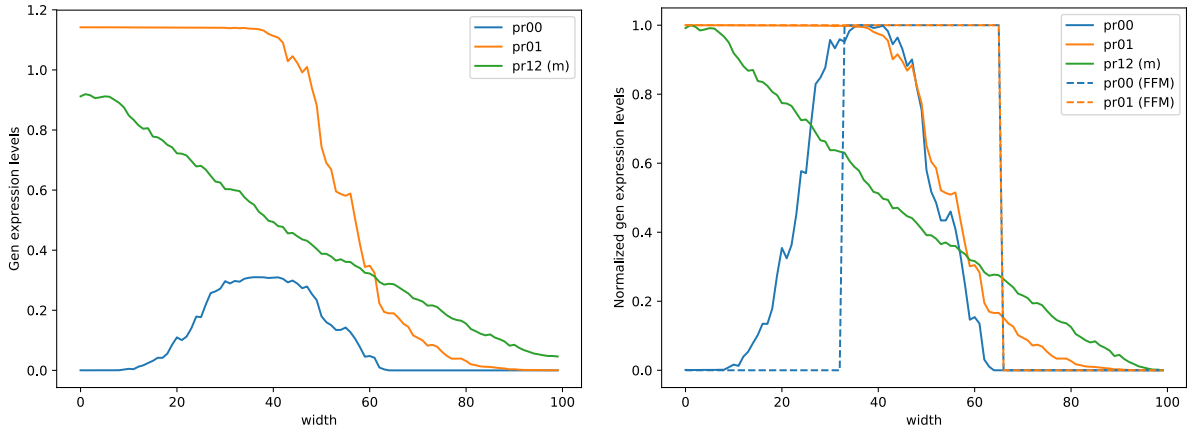
Moreover, constraining the genome length -either hard coding a maximum length or inserting a penalty term in the evaluation function where longer genomes score worse fitness values- would end up leading the population to a local minimum. As an alternative, a periodic change in fitness goal is proposed, where during a first period of generations the algorithm seeks expression levels closer to the French flag model, and later this genome length penalty term is added with increasing weight in the evaluation function in order to obtain simpler GRNs, much in the same line as the parsimony factor used in Section 2.



(a) Fitness evolution across GA generations.

(b) Genome length evolution across GA generations. Sudden maximum and minimum peaks correspond to gene crossover between two average length genomes yielding a very long and a very short offspring.

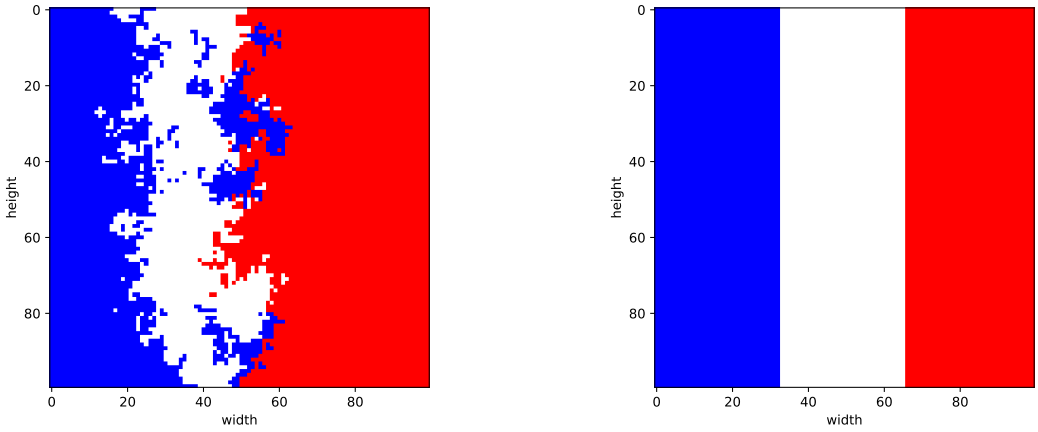
Figure 3.6: Evolution of fitness and genome length across GA generations.



(a) Obtained expression profile.

(b) Comparison between normalized expression profile and theoretical expression profile (FFM: French flag model).

Figure 3.7: Stable expression profiles across the spatial dimension. $pr12$ (green) is the protein sensing the morphogen gradient, $pr00$ and $pr01$ are the color determinants.



(a) Obtained 2D expression map.

(b) French flag model 2D expression map.

Figure 3.8: Comparison between stable expression maps across the 2D spatial domain. To obtain blue, white and red cell types, the normalized concentrations have been colormapped as follows: if the sum of normalized $pr00$ and normalized $pr01$ is smaller than 0.5, the cell type is *red*. If it is between 0.5 and 1.5, it is *blue*, and if it is greater than 1.5, it is a white cell.

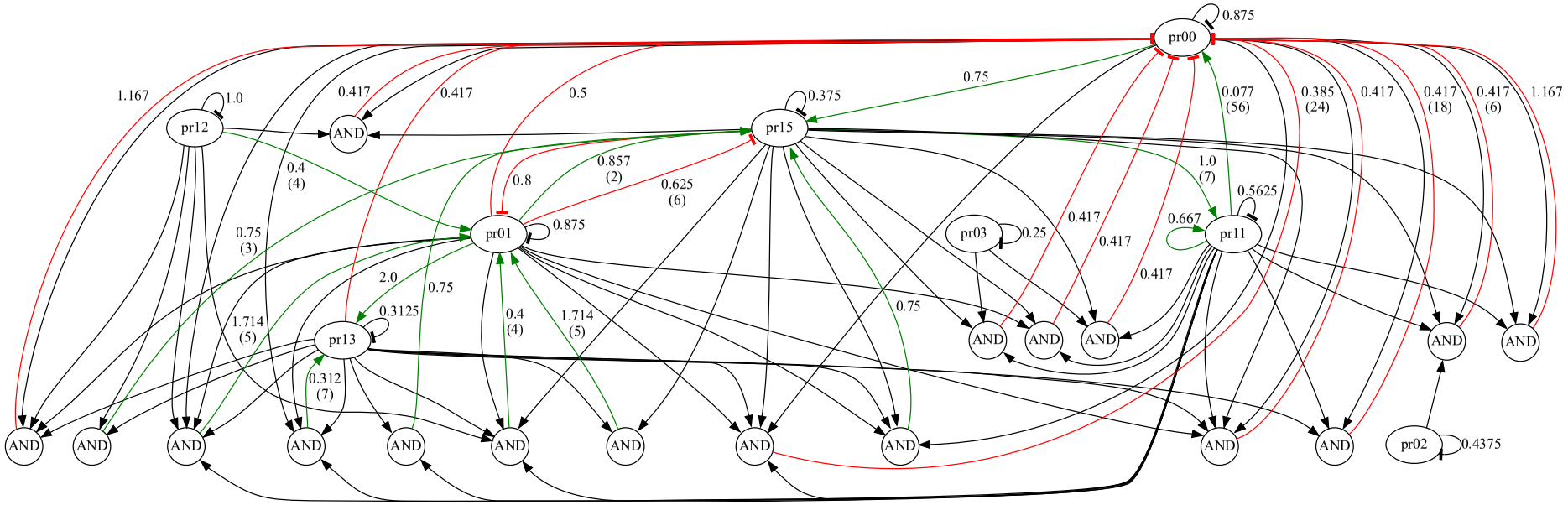


Figure 3.9: Final Gene Regulatory Network obtained to match response with the French flag model. Red edges indicate inhibition regulation, green edges indicate activation regulation, and black edges are inputs to AND gates or nodes self-regulation (decay rate). Decimal numbers are the apparent dissociation constant K_{ij} (3.5), and integers between parenthesis are the Hill coefficient n of the given regulation. If not shown, $n = 1$. Graph layout follows an edited DOT graph description language specification [17].

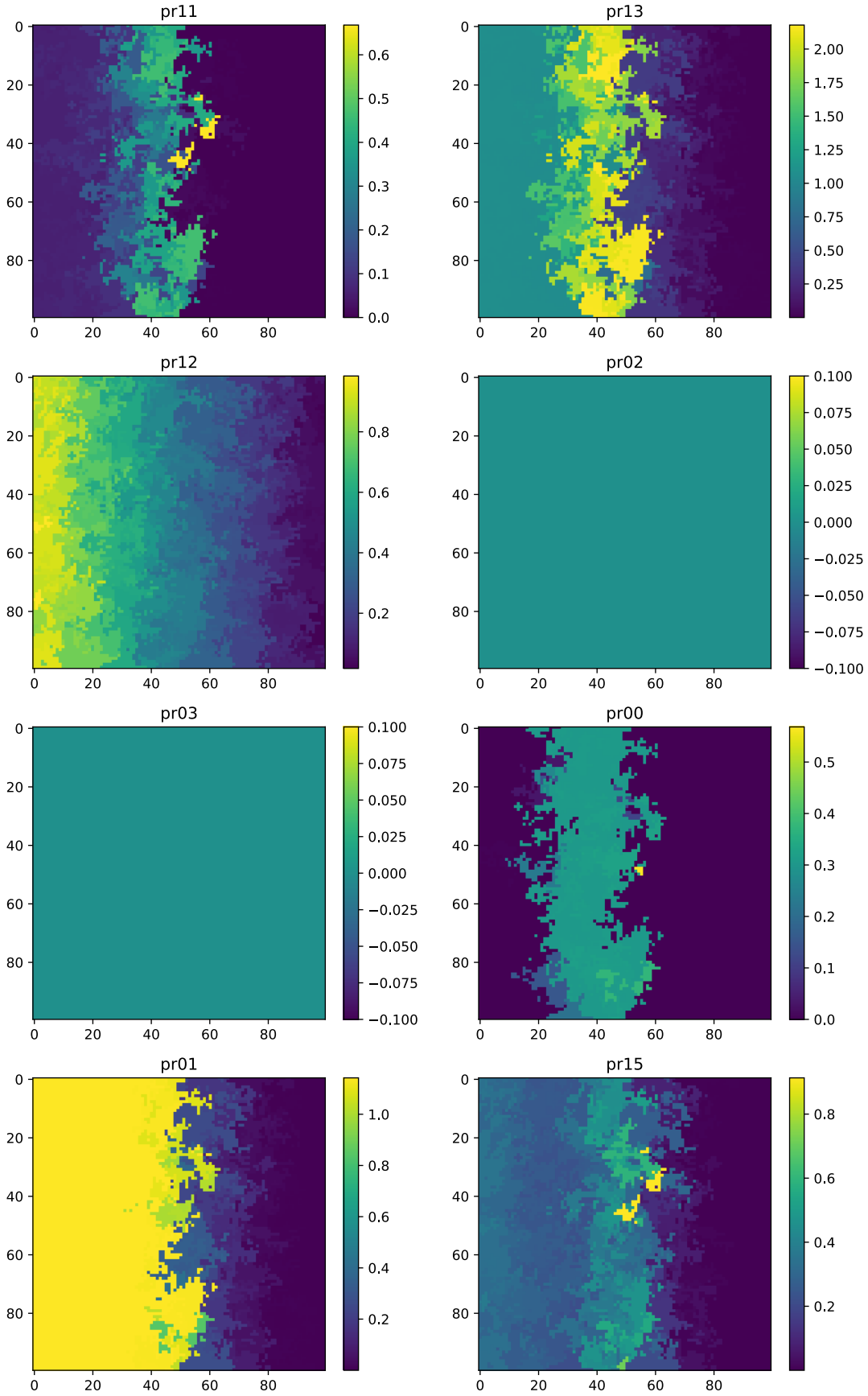


Figure 3.10: Final protein fields in the 2D spatial plane. The morphogen gradient decreasing from left to right (x axis) is sensed by $pr12$. $pr00$ and $pr01$ are the color determinants.

4 | Conclusions

The study of GRNs provides an insight into the mechanisms that real cells like *E. coli* (Fig. 1.4) use to express certain behavior and seeks to capture general laws that govern these biological interactions. Our use of Genetic Algorithms is inspired by the historical nature of Biology, where these networks are the result of evolution.

In this work, we have studied two ways of evaluating *in silico* Gene Regulatory Networks and using a Genetic Algorithm to find firstly arbitrary networks from temporal expression pattern data and secondly, suitable models to replicate behavior like cellular differentiation under the French Flag model from spatial expression pattern data.

The optimization across time (Section 2) yields good matching networks for small sized samples (12 correct edges out of a network with 13 edges, with a edge weight error of 4.35%), but produces networks that are too crowded when trying to find bigger GRNs (e.g. network #3, section 2.5.3).

The optimization across space (Section 3) utilizes a much more complex evaluation tool, namely CompuCell3D, which is able to simulate GRNs in a spatial environment, together with factors not controlled by the network description, like cell movement and behavior (using the Cellular Potts Model). The Genetic Algorithm used to mutate, reproduce and select better GRNs was adapted from the first section to allow the evaluation against the expected behavior (the French flag model spatial pattern seen in 3.8). The result was a rather complex network (Figure 3.9), where much of its complexity was due to an unconstrained genome length which led to many regulation paths. This network was able to successfully express a discernible French flag pattern.

Bibliography

[1] Alon, U. *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Chapman and Hall/CRC, London, 2006.

[2] Ando, S., and Iba, H. Inference of gene regulatory model by genetic algorithms. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)* (2001), **1**, IEEE, 712–719.

[3] Carballido-landeira, J. *Nonlinear Dynamics in Biological Systems*, vol. 7 of *SEMA SIMAI Springer Series*. Springer International Publishing, Cham, 2016.

[4] Chu, Y., and Corey, D. R. RNA Sequencing: Platform Selection, Experimental Design, and Data Interpretation. *Nucleic Acid Therapeutics* ***22***, 4 (2018), 271–274.

[5] Del Vecchio, D., and Murray, R. M. *Biomolecular Feedback Systems*. Princeton University Press, 2014.

[6] Di Caro, G., and Dorigo, M. AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research* ***9*** (2011), 317–365.

[7] Dorigo, M., Stützle, T., and Socha, K. *Ant Colony Optimization*, vol. 20073547 of *Chapman & Hall/CRC Computer & Information Science Series*. Chapman and Hall/CRC, 2007.

[8] Elowitz, M. B., Levine, A. J., Siggia, E. D., and Swain, P. S. Stochastic gene expression in a single cell. *Science* ***297***, 5584 (2002), 1183–1186.

[9] Floyd, T. L. *Digital Fundamentals 11th Edition*, Pearson ed. Pearson Prentice Hall, 2008.

[10] Fortin, F.-A., Gardner, M.-A., Parizeau, M., and Gagné, C. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* ***13*** (2012), 2171–2175.

[11] Gillespie, D. Markov Processes. In *SpringerReference*, Academic Press ed. Springer-Verlag, Berlin/Heidelberg, 2003, 31–38.

[12] Gillespie, D. T. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics* ***22***, 4 (1976), 403–434.

[13] Holland, J. H. *Adaptation in Natural and Artificial Systems*. The MIT Press, Cambridge, MA, USA, 1992.

[14] Hucka, M., Finney, A., Sauro, H. M., et al. The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics* ***19***, 4 (2003), 524–531.

[15] Kanter, I., and Kalisky, T. Single Cell Transcriptomics: Methods and Applications. *Frontiers in Oncology* ***5*** (2015), 53.

[16] Knabe, J. F., Wegner, K., Nehaniv, C. L., and Schilstra, M. J. Genetic Algorithms and Their Application to In Silico Evolution of Genetic Regulatory Networks. In *Journal of Sports Science and Medicine*, D. Fenyö, Ed., *Methods in Molecular Biology* ***673***. Humana Press, Totowa, NJ, 2010, 297–321.

[17] Koutsofios, E., and North, S. Drawing graphs with dot, http://www.graphviz.org/pdf/dotguide.pdf, 1991.

[18] Kyoda, K., Morohashi, M., Onami, S., and Kitano, H. A gene network inference method from continuous-value gene expression data of wild-type and mutants. *Genome informatics. Workshop on Genome Informatics* ***11*** (2000), 196–204.

[19] Maki, Y., Tominaga, D., Okamoto, M., Watanabe, S., and Eguchi, Y. Development of a System for the Inference of Large Scale Genetic Networks. *Pacific Symposium on Biocomputing* ***6***, February 2001 (2001), 446–458.

[20] Mandal, S., Saha, G., and Pal, R. Recurrent Neural Network Based Modeling of Gene Regulatory Network Using Bat Algorithm. *Journal of Advances in Mathematics and Computer Science* ***23***, 5 (2017), 1–16.

[21] Okamoto, A., Tanaka, K., and Saito, I. DNA logic gates. *Journal of the American Chemical Society* ***126***, 30 (2004), 9458–9463.

[22] Okawa, S., Nicklas, S., Zickenrott, S., Schwamborn, J. C., and del Sol, A. A Generalized Gene-Regulatory Network Model of Stem Cell Differentiation for Predicting Lineage Specifiers. *Stem Cell Reports* ***7***, 3 (2016), 307–315.

[23] Ram, R., and Chetty, M. A guided genetic algorithm for learning gene regulatory networks. *2007 IEEE Congress on Evolutionary Computation, CEC 2007* (2007), 3862–3869.

[24] Sakamoto, E., and Iba, H. Evolutionary Inference of a Biological Network as Differential Equations by Genetic Programming. *Genome Informatics* ***12***, 13 (2001), 276–277.

[25] Stegle, O., Teichmann, S. A., and Marioni, J. C. Computational and analytical challenges in single-cell transcriptomics. *Nature Reviews Genetics* ***16***, 3 (2015), 133–145.

[26] Swat, M. H., Thomas, G. L., Belmonte, J. M., et al. Multi-Scale Modeling of Tissues Using CompuCell3D. *Methods in Cell Biology* ***110*** (2012), 325–366.

[27] Turing, A. M. The chemical basis of morphogenesis. *Bulletin of Mathematical Biology* ***52***, 1-2 (1990), 153–197.

[28] Wolpert, L. Positional information and the spatial pattern of cellular differentiation. *Journal of Theoretical Biology* ***25***, 1 (1969), 1–47.

[29] Wolpert, L. Positional information and patterning revisited. *Journal of Theoretical Biology* ***269***, 1 (2011), 359–365.