Genetic Algorithms and their Application to *in-silico* Evolution of Genetic Regulatory Networks

Johannes F. Knabe[‡†], Katja Wegner[†], Chrystopher L. Nehaniv[‡†], and Maria J. Schilstra[†]*

[†]Biological and Neural Computation Laboratory, and [‡]Adaptive Systems Research Group, STRI, University of Hertfordshire, Hatfield, AL10 9AB United Kingdom

*To whom correspondence should be addressed (m.j.1.schilstra@herts.ac.uk)

i. Abstract

A genetic algorithm (GA) is a procedure that mimics processes occurring in Darwinian evolution to solve computational problems. A GA introduces variation through "mutation" and "recombination" in a "population" of possible solutions to a problem, encoded as strings of characters in "genomes", and allows this population to evolve, using selection procedures that favor the gradual enrichment of the gene pool with the genomes of the "fitter" individuals. GA's are particularly suitable for optimization problems in which an effective system design or set of parameter values is sought.

In nature, genetic regulatory networks (GRN's) form the basic control layer in the regulation of gene expression levels. GRN's are composed of regulatory interactions between genes and their gene products, and are, *inter alia*, at the basis of the development of single fertilized cells into fully grown organisms. This paper describes how GA's may be applied to find functional regulatory schemes and parameter values for models that capture the fundamental GRN characteristics. The central ideas behind evolutionary computation and GRN modeling, and the considerations in GA design and use are discussed, and illustrated with an extended example. In this example, a GRN-like controller is sought for a developmental system based on Lewis Wolpert's French flag model for positional specification, in which cells in a growing embryo secrete and detect morphogens to attain a specific spatial pattern of cellular differentiation.

ii. Key Words

Evolutionary Computation; Genetic Algorithm, Genetic Regulatory Network, Modeling, Simulation, Gene Regulation Logic, Developmental Program

1 Introduction

With the sequencing of the human genome, and the inventory of its total protein-coding gene content, a full understanding of the programs controlling metabolism, development, and adaptation may seem to be within reach. However, over the years it has become increasingly clear that regulation of gene expression is achieved through highly complex dynamic interaction networks, often called Genetic Regulatory Networks (GRN's). There is now a large amount of information available about gene expression levels in different cells, tissues, and organisms, at different stages of development, response, and adaptation. Nonetheless, unraveling the intricate structure of the GRN's that underlie these processes is difficult, and requires the integration of a great number of experimental and computational resources. Techniques are being developed to pinpoint and quantitatively describe the interactions--of so-called *cis*-regulatory sites with *trans*-regulatory factors--that lead to enhanced or reduced gene expression. In addition, network structures can, at least partially, be inferred using sophisticated statistical techniques that detect correlations and dependencies in gene expression levels. On the basis of the progress already made in these fields, it is widely expected that one day it will be possible to use the knowledge about the basic interactions, constrained by information obtained with network inference techniques, to automatically create mathematical GRN models that not only show similar dynamics and responses, but also have the same network structure as the intracellular GRN's that they aim to describe. Procedures for automatic or computer-aided dynamic GRN generation will almost certainly make use of computational evolutionary techniques to search for networks that exhibit the intended behavior. It is likely that computational tools will be using a limited number of pre-defined dynamic components that describe processes such as transcription, translation, and transport, and sub-networks such as signaling cascades. The greatest challenges in the development of such tools lie, in the first instance, in deciding what the nature and characteristics of these pre-defined components should be, and in collecting sufficient information to be able to describe their dynamics in adequate detail.

The purpose of this chapter is to give readers an impression of what Evolutionary Computation (EC) is, and what its advantages and limitations are, and how it may be used. We illustrate how EC techniques, specifically a class of techniques called Genetic Algorithms, can be used to generate GRN models that show a particular, relatively complex behavior on the basis of an example taken from our own investigations into the potential of GRN-like structures as control systems. An in-depth discussion of the requirements for quantitative modeling and simulation of biological GRN's is beyond the scope of this chapter, and we will focus mostly on

"conceptual" or artificial GRN's, mathematical models that qualitatively capture the most distinctive features of biological GRN's.

1.1 Evolutionary computation and genetic algorithms

Relatively early in the history of modern computation, engineers had the idea to harness the power of evolution for optimization purposes. After all, evolution, driven by the related mechanisms of *genetic drift* and *natural selection*, has led to the huge diversity of well-adapted species we see on earth nowadays. Evolution, it is argued, is a massively parallel search that tests the ability of millions of populations to adapt to their habitats.

In its most general definition, biological evolution is "change in the gene pool of a population from one generation to the next". A *gene* is a hereditary, functional unit that can be passed on unaltered for many generations, and a *gene pool* is the collection of all genes in a *population*--here: a group of individuals that are able to interbreed and produce fertile offspring[1]. Genetic drift is the accumulation of random change in the gene pool of a population over time. Each individual member of a population has its own, unique *genotype*, the full set of genes in its *genome*. Genetic variation within a population is brought about by mutation, heritable change in the genotype of a single individual, or recombination, the reshuffling and exchange of genetic material between two individuals. An individual's ability to adapt to its environment is determined by its *phenotype*, the physical expression of its genotype in that environment. While mutation and recombination increase the diversity within a population, natural selection makes it decrease. "Fitter" individuals--those whose phenotype is best adapted to the environment--are the ones that, on average, produce more offspring. Greater fitness may be due to greater *viability*, the probability to survive to a given age, or to increased *fertility*, which is related to the total number of offspring produced during a lifetime. In a population whose size remains the same, traits that make individuals fitter will accumulate as the population evolves over many generations, while the less favorable ones will disappear.

The *evolutionary cycle*, in which a number of individuals are selected from a population on the basis of their fitness in step 1, to produce offspring that receives, in step 2, a combination of the--possibly mutated--parental genomes, and where, in step 3, the new generation replaces the whole previous generation in the population, is depicted in Figure 1. This simple cycle of selection, variation, and replacement forms the basis of all evolutionary computation. Evolutionary Computation (EC) has been found particularly appropriate for solving multidimensional problems whose parameter space is too large to search exhaustively or too complex to investigate systematically. By probabilistically searching and improving a "population" of "candidate

solutions"--rather than trying to find a single analytical solution, or applying deterministic search rules--the algorithms used in EC often find good, albeit not necessarily the best, solutions. As with all such global search heuristics, EC techniques are better suited to certain types of problems than to others, and are certainly not guaranteed to solve any problem optimally. However, because they have fewer restrictions, evolutionary algorithms are, on the whole, applicable to a wider range of problems than many other optimization techniques.

Genetic Algorithms (GA's) form a particular class of EC techniques[2], first developed in the 1960s by John Holland in an attempt to use natural adaptation mechanisms in computer science **(*1, 2*)**. GA's use selection methods and so-called genetic operators, such as cross-over, mutation, and inversion, to slowly change the information content in a population of "genomes". Using a set of rules that is different for different problems, the information in each genome is translated into a "phenotype". The characteristics of each phenotype are then compared with a set of desired characteristics, and assigned a "fitness value" on the basis of the comparison. The selection process is set to favor genomes that produce fitter phenotypes, allowing the fitter individuals to produce most of the offspring.

Thus, GA's employ procedures that simulate, in a highly simplified fashion, the processes that operate in biological evolution. The assignment of a fitness value to a genotype on the basis of a comparison with a target behavior is clearly unrealistic, as in nature there are no targets, just survival and reproduction probabilities in a given environment. However, the primary purpose of a GA is not to accurately simulate biological evolution, but to solve computational problems. In that context, GA's are often capable of providing satisfactory solutions to complex optimization problems that cannot so easily be solved by other, more formal methods. For further reading we recommend *An Introduction to Genetic Algorithms* by M. Mitchell **(*3*)**, and the two volumes in the series *Evolutionary Algorithms*, edited by Back, Fogel, and Michalewicz **(*4*)**.

1.2 Genetic regulatory networks

In its most basic definition, a GRN is a collection of regulatory interactions between genes and gene products. A gene product (PR)[3] is a macromolecule that has been constructed on the basis of the information present in the coding region of the gene, and may be a polyribonucleotide (RNA) or a polypeptide (protein). A gene is said to be *expressed* in a cell when its gene products can be found in that cell. Some genes are constitutively expressed, and their PR's always present. Most genes, however, are conditionally expressed: they are only activated under certain conditions. In spite of the fact that all cells that form an organism have the same genome, expression levels within a single organism often vary widely from tissue to tissue, from cell to cell, and over time. This

differential gene expression may originate in variations in certain extracellular conditions, or in variable detection of, or response to the external conditions, but can also be caused by asymmetries in the distribution of certain key molecules after cell division. Thus, the PR's participating in the cell signaling apparatus that detects these differences in conditions form the "input" to GRN's, whereas the PR's that contribute to the cell structure and metabolism are the "output".

Many PR's are involved in the activation or repression of the expression of one or more genes: they are said to function as *trans-regulatory* factors (TF's)[4] to those genes. Thus, the statement "gene A regulates gene B" in actual fact means that the PR of gene A is involved as a TF in the regulation of the expression of gene B. Regulation is mediated through so-called *cis-regulatory* TF binding sites. Physically, a *cis*-regulatory binding site is a stretch of DNA that has a high affinity for a specific TF or TF complex. If TF molecules are present in a given cell over a certain period of time (*i.e.* their genes are being expressed), they will bind to their *cis*-regulatory binding sites for at least part of that time, during which they act to increase or decrease the gene *transcription rate*. A gene may have many *cis*-regulatory sites that bind as many different TF's, each of which may affect the transcription rate differently. TF's often act synergistically, with their combined effect very different from the sum of their individual effects. A simple example of synergy is a situation in which only the complex of two different TF's (a hetero-dimer) 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 TF's--which, as the products of different genes, may require different sets of conditions for their own expression--are being expressed. Because the expression of a single gene may be regulated by many different TF's, and a particular PR may take part in the regulation of multiple genes including its own, the connection patterns in GRN's are often highly convoluted, and feedback loops are in abundance.

Mature PR's are created in a series of consecutive transformations of the primary transcript that include chemical modification (carried out by enzymes, a class of proteins that catalyze chemical reactions), transport from the nucleus to other parts in the cell, and, for proteins, translation of the information in the mRNA into a polypeptide chain, followed by post-translational modification into a mature protein. Activation of mature PR's may require further modification or complex formation. Furthermore, mature RNA and protein molecules are often actively broken down as soon as they are somehow surplus to requirement[5]. Modification, translation, transport, and breakdown are all as tightly controlled as gene transcription. In the following, however, we shall focus on the regulation of transcription, not only because it forms the basis for all higher-level regulation, but

also because it can be highly adaptive, and has the ability to integrate the information contained in a large amount of incoming data.

All GRN models include components that fulfill the roles of genes and TF's in biological GRN's. In their most abstracted form, GRN's are simply represented by "directed graphs": sets of nodes (boxes or circles) connected by arrows. The nodes represent genes, and the arrows indicate control: the arrowhead points at the gene whose expression is being regulated by the PR of the gene connected to the other end of the arrow. The arrows may have weights that express the effectiveness of a PR as a TF.

In more elaborate models, PR's, their precursors, and the various processes that contribute to changes in their levels may have individual representations. Although no one way of representing GRN's is "the best", the inclusion or omission of detail will certainly make particular models more fit for purpose than others.

Graphical representations such as the ones described above specify which processes can and cannot occur. However, the response of a network to an incoming signal can only be assessed on the basis of rules that describe how rapidly the value of a node changes when the values of other system components change. In their most simplified form, GRN's are modeled as Boolean networks, in which the genes are either "on" or "off". Boolean network models usually let a full response to changed conditions take place in a single, discrete time step. More complex dynamic models may include delays and fractional or exponential response mechanisms, and PR levels are allowed to assume (positive) values on discrete or continuous scales. Most models define a functional dependency of the rate of PR production on the amount of TF's present, so that increasing the amount of an activator, or decreasing the amount of a repressor increases the rate of PR production. In order to prevent unlimited growth, a PR breakdown rate or an upper level must be specified. Synergistic effects may be modeled in different ways, some of which are outlined in **(*5*)**.

The parameters associated with the delayed, fractional, or exponential response determine the rate at which these networks can adapt to changed conditions. Time may be modeled discretely or continuously. To simulate the responses of a network over time, discrete-time models require solution of a set of difference equations, whereas continuous-time models translate to sets of differential equations that are numerically integrated. Stochasticity (randomness) may be introduced in various ways in both types of models; however, introduction of stochasticity tends to be computationally costly, and, for highly abstracted models, does not usually yield a more accurate reflection of reality.

Again, no particular modeling framework is necessarily better or more realistic than any of the others. The mechanisms that operate within a biological cell or organism are so diverse and complex, and the models so abstract, that it is possible to find some "biological justification" for most. Altogether, there are far more ways of modeling GRN's than can be evaluated here. Many excellent reviews have appeared over the last decade, some discussing specific systems, some examining specific modeling techniques, and others attempting to evaluate the whole **(*6-28*)**. An example of how the above concepts and considerations can be incorporated in a dynamic model is presented in section 2.2.

1.3 Running example: evolving a GRN-like control system for cellular differentiation into a French flag pattern

Throughout this chapter, we shall illustrate the most important concepts in the field of evolutionary computation and GA's on the basis of an example application in which a GRN is evolved that controls cell division and gradient formation in a system based on Lewis Wolpert's famous French flag model of positional specification in the embryo (**(*29, 30*)** and references therein; see also **(*31*)**). Wolpert proposed this model in 1969 to explain a mechanism for retrieving positional information by arrays of cells in an embryo or tissue. The model is based on the assumption that, during early development, gradients of chemical signals called morphogens build up across the growing embryo. The cells in the embryo detect the morphogen concentration and compare it with certain built-in thresholds, to decide in which domain of the embryo--here: in the red, white, or blue section of the French flag--they are located. Since this model was first proposed, several variations of have been explored, *e.g.* **(*32, 33*)**. Although these models differ in detail, all include a 2D spatial arrangement of cells that can detect the concentration of morphogen in an underlying stratum. In our version, all cells can also produce this morphogen, and secrete it into the stratum. Gradients form when the morphogen diffuses away from the cells that secreted it, and eventually decays. Cells only "communicate" through these morphogen gradients; there is no other exchange of information between cells. Each cell has its own independent control unit that regulates, *inter alia*, its morphogen secretion rate, and determines its color. These control units are instances of a GRN, and have the same components and connectivity in each cell.

The task for the GA in this example is to find a GRN that controls cellular morphogen secretion in such a way that a gradient forms along the long axis of the flag-shaped environment as the cells in the embryo multiply and eventually fill the whole flag. The gradient has to be steep enough to allow the cells to determine whether its level at their position is above, in between, or below two separate thresholds, in order for them to adopt a blue,

white, or red color. More information on the development of this GRN-controlled French flag system, including a detailed description and discussion of the results, is found in **(*34*)**.

2 Materials

2.1 Cellular model

To mimic the growth and multiplication of cells in a developing embryo, we used an existing implementation of the so-called Cellular Potts Model (CPM). The CPM was originally developed by Glazier and Graner to simulate differential adhesion driven cell arrangement **(*35*)**, but has been re-used for a variety of cell-level modeling tasks, recently reviewed by Merks and Glazier **(*36*)**. The CPM has two layers, both consisting of pixel lattices (here of 60 × 40 pixels each). In the top layer, "cells" are represented as domains of adjacent pixels. Every pixel is associated with an integer number that identifies the cell to which it belongs, with zero indicating that it does not belong to any cell. The system advances in time steps in which each pixel attempts once, on average, to copy its value into a randomly chosen neighboring pixel. Copying success is limited by so-called effective energy constraints: each cell has an ideal size (number of pixels in the cell) at which its energy content is minimal, and deviation from the ideal incurs an energy penalty. A copying attempt will succeed in any case if the copying decreases the energy of the whole system. The copying event may even proceed if the overall energy is predicted to increase, but the probability of success decreases exponentially with increasing energy differences. Because of this, cells remain dynamic, even if their energy is close to minimal. Other cell properties that can be constrained, and therefore externally controlled, include shape (the ratio of its projections on the horizontal and vertical axes), its tendency to "stick" to neighboring cells, and the direction in which to divide. Importantly, all cells in our model have the ability to "secrete" one or more types of morphogen into the "stratum" in the bottom layer, and to read out its current level in each time step. The rate of secretion is open to external control. Each type of morphogen has fixed diffusion and decay rates, and its distribution over time over the stratum is computed by numerically solving the partial differential equations that describe the diffusion, decay, and production (secretion) of the morphogen. We used a flexible open source CPM implementation called CompuCell3D (see http://CompuCell3D.org and **(*36*)** for implementation and formalism details), which allowed us to choose pre-defined constraints, and also to define new ones where necessary. CompuCell3D also facilitates the use of external control modules to control the various parameters that determine cell behavior.

2.2 GRN model

The central components of the GRN control unit used to control the French Flag development are genes and gene products (PR's). The genes contain one or more *cis*-regulatory modules (CRMs), and each of these modules consists of one or more binding sites for *trans*-regulatory factors (TF's). Each gene produces one type of PR, and each type of PR can function as a TF in the expression of any gene (including its own) if at least one of the CRMs of that gene contains a binding site for that PR. A PR can function as a TF in the regulation of many genes, and a gene can have many TF binding sites. A PR may have many multiple binding sites on one gene, even within a single CRM. Genes may be constitutive or facultative. Constitutive genes are "on" in the absence of their TF's, and need overall repression to reduce their expression level; facultative genes are "off" by default, and need to be activated by their TF's to produce their PR's. TF's that bind to the same CRM act together, whereas the CRMs make independent contributions to the gene expression level. These contributions may either be activating or inhibitory (repressive).

These effects are quantitatively implemented as follows. At any one point in time, each PR in each cell is associated with a number, *n*, that represents the amount of PR molecules in that cell. The TF's that bind to a single CRM do so (conceptually) as a tight complex (a hetero-oligomer), so that the TF with the smallest number molecules present determines the total amount of complex that can be formed[6]. The contribution of one CRM to the overall expression level is taken to be proportional to the amount of complex formed. Activating CRMs make a positive, and inhibitory CRMs make a negative contribution. The sum S of the individual CRM contributions is then translated into a PR production rate v^p between zero and a maximum rate (*e.g.* 150 PR molecules per time unit). The relationship between v^p and S is sigmoid, with its inversion point below zero for constitutive, and above zero for facultative genes. Figure 2a shows a graphical representation of a single gene, and demonstrates how v^p is calculated from the quantities of its TF's. Each PR also decays at a rate v^d that is proportional to the amount n of PR present: $v^d = k^d \times n$. The proportionality constant k^d is different for each PR.

To simulate the response of the GRN in each cell to the changing conditions, time is divided in discrete steps Δt of unit size ($\Delta t = 1$). The quantity of the i^{th} PR in the j^{th} cell at the end of time step t, $n_{i,j}(t)$, is calculated by simply adding the number of PR molecules formed in time step t to the quantity $n_{i,j}(t-1)$ of the same PR in the previous time step, and subtracting the number of molecules that have disappeared, as shown in equation 1. This is done for all PR's in all cells.

$$n_{i,j}(t) = n_{i,j}(t-1) + (v^{p}_{i,j}(t) - v^{d}_{i,j}(t)) \times \Delta t \quad (1)$$

The GRN for the French Flag system uses 20 genes to produce a maximum of 16 different PR's (justified in section 3.1). All PR's may be used as TF's (to regulate the expression of other genes). Furthermore, a total of 12 PR's have the following pre-defined functions. 1) For each cell, the only input into the GRN comes from the average concentration of two morphogens, $m_1$and m_2, in the part of the stratum directly underneath the cell. This is done at the start of each time step by reading out, and setting the number of molecules of two particular PR to values proportional to $m_1$and m_2. A further 10 PR's form the GRN's output, and act as controllers for the CPM. One PR controls the cell's target size: when there is a large amount present, the cell will try to grow until it reaches the size threshold for division (and then divides); if its quantity is zero, the cell will shrink, and eventually disappear (undergo "apoptosis"). Five more PR's set the cell's shape, stickiness, and division direction targets, essentially in the same way. Another two govern the morphogen production-secretion rate: again, the more PR, the greater the rate. Finally, two PR's determine the color of the cell: high quantities of both (compared with a set threshold) make the cell blue, one high and one low gives white, and two low quantities yields a red cell.

2.3 Simulation

In a simulation round, the CPM is combined with a "wired" (equipped with a particular connectivity pattern, as encoded in the genome; see section 3.1) and parameterized GRN model. The simulation starts with a single cell, and therefore one copy of the GRM. After setting the PR quantities in that cell to their initial values (usually zero), the system is left to develop. In viable individuals, the cell will begin to grow as soon as the PR that controls the target size accumulates, and will divide when its threshold for division is reached. Upon division, the cell's PR content is distributed between the daughter cells in quantities proportional to their sizes, and each daughter receives an exact copy of the GRN. Because of the way the CPM is designed, cells cannot grow on top of each other, or outside the pixel lattice, so that the lattice fills up with cells over the course of the simulation. The fraction of the lattice that is covered by cells by the end of a simulation round (of 200 time steps, in our case) is dependent on the growth rate that is achieved by the system. Numerical integration of the rate equations for the GRN system, as expressed in equation 1, is deterministic: if a simulation is started with the same initial PR quantities in cells with equal and constant sizes, the PR quantities will develop equally in all cells. However, the CPM is a non-deterministic system: it uses a random number generator to choose a neighboring pixel in each

copying attempt (section 2.1). Provided they are carried out with a different sequence of random numbers, no two simulation rounds will be the same[7], even if they start under equal initial conditions.

2.4 Software

Genetic algorithm-aided design of artificial GRN's as controllers for specific system requires a combination of three elements: 1) the device for which a controller is required (here the CPM), 2) a GRN modeling and simulation tool, and 3) a tool that applies the GA.

Obviously, the "device" is different for each application; it may be a virtual system (such as the CPM in our running example), but could equally well be a physical apparatus, a robot of some kind, or simply a table that lists the quantities of particular PR's during a simulation. Whatever the device, it must be possible to formulate a desired behavior for it, and to somehow quantify its accomplishment in comparison with the target.

Because there is no standard way of modeling GRN's, few off-the-shelf tools are available for this purpose. Some relatively mature tools that have been designed specifically for modeling and analyzing GRN's are the Genetic Network Analyser (GNA) **(*37*)**, and GINsim **(*38*)**, which use qualitative simulation methods for predicting gene expression in highly abstracted GRN models. Furthermore, there are many software packages that are suitable for modeling of biochemical reaction networks and dynamical systems in general (see **(*39*)** and **(*40*)** for examples), and in principle it is possible to use almost any of these to create dynamic GRN models and simulate their dynamics. Mathematics packages such as Mathematica, MATLAB, Maple, and Octave, as well as programming languages for which mathematics libraries are available, such as FORTRAN, C/C++, Java, and Python, may facilitate the numerical simulation of GRN's. Moreover, these high-level programming packages often contain generic tools for setting up and using GA's.

In general, because application of GA's to any problem requires the integration of a number of software tools, some of which may have to be developed from scratch, and most will need to be adapted to the current problem, knowledge of a high-level programming or scripting language, or familiarity with a dedicated mathematics package is essential.

As a general rule, application of GA's is computationally intensive. In the case of our running example, an "evolutionary run" lasted for 250 generations, with each generation consisting of 250-300 individuals; that is 250 times 250-300 × 10 simulation rounds of 200 time steps each (see sections 2.3 and 3.3), plus overhead for the computation involved in the fitness assessment for each individual in each generation, the selection

procedure, and the application of the evolutionary operators to each generation. Although it is possible to do these computations using a single processor, they are readily split into segments that can be performed in parallel. We used a specialized workload management system called Condor **(*41*)** to distribute the parallel segments between hundreds of desktop workstations in our University (including many student lab machines), harnessing the CPU power that would have been wasted while they were standing idle, and cutting the overall computation time by one or two orders of magnitude.

3 Methods

As explained in section 1.1 and show in Figure 1, an evolutionary cycle includes three steps:

1. Selection: from a population of candidate solutions to the problem, encoded in "genomes", a number of individuals are selected for reproduction. Each genome is associated with a "phenotype", and selection occurs on the basis of its phenotypic characteristics
2. Variation: the offspring of the selected genomes receive a "mutated" and "recombined" version of their parental genomes
3. Replacement: the previous population is replaced by their offspring, after which event a new cycle begins

An evolutionary run usually starts with a population of random solutions (genomes), and terminates either when a pre-defined level of satisfaction with the solution has been met, or simply after a pre-set number of cycles has been completed. The application of a GA to a specific problem requires tailored specification of:

1. A genome[8]: a representation of the "solution domain" that allows modification by evolutionary operators
2. A phenotype: a representation whose characteristics can be compared to the target of the search, and a collection of rules that stipulate how the information contained in a genome is transformed into a phenotype
3. A fitness function to evaluate the solution contained in the genotype and manifested in the behavior of the phenotype
4. Appropriate mechanisms for selection, mutation and recombination, and reproduction, and a termination strategy

In this section we introduce the most popular approaches to the introduction of variation and selection, and describe in detail how the genome, phenotype, and fitness concepts were tailored to our running example.

3.1 Genome representation

In GA's, genomes are typically represented by strings (linear sequences, as in DNA itself) of characters that somehow represent a solution to the problem that needs to be solved. These strings are often just sequences of zeros and ones, but larger "alphabets" -- sets of permitted characters -- are also allowed. The genome representation in our running example uses an alphabet of 4 characters: 0, 1, 2, and 3. Each genome consists of 20 individual "genes"[9], and also encodes some information that is global to the system. Each gene is subdivided into one "coding" area and several "*cis*-regulatory modules" (CRMs). In turn, each CRM contains a number of "binding sites" for TF's. The characters 2 and 3 function as CRM and gene delimiters only, whereas the information carried by the characters 0 and 1 varies, as explained below. Figures 5b (full genome) and 2b (single gene) illustrate the encoding. The last (rightmost) 7 characters in the full genome are used to set two global properties, whose significance for brevity we will not go into. The next (from the end) 16×4 positions in the string encode the decay rates for each of the 16 (see below) different PR's. The 72^{nd} character from the end, always a 3, indicates the start of the first gene. Although genes may have different numbers of CRMs, they are structured in the same way. The gene delimiter, 3, is followed by a single character (0 or 1) that determines whether the gene is constitutive (0) or facultative (1). The next four characters, all 0 or 1, indicate, as a binary number, the PR encoded by the gene; thus 0000 simply encodes pr00, 0101 corresponds to pr05, and 1111 to pr15[10]. Any zeros or ones following this 5-character area are ignored, and the regulatory region begins at the first CRM delimiter (a 2) to the left of the gene. CRM representations may have different lengths, but the character (0 or 1) that immediately follows the delimiter always indicates whether the overall effect of the TF complex that binds to the CRM is inhibitory (0) or activating (1). The characters (0 or 1) in the "TF-binding area" of the CRM, which extends up to the following CRM or gene delimiter, determine which PR's will bind to the CRM. To this aim, the TF-binding area, for instance 00111110010110, as in CRM α in Figure 2b, is split into as many quadruplets as possible, reading from left to right along the CRM (here, 0011, 1110, and 0101), and a residual (here 10). The residual (10) is taken to be "junk", and ignored, but the quadruplets (0011, 1110, and 0101) specify that pr03, pr14, and pr05 act, in synergy, as TF's in the expression of the gene to which the CRM belongs.

3.2 Genotype-phenotype mapping

In the case of our running example, the rules that specify how the information contained in the genome is transformed into the phenotype are easily understood. The phenotype is formed by the whole machinery of the CPM (section 2.1) and its GRN controller (section 2.2). Most of the equations that govern behavior of the phenotype are contained in the CPM and GRN model themselves, and are therefore unchangeable. However, the connectivity of the GRN is contained in the genome, as well as some of the parameter values that determine the dynamics of the PR's (namely their decay rates). The decay rates are encoded as "words" of four characters (0 or 1), each of which can therefore represent ("address") one of 16 different values. In our example, an even narrower "mapping" was used: the numbers 0 to 9 (0000 to 1001) represent decay rates of 0.0 to 0.9 (in steps of 0.1), and all higher numbers (1010 to 1111) map on a decay rate of 1.0. Furthermore, 12 of the 16 possible PR's were given the following (predetermined, and unchangeable) control functions: color determinants: pr00 and 01; CPM constraint control: pr02, 03 (morphogen secretion), 04 (shape), 05, 06, 07 (stickiness), 08 (size), 10 (preferential division direction); and morphogen concentration sensors (input): pr12 and 13.

3.3 Fitness evaluation

To rank the fitness of the individual genomes, a so-called fitness function, a quantitative measure for their proximity to the target, must be specified. The design of the fitness function depends entirely on the nature of the problem, and may vary from simple root mean square deviation to a sophisticated statistical analytic function.

In our example, fitness was assessed as follows. As stated in section 2.3, each simulation round, in which a phenotype was given a chance to develop from a single cell into a fully grown 2D array of cells, lasted for 200 time steps. At the end of the simulation, the fitness f of each individual was quantified using a pixel-by-pixel comparison of final arrangement of colors $\boldsymbol{R}$ in the cell array (after 200 time steps in the simulation) with the target pattern $\boldsymbol{T}$ (the French flag) as expressed equation 2:

$$f = 1 - \frac{4 - n_C}{w \times h} \sum_{x=1}^{w} \sum_{y=1}^{h} (R_{xy} \neq T_{xy}) \qquad (2)$$

Here, x and y enumerate the width w (60 pixels) and height h (40 pixels) of the lattice, respectively, so that $0 < x \leq w$, and $0 < y \leq h$. The color T_{xy} of the pixel at position (x, y) in the target lattice is compared with the color of the pixel in the same position in the cell array, R_{xy}. If the colors are different, the statement $R_{xy} \neq T_{xy}$ is

true, and if they are the same, it is false. Each "false" is converted to 0, and each "true" to 1, and the resulting zeros and ones for each position are added up, and the answer divided by the total number of pixels, $w \times h$. Thus, the fitness f will evaluate to its maximum value of 1 if all pixels in $\boldsymbol{R}$ have the same color as those in $\boldsymbol{T}$ (so that the sum of all ($R_{xy} \neq T_{xy}$) terms equals 0). The factor $4 - n_C$, where n_C is the total number of colors present in the final cell array, in the second term of the function, puts a "penalty" on the use of fewer than three colors in the final pattern, as follows. Suppose $\boldsymbol{R}$ uses all three colors ($4 - n_C = 1$), and 25% of the pixels in $\boldsymbol{R}$ have the same color as their counterparts in $\boldsymbol{T}$, so that the sum of all ($R_{xy} \neq T_{xy}$) terms equals 0.75. In that case, f evaluates to 0.25. However, if only one or two colors are used in $\boldsymbol{R}$ ($4 - n_C = 3$ or 2), f is much smaller, namely -1.25 or -0.5, respectively. In this way, individuals are "encouraged" to use all three colors early on in the evolution. This refinement of the fitness function was deemed to be necessary when it was found that too many evolutionary runs led to monochrome arrays, indicating that the original fitness function (without the $4 - n_C$ factor) had insufficient discriminatory power to allow the search come to a satisfactory conclusion. Furthermore, the overall fitness of an individual was computed by carrying out ten separate simulation rounds (performed with different random number sequences, see section 2.3) and averaging the values of f obtained in each of these rounds, to ensure that the solution was relatively robust.

3.4 Initial population

To start an evolutionary run, an initial genome population must be created. The composition of this initial population is, again, entirely dependent on the problem itself, on the way the problem has been encoded in the genome, and also on the prior knowledge that is to be taken into account. In the case of the running example, the initial population of 250-300 individuals consisted of genomes in which each of the 20 genes had a single CRM. All mutable characters in the sequences (the zeros and ones) were randomly assigned, yielding a collection of randomly connected network. Note that the genes in these networks produced a subset, and not necessarily all, of the possible PR's.

3.5 Selection and replacement

Once the fitness of each individual in the population has been established, a proportion of the genomes are selected to act as parents for the next generation. Typically, two individuals are selected, and allowed to produce two offspring. During this process, exchange and mutation of the genetic material occurs, so that the offspring differ somewhat from the parents. Selection of parents continues until the total number of offspring is equal to

the number of individuals in the parent population, after which the older population is replaced with its offspring. Several strategies have been developed that let the fittest produce the most offspring, but leave a fair amount of genetic variability in the population. Here we introduce the selection techniques that are most widely used for the selection of the parents.

The first method is "fitness proportionate" selection, a weighted lottery in which the chance to be selected increases with fitness. This method is often depicted as a roulette wheel, as in Figure 3a. As individuals may be selected more than once, their reproduction rate is proportional to their fitness. Another method involves a "tournament", in which a number of individuals are randomly chosen from the current population, and out of these the two fittest are selected for reproduction (Figure 3b). Again, individuals may be selected multiple times, and the fittest may win several tournaments, but here their reproduction rate is not necessarily proportional to their fitness. The last strategy is "elitism", in which the genomes of a number of the fittest individuals are copied exactly into the pool of offspring. The remainder of the offspring is then generated using one of the other procedures. In a strategy that uses elitism, the highest fitness value never decreases in the population, which some consider to be an advantage.

In the example, a combination of elitism with tournament selection was used: the fittest genome was kept unchanged, and the rest of the offspring were obtained from tournaments involving 15 randomly chosen individuals.

3.6 Variation

As in nature, mutations in a genome are achieved by changing a character in a random position along the length of the genome into another character from the alphabet, or by inserting or deleting a string of characters. Cross-over is realized by allowing two parents to exchange part of their genomes. Figure 4 illustrates the typical implementation of point mutation, deletion, duplication, inversion, and cross-over.

In the running example, point mutations are made by changing a 0 into a 1 or a 1 into a 0. Point mutations to delimiters (2 or 3) do not occur. Cross-over requires a pair of genomes, P1 and P2, and is performed as follows. Each genome is parsed into 21 segments, 20 of which containing a gene, and one containing the global parameter section. One segment is randomly selected, and a cross-over position, X, is chosen within this segment. If the chosen segment happens to be the global area of the genome (which has the same length in both parents), the genome parts distal[11] to X on P1 and P2 are swapped (Figure 4 Xeq). If X is situated in one of the

genes, an offset *d* is randomly selected from a Gaussian distribution centered on 0, with a standard deviation of 4 (but cut off in cases where the end of a gene will be crossed). Then the parts distal to X1 = X - d on P1 and X2 = X + *d* on P2 are swapped, as illustrated in Figure 4 Xne. The insertions and deletions that result from this Gaussian-cross-over mechanism may generate a reading frame shift, and thus change the gene's regulatory pattern. Moreover, it allows the total number of CRMs within a gene to change. The relatively small standard deviation in the Gaussian distribution prevents extensive perturbation of existing control patterns. Nonetheless, in the course of the evolutionary run CRMs, and as a result even whole genes, may become non-functional, as CRM size is allowed drop below that of a single TF binding site. "Degenerate" CRMs without TF binding sites are ignored in the computation of the gene expression rate.

Associated with each genetic operator is a value that specifies the frequency with which it is to be applied. In the example, the probability that a 0 or 1 changes into a 1 or a 0 during a single evolutionary cycle is 1%, whereas the probability for a point-mutation in a 2 or a 3 is zero. The deletion, duplication, and inversion parameters were not used, but the Gaussian-offset cross-over frequency per genome in one evolutionary cycle was set at 90%.

4 Results

The results of one successful evolutionary run are summarized in Figure 5. The top panel shows that the fitness of the best individual in the population rises in a number of distinct phases. These phases are characterized by an initial sharp increase in fitness, followed by period in which the rate of increase becomes gradually smaller, sometimes dropping down to almost zero for an extended period. It would appear that an "evolutionary innovation" has been made, and a whole new set of better solutions has been uncovered. The GA then spends some time searching "close to home" for the best solution within this new area, until it hits upon a next innovation. The similarity between the target pattern in the inset in the top panel, and the achieved solution, in the bottom right panel (Figure 5d, *t* = 200, bottom row) is approximately 90% after 250 evolutionary cycles. The genome string of the best individual after the full run and the diagram of the GRN it represents (Figures 5b and c) demonstrate that during the course of the evolution the genome has shrunk from the initial 20 functional genes to 13, and that of these only 8 contribute to the total gene expression pattern (all 5 non-contributing genes are facultative and need activation to start producing their PR; however g04, 15, 17, and 19 have no activating TF's, and g10 is activated only by its own product). Only 8 out of 16 possible PR's are produced, with only one (out of a possible 4) without a predefined function: pr00 and 01 (color 1 and 2), 03 (morphogen 2 secretion), 05

and 06 (stickiness 1 and 2), 08 (size), 10 (division direction), 13 (morphogen 2 concentration sensor), and 15 (the only TF without a predefined function). Note that the system could have used two different morphogens and three different stickiness constraints, but appears to have "discovered" that it needed to use only one morphogen (morphogen 2), and that two stickiness constraints were sufficient.

Figure 5d shows snapshots of the expanding cell array and the morphogen gradients in its stratum at various developmental stages. The simulation begins with a single cell, initially consisting of one blue pixel, which grows and turns to red, then to white, and divides. After 5 time steps, there are two white cells, which then proliferate in the next 5 steps to a total of 14. After 20 time steps, there are about 50 cells, and second (red) color begins to appear to the right of the white cells. At around $t = 40$, the third color (blue) appears to the left of the pattern. The area fills up with cells in 70 to 80 time steps, and, while the cells remain dynamic, the final pattern stabilizes after some 110 time steps. It should be emphasized that the solution presented here is not the only one possible: other evolutionary runs have produced GRN's with an apparently entirely different connectivity and greater complexity that were equally capable of directing the development of a French flag pattern.

5 Concluding Remarks

Just as there is no single best way of modeling GRN's, there is no all-purpose GA, and not all problems lend themselves equally well to the GA approach. Even if a solution domain is easily expressed in some sort of "genome" structure, the genome may turn out not to be "evolvable". Poor evolvability may be inherent in the problem, but it may also be due to the way the solution domain is encoded, the genotype-phenotype mapping, or simply to the choice of evolutionary operators and parameter settings. As a rule, it is a good idea to ensure, if possible, that single mutations mostly lead to relatively small changes in the phenotype. It is also advised to try out different evolutionary operators and parameter settings. Techniques exist to automatically adapt parameter values during an evolutionary run: a widely used algorithm is Simulated Annealing **(*42, 43*)**, but it is also possible to use other EC techniques (which are, after all, good at optimization) for this purpose **(*44*)**. Furthermore, solutions to complex problems may sometimes be found by "easing" the system along by shifting the target, as was done in **(*45, 46*)**. To this aim, the "evolutionary pressure" is gradually increased, by first letting the system find a good solution to a part of the problem, and then gradually changing the target to the parts for which a solution is more difficult to find.

It may have become clear from the above exposition that while GA's borrow ideas from nature, they still lack a lot of biological detail. Because basic GA's have quite a few limitations, their users--engineers, scientists--have been prompted to inject more ideas from natural evolution into GA's to try and boost their performance in optimization problems **(*47*)**. It is generally believed that the introduction of "junk code" (as in the genome representation in our example) and diploidy with dominance-recessivity allows for information buffering and protection. The inclusion of developmental programs in which a single unit grows into a differentiated multi-unit structure may result in improved scalability, modularity, and robustness. Because genetic diversity allows species to adapt more easily to changing conditions in nature, some GA's include strategies designed to preserve diversity. Many of these and similar extensions do indeed improve performance in particular instances, but on other occasions they may only add complexity and increase the search time **(*48*)**.

Notes

[1] For reasons of simplicity we have omitted the concept of species from this discussion.

[2] Other techniques include Evolutionary Strategies, Genetic Programming, and Evolutionary Programming; discussion of their characteristics and application is outside the scope of this article.

[3] We use PR, rather than GP, as the abbreviation for gene product, in order to avoid confusion with GP as an abbreviation for Genetic Programming, another branch of evolutionary computation. Note that PR does not stand for protein, although proteins are a PR category, but could mean Protein/RNA.

[4] TF is often used as an abbreviation for Transcription Factor; however in the definition of *trans*-regulatory factor to which we adhere here, Transcription Factors are a specific class of TF's.

[5] In fact, a capacity for rapid breakdown contributes significantly to a cell's potential to adapt efficiently to changed conditions

[6] Note that this approach is highly simplified, and does not take into account that a complex may contain more than one molecule of a particular PR, and that a PR may take part in more than one type of TF complex

[7] Provided the sequences of random numbers produced by the random number generator (RNG) are different: this is achieved by using a different "seed" value for the RNG in each simulation round

[8] In this context sometimes also indicated as genotype or chromosome

[9] The number of genes was fixed to facilitate further analysis

[10] Thus, a coding area consisting of 4 "bits" (whose value can be 0 or 1) may code for one of a total of $2^4 = 16$ different PR's

[11] Distal to X: the genome section from X to the end of the genome

Abbreviations

CPM, Cellular Potts Model; CRM, *cis*-regulatory module; EC, evolutionary computing; GA, genetic algorithm; PR, gene product (protein or RNA); GRN, genetic regulatory network; TF, *trans*-regulatory factor

6 References

1. Holland, J. H. (1962) Outline for a logical theory of adaptive systems *J ACM* **9**, 297--314.

2. Holland, J. H. (1992) *Adaptation in natural and artificial systems, 2nd edition*, MIT Press, Cambridge, MA.

3. Mitchell, M. (1998) *An introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, USA.

4. Back, T., Fogel, D. B., and Michalewicz, Z. (1999) *Evolutionary Algorithms*, Vol. I and II, IOP Publishing Ltd, Bristol, UK.

5. Schilstra, M. J., and Nehaniv, C. L. (2008) Bio-Logic: Gene expression and the laws of combinatorial logic *Artif Life* **14**, 121-33.

6. Karlebach, G., and Shamir, R. (2008) Modelling and analysis of gene regulatory networks *Nat Rev Mol Cell Biol* **9**, 770-80.

7. Kulasiri, D., Nguyen, L. K., Samarasinghe, S., and Xie, Z. (2008) A Review of Systems Biology Perspective on Genetic Regulatory Networks with Examples *Curr Bioinf* **3**, 197-225.

8. Mitrophanov, A. Y., and Groisman, E. A. (2008) Positive feedback in cellular control systems *Bioessays* **30**, 542-55.

9. Cho, K. H., Choo, S. M., Jung, S. H., Kim, J. R., Choi, H. S., and Kim, J. (2007) Reverse engineering of gene regulatory networks *IET Syst Biol* **1**, 149-63.

10. Goutsias, J., and Lee, N. H. (2007) Computational and experimental approaches for modeling gene regulatory networks *Curr Pharm Design* **13**, 1415-36.

11. Tomlin, C. J., and Axelrod, J. D. (2007) Biology by numbers: mathematical modelling in developmental biology *Nat Rev Gen* **8**, 331-40.

12. Palumbo, M. C., Farina, L., Colosimo, A., Tun, K., Dhar, P. K., and Giuliani, A. (2006) Networks everywhere? Some general implications of an emergent metaphor *Curr Bioinf* **1**, 219-34.

13. Kaznessis, Y. N. (2006) Multi-scale models for gene network engineering *Chem Eng Sci* **61**, 940-53.

14. Facciotti, M. T., Bonneau, R., Hood, L., and Baliga, N. S. (2004) Systems biology experimental design - Considerations for building predictive gene regulatory network models for prokaryotic systems *Curr Genom* **5**, 527-44.

15. Welch, S. M., Dong, Z. S., Roe, J. L., and Das, S. (2005) Flowering time control: gene network modelling and the link to quantitative genetics *Australian J Agric Res* **56**, 919-36.

16. Prusinkiewicz, P. (2004) Modeling plant growth development *Curr Op Plant Biol* **7**, 79-83.

17. Kaern, M., Blake, W. J., and Collins, J. J. (2003) The engineering of gene regulatory networks *Ann Rev Biomed Eng* **5**, 179-206.

18. Thieffry, D., and Sanchez, L. (2003) Dynamical modelling of pattern formation during embryonic development *Curr Op Gen Dev* **13**, 326-30.

19. Bolouri, H., and Davidson, E. H. (2002) Modeling transcriptional regulatory networks *Bioessays* **24**, 1118-29.

20. De Jong, H. (2002) Modeling and simulation of genetic regulatory systems: A literature review *J Comp Biol* **9**, 67-103.

21. Shmulevich, I., Dougherty, E. R., and Mang, W. (2002) From Boolean to probabilistic Boolean networks as models of genetic regulatory networks *Proc IEEE* **90**, 1778-92.

22. Hasty, J., McMillen, D., and Collins, J. J. (2002) Engineered gene circuits *Nature* **420**, 224-30.

23. Van Someren, E. P., Wessels, L. F. A., Backer, E., and Reinders, M. J. T. (2002) Genetic network modeling *Pharmacogenomics* **3**, 507-25.

24. Rao, C. V., and Arkin, A. P. (2001) Control motifs for intracellular regulatory networks *Ann Rev Biomed Eng* **3**, 391-419.

25. Hasty, J., McMillen, D., Isaacs, F., and Collins, J. J. (2001) Computational studies of gene regulatory networks: In numero molecular biology *Nat Rev Gen* **2**, 268-79.

26. Smolen, P., Baxter, D. A., and Byrne, J. H. (2000) Modeling transcriptional control in gene networks - Methods, recent results, and future directions *Bull Math Biol* **62**, 247-92.

27. McAdams, H. H., and Arkin, A. (1998) Simulation of prokaryotic genetic circuits *Ann Rev Biophys Biomol Struct* **27**, 199-224.

28. Kauffman, S. A. (1993) *The origins of order. Self-organization and selection in evolution* Oxford University Press, New York, USA.

29. Wolpert, L. (1969) Positional information and the spatial pattern of cellular differentiation *J Theor Biol* **25**, 1-47.

30. Wolpert, L. (1996) One hundred years of positional information *Trends in Genetics*, 359-64.

31. Jaeger, J., and Reinitz, J. (2006) On the dynamic nature of positional information *BioEssays* **28**, 1102-11.

32. Miller, J. F. (2004) Evolving a Self-Repairing, Self-Regulating, French Flag Organism *Proc Gen Evol Computation - GECCO 2004* **1**, 129-39.

33. Meinhardt, H. (1982) *Models of biological pattern formation*, Academic Press, London.

34. Knabe, J. F., Schilstra, M. J., and Nehaniv, C. L. (2008) Evolution and morphogenesis of differentiated multicellular organisms: Autonomously generated diffusion gradients for positional information, in *Artificial Life XI: Proc Eleventh International Conference on the Simulation and Synthesis of Living Systems*, pp 321-8, MIT Press, Cambridge, MA, USA.

35. Glazier, J. A., and Graner, F. (1993) Simulation of the differential adhesion driven rearrangement of biological cells *Phys Rev E* **47**, 2128-54.

36. Merks, R. M. H., and Glazier, J. A. (2005) A cell-centered approach to developmental biology *Physica A* **352**, 113-30.

37. De Jong, H., Geiselmann, J., Hernandez, C., and Page, M. (2003) Genetic Network Analyzer: qualitative simulation of genetic regulatory networks *Bioinformatics* **19**, 336-44.

38. Gonzalez, A. G., Naldi, A., Sánchez, L., Thieffry, D., and Chaouiya, C. (2006) GINsim: A software suite for the qualitative modelling, simulation and analysis of regulatory networks *Biosystems* **84**, 91-100.

39. Schilstra, M. J., Martin, S. R., and Keating, S. M. (2008) Methods for simulating the dynamics of complex biological processes, in *Biophysical Tools for the Biologist* (Correia, J. J., and Dietrich, W. H., Eds), Methods in Molecular Biology, (Wilson, L., and Matsudaira, P. T., Eds), pp 807-41, Elsevier Inc., San Diego.

40. Szallasi, Z., Stelling, J., and Periwal, V., (Eds.) (2006) *System modeling in cellular biology*, MIT Press, Cambridge, MA.

41. Thain, D., Tannenbaum, T., and Livny, M. (2005) Distributed computing in practice: the Condor experience *Concurrency - Practice and Experience* **17**, 323-56.

42. Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983) Optimization by Simulated Annealing *Science* **220**, 671-80.

43. Cerny, V. (1985) A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm *J Opt Theor App* **45**, 41-51.

44. Beyer, H. G., and Schwefel, H. P. (2002) Evolution strategies: a comprehensive introduction *Natural Computing* **1**, 3-52.

45. Knabe, J. F., Nehaniv, C. L., and Schilstra, M. J. (2006) Evolutionary robustness of differentiation in genetic regulatory networks *Proc 7th German Workshop on Artificial Life 2006 (GWAL-7)*, 75-84.

46. Knabe, J. F., Nehaniv, C. L., and Schilstra, M. J. (2008) Genetic Regulatory Network models of Biological Clocks: Evolutionary history matters *Artif Life* **14**, 135-48.

47. Altenberg, L. (1994) The evolution of evolvability in genetic programming, in *Advances in Genetic Programming* (Kinnear, K. E., Ed.), pp 47-74, MIT Press, Cambridge, MA, USA.

48. Beer, R. D. (2004) Autopoiesis and cognition in the game of life *Artif Life* **10**, 309-26.

49. Longabaugh, W. J. R., Davidson, E. H., and Bolouri, H. (2005) Computational representation of developmental genetic regulatory networks *Dev Biol* **283**, 1-16.

Figure captions

Figure 1. The evolutionary cycle, consisting of parent "selection" (fitter parents produce, on average, more offspring), introduction of genetic variation in the offspring through cross-over and mutation mechanisms, and replacement of the parent generation with their offspring, who will act as parents in the next round. In Evolutionary Computation, an evolutionary run must be initiated with a starting population, and is terminated when a pre-defined fitness or time criterion is met. The population size is kept constant throughout the full evolutionary run.

Figure 2. Gene representation. a. Dynamic model, illustrating how a PR production rate is computed from the TF quantities present in one cell, at one particular point in time. The gene depicted here (as a horizontal line crossed by a bent arrow) is constitutive, and has three CRMs, α, β, and γ, one of which (γ) is activating, and the other two inhibitory. The boxes indicate the names of the PR's that act as TF's to the gene, with their quantities shown in bold above the boxes. For each CRM, the minimum TF quantity is carried through, and negated for inhibitory CRMs. The sum S of these values is then used to compute v^P, the PR production rate. The relationship between S and v^P is sigmoid, as shown in the graphs to the right of the gene. The broken line, here with a midpoint at -5, represents $v^P(S)$ for constitutive genes; the solid line (midpoint at 15) that for facultative genes. b. Gene encoding: genes are encoded in the genome using 3 as the gene delimiter, 2 as CRM delimiters, and zeros and ones immediately to the right of the delimiter to indicate a constitutive (13) or facultative (03) gene (bold, underlined), and an activating (12) or inhibitory (02) CRM (bold). The rest of the gene representation consists solely of zeros and ones, and represent the PR of the gene (the four characters immediately to the left of the 31 or 30), and the TF's (as many quadruplets to the left of each 21 and 20 that will fit). The residual characters (of which there may be zero or more in the PR area, and zero to three in the CRM areas), indicated in gray, do not have a function.

Figure 3. Selection methods. a: Roulette wheel selection: each individual is assigned a slice of the wheel whose size is proportional to its fitness. The wheel is rotated, and when it stops, the individual associated with the slice under the pointer is selected as a parent. b: Tournament selection: a fixed number of individuals is randomly chosen from the pool. The couple with the highest fitness scores is selected as parents.

Figure 4. Introducing genetic variation. Top panel: mutation mechanisms, requiring one parent (P) and producing one child (C). PM, point mutation: a single zero changes into a one or vice versa; Del, deletion: a

substring of characters is deleted in the child genome; Dup, duplication: a copy of substring appears in the child genome (not necessarily adjacent to the original); Inv, inversion: a substring is inverted in the child. Bottom panel: Cross-over mechanisms, requiring two parents (P1, P2), and producing two children (C1, C2). Xeq, equal cross-over: the characters to the left of a position X on two equally sized genomes or genome parts are swapped between the parents; Xne, non-equal crossover: the characters to the left of X - *d* on P1 are swapped with those left of X + *d* on P2. In a Gaussian-offset cross-over mechanism, the offset *d* is randomly chosen from a Gaussian distribution.

Figure 5. Results. a. Fitness value (*f*) of the fittest individual in the population over an evolutionary run of 250 generations. Inset: the "French flag" target pattern, with the dark gray zone on the left and the lighter gray zone on the right representing the blue and the red areas of the flag. b. The genome of the fittest individual in the 250^{th} generation. The gene delimiters are underlined; the area of the genome with the global information is indicated in gray. c. Box-and-arrow GRN representation of the genome in b (visualized using the tool BioTapestry **(*49*)**). The symbols labeled g00, g03, etc, represent genes, and the text boxes labeled pr00, pr01, etc, represent their gene products. Constitutive and facultative genes are represented by, respectively, black and gray symbols. The boxes of PR's with a pre-defined function have a solid outline; black for input and output, and gray for PR's that drive a CPM constraint (see text). Connections ending in arrows (activating) and bars (inhibitory) specify regulatory interactions; the α under g13 indicates that both regulatory interactions happen on one CRM, named α (in this GRN, there are no other CRMs with multiple participants, and none of the genes have more than one CRM). Note, furthermore, that non-functional genes g01, 02, 06, 08, 09, 11, and 16 have been left out of the drawing, and that only pr00, 01, 03, 05, 06, 08, 10, 13, and 15 are produced. d. Four stages (after 10, 20, 50, and 200 time steps) in the simulation of the development controlled by the GRN in b and c. The four top panels show the spread of the morphogen over the stratum (with the cell outlines superimposed; the stratum itself consists of 40 × 60 pixels with no higher level structure); darker grays indicate higher morphogen concentrations. The bottom panels show the proliferation of cells. Blue, white, and red cells are represented in middle gray, white, and black, respectively, and the light gray background color indicates the absence of cells.

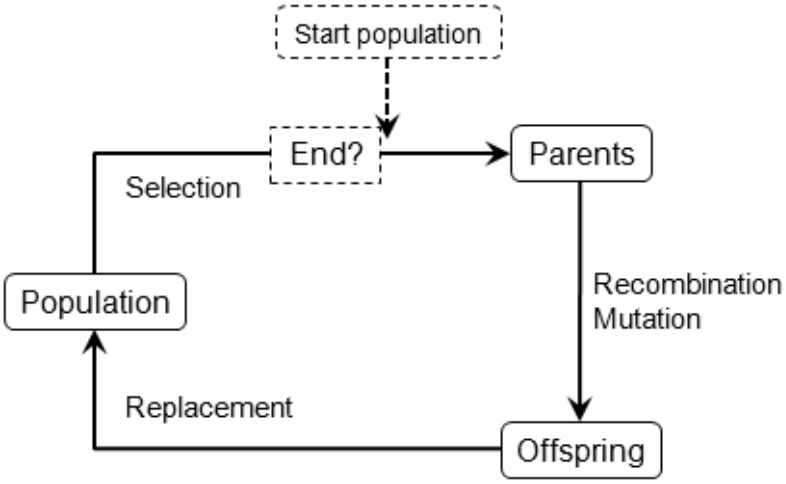

Start population
End?
Parents
Selection
Population
Recombination
Mutation
Replacement
Offspring

Figure 1

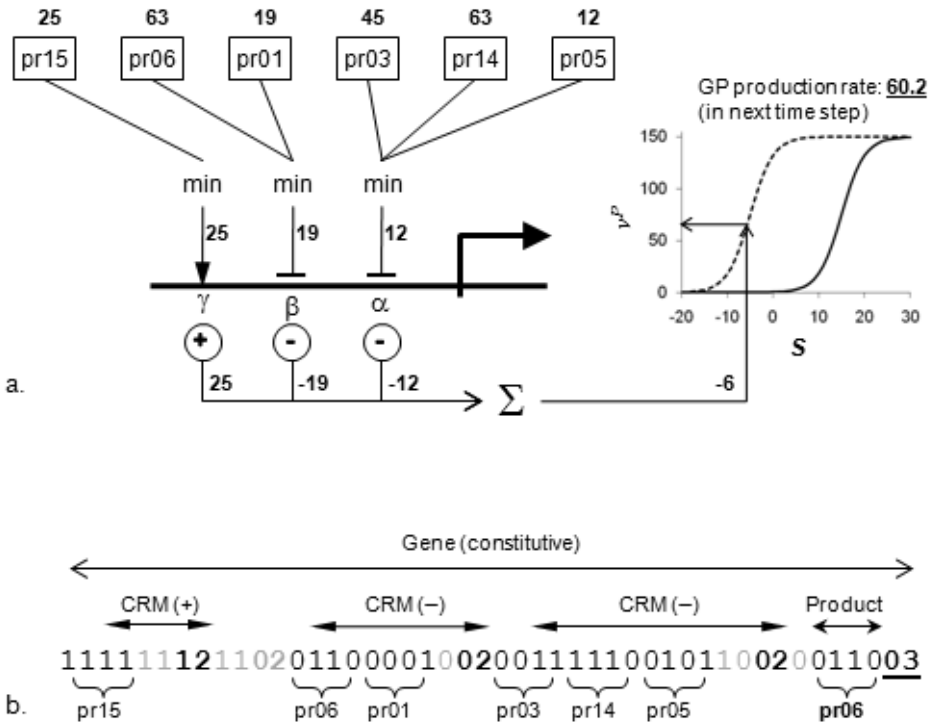
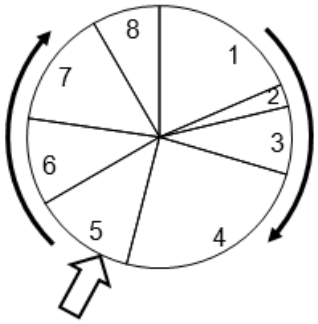

a.
25
63
19
45
63
12
pr15
pr06
pr01
pr03
pr14
pr05
min
min
min
25
19
12
γ
β
α
25
-19
-12
Σ
-6
GP production rate: 60.2
(in next time step)
150
100
50
0
-20
-10
0
10
20
30
S
b.
Gene (constitutive)
CRM (+)
CRM (–)
CRM (–)
Product
pr15
pr06
pr01
pr03
pr14
pr05
pr06

Figure 2

Genome	1	2	3	4	5	6	7	8
Fitness	0.72	0.11	0.32	0.95	0.49	0.42	0.56	0.32

Roulette selection



Selected	1	2	3	4	**5**	6	7	8
a.	0.72	0.11	0.32	0.95	**0.49**	0.42	0.56	0.32

Tournament selection

	1	⇩ 2	3	4	⇩ 5	⇩ 6	⇩ 7	8
	0.72	0.11	0.32	0.95	0.49	0.42	0.56	0.32

Selected	1	2	3	4	**5**	6	**7**	8
b.	0.72	0.11	0.32	0.95	**0.49**	0.42	**0.56**	0.32

Figure 3

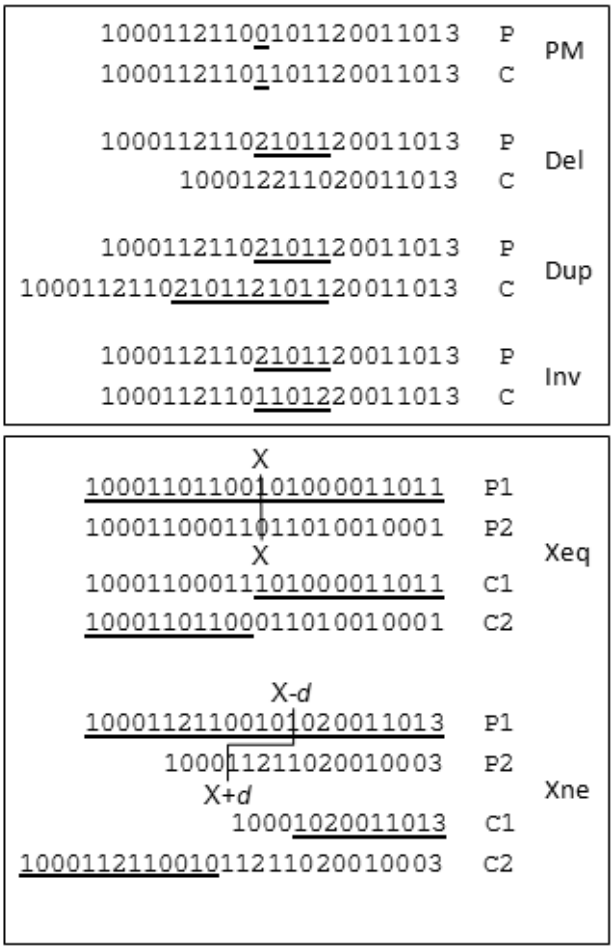
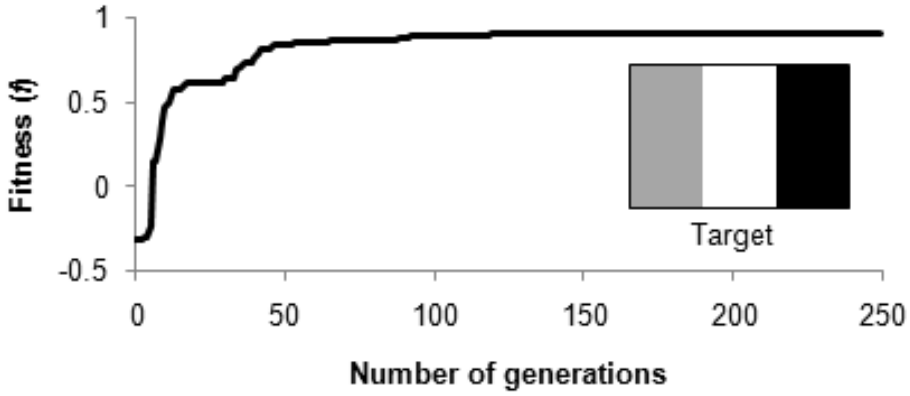
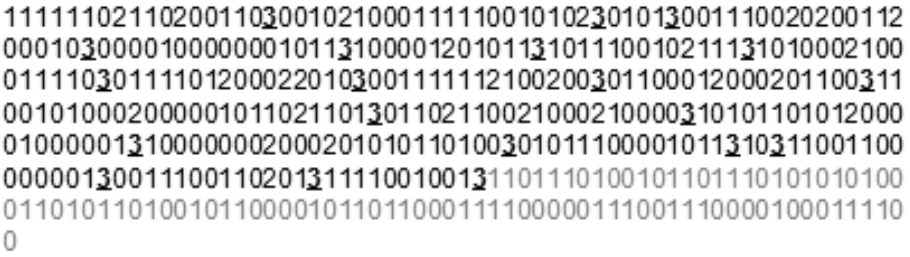

10001121100101120011013 P
PM
10001121101101120011013 C
10001121102101120011013 P
Del
100012211020011013 C
10001121102101120011013 P
Dup
1000112110210112101120011013 C
10001121102101120011013 P
Inv
10001121101101220011013 C
X
10001101100101000011011 P1
10001100011011010010001 P2
X
Xeq
10001100011101000011011 C1
10001101100011010010001 C2
X-d
10001121100101020011013 P1
100011211020010003 P2
X+d
Xne
10001020011013 C1
10001121100101121102001003 C2

Figure 4

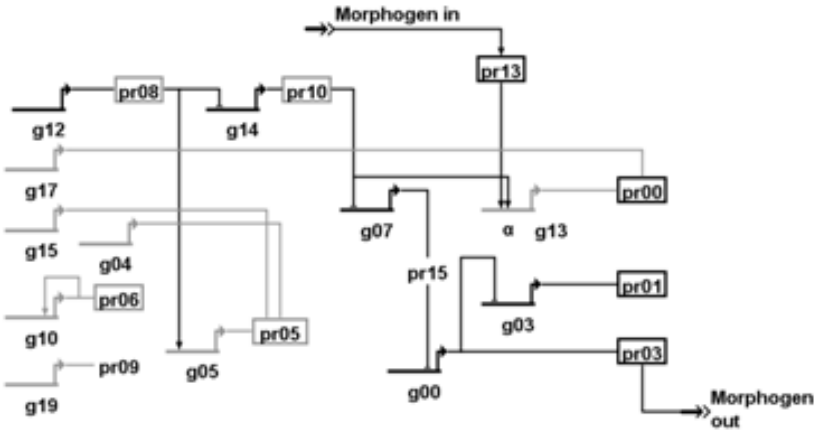
a.



b.



c.



d.

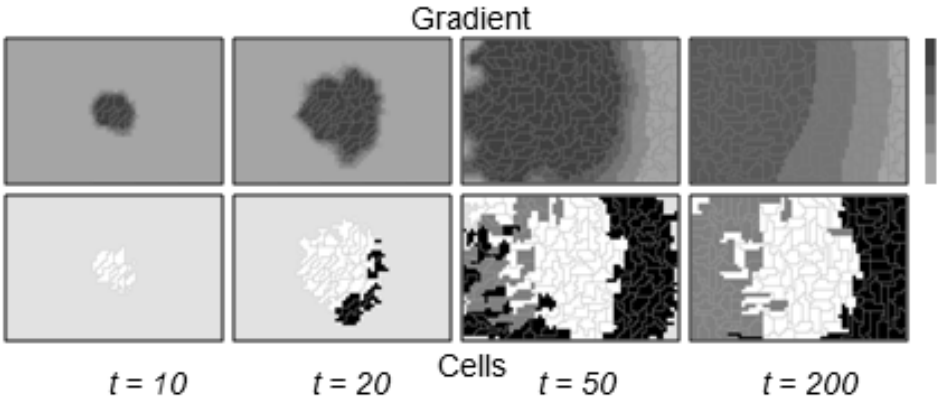


Figure 5