Open Archive TOULOUSE Archive Ouverte (OATAO)

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in: http://oatao.univ-toulouse.fr/
Eprints ID: 13052

To link to this article: DOI: 10.1162/ARTL_a_00136
URL: http://dx.doi.org/10.1162/ARTL_a_00136

To cite this version: Cussat-Blanc, Sylvain and Pollack, Jordan
Cracking the egg: virtual embryogenesis of real robots. (2014)
Artificial Life, vol. 20 (n° 3). pp. 361-383. ISSN 1064-5462

Any correspondance concerning this service should be sent to the repository administrator: staff-oatao@listes-diff.inp-toulouse.fr
Cracking the Egg: Virtual Embryogenesis of Real Robots

Sylvain Cussat-Blanc*
*Brandeis University

Jordan Pollack*
*Brandeis University

Abstract All multicellular living beings are created from a single cell. A developmental process, called embryogenesis, takes this first fertilized cell down a complex path of reproduction, migration, and specialization into a complex organism adapted to its environment. In most cases, the first steps of the embryogenesis take place in a protected environment such as in an egg or in utero. Stating from this observation, we propose a new approach to the generation of real robots, strongly inspired by living systems. Our robots are composed of tens of specialized cells, grown from a single cell using a bio-inspired virtual developmental process. Virtual cells, controlled by gene regulatory networks, divide, migrate, and specialize to produce the robot's body plan (morphology), and then the robot is manually built from this plan. Because the robot is as easy to assemble as Lego, the building process could be easily automated.

1 Introduction

Growing a robot from a single functional unit, in other words a cell, is one of the most exciting dreams for researchers in the modular robotics field. Recent works have allowed the development of bigger and more complex robots. Usually based on spherical or cubical units, they embed complex mechatronics that make them more accurate, more efficient, and more diversified [41]. In this domain, the Molecube may be seen as an example. It was first based on a cube able to turn around its central axis to demonstrate limited self-replication. Lipson and his team went on to develop new modules to build more complex machines [42]. For example, we can cite the gripper modules able to pick up objects, the wheel cubes, and the camera cube. To control these blocks, a neural network must be evolved. However, the user has to design by hand the morphology of the robots. That means that, in case of modification of the machine's purpose, a human has to imagine a new morphology potentially adapted to the new problem before developing its controller.

M-TRAN is another example of a self-reconfigurable robot [31]. This robotic system is composed of 1-DOP modules capable of self-reconfiguration through a magnetic linking and unlinking system. In this case, the robot morphology is usually designed by a human, but the system is capable of changing from one morphology to another by reorganizing its structure. By these means, the robot can adapt to its environment.

* Contact author.
In 2007, Christensen proposed a robotic approach where the behavior of the machine, also composed of rotating cubic blocks, is strongly linked to its morphology [6]. Indeed, instead of having a fixed morphology, the robot is self-reconfiguring in order to accomplish its function. This method yields a very interesting property: The robots are capable of self-healing, as multicellular organisms are. However, the linking and unlinking system makes the cubes very expensive and complex to manufacture.

On the other side, the history on virtual robots, commonly called artificial creatures, dates from the early nineties with Karl Sims' creatures [39]. It took another six years before evolved robots took the step from virtual to real [26]. Sims used a graph-based genotype-phenotype map to generate a morphology controlled by a neural network, whereas Lipson and Pollack directly evolved the morphology. Both systems are coevolved so that the controller is adapted to the morphology as much as possible. L-systems have also been used for the same purpose. Kosominski has created one of the most advanced works in this domain. In this work, a neural network is coevolved with the morphology generator (an L-system) to produce artificial robots. Artificial gene regulatory networks (GRNs) appear to be able to generate complex 2D and 3D morphologies when they control a developmental model [4, 8, 14, 15, 22]. They also have been proved capable of controlling virtual and real agents for various tasks such as pole balancing [33] or food foraging [21]. More recently, Schramm et al. were able to grow a digital aquatic worm that moves by oscillation generated by the same regulatory network used to grow it [36].

In 2009, Jin proposed to name this emerging field of research "morphogenetic robotics" [19] and proposed a classification of existing works into three categories: morphogenetic swarm robotic systems that use biological morphogenesis processes such as gene regulation or hormone systems to organize a swarm of robots [17, 28, 38], morphogenetic modular robots that use morphogenesis principles to self-reorganize their environment [2, 29], and developmental approaches used to design the morphology and the behavior of the robots [10, 18, 25]. However, in the morphogenetic modular and developmental approaches, very few of these works have been designed to build a real robot. One of them was proposed in 2003 by Hornby et al. They proposed a model where an L-system is evolved to control a "turtle" factory that could produce a virtual robot controlled by a neural network [18]. The best robots were 3D printed and assembled by hand. They were capable of moving like the virtual robots.

The main motivation of this article is to use a cell-based developmental system driven by a gene regulatory network to generate real robot morphologies. In our opinion, an evo-devo method is well suited for this range of problems because it is more biologically plausible than other quasi-developmental approaches and thus should scale up better. Using a biologically plausible developmental model to produce robots could in itself help us to tackle crucial properties such as clustering, symmetry, regularity, robustness, and complex self-organization into tissues and organs. All these properties are necessary to produce the extremely complex machines that living systems are, and they would be extremely helpful in robotics.

To present our embryogenic approach to modular robot design, the article is organized as follows. The next section presents the real robotic blocks that will be used to build the robots. Section 3 presents the developmental model, the gene regulatory network, and a novel nonlinear morphogen production system. In Section 4, the evolution of the developmental model is presented through two experiments. First, we have used an interactive evolutionary algorithm to point out the capacities of the system, with regard to how easy it is to obtain complex morphologies, symmetry, and regularity of the robots. We then propose a standard evolution of the robot with a genetic algorithm and present various virtual and real robots capable of locomotion. The article concludes with possible extensions of this work.

2 The Robotic Blocks

In this work, the design of our machines was inspired by living organisms. An organism is usually composed of billions of functional units: the cells. Cells can have various functions (neural, muscular, etc.). Thus, our machine uses multiple block types that have different functions. The blocks must be easy to assemble, so we can conceive an automatic assembly process using off-the-shelf robot arms.
and conveyor belts. When a machine is built, its morphology will no longer evolve (as many grown living beings do not modify their morphology). With this constraint, the blocks do not need a dynamic linking and unlinking system. Finally, each block must be cheap to produce, to keep the overall machine cost low.

With all these requirements, we have designed each block as a cube, scaled to 2 in. (about 5 cm) per side. Currently, these blocks are printed in a 3D printer, but the design could allow easy mass production of the unit through a mold. As presented in Figure 1, three different units are used in the work (the blueprints of the blocks are presented in the Appendix):

- A **nope** block is used to build the structure of the robots (as bones are to build the structure of the body). This block does not have any function except the capacity to be linked to other blocks.

- A **hinge** block can rotate around a central axis within the range \([-\pi/3; \pi/3]\). This block does not move by itself, but can be actuated by a neighboring motor block through two actuators, as tendons actuate a natural articulation. These actuators are made of plastic to be light and inelastic. To ensure ease of construction, the hinge is made of one piece with all the necessary motions of the actuators.

- A **motor** block actuates the hinge block. The actuators protruding from the hinge blocks are tied to the motor so that the motor can actuate the hinge. A servomotor (Futaba S3102 metal gear) is embedded in the motor block.

The choice to separate the hinge and the motor comes from a biological observation. In nature, most articulations and muscles are separated. They are linked together with tendons. With this method, muscles can have various shapes and sizes, according to local constraints (space, needed strength, etc.). We decided to use this idea for our building blocks for two main reasons. First, motors are usually the heaviest parts of the robots. Modifying the position of the motors in the robot can modify its center of mass, which can be useful. Secondly, we can also imagine increasing the strength of a joint by having multiple motors actuating the same hinge (as multiple muscular cells actuate the same articulation).

As our robotic system does not reconfigure itself—each block’s position is given by the evolved body plan of the robot (its morphology)—we can keep the linking system as simple as possible. As presented in Figure 1, it consists of a dovetail latch slipped between two blocks. The latches are also printed on our 3D printer.

We have decided to keep the conception of the blocks simple for three main reasons. First of all, the cost of the blocks has to be as low as possible to keep the total cost of the robots low. This is usually not the case in classical robotic approaches or in modular robotics, where the embedded electronics strongly increases the total price of the robots. Our blocks are made of plastic, and their

---

**Figure 1.** The robots are composed of three different blocks (from left to right): a nope block, a hinge block, and a motor block. Blocks are tied together by use of dovetail latches inserted between the blocks.
only mechanical feature is the servomotor, which has an acceptable price (about $50). The blocks are also simple enough to be easily mass-produced. The production of the plastic block itself can be easily automated, and the only assembly consists of clipping the motor to the actuators, screwing the motor to the structure, and finally closing the hood of the block with two other screws. The protocol takes about 2 min and could be reduced by using clips instead of screws. The last reason is the possible miniaturization of the blocks. For now, the blocks are dimensioned around the servomotor, which is the biggest non-plastic part of the blocks. Miniaturizing this motor could allow a proportional miniaturization of all the blocks. It could allow the assembly of a robot composed of thousands of these blocks.

For the purpose of this article, 16 blocks of each type (noop, hinge, and motor) have been produced and can be assembled with the latches. An external Arduino Mega 2560 microcontroller board is used to control the motors, and the motors are powered via an external source. We are currently designing a brain block that embeds an Arduino Mini and a energetic block that integrates a battery to power the robot. The main task with these blocks is to manage data and power transmission through the different blocks. An electronic study is necessary in order to prove that the new blocks would solve the power distribution problem while enabling the previous successful constructions.

To develop the body plans of the robots, we propose a novel approach based on a developmental model in which cells are controlled by a gene regulatory network coupled to a novel nonlinear morphogen production system. In comparison with other methods such as graphals or L-systems, this approach has the advantage of being more biologically plausible. Moreover, some properties, such as regularity and symmetry, emerge from the system without any prior encoding. The next section presents this model.

3 The Developmental Model

3.1 The Cells and Their Environment

The developmental model is based on cells that act in a discrete 3D environment. Each cell has a unique identifier given at its birth, an orientation vector, and a position in the environment. Two main types of cells are available: (1) undifferentiated cells, or stem cells, that can divide as much as they want to; and (2) differentiated cells coming from a specialization of a stem cell to one of the three following types: B-cells, H-cells, or M-cells. When a cell is differentiated, it can no longer divide. The use of stem cells to drive the organism's growth has already been successfully used to develop complex shapes by Fontana in [16].

At each time step of the simulation, the stem cells can perform one of the following actions:

- **Symmetric division:** When a stem cell performs this action, a new stem cell is placed at the position of the mother cell while the mother cell migrates in the direction given by its orientation vector. Whereas the mother cell keeps the same identifier, the new cell receives a new unique identifier. Its initial orientation vector is the same as the mother's vector.

- **Asymmetric division:** A stem cell can divide to create a new differentiated cell. In this case, the mother cell first produces an exact copy (same internal state) of itself in the direction indicated by its orientation vector. While the newly created cell stays a stem cell, the mother cell differentiates to a B-cell. Thus, this cell can no longer divide.

- **Differentiation:** A stem cell can become a B-cell. Because a differentiated cell cannot divide, this action stops the proliferation of the stem cell.

- **Rotation:** A cell can reorient its orientation vector. When it chooses this action, it can turn in one of the four following directions: right, left, top, or bottom. While turning, the cell stays at the same position.

- **Special division:** A cell can divide asymmetrically twice during one time step. In this case, it first produces an H-cell at the initial position, an M-cell one voxel away from the initial position in the direction of its orientation vector, and a copy of itself two voxels away.

364

Artificial Life Volume 20, Number 3
Naturally, two cells cannot be in the same voxel of the environment. To satisfy this requirement, all inconsistent actions are disabled at a time step of the growth. For example, if a cell wants to divide, the position in the direction of the orientation vector is first checked. If the position is not free (in other words, already occupied by another cell), the division is disabled. The cell will have to choose another action. At least two actions are guaranteed to be always available: rotation and differentiation. Indeed, both actions only affect the cell itself. No new voxels are necessary.

After the growth of the organism from a single cell to a connected set of cells, each cell is translated to a robotic block: B-cells and stem cells are interpreted as no-op blocks, M-cells produce motor blocks, and, finally, H-cells produce hinge blocks.

To evaluate the morphologies generated, a virtual robot is built and dropped into a 3D physics environment. This simulator is used to visualize the produced robots before actually building them. We use the well-known Bullet physics engine coupled with OpenGL for the graphic visualization.

To select an action, cells are using a gene regulatory network. The next section details our model based on a simplification of Banzhaf’s GRN [1].

### 3.2 The Gene Regulatory Network

#### 3.2.1 Background on Artificial Regulatory Networks

Many current developmental models rely on an artificial GRN to simulate cell differentiation. These systems are inspired by the gene regulation systems of living systems. In living systems, the cells of an organism have several functions. They are specified in the organism genome, and their expressions are controlled by a regulatory network [11]. Cells use external signals collected from protein sensors localized on the membrane to activate or inhibit the transcription of the genes. The gene expressions determine the cells’ behaviors.

Eggenberger first used a GRN to generate a 3D organism able to move in its environment by modifying its morphology [15]. Reil then proposed a biologically plausible model with a genome defined as a vector of numbers [34]. In this model, each gene starts with the sequence (0101), named the promoter. Then, a graph is used to visualize the gene activations and inhibitions over time with randomly generated networks. Observations revealed the existence of various patterns such as gene activation sequencing, chaotic expressions, or acyclic expressions. Reil also pointed out that the system was resistant to random deteriorations of the genomes. Banzhaf described an artificial GRN model close to real-world gene regulation [1], detailed further below. Starting from these seminal models, many variations have been explored in order to address various concerns and applications. Several works addressed artificial embryogeny problems with models of GRNs ranging from cellular automaton modeling [5] to stripped-down versions of the GRN combined with complex developmental systems [13, 20]. Some works have also addressed control problems: using GRN as a control function to map a virtual robot’s sensory inputs to its motor actuator values. This has been applied in various tasks, from foraging agents [21] to pole balancing [33].

#### 3.2.2 Our Implementation of the Regulatory Network

We have based our regulatory network on Banzhaf’s model [1]. He designed it to be as close as possible to a real gene regulatory network. As DNA is composed of a sequence of nucleotides, Banzhaf’s network is encoded within a sequence of bits. As a real gene starts with the particular sequence of nucleotides—for example, TATA, a gene in Banzhaf’s network, starts with a particular sequence of 8 bits named the promoter. A gene is then encoded next to this sequence by five 32-bit integers, named the sites. This mechanism allows the generation of a variable number of genes in a fixed-size chromosome. However, as in nature, this encoding generates a certain amount of noncoding DNA (the probability of having a promoter being very low \(2^{-8}\)), which is undesired in a computational model aimed to be evolved with an evolutionary algorithm.

Banzhaf’s model was designed neither to be evolved nor to control any kind of agent. However, Nicolau used an evolution strategy to evolve the GRN to control a pole-balancing cart [33]. Although
the cart has shown consistent behaviors, the evolution of the GRN has been a problem. In our opinion, the difficulty in the evolution is due to: (1) the noncoding DNA and (2) the dynamics of the network.

Based on these observations, we decided to modify the encoding of the regulatory network and its dynamics. In our model, a gene regulatory network is defined as a set of proteins. Each protein has the following properties:

- The protein identifier is coded as an integer between 0 and $p$. The upper value $p$ of the domain can be changed in order to control the precision of the GRN. In Banzhaf’s work, $p$ is equal to the size of a site, which is 32 bits. We decided to increase the precision $p$ to 64.
- The enhancer identifier is coded as an integer between 0 and $p$. The enhancer identifier is used to calculate the enhancing matching factor between two proteins.
- The inhibitor identifier is coded as an integer between 0 and $p$. The inhibitor identifier is used to calculate the inhibiting matching factor between two proteins.
- The type determines if the protein is an input protein (whose concentration is given by the environment of the GRN and that regulates other proteins but is not regulated), an output protein (whose concentration is used as output of the network and that is regulated but does not regulate other proteins), or a regulatory protein (an internal protein that regulates and is regulated by other proteins).

This encoding removes the problem of noncoding DNA in Banzhaf’s approach. Each integer is used in the regulatory network, and a modification of one of them will automatically imply a modification of the network.

The dynamics of the GRN are calculated as follows. First, the affinity of a protein $a$ with another protein $b$ is given by the enhancing factor $n_{ab}^+$ and the inhibitory factor $n_{ab}^-$:

$$n_{ab}^+ = p - |\text{enh}_a - \text{id}_b|, \quad n_{ab}^- = p - |\text{inh}_a - \text{id}_b|$$

where $\text{id}_a$ is the identifier, $\text{enh}_a$ is the enhancer identifier of protein $a$, and $\text{inh}_a$ is the inhibitor identifier.

Then, the proteins are compared two by two, using the enhancing and the inhibiting matching factors. For each protein of the network, the global enhancing and inhibiting values are given by the following equations:

$$g_i = \frac{1}{N} \sum_{j=1}^{N} \gamma e^{(\beta j^{\text{enh}} - \lambda_{j_{\text{max}}})}, \quad h_i = \frac{1}{N} \sum_{j=1}^{N} \gamma e^{(\beta j^{\text{inh}} - \lambda_{j_{\text{max}}})}$$

where $g_i$ ($h_i$) is the enhancing (inhibiting) value for a protein $i$, $N$ is the number of proteins in the network, $j$ is the concentration of protein $i$, and $\lambda_{j_{\text{max}}}$ ($\lambda_{j_{\text{max}}}$) is the maximum enhancing (inhibiting) matching factor observed. $\beta$ is a control parameter described hereafter.

The final modification of protein $i$’s concentration is given by the following differential equation:

$$\frac{dc_i}{dt} = \frac{\delta (g_i - h_i)}{\Phi}$$

where $\Phi$ is a function that keeps of the sum of the output and regulatory protein concentrations equal to 1.

$\beta$ and $\delta$ are two constants that set up the speed of reaction of the regulatory network. The higher their values, the more sudden the transitions in the GRN. The lower they are, the smoother the transitions are.
The GRN is defined as a set of these proteins. A GRN can have a variable number of them. To be evolved, the GRN can be encoded in a chromosome defined as a variable-length chromosome of indivisible proteins, as presented in Figure 2. Each protein is encoded with four integers: three between 0 and 4 for the three different identifiers, and one between 0 and 2 for the type of the protein. If an evolutionary algorithm is to evolve this chromosome, the modification operators have to be redefined. First, the crossover consists in exchanging subparts of two different networks (see Figure 3). Because proteins are indivisible, the crossover points have to be chosen between two proteins. That ensures the integrity of each subnetwork. The local connectivity is thus kept. Only new links between the different subnetworks are created. As depicted in Figure 4, the mutation can be applied in three equiprobable ways: mutating an existing protein by randomly changing one of its four integers, adding a randomly generated new protein, or removing one randomly chosen protein from the network.

3.2.3 Encapsulation into the Cells

This section presents how the above-mentioned gene regulatory network is embedded within the cells of our developmental system. Cells have five different actions: symmetric division, asymmetric division, differentiation, reorientation (rotation), and special division. However, if a cell chooses to rotate, it also has to decide in which direction (left, right, top, or bottom). To encode this in the GRN, we decided to subdivide the outputs into two groups of output proteins: The first group contains five output proteins that match with the five possible actions, and the second group contains four output proteins that...
encode the four possible rotation directions when the rotation is selected in the first group of proteins. Thus, nine output proteins are necessary to encode the possible actions of the cells.

As illustrated by Figure 5, these nine output proteins are encoded at the beginning of the genome. Their three identifiers (protein, enhancer, and inhibitor) can be mutated, but the type cannot. The remaining elements of the genome are composed 25% of input proteins and 75% of regulatory proteins (empirically chosen, determined by a series of pre-tests to find a reasonable range). They can be mutated and crossed over as previously presented.

At each step of the simulation, the cells execute the regulatory network for 25 steps (found to be within reasonable range during preliminary tests) and select the action corresponding to the maximum protein concentration of the first five proteins: If the first protein has the maximum concentration, the cell divides symmetrically; if the second protein has the maximum concentration, the cell divides asymmetrically; and so on. When the rotation action is chosen, it gives access to the second group of output proteins. The cell calculates the maximum value of the last four output proteins' concentrations to determine in which direction to turn.

The coefficients $\beta$ and $\delta$ presented in the dynamics model are also encoded in the genome. They modify the dynamics of the network. In the differential equation, $\beta$ affects the importance of the matching factor, and $\delta$ affects the level of production of the protein. The lower either one of these values is, the smoother the regulation; the higher either one of these values is, the more sudden the

Figure 5. The chromosome that encodes the GRN is subdivided into two parts: The first nine proteins are output proteins, and the remainder of the chromosomes encode input and regulatory proteins.
3.3 Nonlinear Morphogen Production

The genotype of our developmental model also describes a novel nonlinear morphogen production system. This system generates the inputs to the gene regulatory network based on the time step and the index number of the cell. In order to be efficient, each input protein of the GRN must have a different morphogen production profile that provides the concentration of the input protein all through the simulation. Moreover, this profile must also be different for each stem cell in the organism. Indeed, if all the cells had the same profile, it would not allow any variation of their behaviors. They would all perform the same sequence of actions. In summary, the concentration $c$ of an input protein $i$ is related to two parameters: the current time step $t$ of the simulation, and the identifier of the cell in which the concentration is calculated.

Thus, the morphogen production profile of an input protein can be coded by a 3D surface. The first two dimensions of the surface correspond to the simulation time on the abscissa and the identifier of the cells on the ordinate. The third dimension gives the concentration of the corresponding input protein’s concentration according to the first two dimensions. The global morphogen system is thus defined as a set of $p$ 3D surfaces. Each surface is associated with the identifier of the input protein (which varies between 0 and $p$, $p$ being the precision of the GRN).

To generate these surfaces, we used Bézier’s algorithm [3]. Originally used to design car bodywork, it allows the generation of a surface by means of a given set of checkpoints. In this work, we use a Bézier surface of order $(n = 10, m = 10)$. This order corresponds to the number of checkpoints. For a $(10, 10)$-order surface, $(10 + 1)(10 + 1) = 121$ checkpoints $k_{ij}$ are necessary. With these checkpoints, the $z$ coordinate of every point with coordinates $(x, y, z)$ of the surface is given by the following equation:

$$z = \sum_{i=0}^{10} \sum_{j=0}^{10} B_i^n(\xi) B_j^m(\eta) k_{ij}$$

where $B_i^n(\xi)$ is the Bernstein polynomial:

$$B_i^n(\xi) = \frac{n!}{i!(n-i)!} \xi^i (1 - \xi)^{n-i}$$

Figure 6 presents some examples of randomly generated surfaces using this method. They show the diversity of possible landscapes generated with only 121 checkpoints.
Each possible input protein must have a Bézier surface assigned. To do so, we have decided to allocate a surface to each possible protein identifier. When the identifier of an input protein is changed by mutation to another value, a new surface is thus assigned. With this method, $p$ surfaces are necessary for a GRN of precision $p$ (a protein identifier is coded with an integer between 0 and $p$; here $p = 64$). To encode this system into the genome, the checkpoints are encoded in the third chromosome, which consists of a common set of $121p$ double-precision floats. Each value can vary in the range $[-1.0; 1.0]$. This chromosome can be classically crossed over and mutated. In order to not overflow the regulatory network with the input proteins, the input concentrations given by Bézier surfaces are normalized between 0 and 0.05.

The main inconvenience of this method is the number of points necessary to describe the curve. Here, 7,744 points are necessary. To reduce this amount of data, we also propose to use a 4D meta-surface of order $(5, 5, 5)$ instead of a 3D surface of order $(10, 10)$. The equation previously presented has been extended to one more dimension as follows:

$$t = \sum_{i=0}^{5} \sum_{j=0}^{5} \sum_{k=0}^{5} B_{i}^{5}(x) B_{j}^{5}(y) B_{k}^{5}(z) k_{i,j,k}$$

Here, the third dimension ($z$) specializes the production profile of the morphogen according to the corresponding input protein ID. In comparison with the previous method, this approach only needs 216 rather than 7,744 checkpoints to generate the surface. The compression gain is very important (97%), and we expect the same kinds of results. However, the complexity is increased due to the addition of the fourth dimension. A comparison of the two approaches to the robot morphology and their influence on the convergence of the genetic algorithm will be given in Section 4.2.

### 3.4 Summary of the Model

To summarize, cells are acting in a 3D discrete environment. Stem cells can divide symmetrically, asymmetrically, or twice in the same time step to produce an articulation. They can also reorient their orientation vector and differentiate to B-cells. To control their developmental behavior, cells have a gene regulatory network coupled to a nonlinear morphogen production system. The gene regulatory network is used to control the cell behavior. Each input protein concentration is given by a morphogen concentration. Each morphogen is produced in the cells according to a profile given by a set of 3D Bézier surfaces or a single 4D Bézier surface whose checkpoints are encoded in the cells' genome.

In order to evolve the organisms (and thus the robots' body plans), both systems are encoded in a genome composed of three independent chromosomes: The first chromosome encodes a set of indivisible proteins used to build the architecture of the GRN, the second chromosome encodes the dynamics parameters, and the third chromosome encodes the checkpoints necessary to generate the morphogen production system's Bézier surfaces.
In order to evaluate the complexity of possible morphologies generated by this model, we used a blind watchmaker with the aim to study the capacity of the GRN to inherently produce properties that other models usually include, such as symmetry or regularity [10]. We also want to verify that our system is able to generate a wide variety of morphologies, keeping in mind its possible evolution with an evolutionary algorithm. Then, we used a standard genetic algorithm to evolve the systems for locomotion and build the corresponding robots in reality. The next section presents these two methods of evolution.

4 Evolution of the Robots

4.1 Interactive Evolution

The blind watchmaker is an interactive evolutionary method first proposed in 1986 by Richard Dawkins [12]. He originally used this method to sustain the theory of natural evolution using a pedagogical model called *bionorphs*, fractal-like creatures generated with a small set of genes. This method gave birth more recently to the field of interactive evolution. Many applications are based on using this principle to solve various problems. For example, it has been used with genetic programming to generate realistic camouflage [35], and with HyperNEAT to generate 2D pictures [37] or 3D shapes [7].

In this work, we have used the blind watchmaker to evaluate the capacity of the developmental model to produce complex morphologies with emerging properties such as regularities or symmetries. To do so, we first generate nine random genotypes. The developmental model is then executed to generate the corresponding virtual robots. When the robots are created, they are dropped into the physics simulator at nine different points of the environment such that they cannot interact together. Figure 7 illustrates nine random robots evolving in the physics simulator. In this particular experiment, we have limited the number of developmental steps to 20 and the number of stem cells to 10 in order to control the scale of the morphologies. Later, these limits will be based on the actual availability of robotic cells or could be simulated by biologically plausible mechanisms.

During the first 5 s of the simulation, the robots are still, in order to stabilize their structure on the floor. Then, to make the robots move in the environment, an oscillator controls the movement of every hinge of the robots: They rotate clockwise for 3 s (the rotation angle of a hinge cannot be

![Figure 7. Nine robots evolve at the same time in the physics simulator. A 6-s phase oscillator generates the robots' movements. The user can select the robot chosen to evolve. The evolution only consists in mutating 10% of the genome. Spacing between robots has been reduced for this particular picture.](image-url)
above $-\pi/3$ and then counterclockwise for 3 s (within a $\pi/3$ limit). A third of the oscillators are inverted: When they should turn clockwise, an inverted hinge actually turns counterclockwise and vice versa. Although this method generates only simple movements, it is sufficient for visualizing the potential of the generated morphologies.

The user can navigate from one robot to another to observe its movement and its properties. The user can also select the robots chosen to evolve. The application then generates nine new robots by mutating 10% of the selected robot's genome. We have decided not to use the crossover operator to enhance the diversity of generated robots. For the same reason, the mutation rate has been deliberately chosen high. Here, only the 3D surfaces are used to generate the morphogen production profiles.

Figure 8 presents five robots obtained during the same run of the interactive evolutionary algorithm. These examples show the diversity of morphologies obtained.

The first observation we can make about these robots is the capacity of the regulatory network to generate robots of different sizes. The GRN is able to generate small robots (such as robot R4) as well as bigger ones (such as R1 or R2). This is mainly possible by using the differentiation action that stops the proliferation of the stem cells.

Moreover, interesting properties emerge during the evolution. After only nine generations, the robot R1 already presents some regularity in its morphology. By a regularity we mean a set of cells with the same organization (relative positioning and differentiation of the cells) that appears at different places in the organism. For example, the robot R1 is composed of three similar legs and two identical structures built with a double-row structure of cells (see the second picture of R1 in Figure 8). The robot R2 also has some regularities. It is composed of two long legs of the same size. The same observation can be made on robots R3 and R5 (two small legs on one side for the robot R5).

In some cases, these regularities can be very interesting: On the robot R4, they lead to a double row of H-cells and M-cells, which generates a double joint in the robot. This structure multiplies the strength of the resulting joint. Figure 9 presents the growth sequence of the robot R4. The double-row cell structure appears at the beginning of the growth, from step 1 to 3. The initial cell divides first.
Then both cells perform the same actions for two steps, before changing their behaviors, certainly because of a change in the morphogen profile of some input proteins at this particular point of the simulation. While the first cell produces the left part of the robot, the second cell quickly stops its development after a few more (parasitic) reorientations.

As proven by the blind watchmaker approach, some robots can be symmetric, such as robots R2 and R3 in Figure 8. This property is particularly interesting because it is usually expected in robots' morphologies. A symmetric machine will have more chances to be stable. The symmetry seems to be achieved using a similar method to the one that achieves regularity—but with a rotation before two equivalent sequences of actions are performed.

The use of a developmental model to generate robot body plans seems to be a promising approach. Some desirable properties such as regularity and symmetry are easily obtained with this system. As presented above, we have obtained these properties in a few generations with a very small population. This is not necessarily the case in works based on L-systems, which require many more generations of evolution. Moreover, the nonlinear production of the morphogens provides for much more creativity than a standard linear morphogen production regulated by the GRN.

The robot R4 has been built in reality. To do so, we followed step by step the plan provided by the developmental model and described in Figure 9. The oscillator behavior is loaded in an Arduino

<table>
<thead>
<tr>
<th>Step</th>
<th>Cell 1</th>
<th>Cell 2</th>
<th>Cell 3</th>
<th>Cell 4</th>
<th>Organism</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Birth</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SyD2</td>
<td>Birth</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TurnB</td>
<td>TurnB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SpD</td>
<td>SpD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SyD3</td>
<td>TurnB</td>
<td>Birth</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TurnB</td>
<td>TurnB</td>
<td>SyD4</td>
<td>Birth</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TurnB</td>
<td>Diff</td>
<td>Diff</td>
<td>TurnB</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>TurnB</td>
<td>-</td>
<td>-</td>
<td>Diff</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>SpD</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>AsD</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Diff</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9. Actions performed by the stem cells only. Each line corresponds to a growth step. Signification of the acronym: SyDx = symmetric division to create a new cell x; AsD = asymmetric division, SpD = special division, Diff = differentiation, and TurnB = turn to bottom.
board to actuate the joint as in simulation. The robot obtained has a comparable motion and is walking in the environment as presented in Figure 10.

### 4.2 Evolving for Locomotion with a Genetic Algorithm

The second experiment consists in evolving the developmental model in order to produce robots capable of locomotion. The developmental system is encoded in three independent chromosomes (one for the set of proteins of the GRN, one for dynamics coefficients of the GRN, and one for checkpoints of the morphogen production system), as presented previously.

This experiment uses the library OO-GA (Object-Oriented Genetic Algorithm) to evolve the robots. This library specifies an object-oriented encoding of the genes, which is well adapted to our object-oriented GRN. Each protein is encoded as a gene that can only be altered by mutation by the genetic algorithm, as defined in the GRN's model. In the work, the following parameters have been used to configure the genetic algorithm:

- Population size: 200
- Crossover rate: 70%
- Mutation rate: 10%
- Selection: 3-individual tournament with elitism
- Minimum GRN size: 9 output proteins + 5 input and/or regulatory proteins
- Maximum GRN size: 9 output proteins + 50 input and/or regulatory proteins.

In order to evolve different kinds of robots, two different fitness functions have been used. First, the locomotion is calculated as the distance traveled by the center of mass of the robot in the physics simulator. The initial point is taken after a stabilization period of 5 s to avoid folding creatures. The final point is taken after 40 s more of simulation. An optional term can be added to describe the complexity of the morphological conception. To do so, we use the number of kinds of actions used during the developmental process. Cells can choose among eight actions (three divisions, four rotations, and one differentiation). If the developmental process requires only three of the eight possible actions, the complexity score is 3/8. Finally, the second optional term of the fitness function evaluates the size of the creature. Due to the availability of real robotic blocks, the maximum size is 48 blocks (16 noops, 16 hinges, and 16 motors). The size score only consists of the number of blocks used in the final morphology. In order to produce bigger robots, we have decided to maximize this size. In summary, the fitness function is given by one of the following equations:

\[
\text{fit}_a = \text{distance with distance} \in [0, 2]
\]

\[
\text{fit}_b = \text{distance} + \frac{\text{actions}\_\text{used}}{8} + \frac{\text{max}\_\text{size}}{\text{max}\_\text{size}}
\]

Whereas the first fitness function generally leads to smaller robots, the second one first maximizes the size of the robots (usually easier to evolve than the locomotion) and then optimizes the positions of the hinges.
Figure 11 presents some of the robots obtained by evolution. Again, many of them present interesting properties such as regularities, in particular Figure 11a, where the same identical structure, composed of two passive blocks, a hinge, and a motor, is produced three times in a row. This feature is perfectly visible in the developmental process of this robot, where only one stem cell is acting, as presented in Figure 12. We can observe an almost perfect repetition of the sequence of action achieved by the stem cell to produce the robot. It is also visible in Figure 13, which represents the evolution of the concentration of the output proteins in the gene regulatory network that controls the cell. The GRN cyclically produces the proteins, corresponding to the stem cell’s action, that make the regularities possible.

The robot in Figure 11b has been evolved for locomotion, diversity, and size, whereas the one in Figure 11a was only evolved for locomotion. The aim of adding the diversity and the size to the fitness function is to generate bigger creatures in order to test the scalability of the approach. Indeed, not adding the size of the robot to the fitness function usually leads to small robots (e.g., the worm presented in Figure 11a), because they are lighter and thus easier to move. The robot in Figure 11b also presents regularities. We can first observe two almost identical legs at the rear. A double-hinge structure also appears in the front of the robot. Moreover, this creature presents another interesting property: It has a symmetry plane orthogonal to its development plane. Once again, this symmetry is not perfect (as is the case in many living beings), but is sufficient to obtain efficient locomotion.

Finally, whereas previous robots have been obtained with 3D morphogen profiles, Figure 11c and d present two creatures obtained by using 4D morphogen profiles. With the 4D encoding of the morphogens, only 216 checkpoints are necessary to generate all the concentration values of the input proteins. As shown with the robots in Figure 11c,d, the complexity of the creatures obtained

---

1 Videos can be found at the following addresses:
(a) http://demo.cs.brandeis.edu/video/worm_simu.mov
(b) http://demo.cs.brandeis.edu/video/turtle_simu.mov
(c) http://demo.cs.brandeis.edu/video/runner_simu.mov
(d) http://demo.cs.brandeis.edu/video/the_thing_simu.mov
is the same. The same property of regularities and symmetry is also notable. To measure the effect of the use of a 4D meta-surface instead of 64 3D surfaces on the fitness function, we have run the genetic algorithm 15 times with both systems. Figure 14 presents the average best fitness scores obtained during the evolution. The quality of the solutions obtained by the genetic algorithm is not affected by the 4D morphogen production system. Indeed, the average best fitnesses are equal after 200 generations, which is sufficient for the genetic algorithm to converge. However, the convergence curve of the 4D morphogen production system is below that of the 3D morphogen production system during the whole evolution. This can be explained by the complexity implied by the use of a fourth dimension. In conclusion, this experimentation shows that the 4D morphogen production systems generate sufficient complexity to enhance the GRN dynamics and keep the evolution feasible. Moreover, the relatively small number of checkpoints is a big advantage of the 4D morphogen profiles in comparison with the 3D ones.

The robots presented in Figure 11 have been built in reality. They are presented in Figure 15. The same oscillator has been uploaded in these robots and in the evolved virtual creatures so that they can

Figure 12. Actions executed by the stem cell of the C. elegans. A video of the growth can be downloaded at the following address: demo.cs.brandeis.edu/videoworm_growth.mov

Figure 13. Evolution of the GRN’s output protein concentrations through the development steps. A perfect oscillation of the GRN is observable that produces the regularities in the robot morphology.

subito e.V. licensed customer copy supplied and printed for INIST-CNRS (S104X00627E)
Figure 14. Influence of the morphogen production systems on the evolution of the creatures: best fitness values per generation, averaged on 15 runs. Error bars represent first and third quartiles.

express the same behavior. The motion of the real robots is close to what we can observe in simulation. However, big robots such as those in Figure 11b and c are slower than their simulated counterparts. This is due to lack of strength of the servomotors used in the motor blocks. In Figure 15, this is particularly observable on the front leg of the robot in part b. This leg is powered by only one servomotor, which has to move the whole top part of the robot. In reality, this leg is motionless because of the robot’s weight. In simulation, the motor can move it slightly due to the flexibility of the hinge joints.

4.3 Influence of the Robot’s Size on the Fitness Function

In order to evaluate the influence of the robot’s size on its motion capacity, we have optimized in simulation robots of various sizes, without taking into account the availability of the real blocks. With this aim in mind, we have used the 4D morphogen production profile and the genetic algorithm based on the second fitness (which takes into account the locomotion, the complexity, and the size of the robots) in order to produce robots with a target size and with maximum locomotion capacities. Thus, we ran 16 evolutions with seven different size limits. The size limit is managed by limiting the number of each cell type to 4, 8, 12, 16, 24, 32, or 48 per type. Thanks to the fitness size objective, the generated robots usually use all the blocks available and are composed of 12 to 144 blocks.

Figure 16 shows the results of the evolution for each size limit. The optimal robot size is between 8 and 12, and the evolution with these size limits always produces better results (in the electronic version, crossed blue and starred green curves). Increasing the number of blocks reduces the efficiency of the robot’s motion. This confirms the preliminary conclusion proposed in the previous section, where bigger robots were less efficient than expected for motion. Here, the motion reduces from about 0.8 m in 40 s for small robots (with 8 and 12 blocks per type) to 0.7 m in 40 s for robots with 16 blocks per type, and up to about 0.55 m in 40 s for bigger robots. In our opinion, the motors are not powerful enough to move the large robot structure. However, this reduction is acceptable: The robots are still able to move forward even when they are composed of 144 blocks (48 blocks per type). Moreover, between 24 and 48 blocks per type, the results are very comparable: The robot motion deterioration seems to be stabilized when it reaches this size limit.

2 Videos are available at the following addresses:
(a) http://demo.cs.brandeis.edu/video/worm_real.m4v
(b) http://demo.cs.brandeis.edu/video/turtle_real.mov
(c) http://demo.cs.brandeis.edu/video/runner_real.mov
(d) http://demo.cs.brandeis.edu/video/the_thing_real.mov
Another interesting observation concerns the robots with the 4-block-per-type limit. Even if they are smaller robots, they usually generate worse walkers, with an average distance traveled of about 0.6 m. In our opinion, even though they are lighter, the robots are too small and cannot generate complex enough morphology to produce efficient robots.

5 Discussion

We have shown in this article that a cell-based developmental model can produce the morphologies of modular robots and inherently induces symmetries and regularities. Self-repairing is another key property often desired in modular robots. Many research teams have explored this property since the birth of modular robotics, and many algorithms have been proposed to obtain it [6, 23, 31, 41].
Having this feature inherently supported by the morphology design algorithm is a strong advantage of the approach. Developmental models are also known to create inherently self-repairing developed structures, in other words, to recover the function and/or morphology from a faulty cell [9, 27, 30, 40]. Because this model is based on the same principles, our developmental model should have the same property. A full study would be necessary to evaluate the capacity of this model to recover from faulty modules, but we expect a local reorganization of the morphology. Of course, because the modules are by design unable to self-reorganize, a new assembly stage will be necessary, but the developmental model might once again provide the modification steps to achieve reconfiguration.

The current developmental model produces usable blueprints to assemble the robots—manually at present, but, as a long-term objective, machinery could be able to use these step-by-step blueprints to assemble and disassemble the robot automatically. The idea is to produce on-the-fly robots whose morphologies and behaviors are optimized for a given function (possibly by including more types of modules—grippers, wheels, tools, etc.—still managed by cell specialization) and assemble them with an automatic process. Once assembled, the robots would perform their missions before coming back to the base and being disassembled or modified for another mission. Figure 17 shows a schematic of this process. This way to proceed is cheaper because multiple kinds of robots can be produced with simple and cheap modules. In comparison, reconfigurable robots need complex modules with linking and unlinking systems, sometimes leading to failure. The assembly and disassembly machinery could be off-the-shelf systems such as robotic arms and conveyor belts.

6 Conclusion and Perspectives

This article presents an innovative system to generate robot body plans. It is based on a developmental model in which cells are controlled by a gene regulatory network. We have based our GRN model on Banzhaf’s by modifying the encoding of the GRN into a chromosome without any noncoding DNA and by modifying its dynamics. The resulting GRN is easier to evolve and is more adapted to a computational purpose. Instead of using morphogens as in most developmental models, we have

![Diagram](image_url)

**Figure 17.** Using a developmental model could lead to an automatic assembly and disassembly process of robots in which the only human interaction consists of describing the robot’s objectives. An evolutionary algorithm would optimize the robot morphology and behavior through the embryogenesis process. This later would produce the blueprint used by an assembly robot that builds the final robot using pre-manufactured blocks. Once assembled, the modular robot would achieve its mission in the environment and return to the base, where it would be disassembled until the next mission.
decided to provide inputs to the GRN using a morphogen production system that produces different production profiles according to the cell identifier, the input protein identifier, and the development step.

We first produced various virtual robots by contemporarily evolving the GRN and the morphogen production system with a blind watchmaker. The resulting robots have various sizes, from small to very large, and two interesting properties emerge from the systems: Many creatures have regularities in their morphologies, such as legs or double joints, and some of them are symmetric. This is possible because of the oscillations, which are a well-known characteristic of GRN systems. Running a standard genetic algorithm on our GRNs has also produced interesting results, showing various kinds of robots capable of locomotion in their environment.

Some of the virtual robots obtained by evolution, both interactive and automatic, have been built in reality. The assembly of these robots is fast and simple thanks to the blueprints provided by the developmental model. They also present the same behavior as their simulated counterparts. The distances they travel in simulation and in the real world are very close.

We have only begun to crack open the interface between GRN evolution and robot development. The joints’ controllers could be improved. They currently consist of oscillators moving the joints clockwise and then counterclockwise. Two main solutions are currently being considered to improve these controllers. On the one hand, we can imagine that the same GRN also controls the joints when the growth of the robot is finished. Schramm et al. have already successfully used this method [36]. Another possibility is to plug a neural network or an equivalent method into the joints. Here, two approaches are possible: Either the neural network and the developmental model can be evoked at the same time, or the controller can be evoked after the evolution of the morphology.

Secondly, in order to generate efficient robots, the fitness function has to be improved, as the field needs to go beyond linear movement in a simulated environment. To have other kinds of morphologies, we have included simple measurement of complexity and size. The measures produce interesting results, but should be improved in order to overcome the motor strength problem. It could be interesting to add a symmetry score to the fitness function and use a more developed complexity fitness. Increasing the complexity of the environment could also lead to more complex morphologies and behaviors. However, complexifying the objective of the robots is not an easy task. Most current work on modular and evolutionary robotics still focuses on locomotion. Our earliest work in evolution of Lego structures relied on handwritten fitness functions for each objective. We feel that in order to address this problem, hybrid evolution could be used, alternating interactive and standard evolution. On the one hand, the use of the blind watchmaker helped us to point out the capacity of the developmental model to inherently generate wanted properties such as symmetry and regularities. However, it requires a lot of work by the user to evaluate each robot. On the other hand, the genetic algorithm was efficient in evolving robots with a given simple objective. To increase the complexity of this objective, user expertise could be required at different steps of the evolution in order to lead the genetic algorithm to better solutions, to cut irrelevant branches of the tree, or to reduce plateauing of the genetic algorithms. This hybridization will of course require more involvement of the user to evaluate some of the robots, but less than in a fully interactive evolution.

Finally, the main challenge of this approach concerns the size and the diversity of the robotic blocks. Because of their size (cubes of 2-in.—5-cm—side) and weight (between 300 and 700 g), the robots obtained do not scale up as much as we would like to. The size of the blocks is constrained by the size of the servomotor used in this work. However, with the miniaturization of mechanics currently in progress, we can imagine that smaller and more powerful motors will be available in the future. Two ways of miniaturization are currently possible. On the one hand, servomotors tend to be smaller and stronger. Here, we use Futaba S3102 metal gears because they are inexpensive and are available in our lab, but brushless micromotors such as the Faulhaber 0515B are now more and more common. Their sizes would be perfectly adapted to our robots, but their torques are still not adequate to our use. On the other hand, artificial muscles are used nowadays for artificial jellyfish propulsion [32]. We could have such a muscle directly included in the hinge blocks and leave out the motor blocks entirely. This will allow us to miniaturize our blocks and thus our robots. Having blocks a few millimeters on a side could make possible robots composed of thousands of blocks, which would be closer to living organisms, which are composed of billions of cells.
Acknowledgments

The 3D printer used for this work was supplied by DARPA under grant W911 NF0710639. Partial support for this work was provided by the NSF under the Creative-JT program, grant 0757452. The OCGA library has been developed by the VORTEX team, University of Toulouse, CNRS, UMR 5505.

References


34. Reif, T. (1999). Dynamics of gene expression in an artificial genome—implications for biological and artificial ontogeny. In D. Floresano et al. (Eds.), *Advances in Artificial Life*.


Appendix: Block Blueprints

See Figure 18.

Figure 18. Block Blueprints.