A Developmental Model for the Evolution of Artificial Neural Networks

J. C. Astor¤
C. Adami
Computation and Neural Systems
and Kellogg Radiation Laboratory
California Institute of Technology
Pasadena, CA 91125, USA

Keywords
artificial neural network, (distributed) genetic algorithms, neurogenesis, gene expression, artificial chemistry, artificial life, Java

Abstract We present a model of decentralized growth and development for artificial neural networks (ANNs), inspired by developmental biology and the physiology of nervous systems. In this model, each individual artificial neuron is an autonomous unit whose behavior is determined only by the genetic information it harbors and local concentrations of substrates. The chemicals and substrates, in turn, are modeled by a simple artificial chemistry. While the system is designed to allow for the evolution of complex networks, we demonstrate the power of the artificial chemistry by analyzing engineered (handwritten) genomes that lead to the growth of simple networks with behaviors known from physiology. To evolve more complex structures, a Java-based, platform-independent, asynchronous, distributed genetic algorithm (GA) has been implemented that allows users to participate in evolutionary experiments via the World Wide Web.

1 Introduction

Ever since the birth of computational neuroscience with the introduction of the abstract neuron by McCulloch and Pitts in 1943 [19], a widening gap has separated the mathematical modeling of neural processing—inspired by Turing's notions of universal computation—and the physiology of real biological neurons and the networks they form. The McCulloch–Pitts neuron marked the beginning of over half a century of research in the field of artificial neural networks (ANNs) and computational neuroscience. Although the goals of mathematical modelers and neuroscientists roughly coincide, the gap between the approaches is growing mainly because they differ in focus. Whereas neuroscientists search for a complete understanding of neurophysiological phenomena, the effort in the field of ANNs focuses mostly on finding better tools to engineer more powerful ANNs. The current state of affairs reflects this dichotomy: Neurophysiological simulation test beds [4] cannot solve engineering problems, and sophisticated ANN models [7, 11] do not explain the miracle of biological information processing.

Compared to real nervous systems, classical ANN models are seriously falling short owing to the fact that they are engineered to solve particular classification problems and analyzed according to standard theory based mainly on statistics and global error reduction. As such, they can hardly be considered universal. Rather, such models *define* the network architecture a priori, which is in most cases a fixed structure of homogeneous computation units.

¤ Present address: Siemens Business Services GmbH & Co. OHG, Otto-Hahn-Ring 6, 81739 Munich, Germany

Models that support problem-dependent network changes during simulation are few and far between, but see [7, 9]. Other approaches try to shape networks for a particular problem by evolving ANNs either directly [1], or indirectly via a growth process [10, 13]. The latter allows for more efficient—and adaptable—storage of structure information. Indeed, the idea that development is the key to the generation of complex systems slowly seems to be getting a hold [14].

Some research has [17, 21, 22] included a kind of artificial chemistry that allows a more natural and realistic development. An important step was taken by Fleischer [8], who recognized the importance of development and morphology in natural neural networks and succeeded in growing networks with interesting topology, morphology, and function within an artificial computational chemistry based on difference equations. Still, in these models neurons are unevolvable homogeneous structures in a more or less fixed architecture that appears to limit their relevance to natural nervous systems. This situation is motivation enough to design, implement, and evaluate a more flexible and evolvable ANN model, with the goal of shrinking the gap between models based on neurophysiology and engineered ANNs. The approach we take here is rooted in the artificial life paradigm: to start with a low-level description inspired (if not directly copied) from the biological archetype, and to design the system in such a way that the complex higher-order structures can emerge without unduly constraining their evolution [2, 16].

The developmental aspect has often been ignored or considered half-heartedly. Also, the heterogeneity of neurons was never taken into account in a flexible way. It is therefore possible that more complex and universal information-processing structures can be grown from a model that, at a minimum, follows the four basic principles of molecular and evolutionary biology: coding, development, locality, and heterogeneity, discussed below. While models for ANNs currently exist that implement a selection of them, the inclusion of all four opens the possibility that, given enough evolutionary time, novel and powerful ANN structures can emerge that are closer to the natural nervous systems we endeavor to understand. The four principles thus are

- *Coding.* The model should encode networks in such way that evolutionary principles can be applied.
- *Development.* The model should be capable of growing a network by a completely decentralized growth process, based *exclusively* on the cell and its interactions.
- *Locality.* Each neuron must act autonomously and be determined only by its genetic code and the state of its *local* environment.
- *Heterogeneity.* The model must have the capability to describe different, *heterogeneous* neurons in the same network.

While eschewing the simulation of every aspect of cellular biochemistry (thus retaining a certain number of logical abstractions in the artificial neuron), the adherence to the fundamental tenets of molecular and evolutionary principles—albeit in an artificial medium—represents the most promising unexplored avenue in the search for intelligent information-processing structures. One of the key features of a model implementing the above principles will be the absence of explicit activation functions, learning rules, or connection structures. Rather, such characteristics should emerge in the adaptive process.

The model presented here takes certain levels of our physical world into account and is endowed with a kind of artificial physics and biochemistry. Applied to the substrates, this allows the simulation of local gene expression in artificial neural cells that finally results in information-processing structures.

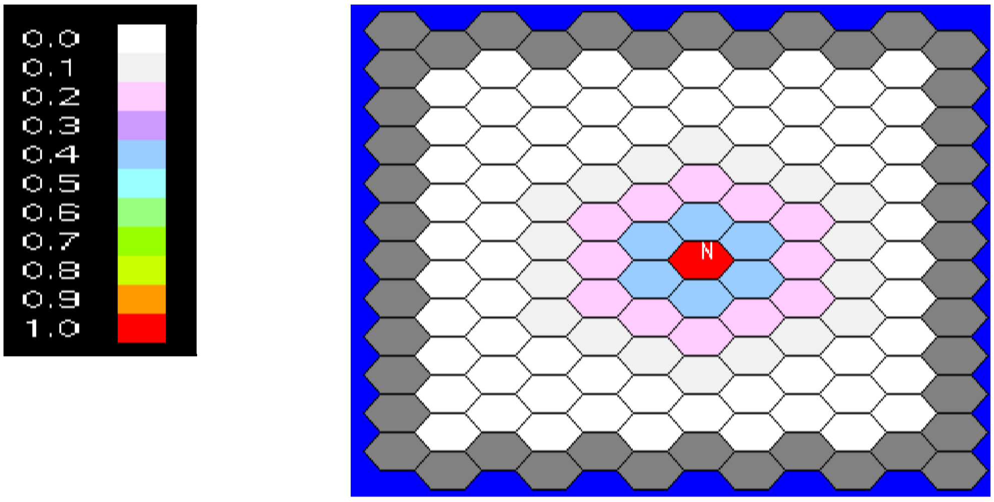


Plate 1. Hexagonal grid with boundary elements. Diffusion occurs from a local concentration peak at grid element *N*.

Gene expression is the main cause for the behavior of neurons in this model. Without gene expression, neurons do not become excited or inhibit; they do not split or grow dendritic or axonal extensions, nor do they change synaptic weights. Thus, this behavior must be encoded in the genome of the somatic cell from which the network grows. While each neural cell has a genotype, we choose to have certain physiological traits (like the continuous production of a cell-specific diffusive protein and the transmission of cell stimulation) "hardwired" (i.e., implicit and not evolvable). Also, special neurons called actuators and sensor cells are used as interface to the outside world, which are static, unevolvable, and devoid of any genetic information.

In the following section we introduce the model, its artificial physics and biochemistry, as well as the genetic programming language that allows for growth, development, gene regulation, and evolution. We present hand-written example genomes in Section 3, discuss their structure, and analyze the physiology of the networks they give rise to. Section 4 discusses the distributed genetic algorithm (GA), its genetic operators and the client–server structure used to set up worldwide evolution experiments. We close with a discussion of the present status of the system as well as future developments.

2 Model

2.1 Artificial Physics and Chemistry

The model is defined on a kind of "tissue" on which information-processing structures can grow. The arrangement of spatial units in this world must meet the principle of locality in an appropriate way, implying that neighboring locations always have the same distance to each other. The simplest geometry that achieves this goal is a hexagonal grid (Plate 1). Each hexagon harbors certain concentrations of substrates, measured as a percentage value of saturation between 0.0 and 1.0. As all sites are equidistant in the hexagonal lattice, the diffusion of substrate k in cell i can be modeled discretely as

$$C_{ik}(t+1) = \frac{D}{6}\sum_{j=1}^{6}\left(C_{ik}(t) - C_{N_{i,j}k}(t)\right), \qquad (1)$$

where $C_{ik}(t)$ is the concentration of substrate k in site i, D is a diffusion coefficient ($D < 0.5$ to avoid substrate oscillation), and $N_{i,j}$ represents the jth neighbor of grid element i. Accordingly, a local concentration of substrate will diffuse through the tissue under conservation of mass. The tissue itself is surrounded by special boundary elements that absorb substrates (Plate 1), thus modeling diffusion in infinite space. Note that hexagons are sites that *may* harbor neuron cells but otherwise only represent a convenient equidistant discretization of space to facilitate the distribution of chemicals via diffusion.

Each hexagon of the discretized space—whether it harbors a neuron cell or not—contains concentrations of substrates. Substrate has to be produced by (neuron) cells and does not just exist a priori. The artificial biochemistry model distinguishes four different classes of substrates: external, internal, cell-type proteins, and neurotransmitters (see below). Each substrate has an identification (e.g., `EP0`, `IP0`, `eNT`) and belongs to a particular class of substrates that defines its properties.

External proteins are not constrained by the confines of the cell in which they are produced. They undergo diffusion and spread over the lattice according to Equation (1). As such, they can be used as indirect messenger substrates just like hormones in biochemistry. At each grid element, the concentration of an external protein can be measured. By comparing all concentration values of neighboring cells, a local gradient can be calculated pointing in the direction from which the external protein is emanating.

Internal proteins are non-diffusive and, as they cannot cross the cell membrane, stay inside the cell in which they have been produced. Internal proteins can still play the role of internal messengers, however. After production, they decay with a fixed rate at every time step.

Cell-type proteins are external proteins produced by every neuron and are specific to its *type*. The protein's production is part of the implicit behavior of neurons. Cell-type proteins are diffusive just like external proteins.

Neurotransmitters are a special type of internal protein that play an important role in the direct information exchange between neurons.

2.2 Artificial Cell

The tissue shown in Figure 1 can harbor different cell types and connections. A neural cell (neuron) is represented by the hexagon it occupies, including its connections (dendritic or axonic) to other cells. Each particular neuron belongs to one of three classes: *actuator cells*, *sensor cells*, or *common neurons*. Actuator and sensor cells build the interface to a simulated outside environment to which the network adapts and on which it computes. Compared to classical ANN models, actuator and sensor cells replace the input and output layer of the network, while the common neurons (subsequently just called neurons) represent hidden neurons.

Neurons of all three types can be excited to a real-valued level between 0.0 and 1.0 and can take part in the information transfer via dendritic or axonal connections. Each type of cell is also characterized by its own cell-type protein, which it produces continuously at a certain rate. These cell-type proteins diffuse over the tissue (Figure 1) and can signal cell existence to other cells. They can be compared to the role of *growth factors* in the development of real nervous systems. Even though we know from biology that not every cell type produces its own diffusive messenger substrate, membrane proteins exist that make the different cell types "chemically" distinguishable.

2.2.1 Neurons

A neuron's function is mostly determined by its hereditary information and the local concentrations of substrates. This aspect will be explained in depth in Section 2.3. Even so, as mentioned before, a neuron's artificial physiology is not completely malleable; a certain number of features are hard-coded. For example, the aforementioned cell-type protein production is fixed. Furthermore, a neuron produces at every time step (i.e., simulation cycle) specific amounts of a particular neurotransmitter and "injects" it via its axons into cells to which it is connected. Either each neuron can use its default neurotransmitter called `eNT`, or gene expression can define the usage of another neurotransmitter. However, a particular neuron uses only one type of neurotransmitter (*Dale's Law*). The specific amount $[\mathtt{NT}_x]_{ij}$ of neurotransmitter $\mathtt{NT}_x$ injected at the connection between neuron i and neuron j at time step t is

$$[\mathtt{NT}_x]_{ij} \; \mathsf{D} \; a_i, \tag{2}$$

where a_i is the current activation value $(0.0 \cdot a_i \cdot 1.0)$ of neuron i at time step t, and $\mathtt{NT}_x$ stands for the specific neurotransmitter chosen. Consequently, neuron j receives a flux of neurotransmitter from one of its dendrites. Each dendrite has a neurotransmitter-specific weight (an amplification factor) w_{ij}. As we shall see later, the manipulation of weights is in response to gene expression only, rather than being defined a priori.

At each time step, thus, a neuron harbors certain amounts of (possibly *different*) neurotransmitters received from weighted dendritic influx. This is comparable to the weighted sum of inputs in classical ANNs. However, unlike in standard ANN models, this *does not imply* an automatic activation stimulation (i.e., firing) of the neuron, unless such behavior is explicitly encoded in the neuron's genome. Thus, there is no implicit activation function or learning rule. Weights remain always at the amplification value 1.0 (their initial value) if not modified through gene expression.

2.2.2 Actuators and Sensor Cells

Actuators and sensor cells do not carry genetic information; they are used solely as interfaces to the environment (input–output units). They represent *sources* and *sinks* of signal. Consequently, their behavior is hard-wired (i.e., implicit) and does not depend on gene expression.

Sensor cells can simply be set to a certain activation level. This is usually done every time step in accordance to the signal received from the environment. Sensor cells provide information about their activation just like neurons do: They inject the appropriate amounts of neurotransmitter in response to their activation into all neuron cells to which they are connected. As sensor cells do not carry genes, they always use the default neurotransmitter `eNT`.

At every time step, actuators receive a flux of neurotransmitters from their dendritic synapses. The weights of their dendrites cannot be modified (again as they are not subject to gene regulation) and remain always at 1.0 (the initial value). As neurotransmitters cannot be produced through gene expression, the amount of neurotransmitter a cell harbors at a particular time step must be due to injection by pre-synaptic neurons, and therefore neurotransmitter concentration is an input signal. Each actuator k becomes stimulated at time step t according to

$$a_k(t) \; \mathsf{D} \sum_{j,x} [\mathtt{NT}_x]_j(t), \tag{3}$$

where $[\mathtt{NT}_x]_j(t)$ is the dendritic influx concentration of neurotransmitter $\mathtt{NT}_x$ from neuron j at time t. Of course, as the activation value always has to be in the range `{0.0;1.0}`, $a_k(t)$ is taken modulo 1.

Table 1 summarizes the cell types and how they interact with other computational elements used in the model.

Table 1. Features of different cell types in an artificial tissue: A: participates in diffusion; B: can be stimulated; C: behavior depends on gene expression; D: can have axons; E: can have dendrites; F: produces a diffusible cell-type protein.

Type	A	B	C	D	E	F
Neuron	x	x	x	x	x	x
Sensor	x	x		x		x
Actuator	x	x			x	x
Grid element	x					
Boundary element						

2.3 Genetic Code and Gene Regulation

Perhaps the most important aspect of this neurogenesis model is not that form and function are determined by genes, but that these genes are regulated and regulate each other. One of the more profound insights in evolutionary biology seems to be that starting with the late Phanerozoic, only few proteins and enzymes have been created de novo while evolution has proceeded at a torrid pace [5, 6]. It can thus be speculated that from a certain level of complexity on, evolution proceeds mainly by adjusting the mechanisms of gene regulation. In order to tap this enormous potential, the regulation of genes (by means of activation and repression of expression) is central to this model.

Each neural cell carries a genome that encodes its behavior. Genomes consist of *genes* that can be viewed as a genetic program that can either be executed (*expressed*) or not. The genetic program is written in a customized genetic language that we begin to describe below, that is designed to be both flexible and evolvable, and that permits regulation. Regulation is achieved via *gene conditions* that determine the level of expression of a gene (e.g., the rate of production of any type of protein or substrate).

2.3.1 Conditional Regulation

A gene condition is a combination of several condition *atoms*, usually related to local concentrations of substrates. The expression of a gene (an ensemble of expression commands) can result in different behaviors such as the production of a protein, cell division, axon/dendrite growth, cell stimulation, and so forth. Thus, gene conditions model the influence of external concentrations on the expression level of the gene, that is, they model activation and suppression sites. Figure 1 illustrates the structure of this genetic code.

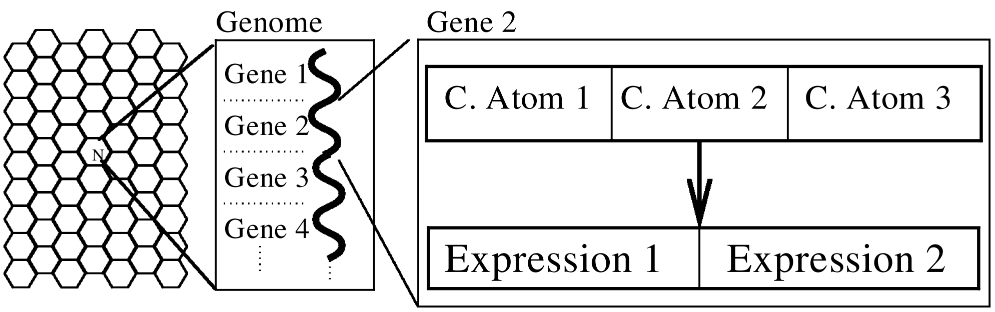


Figure 1. Genetic structure of neural cells. Gene conditions (consisting of condition atoms related to substrates) trigger gene expression leading to cell division, axon/dendrite growth, substrate production, stimulation, and so forth.

Table 2. Condition atoms with veto power. Here, CTPx stands for the cell-type protein CTPx, while {PTx} is the current local concentration of protein PTx. CTPx can be replaced by any kind of cell-type protein, whereas PTx can be any type of protein (either cell-type, internal, external, or neurotransmitter protein).

Type	Represses its gene if ...
SUP{CTPx}	cell is not of type CTPx
NSUP{CTPx}	cell is of type CTPx
ANY{PTx}	{PTx} D 0
NNY{PTx}	{PTx} Ð 0

Table 3. Evaluative condition atoms calculate expression levels from an initial condition Θ and the current local concentration of assigned protein (here: PTx). {PTx} can stand for any kind of protein (either cell-type, internal, external, or neurotransmitter protein). To handle values outside the limits {0.0;1.0}, $R_0^1()$ takes the value modulus 1, to keep it in the desired range.

Type	Evaluation value Φ	
SUB{PTx}	Φ(Θ, SUB[PTx])	D R_0^1(Θ ¡ {PTx})
ADD{PTx}	Φ(Θ, ADD[PTx])	D R_0^1(Θ C {PTx})
MUL{PTx}	Φ(Θ, MUL[PTx])	D Θ¤{PTx}
AND{PTx}	Φ(Θ, AND[PTx])	D min(Θ, {PTx})
NAND{PTx}	Φ(Θ, NAND[PTx])	D 1 ¡ min(Θ, {PTx})
OR{PTx}	Φ(Θ, OR[PTx])	D max(Θ, {PTx})
NOR{PTx}	Φ(Θ, NOR[PTx])	D 1 ¡ max(Θ, {PTx})
NOC	Φ(Θ, NOC)	D Θ; the neutral condition
NNY{PTx}	Φ(Θ, NNY[PTx])	D Θ, if {PTx} D 0
ANY{PTx}	Φ(Θ, ANY[PTx])	D Θ, if {PTx} Ð 0

There are two different classes of condition atoms: *repressive* and *evaluative* atoms.

Repressive condition atoms. Members of this class can completely silence the expression of genes to which they are assigned by means of a Boolean condition that—if it turns out to be false—vetoes the expression of the associated gene regardless of other condition atoms in the same gene condition. Examples of repressive condition atoms can be found in Table 2.

Evaluative condition atoms. Such condition atoms always lead to real-valued results in the range of {0.0;1.0}. To evaluate a condition atom of this type, two parameters are necessary: an initial condition[1] and the *local concentration* of a particular protein, both in the range {0.0;1.0}. An evaluative condition atom computes its result using both parameters according to its type. Table 3 gives an overview of all such condition atoms.

Let C be a vector of condition atoms of length n, and a_i a particular condition atom at position i, $1 < i \cdot n$:

$$C \text{ D } (a_1, a_2, \ldots, a_n). \quad (4)$$

1 Initial conditions will be explained below.

Suppose that m of the n atoms are evaluative condition atoms, and l of the n atoms are repressive condition atoms.[2] C can be written as two vectors:

$$C_2 \text{ D } (c_1, c_2, \ldots, c_{n \text{ ¡ } m}), \tag{5}$$

$$C_1 \text{ D } (b_1, b_2, \ldots, b_m), \tag{6}$$

where C_2 contains all repressive condition atoms of C and C_1 is the vector of all evaluative condition atoms in C. Now, C can be evaluated in two steps:

1. Check if C_2 contains an atom that exercises its veto. If so, stop the evaluation of this gene condition and do not express its assigned gene at all. Otherwise,
2. Evaluate C_1 and use the result as the overall condition value.

Assume that none of the repressive condition atoms exercised a veto during step 1. Now we have to follow step 2 to evaluate the overall condition value of C. Consider $\Phi(\Theta, b_i)$ as the evaluation function (Table 3) of condition atom b_i under the assumption that Θ is an *initial condition.* As the condition is evaluated *in the order* of the evaluative condition atoms $b_1, \ldots, b_m$, the overall condition value Φ_C of C can be computed as

$$\Phi_C \text{ D } \Phi(\ldots\Phi(\Phi(\Theta_0, b_1), b_2), \ldots, b_m), \tag{7}$$

where Θ_0 is the initial condition of the first condition atom in C_1. Thus, the evaluation result of each condition atom is used as the initial condition for the evaluation function of the following condition atom in the order of their appearance. This recursive rule is obviously not applicable to the *first* condition atom b_1, for which the initial operand Θ_0 of its evaluation function is fixed as:

$$\Theta_0 \text{ D } \begin{cases} 1, & \text{if } b_1 \text{ is of type MUL, AND, NAND, SUB, ANY, NNY} \\ 0, & \text{if } b_1 \text{ is of type ADD, OR, NOR} \end{cases} \tag{8}$$

To further clarify gene conditioning, consider Figure 2. In this figure, let the local concentrations of `IP0, EP0, EP1, EP2, CPT` be f`0.3, 0.5, 0.6, 0.1, 0.9`g. Furthermore, suppose that this example takes place in a cell characterized by the cell-type protein `CPT`. Figure 2 then shows how the regulatory cascade gives rise to a final expression level of 0.18: the final condition value.

It is obvious that gene conditions that use more than one different type of evaluative condition atoms are *not* commutative: In Figure 2 the order of condition atoms influences the evaluation result, whereas, for example, in a gene condition consisting of repressive condition atoms only and the `ADD` condition atom, the order is of no relevance.

2.3.2 Gene Expression

We now discuss the second element that makes up the gene: the assigned expression. This is in fact a set of several *independent* expression commands in no particular order. Evaluation of the condition leads to an overall condition value between 0.0 and 1.0. This value can then be applied to each single expression command for which it has a particular meaning.

2 Note: $l \text{ C } m > n$, if C contains any condition atoms of type `ANY` or `NNY`. This is because `ANY` and `NNY` share the functionality of *repressive* and *evaluative* condition atoms (see Tables 2 and 3).

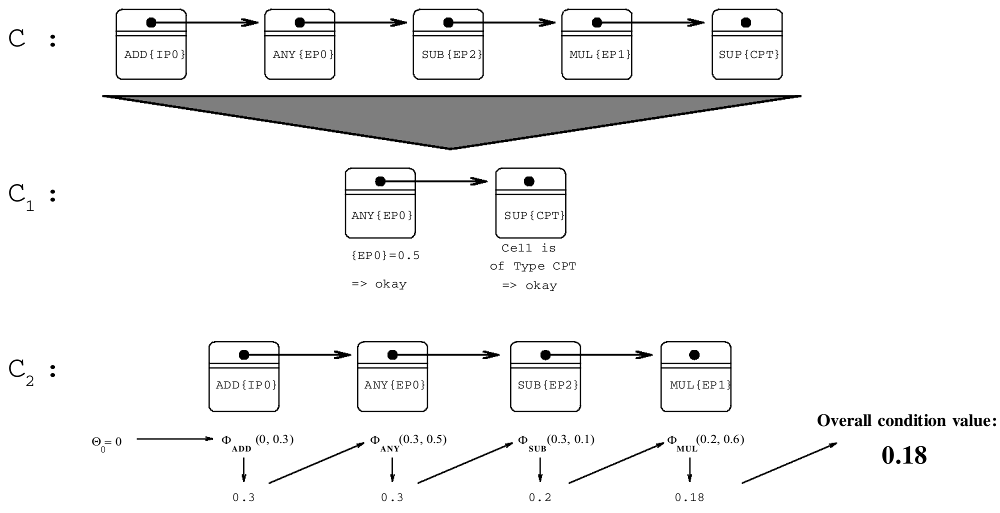


Figure 2. A gene condition C can be represented as two chains of subconditions: C_1 contains the repressive condition atoms, C_2 contains all evaluative condition atoms. First, C_1 is checked for condition atoms leading to complete gene repression. If there are none—like in this example—C_2 is evaluated numerically and determines the overall condition value.

In the following, we discuss every expression command in detail. An overview is given in Table 4.

2.3.3 Developmental Commands

To encode the growth process, special processes are necessary. Developmental commands control the construction of new connections between neurons, as well as cell division.

Table 4. Overview of expression commands. Growing axons/dendrites follow the substrate gradient until a local maximum is reached then connect to the cell if one exists at that location. Strengthening/weakening is a percentage increase/decrease of connection weights, determined by the product of the last neurotransmitter (here NTx) influx at each connection and the value of the gene condition. The cell-type protein assigned to a cell division command determines the type of the future offspring cell. In this example, the offspring will be of type CTPx and therefore produce cell-type protein CTPx continuously. "Relaxing" weights means bringing the specific weights for neurotransmitter NTx influx slightly closer to the initial value 1.0.

Expression	Command description	Influence of condition value
PRD{XY}	produce substrate XY	production quantity
GDR{XY}	grow dendrite following gradient of XY	probability to grow
GRA{XY}	grow axon following gradient of XY	probability to grow
SPL{CTPx}	divide. Offspring is of type CTPx	probability to split
EXT	excitatory stimulus	increase rate
INH	inhibitory stimulus	decrease rate
MOD+{NTx}	increase connection weights	strengthening factor
MOD-{NTx}	decrease connection weights	weakening factor
RLX{NTx}	relax weights slightly	multiplier
DFN{NTx}	define the type of neurotransmitter	none
NOP	null action, neutrality	

The growth of axons and dendrites must be guided in some way to their target. In vivo this is accomplished either by guidance via surface proteins or with the help of diffusible *growth factors*. To implement the latter, two expression commands exist in this model:

- `GDR` (grow dendrite), and
- `GRA` (grow axon).

In the genome, these commands are always used in conjunction with a diffusible substrate, like external proteins or cell-type proteins (e.g., `GDR{CTP0}`). Once expressed, a connection—either a dendrite or an axon—starts to grow following the local gradient of the substrate (here the protein `CTP0`). If the growing connection reaches the location with the highest local concentration of the particular substrate, it tries to connect to the cell. If no cell is present at this location, or the cell is not of the right type (e.g., it does not make sense to connect a dendrite to an actuator), the growing connection dies. In all other cases, the connection will be established. Plate 2 shows an example of dendritic growth.

Another important feature of development is cell division. In this model, a neuron cell can create an offspring cell via the expression command `SPL`, which is always used in combination with a cell-type protein. For example, the expression of the command `SPL{CPT0}` gives rise to the following behavior: First, one of the surrounding free grid elements (those locations that do not harbor a neuron cell) is chosen randomly. It is subsequently transformed to a neuron under conservation of its current diffusible substrates. Finally, after separation from the mother cell, the new offspring cell bears the same genotype as its mother, except that it can be of a different type (indicated by the cell-type protein). In vivo, differentiation through cell lineage is very common.

The role condition values play for the expression of developmental commands is subject to the configuration of the simulation environment. The most important case seems to be its interpretation as a probability for gene expression, leading to *conditioned* development. For instance, a gene like `ADD{EP0} -> GDR{CPT0}` would lead to dendritic growth as a function of the concentration of the diffusible substrate `EP0`, just like growth factors do in vivo.

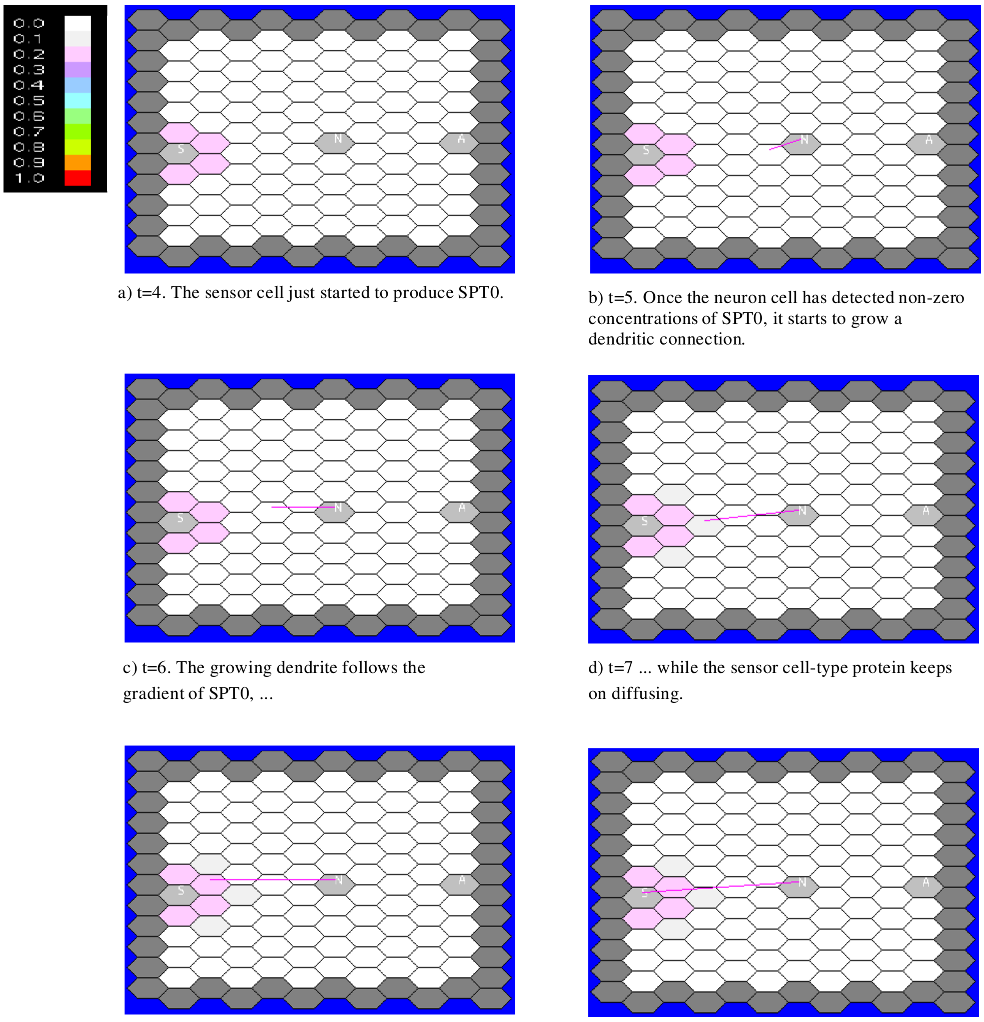
2.3.4 Learning

As is well known, the actual concentrations of neurotransmitter inside of a neuron depend on the *weighted* influx. A specific weight for each type of neurotransmitter is assigned to each dendritic connection. Initially, the weight of a new connection is set to 1.0, which is considered the *relaxed state*. However, it can be manipulated by three different expression commands:

- `MOD+{NTx}` strengthens dendritic connections,
- `MOD-{NTx}` weakens dendritic connections,
- `RLX{NTx}` relaxes dendritic connections.

Each command is specific to a neurotransmitter. In this example the commands influence weights used for influx of neurotransmitter `NTx`. The first two commands—if expressed in neuron *j*—change each synaptic weight in accordance to

$$8i\colon\ \Delta w_{ij}[\mathrm{NTx}]\ \mathrm{D}\ \S\,(w_{ij}[\mathrm{NTx}]\ ¤\ a_i\ ¤\ c)\,, \tag{9}$$



a) t=4. The sensor cell just started to produce SPT0.

b) t=5. Once the neuron cell has detected non-zero concentrations of SPT0, it starts to grow a dendritic connection.

c) t=6. The growing dendrite follows the gradient of SPT0, ...

d) t=7 ... while the sensor cell-type protein keeps on diffusing.

e) t=8. Finally, the growing dendrite approaches the sensor cell ...

f) t=9 ... and establishes the dendritic connection.

Plate 2. A neuron cell starts to grow a dendritic connection following the gradient of the sensor cell-type protein `SPT0`. Finally, a new dendrite becomes established. The gene that was responsible for this behavior is of the form `... , ANY{SPT0} -> GDR{SPTO} , ....` Note that due to the coarse-grained coloring scheme white grid elements do not imply 0.0 concentrations of `SPT0` at this location.

where w_{ij}[`NTx`] is the current specific weight of type `NTx` between neuron i and neuron j, a_i is the activation of neuron i, and c is the overall condition value of the gene that expresses the `MOD+` (respectively `MOD-`) command. As the multipliers a_i and c are ranged in `{0.0;1.0}` and furthermore all weights w_{ij}[`NTx`] are set to 1.0 initially, it can be concluded that

$$8 i, j\colon\ w_{ij}[\mathtt{NTx}]\ ¸\ 0\ . \qquad (10)$$

Non-negative weights may appear to be a limitation, because they do not allow real inhibitive influence. The usage of several types of neurotransmitter, however, makes inhibitive stimulation possible anyway (Figure 3).

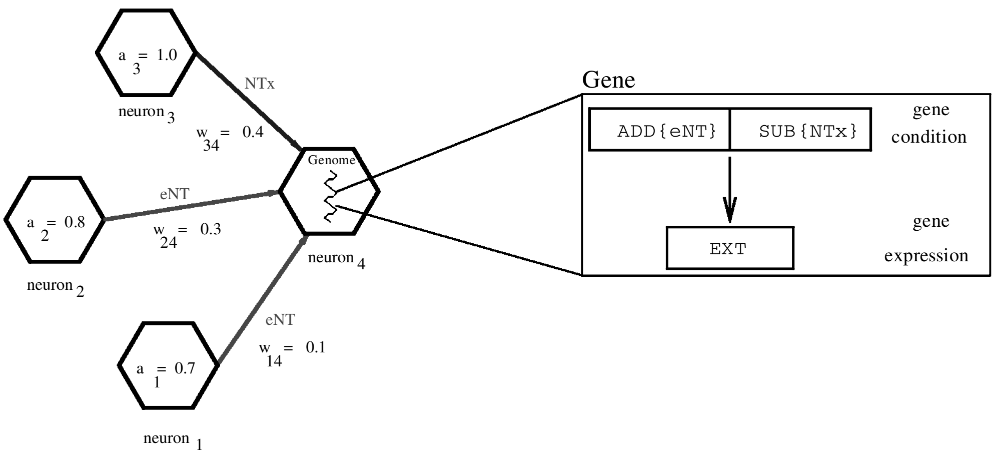


Figure 3. Simple inhibitive connection. The dendritic input from neuron$_1$ and neuron$_2$ produces an eNT concentration of 0.31. Neuron$_4$, however, does not become excited because the NTx concentration represses the gene.

Unlike MOD+ and MOD-, the RLX expression command relaxes weights. In lieu of shaping them, RLX changes individual weight characteristics towards their initial value 1.0. Applying RLX to the specific neurotransmitter NTx in neuron j has the following effect:

$$\mathrm{RLX[NTx]}\ \mathsf{H)}\ 8i\ \Delta w_{ij}[\mathrm{NTx}]\ \mathsf{D} \begin{cases} \mathsf{C}\, w_{ij}[\mathrm{NTx}] ¤ C_p ¤ C_c, & \text{if } w_{ij}[\mathrm{NTx}] > 1.0 \\ ¡\, w_{ij}[\mathrm{NTx}] ¤ C_p ¤ C_c, & \text{if } w_{ij}[\mathrm{NTx}] < 1.0 \\ 0.0, & \text{otherwise ,} \end{cases} \tag{11}$$

where w_{ij}[NTx] is the specific weight controlling the flux of neurotransmitter NTx between neuron i and neuron j, C_p is a fixed percentage constant and C_c is the overall condition value of the gene that expresses the RLX{NTx} command. An internal mechanism avoids oscillation around 1.0, by ensuring that if a weight is currently greater than 1.0 then its future value will be ¸ 1.0 also (and vice versa for weights less than 1.0).

2.3.5 Stimulation

Two commands influence cell stimulation in a simple and direct way. Depending on the value of the gene condition, EXT increases the activation level, while INH command decreases it. Both commands do this with respect to the activation range {0.0;1.0}. If during the same time step EXT or INH are expressed more than once, then the effective change of activation is just the sum of all single increases (respectively, decreases). Thus, a gene like ADD{IP0} MUL{EP1} -> EXT EXT where the current concentrations of IP0 and EP1 are 0.4 and 0.3 would lead to an activation increase of 0.24. Unless it is already completely inactive, the cell activation normally decreases at each time step with an adjustable rate if no stimulation command is expressed.

Obviously, cell stimulation *does not necessarily depend on the weighted sum of inputs*. Influences of *every kind* can manipulate the degree of activation of a cell. Of course, if the modeling of a neuron cell in accordance to classical ANN models is necessary, this can still be achieved by using neurotransmitter-related condition atoms *only* in gene conditions. For example, a gene like ADD{eNT} -> EXT stimulates the neuron according to the weighted sum of inputs (of the default neurotransmitter), just as standard ANN models do. Keep in mind, however, that in vivo, hormones, neuromodulators, and other substrates can have a tremendous influence on cell stimulation. This can be easily modeled with, for example, a gene like: `ADD{eNT} MUL{EP0} -> EXT`. Here, the neuron becomes stimulated according to the weighted sum of inputs *and* the current concentration of a "neuromodulator" `EP0`.

2.3.6 Other Commands

Command `PRD`, when expressed, produces new substrate (i.e., increases the concentration of a particular substrate). By definition, only the synthesis of external and internal proteins is allowed, as neurotransmitters and cell-type proteins have an *implicit* role in the model.

For the expression command `PRD`, the overall condition value simply determines how much of the assigned substrate has to be produced. For example, if the current concentrations of `IP2` and `eNT` are 0.6 and 0.1, the gene `ADD{IP2} AND{eNT} -> PRD{EP2}` produces 0.1 of `EP2`.

Another command is `DFN{NT}` where `NT` stands for any neurotransmitter. Once expressed, it changes the defined neurotransmitter used for dendritic-axonal injection. The condition value is of no importance for this expression command, as long as it is not equal to 0.0. This command is added because it is known that in vivo, some neurons change the type of neurotransmitter used at their synapses. Starting with several types, the maturation process of each neuron eventually defines the right type.

Finally, the last expression command to mention is `NOP`—the neutral command—which takes the role of a placeholder for future genetic changes (i.e., mutations).

2.4 Simulation of the Organism

The tissue of cells produced by gene expression and cell growth is termed an *artificial organism.* It receives input from the environment (the outside world) and can act on it by signaling to the environment via its actuators. In the simplest case, the organism receives and generates patterns of activation.

A simulation always starts by creating sensor and actuator cells. Their number is determined only by the complexity of the outside world and is not coded for in the genome. In other words, these cells really represent *possible* signals and actuations in the world, not actual signals and actuations performed by the organism. An organism chooses to receive input or perform an actuation by connecting to these cells. If needed, an additional *reinforcement cell* can be created. This is a special sensor cell (with its own cell-type protein) used to provide a reinforcement signal from the world about the behavior of the organism. Whether or not this signal is used depends on the genome.

At the start of simulation for each new artificial organism, one initial neuron is placed in the center of the grid. After initialization, the simulation begins. Input from the world is provided to the sensor cells, cell-type proteins and external proteins diffuse, and neurons execute their genetic code. This is done synchronously to guarantee consistency in the artificial chemistry.

Depending on its gene expression, a neuron starts growing axons and dendrites, produces offspring cells, and might initiate cell differentiation. Gene expressions may lead to protein production cascades, stimulation, and ultimately information exchange between neurons. After every simulation cycle the network's "fitness" is determined by comparing any inputs and outputs to what is expected in this particular world, producing a real-valued reinforcement signal between 0.0 (punishment) and 1.0 (reward). This signal can be used by the organism if a reinforcement sensor is present and if the organism chooses to connect to it.

3 Examples: Genes and Networks

The best way to validate and evaluate this model would be to let the system evolve ANNs for specific environments. This, however, is not done yet for reasons discussed later on. Except for some experiments in which simple logical functions were evolved from scratch [3], most genomes were handwritten. To show that the model is able to give rise to networks that perform some key behaviors known from biology, examples of handwritten genotypes and the resulting phenotypes are given in the following, performing self-limiting growth, networks that perform logical functions, and classical conditioning. Other examples of physiologically important structures that have been reproduced are the regeneration of injured networks, pacemaker behavior, as well as sensitization and habituation behavior [3].

3.1 Self-limiting Growth

The development of natural systems is based on a decentralized growth process that arises from local gene expression in each single cell. This, combined with the fact that inhibition is necessary to avoid unbounded growth, means that self-inhibition is an essential property the model has to encompass. For example, self-limitation is essential to avoid cancerous growth. The following genome shows how self-limited growth can be achieved in a simple way in a completely local manner. In the genome below, `C` refers to the regulatory (conditional) part of the gene, and `E` is the code to be expressed:

```
Gene 1      C: SUP{cpt}    NNY{ip0}
            E: PRD{ep0}    PRD{ip0}
Gene 2      C: ADD{ep0}
            E: SPL{acpt0}
Gene 3      C: NNY{ip0}
            E: GRA{acpt0}  PRD{ip0}
Gene 4      C: ANY{ip0}
            E: PRD{ip0}
```

The first gene is expressed initially by the stem cell that produces cell-type protein `cpt`. As no internal protein `ip0` is present initially, gene 1 is expressed and leads to the production of the external protein `ep0` and the internal protein `ip0`. Just like every external protein, `ep0` diffuses within the tissue so that the initial peak concentration in the stem cell becomes weaker while surrounding cells receive concentrations of `ep0`. Gene 4 guarantees that once a cell harbors any `ip0` it is going to produce `ip0` constantly from then on. This, however, suppresses gene 1, which consequently will never produce `ep0` again. The internal protein `ip0` also suppresses gene 3, whose expression leads to axon growth and `ip0` production. Thus, a cell in which initially no `ip0` is present starts growing an axon and then suppresses itself continuously via gene 4.

Gene 2 can lead to cell division. Every executed split command creates a new offspring cell of type `acpt0`. However, cell division occurs only with a probability that is proportional to the current concentration of `ep0`. Thus, as `ep0` diffuses away over time, cell division becomes less and less probable. Consequently the growth process remains limited. This effect can be observed in Plate 3, which shows snapshots taken from a simulation of the genome above.

3.2 Logical Functions

Research in neurobiology has shown [15] that some neural structures essentially implement fixed logical functions. Using two different kinds of neurotransmitter, logical gates can be simulated easily within the present model (Figure 4).

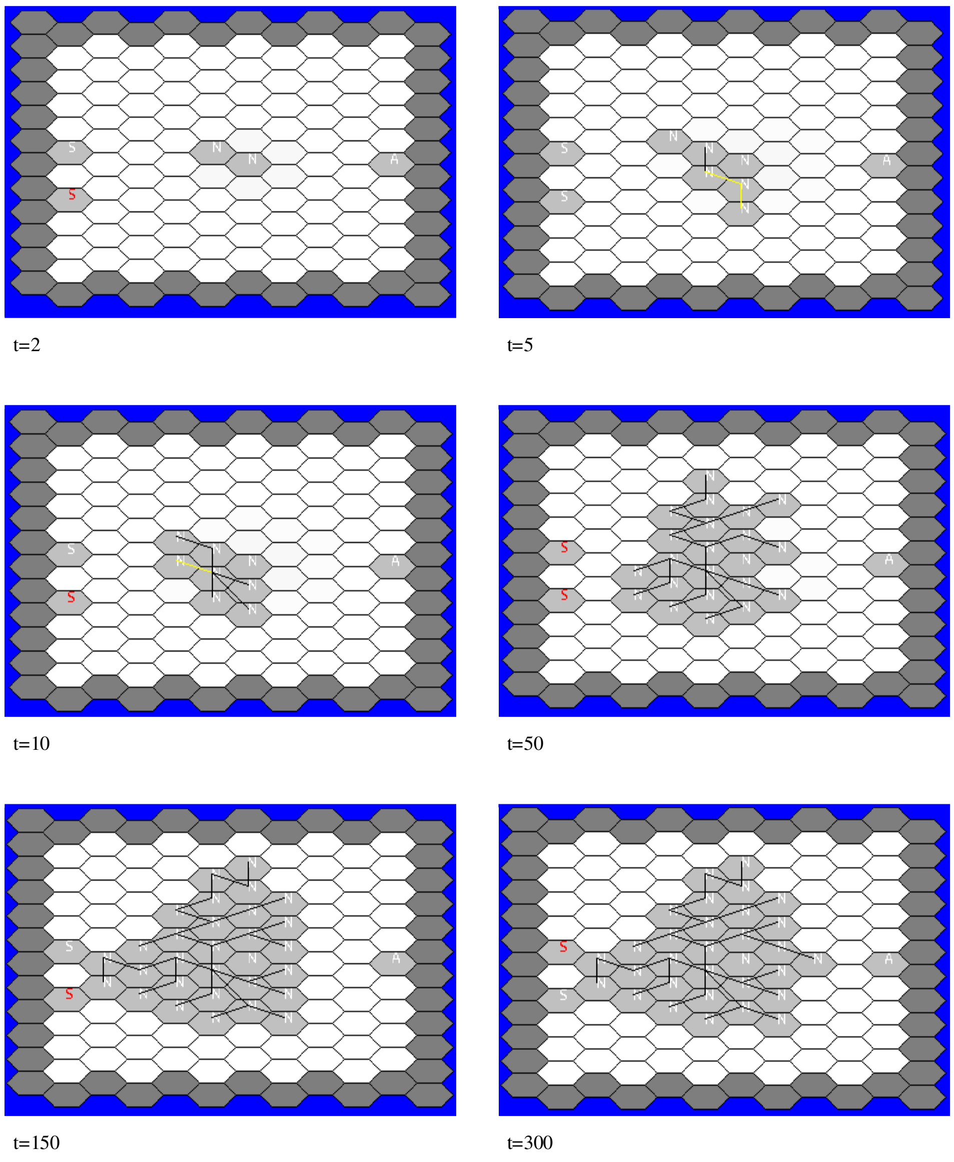


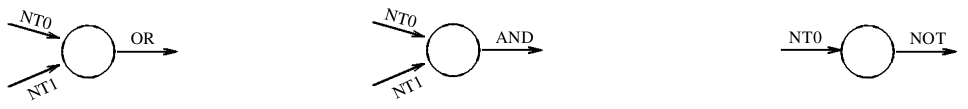
Plate 3. Self-limiting growth. An initially high peak of a diffusible substrate facilitates cell division. As the substrate vanishes over time, fewer cell divisions occur.

The basic logic operations AND, OR, NOT are sufficient for the construction of any logical function. For example, the genome shown below constructs the function $D \Leftarrow (A \rightarrow B) \vee C$, where A, B, C are sensor signals and D is the resulting actuator signal. To model this logical function, one has to combine an implication gate (i.e., $\neg A \vee B$) with an OR gate.

Using both the genetic fragments of Figure 5, a genotype can be constructed that grows the appropriate ANN structure and describes the stimulation behavior:

```
Gene 1  C: SUP{cpt}    NNY{ip0}    ANY{spt0} ANY{apt0}
        E: SPL{acpt0} GRA{apt0}    GDR{spt0} PRD{ip0}
```

Simple binary-logic gates:



Genes:

```
ANY{NT0} -> EXT
ANY{NT1} -> EXT
```

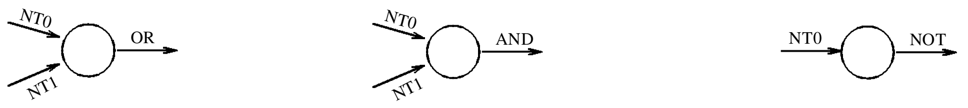
Gene:

```
ANY{NT0} ANY{NT1} -> EXT
```

Gene:

```
NNY{NT0} -> EXT
```

Simple fuzzy-logic gates:



Gene:

```
ADD{NT0} OR{NT1} -> EXT
```

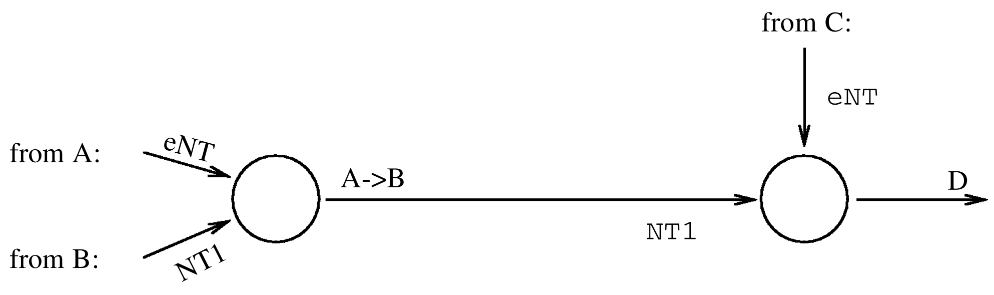
Gene:

```
ADD{NT0} AND{NT1} -> EXT
```

Gene:

```
NAND{NT0} -> EXT
```

Figure 4. Logical gates and the genomes that perform them. Any logical gate can be simulated easily in the model as long as the incoming signals use two different types of neurotransmitter.



Genes:

```
ANY{eNT} ANY{NT1} -> EXT
NNY{eNT} -> EXT
```

Genes:

```
ANY{eNT} -> EXT
ANY{NT1} -> EXT
```

Figure 5. The stimulation model for an ANN structure performing D D $(A$! $B)$ _ C.

```
Gene 2  C: SUP{acpt0} NNY{ip0}    ANY{spt2} ANY{cpt}
        E: GRA{cpt}   SPL{acpt1}  GDR{spt2} DFN{NT1}   PRD{ip0}
Gene 3  C: SUP{acpt1} ANY{spt1}   NNY{ip0}  ANY{acpt0}
        E: DFN{NT1}   GRA{acpt0}  PRD{ip0}  GDR{spt1}
Gene 4  C: ANY{ip0}
        E: PRD{ip0}
Gene 5  C: SUP{cpt}   ANY{NT1}
        E: EXT
Gene 6  C: ANY{eNT}   NSUP{acpt0}
```

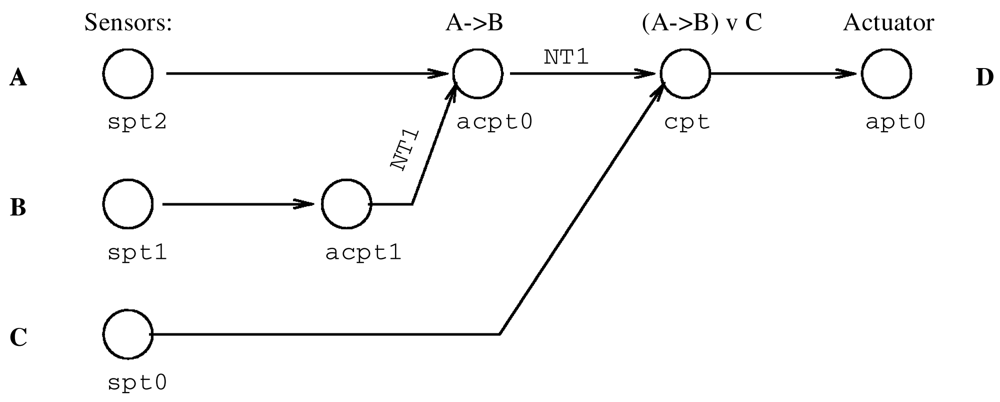


Figure 6. Schematic representation of the ANN realization of *D* D (*A* ! *B*) _ *C*. The type of neurotransmitter is marked next to the axon if a nondefault neurotransmitter is used. Cells are named by the cell-type protein they produce. The cell of type `acpt1` is used as "neurotransmitter-switch."

```
        E: EXT
Gene 7  C: SUP{acpt0} ANY{eNT}     ANY{NT1}
        E: EXT
Gene 8  C: SUP{acpt0} NNY{eNT}
        E: EXT
```

Genes 1–3 lead to the morphological structure of the network. The neuron of cell-type protein `cpt` produces an offspring cell of type `acpt0` that models A ! B, while the `cpt` cell represents the OR gate. Three different cell types are needed because the logical function requires different kinds of neurotransmitter whereas the sensors use only `eNT`. Thus, an intermediate cell is necessary to overcome this problem. Gene 4 is the usual suppression gene that becomes active once one of the first three genes is expressed. Genes 5 through 8 encode the stimulation behavior in accordance to the different types of cells.

The genome for this task might, at first sight, appear rather long. However, one should keep in mind that it describes the construction of a *fully* deterministic structure, which is more difficult for a growth model than for an approach based on direct encoding. Furthermore, simpler genomes can be built that model the same logical function. This example is intended to show that every logical function can be constructed by *connecting* simple gates leading to hierarchical structures not unlike those found in real networks. A schematic representation of the resulting structure is shown in Figure 6. As the paths of the input signals from the sensors to the computation units are of different lengths, a particular input has to be present at the sensors for several subsequent time steps before a *stable* output signal will be produced.

3.3 Classical Conditioning

Plate 4 documents the development of a simple ANN that displays conditioned reflex behavior as in Pavlov's classical experiment [18]. Suppose the sensor on the lower left side in Plate 4 is stimulated at the sound of a bell. Further, suppose the upper left sensor is an *optical stimulus* representing the presence (or absence) of food. Finally, let us imagine that the actuator on the right side triggers a salivary gland if food is present. This behavior is the unconditioned reflex. The above network can learn to associate this reflex with a condition: the sound of the bell. If

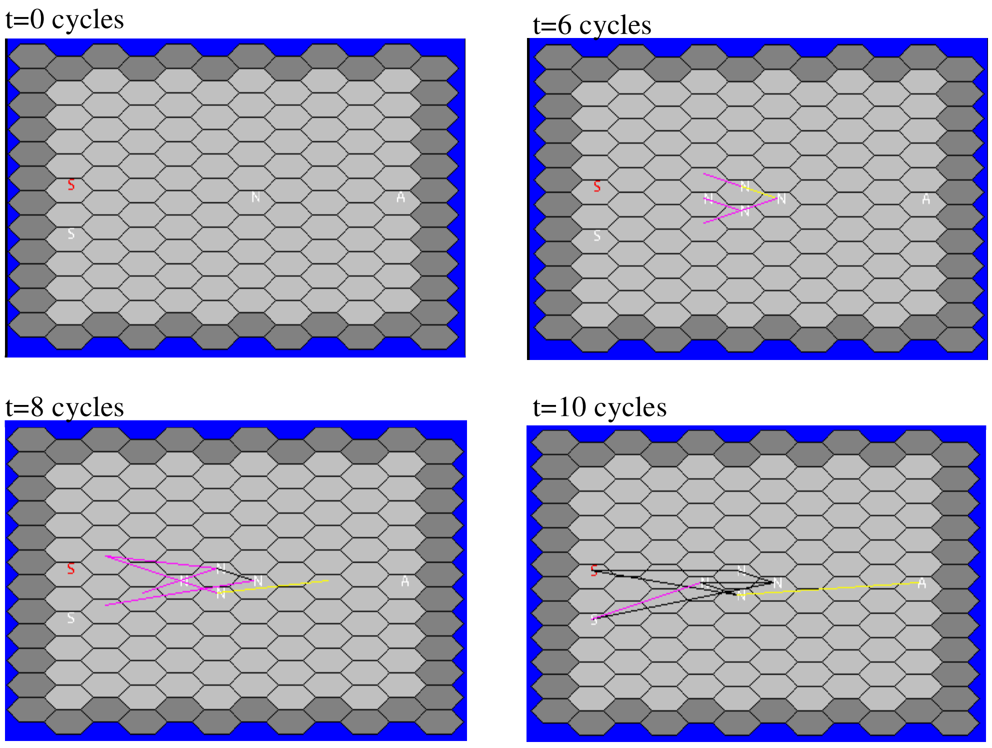


Plate 4. Development of the network for classical conditioning.

presence of food and the ringing of the bell are associated repeatedly, the network will learn to trigger the gland even if *only* the bell rings. If after the conditioning the bell rings without presence of food, the association will gradually, but steadily, weaken.

Such a behavior can be modeled using different kinds of cell types. One cell ("C" cell) is activated if the network is in the conditioned state, which means that the acoustical and optical stimulus have been present together before. Another cell ("E" cell) is activated if the acoustical stimulus is currently present and the network is in the conditioned state at the same time. If so, a cell ("G" cell) representing the trigger of the salivary gland is activated. Of course, the "G" cell also has to be activated if only food is present. This is the unconditioned reflex. A schematic drawing of the network is shown in Figure 7.

The genome that encodes the development and behavior of this network is shown below.

```
Gene 1    C: NNY{ip0}     SUP{cpt}     ANY{spt0}
          E: SPL{acpt0}   PRD{ip0}     SPL{acpt2}  GDR{spt0}  DFN{NT1}
Gene 2    C: NNY{ip0}     SUP{acpt0}   ANY{spt1}   ANY{cpt}
          E: PRD{ip0}     GDR{spt1}    GRA{cpt}    DFN{NT1}
Gene 3    C: ANY{spt1}    SUP{acpt2}   NNY{ip0}    ANY{apt0}
          E: SPL{acpt1}   GDR{spt1}    PRD{ip0}    GRA{apt0}
Gene 4    C: ANY{acpt2}   SUP{acpt1}   ANY{spt0}   NNY{ip0}
          E: GRA{acpt2}   GDR{spt0}    GDR{cpt}    PRD{ip0}
Gene 5    C: ANY{ip0}
```

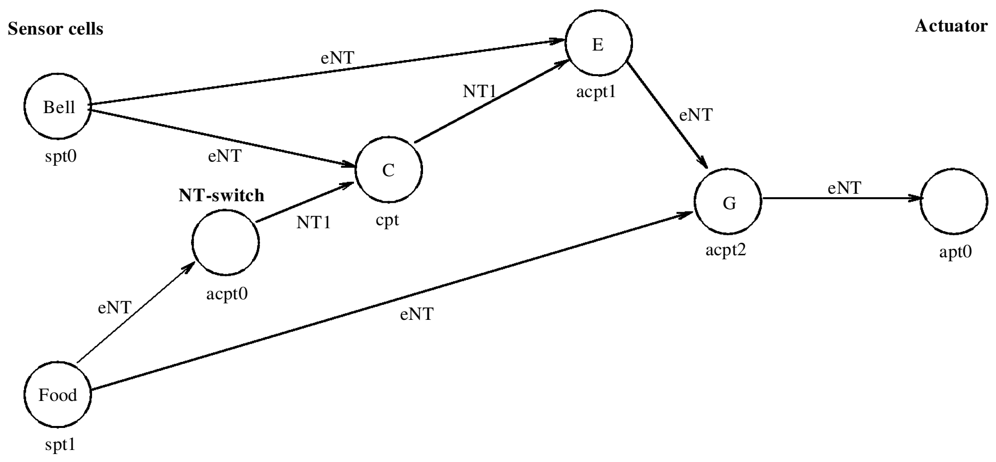


Figure 7. A schematic representation of the network for classical conditioning. The types of neurotransmitter used are marked next to the axons. The cell-type protein used by each cell is indicated near the cell body.

```
        E: PRD{ip0}
Gene 6  C: NSUP{cpt}   NSUP{acpt1} ADD{eNT}
        E: EXT
Gene 7  C: SUP{acpt1}  ADD{NT1}    MUL{eNT}
        E: EXT
Gene 8  C: ADD{eNT}
        E: PRD{ip1}
Gene 9  C: ADD{ip1}
        E: PRD{ip2}
Gene 10 C: SUP{cpt}    ADD{NT1}    MUL{ip2}
        E: PRD{ep0}
Gene 11 C: SUP{cpt}    ADD{ep0}
        E: EXT
```

Genes 1 to 4 control cell division into the different types that are needed, as well as the growth of axons and dendrites. Gene 5 is the usual stop gene. To be able to distinguish between two different kinds of signals, two types of neurotransmitter have to be used. The “C” cell, for example, uses `NT1` as neurotransmitter so that the “E” cell can check for inputs from the acoustical sensor and from the “C” cell at the same time. The manifestation of this can be seen in the genotype: The cells of type `acpt0` and `cpt` use `NT1` instead of the default neurotransmitter `eNT` (note the expression command `DNF{NT1}`).

The “C” cell also has to make the distinction between two signals: the acoustical and the food sensor input. As sensor cells use by definition only the default neurotransmitter, a kind of neurotransmitter-switch cell is needed between the food sensor and the “C” cell. This is why an additional cell of type `acpt0` is needed (gene 2).

While gene 6 takes care of the unconditioned reflex stimulation, genes 7–11 control the conditioning. If food is present and the bell rings, gene 10 produces certain amounts of external protein `ep0` (through a time-delay cascade via genes 8 and 9). The concentration of `ep0` influences cell stimulation (gene 11). Due to diffusion, `ep0` diminishes over time, so the conditioning decreases accordingly.

The time delay of `eNT` inputs via genes 8 and 9 and finally 10 is necessary because the dendritic paths from both sensors (food and sound) to the “C” cell are not equally

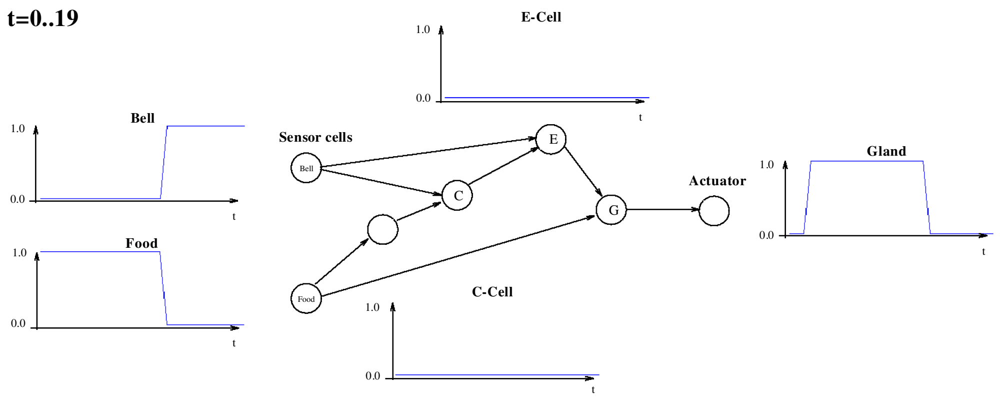


Figure 8. Conditioning. First, only food is present. This triggers the gland because of the unconditioned reflex, while both the "C" cell and "E" cell remain inactive. Later, only the bell signal is present. Due to the fact that the network is not yet conditioned, none of the cells become active.

long. While the sound signal inputs directly to the "C" cell, signals of the food sensor have to pass through the neurotransmitter-switch cell. Both signals, however, must be processed at the same time in the "C" cell. Thus the sound signal has to be delayed via a production cascade.

The behavior of the resulting phenotype network is documented in Figures 8–10.

4 Evolution of Networks

While the neurogenesis model uses a predefined genetic code for the description of growth and behavior of ANNs, this code is not designed to be convenient for humans. However, it was designed in such a manner that evolutionary pressure can be used to find efficient genotypes. To apply principles of artificial evolution we designed a genetic algorithm to find novel neural circuits and architectures.

First of all we have to be aware of the fact that the genetic search space the model creates can be gigantic. Not only the number of different substrates simulated in the artificial chemistry but also the length of gene conditions have an immense influence

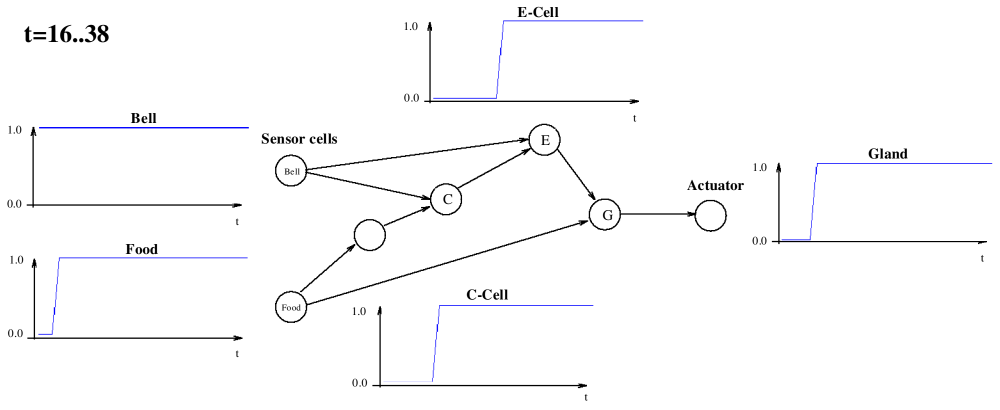


Figure 9. Conditioning. Both sensors, food and sound, are stimulated. Consequently, the ANN becomes conditioned ("C" cell) and the gland is triggered because of the presence of food.

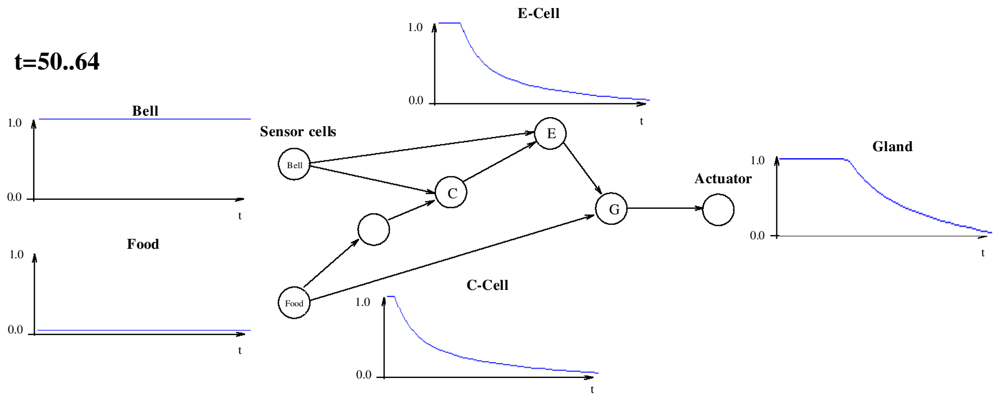


Figure 10. Conditioning. Being in the conditioned state, the food sensor suddenly becomes inactive while the bell keeps on ringing. Thus, the activation of the "C" cell becomes weaker. This implies a decrease of activation of the "E" cell that eventually results in a decline of gland activity.

on the number of possible different genotypes. The size of the genetic space is an important issue for the design of a GA. The bigger it is, the more computation has to be spent on evolutionary search in order to find genotypes that code for interesting ANNs. To compound matters, unlike in usual ANN models, a computational overhead arises due to the simulated biological world. In a neurogenesis model such as this, the map from genotype to phenotype is highly implicit as the networks are grown from single stem cells in a developmental process that involves an enormous computational effort: Substrates are created through gene expression; they undergo diffusion within the tissue; gene conditions have to be evaluated; and so on. Thus, extensive computation is necessary until the fitness of a particular genotype can be determined, and even more to apply evolutionary search. An increase in computational power can be achieved by a *distributed* GA that allows a massively *parallel* search. Details of the implementation of the GA are relegated to Section 5.

4.1 Fitness Evaluation

The fitness of a particular genotype can only be assessed through the fitness of the phenotype it gives rise to in a particular environment. The phenotype's fitness itself is defined as the average reinforcement signal received from its environment (see Section 2.4).

Three levels of assessment determine if an organism passes or fails the evaluation request:

1. Knockout criteria: It seems to be reasonable to assign an organism a zero fitness if it does not fulfill some minimal criteria. For example, organisms in which the actuator and sensor cells are not connected (no dendrite/axon grew to them) are assigned zero fitness by default. Furthermore, organisms in which the actuators are constantly activated or inactivated are not considered fit either, as no computation can be achieved. Such organisms are replaced immediately.

2. Organisms that passed the first level are compared with each other on the second level: All organisms occupy a particular position on a ranked list of decreasing fitness. If the *relative* fitness of the organism that sent the evaluation request is below a fixed percentage threshold rank (i.e., it is better than most), then it passes automatically (Figure 11).

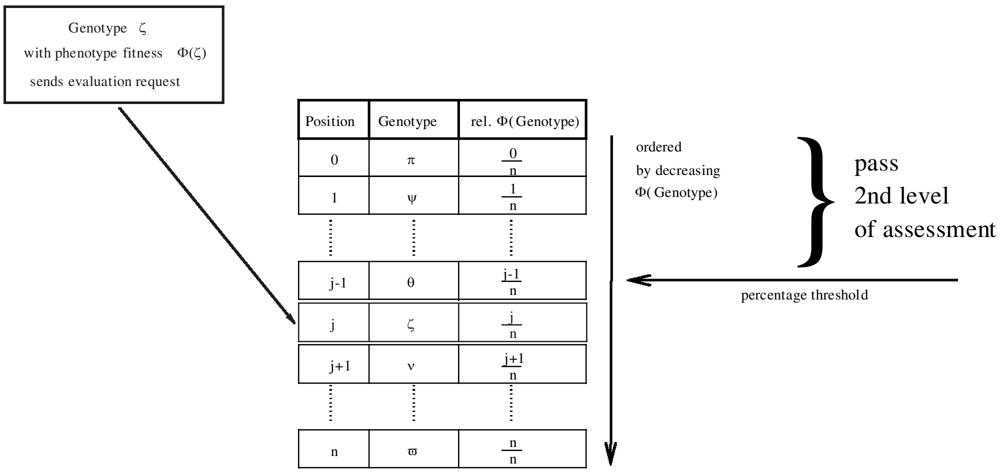


Figure 11. Organisms belonging to the elite (defined by a percentage threshold) always survive their evaluation request. In this case, organism ζ does not pass the comparison test (i.e., the second level of assessment).

3. As the threshold-based assessment is rather rough, organisms that do not pass this step get a second chance: They survive by chance with a probability

$$P(X < e^{-c \cdot p_{\text{rel}}}) \tag{12}$$

where X is taken from a uniform probability distribution over range `{0.0;1.0}`, c is a constant, and p_{rel} is the relative position (based on fitness) in the ranked list of all genotypes.

Obviously, the size of the population does not matter in applying these criteria and can therefore vary over time.

4.2 Genetic Operators

Selection, recombination and mutation are the genetic operations that are used to build new genotypes from a genetic pool. As new genotypes can be constructed in many different ways (by combining the aforementioned operations together with *replication*), the genetic operations will be presented first before the description of the actual construction mechanism follows (Section 4.3).

4.2.1 Selection

Occasionally, the GA-server has to *select* genotypes from its database. According to the principles of artificial evolution, selection has to be based on the fitness of genotypes. As genotype fitness cannot be measured objectively, usually a selection mechanism is applied that chooses genotypes with a probability proportional to their phenotype fitness. Accordingly, the GA server uses a *roulette wheel selection*.

4.2.2 Recombination

To recombine two genotypes they have to be aligned. Subsequently, one or more crossing points have to be chosen at which the hereditary information is exchanged. However, crossover points can only be chosen *within* genes, so that each gene is protected and cannot be torn apart across two genotypes. In other words, different genes do not swap code fragments. This principle is also found in nature: Only whole genes undergo recombinational exchange and never parts of genes. The number of crossovers is chosen from a binomial probability distribution with an expectation value close to one.

Due to mutation (see below) genotypes can differ in length. Naturally, in the case of crossover of genotypes of different length, the crossover points are chosen inside of the overlapping area of the two genotypes.

4.2.3 Mutation

Mutation is necessary to produce diversity in the genetic population. It is applied by the GA-server as a genetic operation and changes a randomly chosen genotype "slightly" (as explained below). In this model, mutation can take place in two ways:

- Point mutations: These are restricted to one gene. A randomly chosen condition atom or expression command of a randomly chosen gene changes to its nearest neighbor in genetic space. This means that if, for example, the condition atom `ADD{IP2}` has been chosen, mutation changes either the substrate (e.g., to `ADD{IP1}`) or the operation (e.g., to `MUL{IP2}`) but never both.
- Genome mutation: Another important kind of mutation that is found in nature as well is mutation on the genome level: *deletion, insertion*, and *doubling* of entire genes. For example, successful genes (e.g., structure genes that build a layer of neurons) can be doubled while other useless genes may vanish entirely. Gene doubling has become a success story in evolution. Indeed, it was shown recently [20] that initially simple morphological genes (sometimes even entire genomes) often doubled during evolution, only to subsequently specialize.

4.3 Construction of Genotypes

When a genotype has to be replaced because of low fitness or if the size of the population grows, a new genotype must be constructed. This can be done in several ways: from a random genome, by recombination, by pure "asexual" reproduction (i.e., replication), with or without point mutation, gene insertion, doubling, or deletion. There is no way to determine a priori which of them plays the most important role. For example, we do not know if recombination of two fit genotypes is in principle better than constructing new genotypes with asexual copies of fit organisms. In nature, both principles are applied. Therefore, the different possibilities should all be integrated using tunable probabilities. Figure 12 describes the algorithm used to construct new genotypes in this GA as a probability tree.

5 Implementation

A distributed GA can be implemented in two ways: either by using a massively parallel machine, or by distributing the system over a network of computers. For our purpose, the latter seems to be the better choice, especially if the GA is not limited to a local area network (LAN) and allows platform-independent use. The importance of *heterogeneous* computer networks has been increasing enormously during the last decade and will most probably be the architecture of the future. Designing the distributed GA as an open system on a wide area network (WAN, e.g., the Internet) enables us to tap the *unused* CPU power of a very large number of computers, bringing an extraordinarily computationally expensive task into the realm of possibility.

The basic design can be described as follows: Using Sun's Java technology, an asynchronous, distributed GA system was built that allows a massively parallel search for genomes based on evolutionary principles. It consists mainly of a central server

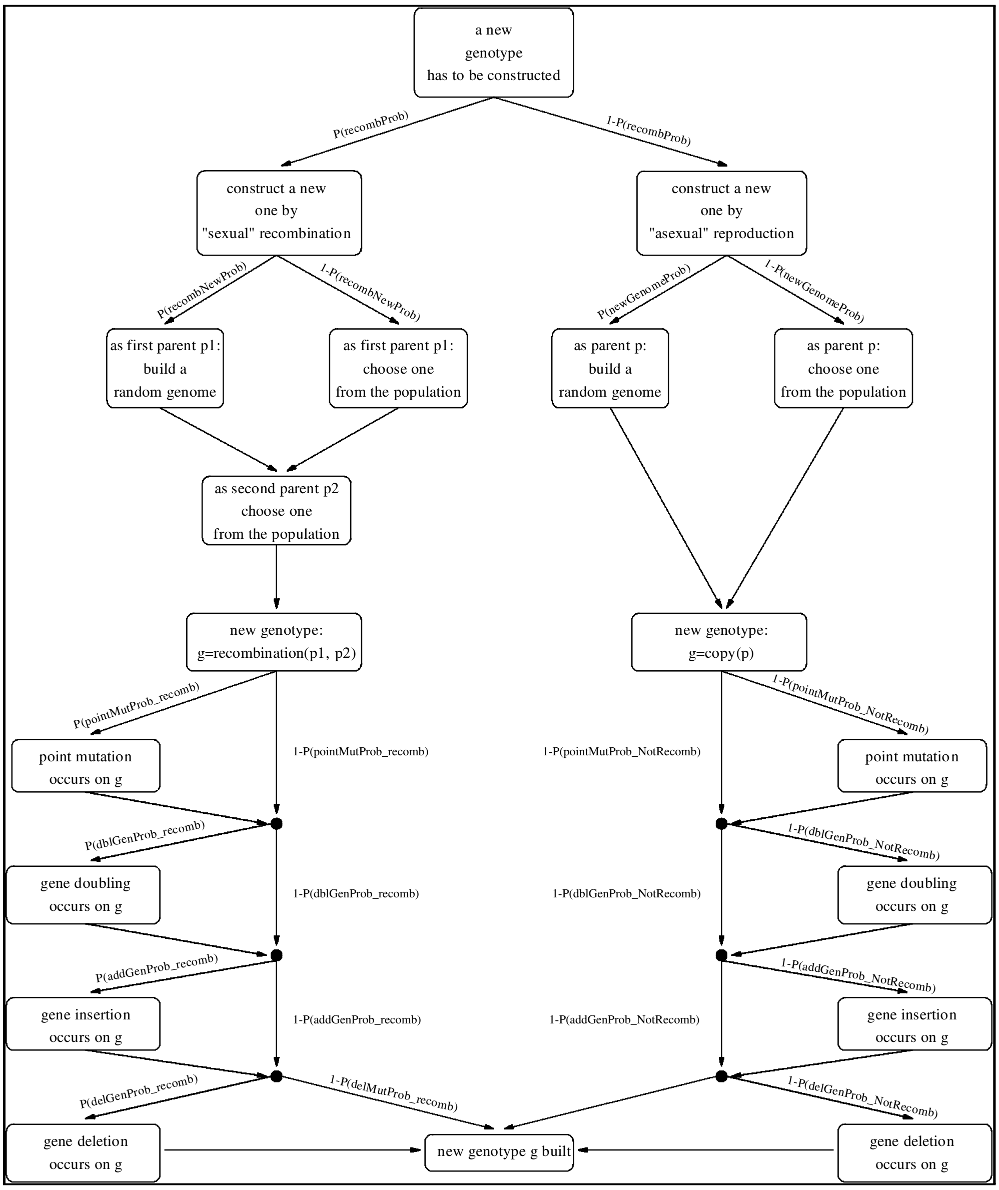

a new genotype has to be constructed
P(recombProb)
1-P(recombProb)
construct a new one by "sexual" recombination
construct a new one by "asexual" reproduction
P(recombNewProb)
1-P(recombNewProb)
P(newGenomeProb)
1-P(newGenomeProb)
as first parent p1: build a random genome
as first parent p1: choose one from the population
as parent p: build a random genome
as parent p: choose one from the population
as second parent p2 choose one from the population
new genotype: g=recombination(p1, p2)
new genotype: g=copy(p)
P(pointMutProb_recomb)
1-P(pointMutProb_recomb)
1-P(pointMutProb_NotRecomb)
1-P(pointMutProb_NotRecomb)
point mutation occurs on g
P(dblGenProb_recomb)
1-P(dblGenProb_recomb)
1-P(dblGenProb_NotRecomb)
1-P(dblGenProb_NotRecomb)
gene doubling occurs on g
P(addGenProb_recomb)
1-P(addGenProb_recomb)
1-P(addGenProb_NotRecomb)
1-P(addGenProb_NotRecomb)
gene insertion occurs on g
P(delGenProb_recomb)
1-P(delMutProb_recomb)
1-P(delGenProb_NotRecomb)
gene deletion occurs on g
new genotype g built

Figure 12. Starting from the top, the diagram describes the different ways to construct a new genotype. While unlabeled arrows carry the probability one, constants are assigned to others describing the probability of the arrow to be chosen. The genetic operations selection ("choose from population"), recombination, and mutation are defined as described in the previous sections.

application and many clients, each of which hosts *one* individual of the current GA population. While the central server drives the evolution via genetic operations (i.e., recombination, selection, and mutation), the clients represent the population, which is changing continuously in size and location. By starting a client locally, it can latch onto the server, and by being a host for one genotype it becomes part of the evolutionary process. It can, of course, detach itself from the evolutionary process at any time. In this case, data about the hosted genotype (and the phenotype it has grown) are sent

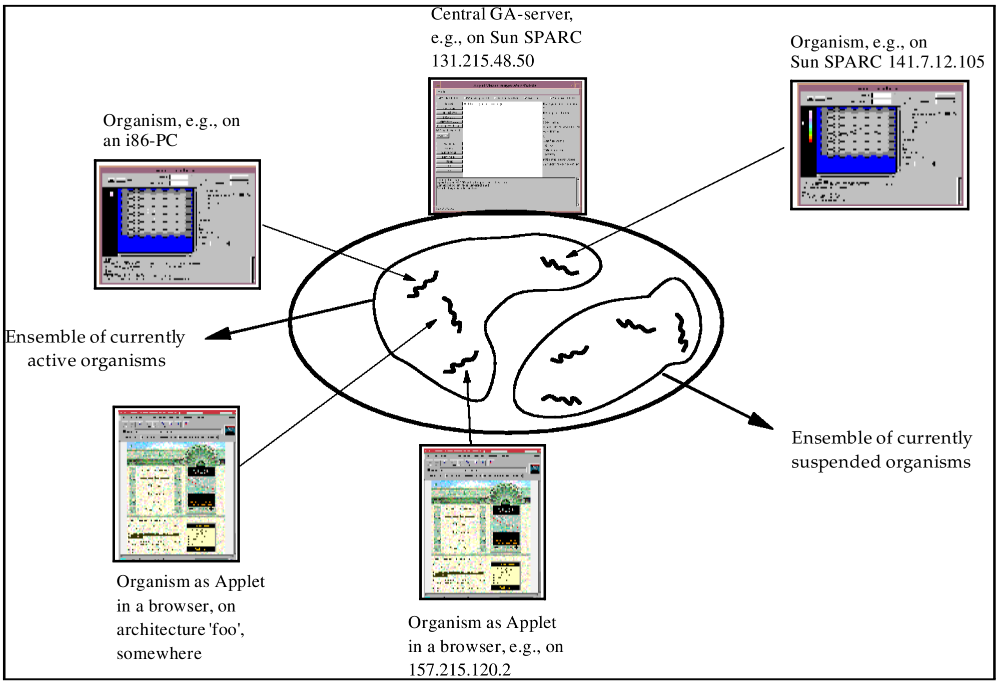


Plate 5. Genotype evaluation in clients of different architecture, using TCP for communication with the asynchronous GA.

back to the server where they will be stored until another client receives the data on its registration request.

In the following, we will use both the terms *organism* and *phenotype* synonymously for the cell structure that has been grown from one stem cell of a particular genotype. We use the term *new organism* for a newly created stem cell that has yet to perform any gene expression. Note that each organism is distinguished by its particular genotype and contains it as its genome inside. Thus, an organism always contains the hereditary information from which it was grown. Consequently, the server only has to maintain a database of organisms.

As Java is supposed to be platform independent, clients can be started from every computer for which an accurate Java virtual machine or browser exists and can then communicate via the *Transport Control Protocol* (*TCP*) with the central server (Plate 5).

To make the system as flexible as possible the clients can be designed as hybrids, which means that they can be started either as a Java Applet by choosing a specific HTML (Hypertext Markup Language) page from a particular WWW server, or as a Java application with the help of a bootloader program that dynamically downloads the client and starts it as a Java application; in the latter case, no browser is necessary.

This dynamical and platform-independent design balances Java's disadvantage in being a slow, interpreted language. Increasing computer performance and faster execution of Java byte code via just-in-time (JIT) compilation will make Java even better suited for this purpose in the future.

5.1 Communication Between Clients and GA Server

A client automatically sends a request-to-register to the central GA server after it is started and receives from the server a (new) organism ζ. The client then starts up a

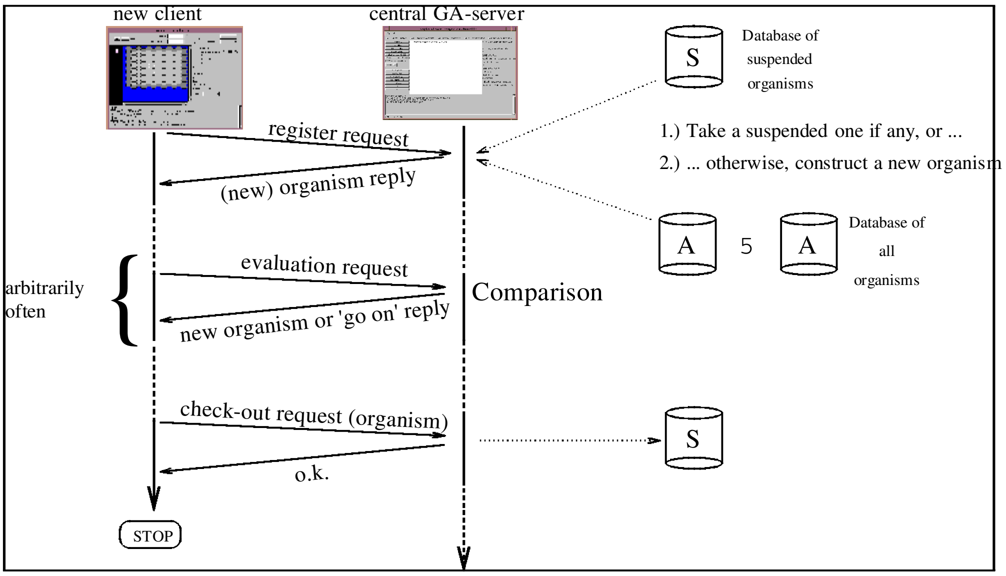

new client
central GA-server
Database of suspended organisms
S
1.) Take a suspended one if any, or ...
2.) ... otherwise, construct a new organism
register request
(new) organism reply
A
5
A
Database of all organisms
arbitrarily often
evaluation request
new organism or 'go on' reply
Comparison
check-out request (organism)
o.k.
S
STOP

Figure 13. Communication between GA server and client hosting an organism.

simulation as described in Section 2.4. After a certain number of simulation cycles, it sends the organism's fitness $\Phi(\zeta)$ to the server. By comparing $\Phi(\zeta)$ to the fitness of other phenotypes in the database (as described in Section 4.1), the server decides if it is worthwhile to keep this organism or if the client should be assigned a different one. If the server has to send a different organism, it either takes a suspended one out of its database, or *constructs* one through the processes of recombination or asexual copying from genotypes of known fitness already present in the population (Figure 13). Fitter genotypes are more likely to be selected for recombination or asexual copying than genotypes of lower fitness. This leads to an increase of the average fitness of phenotypes over time.

A particular client repeats its evaluation request arbitrarily often and always either obtains the permission to go on with the same organism or receives a different organism to evaluate. If the client (or better: the human who started the local client) decides to end its participation, it just sends the phenotype back to the GA server, which stores it in its database of currently suspended organisms.

5.2 Participating in Evolution Experiments

To facilitate the participation in evolutionary experiments across the Internet, site `http://norgev.alife.org` has been created, from which the Java application can be downloaded. This site also offers instructions and more detailed system requirements.

5.3 Performance and System Limitations

Up to now, only very simple ANNs for logical basic functions have been evolved de novo [3]. The system was able to accomplish this in a short amount of time with a population of about 40 genotypes running on about 10 Sun SPARC Stations 2. While this experiment showed that the genetic algorithm works for small populations, it has never had to deal with populations of a large size.

If the system is to be opened to the whole Internet community, a more appropriate database organization may be necessary. While the implementation with Java hash tables might be good enough for small populations, it is surely restrictive for large populations. Instead, the system ought to be connected to a professional database system in order to be able to deal with high rates of server requests and large amounts of data.

The current implementation uses Java on both sides: Clients (Java Applets) and server are fully based on Java technology aided by the remote method invocation technique. However, in principle there is no need for the server to be implemented in JAVA. A better approach would be to implement the GA server as a plain CORBA server in a native language such as C++. This would provide a significant performance boost on the server side and, furthermore, would allow easier access for other applications due to the open CORBA standard.

The system has been implemented with the help of Java Development Kit (JDK) 1.1.5. It has been tested on three different kinds of operating systems: MS-Windows 95, Solaris 2.5.1, and FreeBSD 2.2.2. Using the Java virtual machines included in the JDK 1.1.5 (FreeBSD and Win95) and JDK 1.1.3 (Solaris) the system appears to work fine. However, no tests have been done on any versions newer than JDK 1.1.5. To take advantage of the new system improvements of Java 2 (such as the aforementioned CORBA architecture), the entire system would have to be migrated.

Although computer performance increases continuously, Java—as an interpreted language—is still slow and its computational power therefore very limited compared to compiled program code. As a consequence, we chose to fix the size of the artificial tissue to 12 £ 12, even if any arbitrary size can be chosen. Even if the emergence of the JIT compilation techniques by themselves improved the system's performance, more computational power on the client side seems to be necessary to overcome the limitations mentioned. Increasing the size of the tissue would have several advantages and allow the growth and evolution of larger networks. Also, artifacts caused by the boundary elements and the rough granulation (which affects the smooth diffusion of substrates) might be avoided in such larger tissues.

6 Conclusions

There are many steps that remain to be taken in the future. First of all, the system has to be evaluated in more depth. Experiments should be performed that attempt to evolve simple ANNs for a given world, rather than the simple examples presented here that were largely world independent. Finally, evolution experiments should be started in earnest to validate the central premise of this work, namely that complex networks with novel characteristics can emerge within the present setting.

Apart from this, there is no doubt that the genetic language (the artificial chemistry), invented rather spontaneously and without much rigorous and quantitative testing, can be optimized. For example, it would be interesting to see if the set of condition atoms can be reduced to a minimum. Simple considerations about the size of genetic space and the fitness landscape it gives rise to [3] show that reducing the number of different condition atoms would make the computational overhead much less daunting.

Further, certain enhancements might be suitable for future versions of the model. Fleischer [8] showed the importance of a dynamic morphology for the development of natural-like systems. At present, our model does not feature morphology. Cells are located at the same place from their birth on. Cell death does not exist. Additional expression commands that allow, for example, cells to move along a chemical gradient would be a good way to incorporate morphology into the model. Cell death could be another interesting issue that would bring the model closer to real neural development.

One of the driving principles of this model is that of locality. First and foremost, the behavior of each neuron is determined by local concentrations. A more sophisticated model would take other local structures (besides neurons) into account as well. Dendrites and axons exist in the current model only as shortcuts between neurons. They do not, at present, have any local structural features. For future versions maybe a separate part of the genome could describe the behavior and structure of dendrites and axons. If so, synaptic changes could be coded and based on local conditions rather than *cell-global* conditions. For instance, external proteins could then directly affect synapses at the site of the synapse just like neuromodulators do in vivo.

During natural cell division, cellular substrates are not divided equally between precursor and progeny cell. Differences in both cells can lead to different gene expressions that are important for development based on cell lineage. Currently, this is not the way cell division takes place in the model. Even more unrealistically, the progeny cell simply inherits all substrates of the grid element that it replaces. This could be refined in future versions.

To bring the model closer to biology, a possible consideration is to consider including cell membranes in the model. At the moment, a cell membrane exists only in the sense that internal proteins and neurotransmitters cannot diffuse out of the cell. All other types of substrate can go into and out of the cell plasma. Natural cell membranes are known to be functional (they trigger cell-internal events) and highly selective, and therefore they influence the cell's information processing to a high degree. It is, however, difficult to tell if including this would improve the model or if computational performance would only suffer even more from the increase of complexity.

This work introduced a developmental and behavioral model based on artificial gene expression that shares key properties with natural neural development. The heart of the model is the coding scheme of genotypes. The genetic code allows for the description of information-processing structures that show behavior similar to certain natural systems, within a coding scheme that borrows heavily from the gene-regulation paradigm. Without undue effort, genotypes for ANNs can be constructed that are believed to be essential [12, 15] for higher self-organizing information-processing systems, such as deterministic structure development, self-limiting cell growth, regenerative structures, growth following gradients of diffusive substrates, computation of logical functions, pacemaker behavior, and simple adaptation (sensitization, habituation, associative classical conditioning). Assuming that these phenomena are essential for natural intelligence, it may be surmised that the model allows—in principle—the description of more complex information-processing structures.

Our primary thrust in reducing the gap between the physiology of neurons and the abstract models that are supposed to model them was to reduce the mathematical abstraction of neurons by reverting to more low-level structures, in the hope that abstract neurons would *emerge*. This is the classical artificial life approach that has served well in many other applications. Unfortunately (but predictably) the freedom gained is associated with large genotypes (noncompact descriptions) and an exponentially large genetic search space. We described how an evolutionary search for genomes coding for information-processing network structures can be distributed in a *platform-independent* manner such that the unused CPU power of the Internet can be tapped to search for ANNs that reduce the gap between the abstract models and neurophysiology.

Experiments on a large scale remain to be done and the model needs to become more sophisticated. However, this approach shows that physiology and architecture can be encoded in one genotype, such that it gives rise to the development of ANNs based on local interactions only. Further, the *platform-independent* and *fully WWW-distributed* system is a new and sophisticated strategy to approach problems based on evolutionary search.

Acknowledgments

This work was supported in part by the NSF under grant PHY-9723972, as well as a fellowship of the *Studienstiftung des Deutschen Volkes* to J.A.

References

1. Ackley, D. H., & Littman, M. L. (1992). Interactions between learning and evolution. In C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen (Eds.), *Artificial life II* (pp. 487–509). Redwood City, CA: Addison-Wesley.
2. Adami, C. (1998). *Introduction to artificial life.* Santa Clara, CA: TELOS Springer.
3. Astor, J. C. (1998). *A developmental model for the evolution of artificial neural networks: Design, implementation, and evaluation.* Diploma thesis, University of Heidelberg, Germany.
4. Bower, J. M. (1995). *The book of GENESIS: Exploring realistic neural models with the GEneral NEural SImulation System.* Santa Clara, CA: TELOS Springer.
5. Britten, R. J., & Davidson, E. H. (1969). Gene regulation for higher cells: A theory. *Science, 165*, 349–357.
6. Britten, R. J., & Davidson, E. H. (1971). Repetitive and non-repetitive DNA sequences and a speculation on the origins of evolutionary novelty. *Quarterly Review of Biology, 46*, 111–138.
7. Fahlman, S. E., & Lebière, C. (1990). The cascade-correlation learning architecture. In D. S. Touretzky (Ed.), *Advances in neural information processing systems 2* (pp. 524–532). San Mateo, CA: Morgan Kaufmann.
8. Fleischer, K. (1995). *A Multiple-mechanism developmental model for defining self-organizing geometric structures.* Unpublished doctoral dissertation, California Institute of Technology.
9. Fritzke, B. (1992). Growing cell structures—A self-organizing network in *k* dimensions. In I. Aleksander & J. Taylor (Eds.), *Artificial neural networks II* (pp. 1051–1056). Amsterdam: North-Holland.
10. Gruau, F. (1992). Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. In D. Whitley & J. D. Schaffer (Eds.), *Combination of genetic algorithms and neural networks.* IEEE Computer Society Press.
11. Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, 79*, 2554–2558.
12. Kandel, E. R., & Schwartz, J. J. (1991). *Principles of neural science.* Amsterdam: Elsevier.
13. Kitano, H. (1990). Designing neural network using genetic algorithm with graph generation system. *Complex Systems, 4*, 461–476.
14. Kitano, H. (1998). Building complex systems using developmental process: An engineering approach. *Lecture Notes on Computer Science, 1478*, 218–229.
15. Koch, C., & Poggio, T. (1982). Biophysics of computation: Neurons, synapses and membranes. In G. M. Edelman, W. E. Gall, & W. M. Cowan (Eds.), *Synaptic function* (pp. 637–697). New York: Wiley.
16. Langton, C. G. (Ed.). (1995). *Artificial life: An overview.* Cambridge, MA: MIT Press.
17. Michel, O. (1996). An artificial life approach for the synthesis of autonomous agents. In J. M. Alliot, E. Lutton, & E. Ronald (Eds.), *Artificial evolution*, Lecture Notes in Computer Science 1063 (pp: 220–231). Berlin: Springer.
18. Pavlov, I. P. (1927). *Conditioned reflexes: An investigation of the physiological activity of the cerebral cortex.* London: Oxford University Press.
19. Pitts, W., & McCulloch, W. S. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*, 115–133.

20. Postlethwait, J. H., Yan, Y. L., Gates, M. A., Horne, S., Amores, A., Brownlie, A., Donovan, A., Egan, E. S., Force, A., Gong, Z. Y., Goutel, C., Fritz, A., Kelsh, R., Knapik, E., Liao, E., Paw, B., Ransom, D., Singer, A., Thomson, M., Abduljabbar, T. S., Yelick, P., Beier, D., Joly, J. S., Larhammar, D., Rosa, F., Westerfield, M., Zon, L. I., Johnson, S. L., & Talbot, W. S. (1998). Vertebrate genome evolution and the zebrafish gene map. *Nature Genetics, 18,* 345–349.

21. Vaario, J. (1994). Modeling adaptive self-organization. In R. Brooks & P. Maes (Eds.), *Proceedings of Artificial Life IV* (pp. 313–318). Cambridge, MA: MIT Press.

22. Vaario, J., & Shimohara, K. (1995). On formation of structures. In *Advances in artificial life.* Lecture Notes in Computer Science 929 (pp. 421–435). Berlin: Springer.