Evolution and Development of Control Architectures in Animats

Jérôme Kodjabachian and Jean-Arcady Meyer

Ecole Normale Supérieure
Groupe de BioInformatique
75230 Paris cedex 05, France
E-mail: kodjaba@wotan.ens.fr, meyer@wotan.ens.fr

Abstract

This paper successively describes the works of Boers & Kuiper, Gruau, Cangelosi et al., Vaario, Dellaert & Beer, and Sims, which all evolve the developmental program of an artificial nervous system. The potentialities of these approaches for automatically devising a control architecture linking the perceptions and the actions of an animat are then discussed, together with their possible contributions to the fundamental issue of assessing the adaptive values of development, learning and evolution.

Keywords

Evolution – Development – Animats – Sensorimotor Control – Learning

1 Introduction

An animat [1, 2, 3] is an artificial organism — either a simulated animal or an animal-like robot — the structure and functionalities of which are based substantially on mechanisms observed in real animals. It is usually equipped with sensors, with actuators, and with a behavioral control architecture that relates its perceptions to its actions and allows it to survive in its environment.

Usually, an animat's control architecture is fixed by a human designer. However, because the field of animat research still lacks any systematic comparisons — where a specific architecture would be confronted with various survival problems or where various architectures would have to cope with the same environment [4, 5, 6] — such a procedure relies more on the designer's intuition or technical idiosyncrasies than on basic principles. Actually, several researchers [7, 8] even doubt that such principles will ever prove to be efficient for designing truly autonomous systems, i.e., systems that have to survive in numerous challenging circumstances that are impossible to predict. Therefore, they advocate the use of automatic designing procedures that would bypass human intervention insofar as possible, and that would adapt the control architecture of an animat to the specific environment it lives in, and to the specific survival problems it has to solve.

It turns out that Nature has invented three such automatic designing procedures, namely those of evolution, development and learning. The first two determine the overall organization of a given organism, while the third is used to fine-tune the organism's adaptation to environmental constraints. Although these processes occur on different time scales, they certainly interact in complex ways, which are still not fully understood.

Be that as it may, because the study of animats is grounded in biology, it is not surprising that several research efforts have already let an animat's control architecture evolve, or learn,

or simultaneously both evolve and learn [4, 5, 6]. This paper will focus on the evolutionary design of neural networks, a particular class of control architectures liable to exhibit a wide variety of dynamic behaviors and that may prove to be particularly suitable for automatic design approaches because the low-level, sub-symbolic primitives they are built upon — the elementary neurons — are easily recombined into changing or growing configurations. However, this paper will argue that, to be effective, such an approach needs to be coupled with a non-trivial developmental process that allows complex morphologies to be specified by simple programs. A few such applications — which combine development, evolution and, possibly, learning — have recently been published and will be reviewed herein.

In the following section, we introduce the basic principles of evolutionary algorithms, a set of search algorithms that simulate an evolutionary process. We then discuss the encoding problems that are encountered when evolving neural networks with such methods, problems to which developmental processes appear to provide promising solutions. In the three next sections we review several applications that combine development and evolution. First, we describe two methods that use a simple developmental scheme based on rewriting rules and that exhibit some interesting properties. Then, we describe two models incorporating a process of axonal growth. Finally, we present two research efforts that allow the morphology of an animat to co-evolve with its control architecture. The paper ends with a discussion of the foreseeable difficulties and potentialities of such approaches, notably for autonomous robotics.

2 Simulation of evolution

Several optimization algorithms are inspired by the mechanisms of evolution. Fogel [9] classifies them according to the description level of the genetic mechanisms they are based upon. *Genetic algorithms* [10] and *genetic programming* [11], which make a distinction between a genotype level and a phenotype level, implement the most fine-grained version of these mechanisms. *Evolutionary programming* [12], on the contrary, relies upon high-level mutation operators that directly modify phenotypical traits. Finally, another family of algorithms — *evolution strategies* [13] — can be situated between the preceding, as it operates on vectors of scalars, an intermediate level of representation between the elementary genes-as-bits used in genetic algorithms and the sophisticated traits of evolutionary programming. Another classification has been suggested by Angeline [14], and relies upon the fact that the structures manipulated are either static or dynamic. According to such a classification, genetic algorithms and evolution strategies belong to the first class, genetic programming belongs to the second, and evolutionary programming belongs to either, depending upon the corresponding application.

In this section, we introduce genetic algorithms and genetic programming, two paradigms on which the works reviewed in this paper are substantially based. Then we discuss the encoding problems that occur when the topology of a neural network is evolved.

2.1 Genetic algorithms and genetic programming

Genetic algorithms are randomized search algorithms based on the mechanisms of natural selection and genetics. In their standard version [15], they operate on a set of bit-strings representing the *genotypes* of various individuals in a population. Each bit-string — that is often called a *chromosome* — can be translated into a *phenotype*, which can be a mere parameter or a whole architecture, and which represents a possible solution to a given problem. Each of these chromosomes must be assigned a *fitness* that assesses the corresponding solution. The application of

the genetic algorithm accordingly consists in causing the population to evolve from generation to generation while rendering the probability of reproduction of each chromosome proportional to its fitness and using *genetic operators* such as *mutation* or *recombination* to give rise to new solutions in the population. Under these circumstances, this type of evolutionary process causes chromosomes of ever-increasing fitness to be generated until the optimal value is reached, or sufficiently nearly so for all practical purposes.

An iteration of the algorithm is divided into three stages:

1. The fitnesses of the different chromosomes are assessed by an evaluation procedure (evaluation stage).

2. A subset of the population is selected, depending upon the fitness values (selection stage).

3. The genotypes corresponding to the selected phenotypes are modified by genetic operators and form a new population, the next generation (reproduction stage).

Iteratively, successive generations are produced. The combined influences of selection — that favors the reproduction of the fittest individuals — and variation — that is caused by the application of the genetic operators — allow for a directed search in the set of all possible genotypes. As the genetic operators are stochastic, the search is said to be randomized.

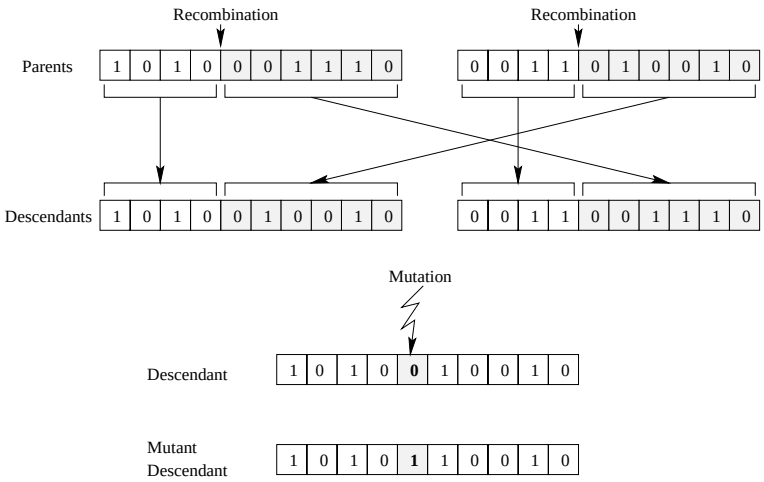

Figure 1. Recombination and mutation in the standard genetic algorithm.

Several types of genetic operators exist. The operator of recombination — also called *crossing-over* — takes two genotypes and exchanges parts between them. It is generally considered the most important operator, because the long-distance jumps it produces allow for a global exploration of the genotype space. Another important operator, mutation, randomly flips bits in the bit-strings. It is usually considered as a novelty source that is used with a low frequency in order to prevent *alleles* (0's or 1's in specific positions within a chromosome) from being ruled out of the population. These two operators are illustrated in Figure 1.

Genetic algorithms can be considered as optimization algorithms acting upon the fitness function because they search for an individual associated with an optimal evaluation. These algorithms are different from other random search techniques in that they evaluate in parallel a population of different individuals that can be modified or recombined at reproduction time. Also, genetic algorithms differ from standard numerical analysis techniques in that they do not assume any continuity, nor the existence of derivatives, for the fitness function.

The theory of genetic algorithms [10] helps to understand the reasons why these methods are efficient and, in particular, to assess the specific role of the recombination operator. The so-called *schemata-theory* is based on the notion of *schema* (or *similarity template*) and its main theorem states that, if the size of the population is N, then an order of N^3 schemata are processed by the genetic algorithm. This property is called *implicit parallelism*. Then, if short schemata implementing potentially useful functionalities exist in the population, they can be efficiently used as *building blocks* by the recombination operator to form fitter individuals, which will constitute the next generation.

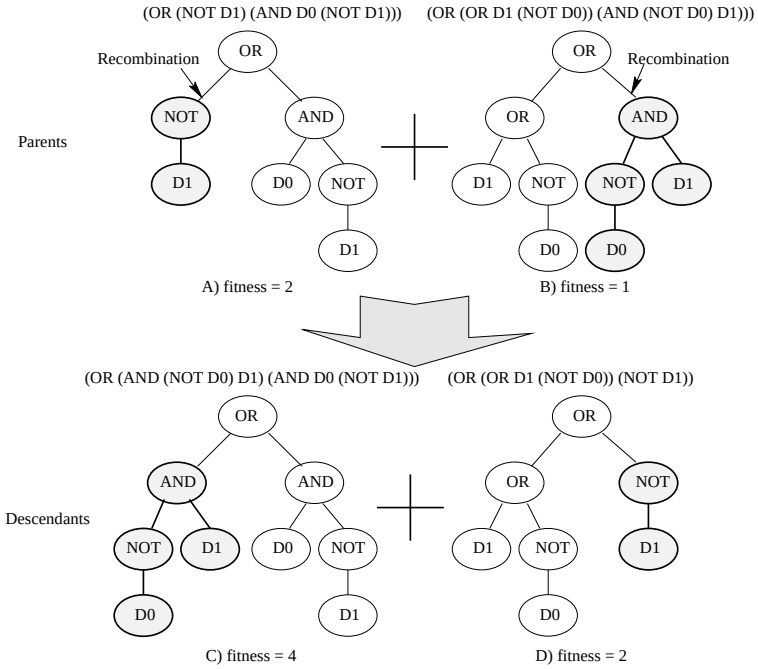

Figure 2. The discovery of a program able to evaluate the XOR of two logical variables D0 and D1 by genetic programming. The application of the recombination operator, which randomly exchanges sub-trees between two incorrect programs, (A) and (B), produces programs (C) and (D). Program (C) is a correct solution for the XOR problem because its upper node gives the desired value in each of the four possible combinations of D0 and D1. The fitness values displayed are the numbers of correct answers, out of a maximum of four, given by the different programs. These programs are expressed either as parenthesized expressions or as equivalent trees.

Another family of evolutionary techniques, genetic programming, operates on Lisp-like S-expressions instead of bit-strings [16, 11, 17]. An S-expression corresponds directly to the sparse-tree created by a Lisp-compiler at compilation time and can be represented either as a parenthesized expression or as a tree. Thus, sub-S-expressions (a single symbol — i.e., a terminal leaf — or a parenthesized functional sub-S-expression — i.e., a sub-tree) are meaningful building-blocks that can be easily manipulated. This particular syntax of the Lisp language allows the main genetic operators to be redefined in an efficient way. Mutation consists in modifying a sub-S-expression. Recombination consists in exchanging sub-S-expressions between two programs, as illustrated on Figure 2. In some applications, an individual is encoded by several S-expressions instead of only one, in which case recombination can be applied either between corresponding S-

expressions of the two parent programs, or in an unconstrained manner. All these manipulations have the advantage of producing syntactically valid S-expressions. The size of the programs, however, can vary due to differences in the sizes of the sub-expressions manipulated.

2.2 Encoding schemes for neural networks

Both genetic programming and genetic algorithms have already been applied to the synthesis of neural networks that can be used as control architectures in animats. These attempts have helped to single out the corresponding main difficulties.

In [11], for instance, a specific application of genetic programming to the design of a network architecture is presented. However this method evolves highly constrained architectures, because a separate sub-network — coded in a corresponding chromosome-tree — has to be designed for each output unit in a feed-forward network. Thus, the method does not exploit the sharing of internal units between different parts of the network, which seems to be a desirable characteristic of neural networks.

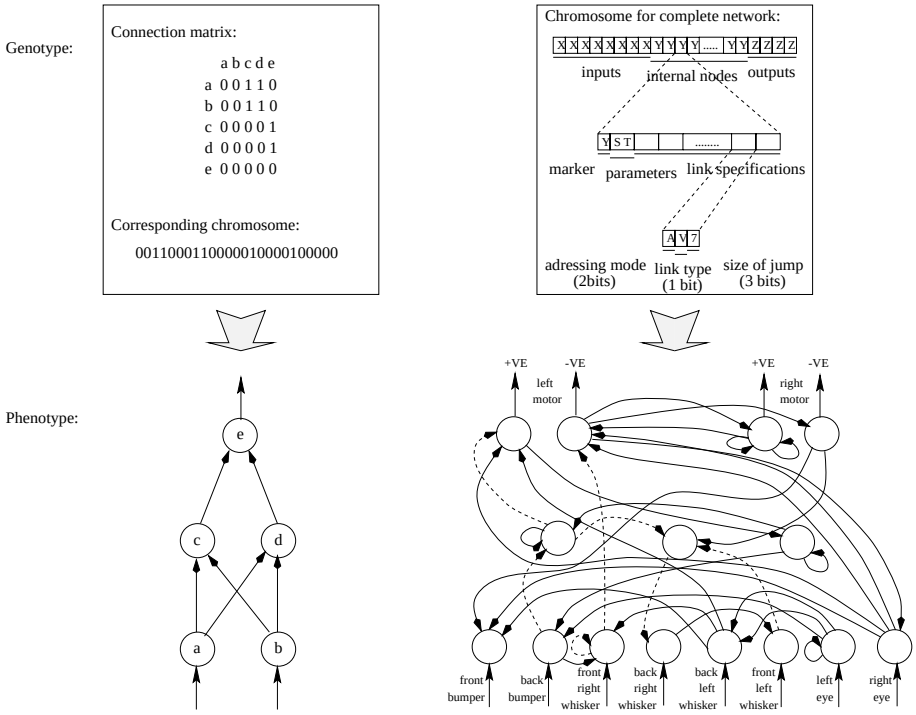

Figure 3. Two types of direct encoding schemes. *Left*: Simple encoding of a neural network topology. The weights are learned by a specific algorithm, such as back propagation (after [18]). *Right*: Variable length encoding of both the architecture and the weights of an animat's nervous system. Although input (sensors) and output (motors) nodes are imposed by the programmer, because the genetic encoding allows for variable numbers of internal nodes (neurons) and links (connections), the resulting control architecture does not necessarily use each of the available sensors or motors. Within such an encoding scheme, the addressing mode can be forward or backward, and relative or absolute. Two types of links can also be used: normal or veto (after [8]).

Likewise, genetic algorithms have often been applied to the design of neural networks (see [18] for a review). The earlier applications were limited to the optimization of the synaptic

weights within fixed architectures (e.g., [19]). Later, several researchers have also genetically coded the topology of a network using *direct encoding schemes*, in which there is a one-to-one correspondence between the data in the genotype and specific parts — neurons and connections — in the phenotype. The *connection constraint matrix* of Miller et al. [20] is a simple example of such an approach. Figure 3 compares this kind of approach with a more sophisticated variable-length scheme that specifies both the architecture and the weights of an animat's nervous system [8]. A less direct encoding scheme, which specifies the overall architecture of a network only in terms of *areas* — instead of neurons — and *projections* — instead of connections — has been proposed by Harp et al. [21]. Thus, the exact connectivity of a network is randomly generated according to the projection parameters that determine both target regions within areas and connection densities. However, all such methods were quickly seen to be hampered by several limits, such as their lack of scalability or the absence of modularity in the resulting architectures.

In order to cope with these limits, several researchers have proposed to get rid of direct genotype-to-phenotype mappings and to use complex, non-linear developmental processes that might exhibit desirable properties [21, 20, 22, 8, 23]. Among these properties, that of scalability is particularly interesting. In [24], Kitano regrets that direct encoding schemes do not generally scale well when the size of the desired network grows, because the size of the code is linearly related to the size of the network. To solve this problem, Kitano proposes an evolutionary algorithm that manipulates dynamic rewriting rules that allow the connectivity matrix of a neural network to develop.

Modularity is another property also concerned with the reduction of the size of the genotypes, which appeared to be missing in direct encoding schemes [25, 8]. According to Gruau's definition [25], an encoding scheme is modular if the genotype can be decomposed into some parts that specify the organizations of sub-networks, and other parts that describe how to interconnect these sub-networks. Thus the same pattern of connectivity can be expressed several times within the same network.

In the three following sections, six recent applications combining a developmental process with an evolutionary process are described.

3 Rewriting rules

The research efforts described in this section evolve genotypes made up of sets of rules that are used to rewrite symbols whose meaning is such that, at the end of the rewriting process, a functional neural network is obtained. In the approach of Boers and Kuiper, the synaptic weights of the network still have to be learned, while in Gruau's approach they are specified by the genotypes. A variation of this second approach also allows modular networks to be developed.

3.1 Boers and Kuiper

The work of Boers and Kuiper [26] combines a genetic algorithm and a learning procedure with an L-system grammar [27] that models development. Basically, the genetic information on which the genetic algorithm operates codes for a set of production rules which are applied to an axiom over a number of iterations. The resulting string is transformed into a structural specification for a classical feed-forward neural network. The weights of this network are trained by back-propagation, which provides a fitness estimate that is fed back into the genetic algorithm and used to assess whether the network should be eliminated or passed on to the next generation.

The strings used in this work are made up of 16 symbols from the alphabet {A-H, 1-5, [,]} U {,}. A letter (A-H) designates a specific neuron in the network and two adjoining letters are automatically connected feed-forward. If two letters are separated by a comma (,), no connection is made. Modules can be created by grouping neurons or other modules between square brackets ([,]): two adjoining modules are connected in such a way that all output neurons from the first module are connected to all input neurons from the second module, and two modules separated by a comma are not connected.

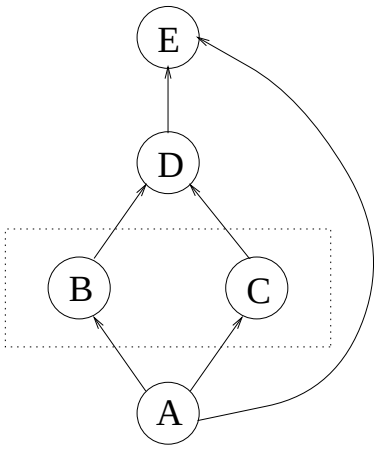

Figure 4. The string [A2[B,C]D]E developed (after [26]).

1 : A → BBB
2 : B > B → [C, D]
3 : B → C
4 : C < D → C
5 : D > D → C1

Figure 5. Production rules sample (after [26]).

Single digits are used to denote a skip within the string. For instance, the string [A2[B,C]D]E codes for the network of Figure 4, where neuron A is connected to neurons B and C because both are input neurons in the [B,C] module — i.e., they receive no connection from other neurons in the same module. Neuron A is also connected to neuron E because the connection skips both module [B,C] and neuron D.

The L-system used for generating such strings is a 2L-system, in which every production rule can have both left and right contexts and is therefore divided into four (possibly empty) parts: $L < P > R \rightarrow S$.

Basically, such a rule means that sub-string P (the predecessor) should be replaced by sub-string S (the successor) if P is connected to every neuron described in L (left- or lower-level context) and in R (right- or upper-level context). Thus, if the five production rules of Figure 5 are applied to a single original neuron A — given as an axiom — (Figure 6a), the string BBB is generated after one rewriting step (Figure 6b). During the second rewriting step, the first (bottom) and the second (middle) B's are rewritten using rule 2, because they both are connected to a higher B. On the contrary, the third (top) B — which has no connection with a higher B — is rewritten according to rule 3, instead of rule 2 (Figure 6c). Likewise, during the third rewriting step, the first (bottom) D is rewritten according to rule 5 and the second (middle) according to rule 4. As no more rules apply, the final network obtained corresponds to string [C, C1] [C,C] C, shown in Figure 6d.

To separate the constituent parts of each production rule, Boers and Kuiper used a special symbol (an asterisk) and, to relate each of the 17 possible symbols in a production rule to the

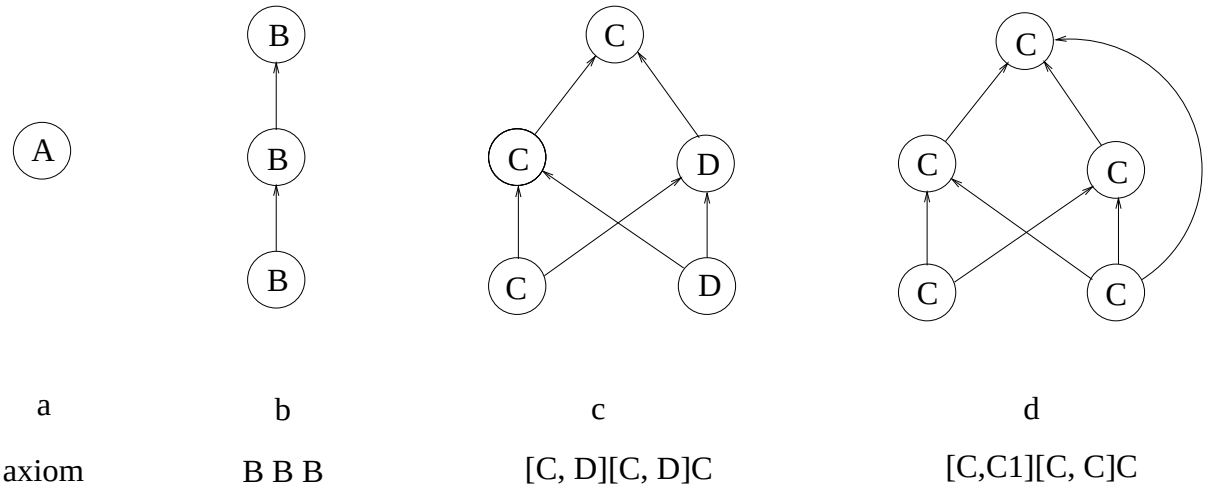

Figure 6. The development of a neural network with rules from Figure 5 (after [26]).

	00	01	10	11	
00	3 3 * *	[[[[	D D 2 2	]] 2 5	00 01 10 11
01	* * * *	1 1 1 1	E E F F	]]]]	00 01 10 11
10	2 2 2 4	A A A A	G G H H	[[ ]]	00 01 10 11
11	, , , ,	B B B B	* * [[	C C C C	00 01 10 11

Figure 7. The genetic code used in [26]. For example, the symbol corresponding to string 100100 is the first A in the Table.

genetic information processed by the genetic algorithm, they used the genetic code described in Figure 7. Thus, in this application, the genetic code relates 17 symbols to 64 6-bit strings, instead of relating 20 amino-acids to 64 triples with four bases.

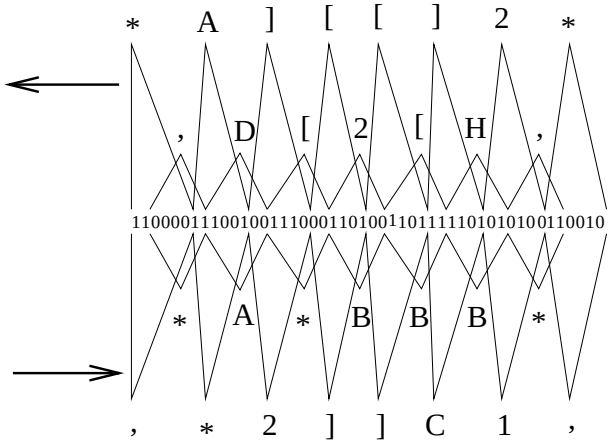

Figure 8. Four possible translations of a given chromosome in Boers and Kuiper's application (after [26]).

Furthermore, the genetic information on a given chromosome can be read in twelve different ways — starting at any of the first six bits and reading forward, or starting at any of the last six bits and reading backwards — thus providing the genetic algorithm with a much higher level of implicit parallelism than in traditional applications. Figure 8, for instance, describes four different translations of a chromosome with a length of 48.

Finally, the software developed by Boers and Kuiper also contains several functions capable of repairing faulty strings, i.e. strings with extraneous brackets, useless commas, or succeeding digits.

To our knowledge, this software has so far been used in only a few very simple applications. For instance, it has evolved neural networks capable of solving the XOR problem or of recognizing handwritten digits 0,1 ... 9 presented on a 5x5 grid. However, it could prove useful for designing the architecture of feed-forward networks that are trained by supervised learning procedures and that could be used to control the behavior of an animat. Such a procedure has been used, for instance, by Pomerleau [28] to control the NAVLAB, i.e., the autonomous vehicle of CMU.

3.2 Gruau

The work of Gruau [29, 30] also encodes a rewriting grammar in a chromosome. However, this encoding scheme — called *cellular encoding* — rewrites neurons instead of symbols. In its simplest version, it is used to develop feed-forward networks of Boolean neurons with integer thresholds and +1 or -1 connections, but more elaborate versions of this encoding scheme [31, 32] can deal with more complex neurons and connectivities.

In Gruau's model, each cell in a developing network has a copy of the chromosome that codes the developmental process, and each cell reads the chromosome at a different position. The chromosome is represented as a grammar tree, with ordered branches whose nodes are labeled with character symbols. These character symbols represent instructions for cell development

that act on the cell or on connections that fan-in to the cell. During a given step of the developmental process, a cell executes the instruction referenced by the symbol it reads and moves its reading head down in the tree. Depending on what it reads, a cell can divide, change some internal registers and finally become a neuron. For instance, when a cell reads and executes the *sequential division* (denoted by S), it divides into two linked cells: the first daughter inherits the input links, the second daughter inherits the output links of the parent cell. When a *parallel division* (denoted by P) is executed, both daughter cells inherit the input and output links from the parent cell. Since a given cell yields two daughter cells, S and P nodes are of arity two: the first daughter moves its reading head to the chromosome's left subtree and the second daughter moves its head to the right subtree. Finally, when a cell divides, the values of the internal attributes of the parent cell are copied into the daughter cells.

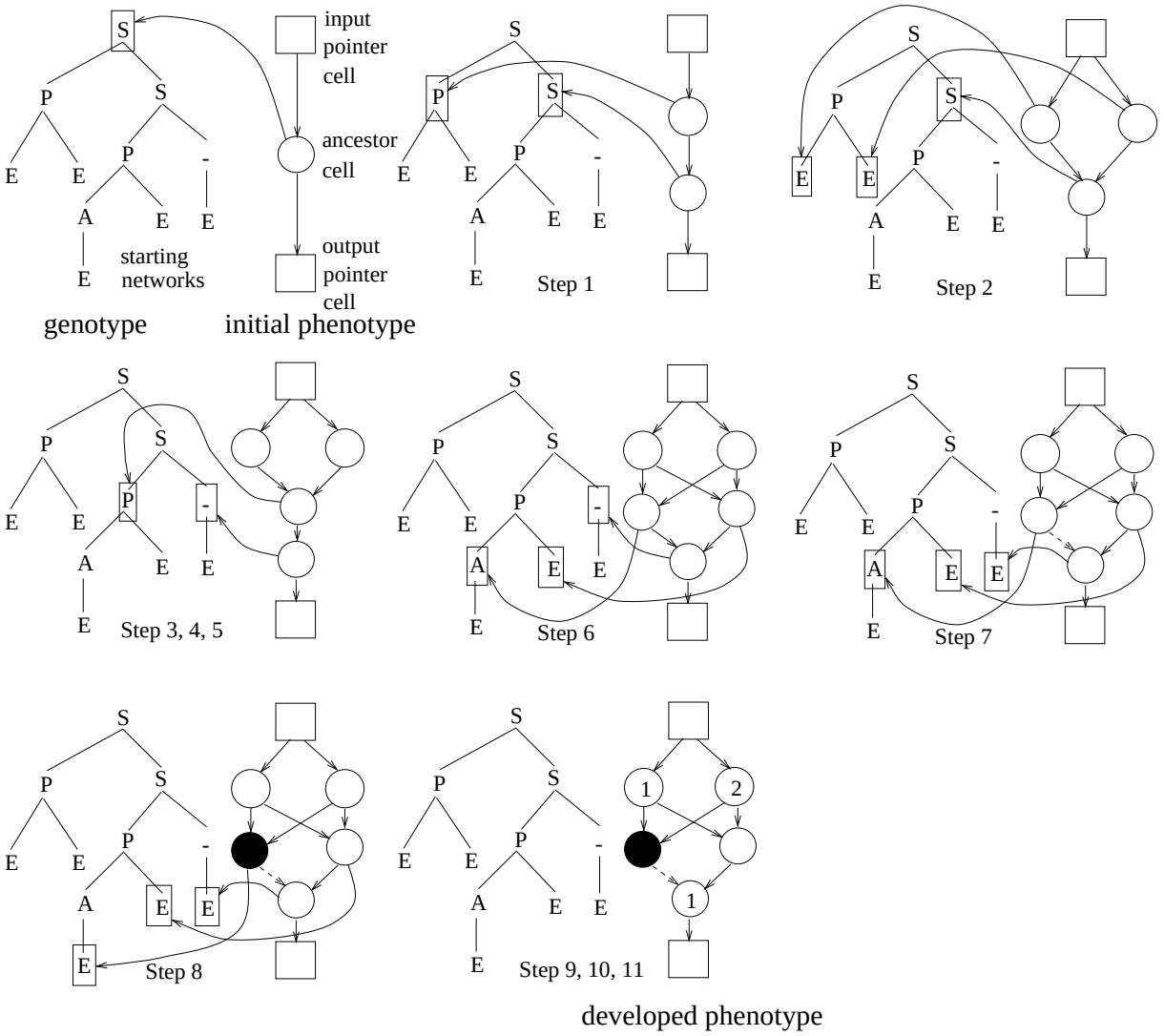

Figure 9. Cellular encoding and development of a XOR network. Explanations are provided in the text (after [29]).

Other symbols change the values of internal registers in the cell. Such registers are used

during development — like the *link register*, for instance, which points to a specific fan-in link or connection into a cell — while others determine the weights and thresholds of the final neural network. Thus, symbols I and D respectively increment and decrement the value of the link register, causing it to point to a different connection. Likewise, symbols A and O respectively increment and decrement activation thresholds, and symbols + and - respectively set to +1 and -1 the weight of the input link pointed by the link register. The *ending program* symbol E causes a cell to lose its reading head and become a neuron.

Figure 9 represents the development of a XOR network. Circles represent active cells or neurons, while rectangles represent reading heads. Empty circles correspond to thresholds set to 0, black circles correspond to thresholds set to 1. Squares represent input/output pointer cells. Solid connections have a weight of 1, dashed connections have a weight of -1.

Since Gruau's chromosomes have the same structure as those used within the genetic programming paradigm, they can be subjected to the same kind of genetic operators, notably to mutations and recombinations.

Cellular encoding has been used by Gruau [32] to evolve a neural network capable of controlling the motion of a six-legged animat. This problem has already been solved by Beer and Gallagher [33] who, instead of forming a locomotion controller by fully interconnecting six individual leg-controllers, took advantage of the various symmetries that such a controller is supposed to exhibit and devised a controller made of six copies of the same sub-network. Gruau solves a slightly simpler version of the problem, but does not help the evolutionary algorithm by using any a priori knowledge about symmetries (except for the order of presentation of the different inputs and outputs). Instead, symmetries are discovered and exploited by the developmental process, because such a process is capable of generating a sub-network that solves a sub-problem, then of producing and combining copies of this sub-network, to build a higher-level network that solves the problem. The genome-splicing technique advocated by Koza [17] seems especially useful for such a modular approach. The control architecture of Figure 10 consists in a sub-network repeated three times and allows the animat to exhibit a tripod gait. Each sub-network contains a neuron with a recurrent connection that acts as a latch register and makes the three controllers work in phase, while insuring that the two legs controlled by a specific controller work in anti-phase.

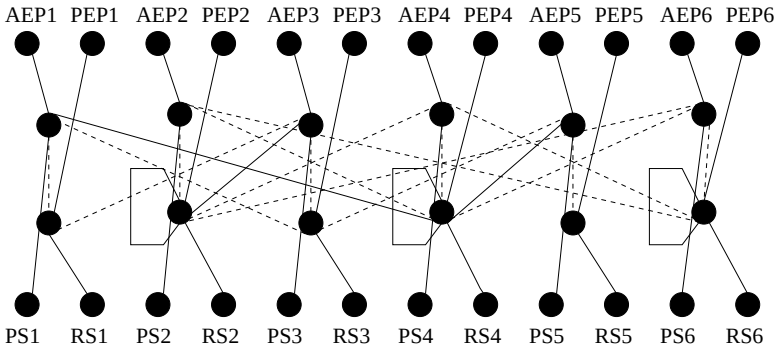

Figure 10. The neural network evolved in Gruau's approach that controls tripod gait. AEP and PEP are sensory input units detecting anterior extreme positions and posterior extreme positions of the legs. PS and RS are motor output units controlling the power stroke and the return stroke (after [32]).

Gruau and Whitley [30] have added a variety of Hebbian learning to cellular development

and evolution. In particular, following Hinton and Nowlan [34] and Belew [35], they have compared results obtained with fitness evaluations depending on a developed neural network alone to results obtained with fitness evaluation depending on a developed neural network with some of its weights changed by a learning procedure. It thus appears that such a modification changes the fitness landscape explored by the genetic algorithm and, eventually, accelerates the speed of evolution — a result known as the *Baldwin effect*. Likewise, Gruau and Whitley have studied how the so-called *developmental learning* could affect evolution. Such learning can occur when some recursive encoding is used by the cellular encoding method, thus allowing a given subtree of the chromosome to be read and executed repeatedly. In such circumstances, indeed, it is possible to learn and change the weight of a connection between two iterations of the recursive loop.

It should be stressed that neither the Baldwin effect, nor the developmental learning, pass the values of learned weights from parents to offspring and, thus, that they do not implement any Lamarckian inheritance of acquired characters.

4 Axonal growth

The approaches to be described in this section simulate a process of axonal growth that determines the connectivity of the networks. While, in Cangelosi et al., the growth process depends upon the genotype only, in Vaario's work it can be regulated by the environment.

4.1 Cangelosi, Parisi and Nolfi

The work of Cangelosi, Parisi and Nolfi [36] is concerned with the evolution of animats equipped with *motivational units* that can inform them about some internal needs. The control architecture of each animat is a bidimensional network that develops from an initial egg cell — during five cell division and migration cycles, followed by five cycles of axonal growth.

Each cell contains three varieties of information coded in its genome:

1. The type of the cell. There are 16 cell types.
2. Several parameters characterizing its neuronal properties:
   - the branching angle of the neuron's axon (there are five branching cycles; each branching is binary; all branchings have the same angle);
   - the length of a branching segment of the neuron's axon;
   - the point on the neuron's surface where the axon starts growing;
   - the threshold of the neuron;
   - the weight of the connections emanating from the neuron (all the connections emanating from the same neuron have the same weight).
3. A set of 16 rules of cell division. Each rule has the following form:
   $$\text{Rewrite } TypeN \text{ as } TypeN' + TypeN''$$
   where N ranges from 1 to 16 (a division rule may rewrite a mother cell into a single daughter cell, or even into no cell at all). Each division rule also specifies:
   - a number of changes to be made to the neuron parameters (cf (2) above) when they are inherited by each of the two daughter cells (the changes need not be identical for the two daughter cells);

- the location of the two daughter cells relative to the location of the mother cell.

The developmental process occurs within a bidimensional neural space divided into three horizontal bands. It begins with the egg cell located in the center of the space. At the end of five cell-division and migration cycles, up to 32 cells are obtained, which specialize into neurons with functionalities depending upon their spatial locations and their cell types. Thus, a neuron that ends up in the lower band of the neural space will work as one of the network's sensory or motivational units, depending upon the corresponding cell type. Likewise, a neuron that ends up in the upper band will work as a motor unit, and a neuron that ends up in the intermediate band will work as a hidden unit, the details of their functionalities being determined by their cell types.

At the end of the division and migration cycles, an axonal growth process begins. During five growth cycles, each neuron grows its branching axon according to the corresponding values of its branching-angle and segment-length parameters. If a growing axon touches another neuron, the two neurons become connected. However, only connections belonging to a pathway between a sensory or a motivational unit to a motor unit are considered as functional and participate in the control of the developed animat.

In order to let such control architectures evolve, an initial population of egg cells is generated, the genome of each being constituted randomly (after development, every individual in the population is evaluated during a fixed life-time, according to how successfully its nervous system allows to cope with a given task. At the end of their lives, individuals reproduce selectively according to their fitness, and random mutations are eventually applied to each component of any reproducing genotype.

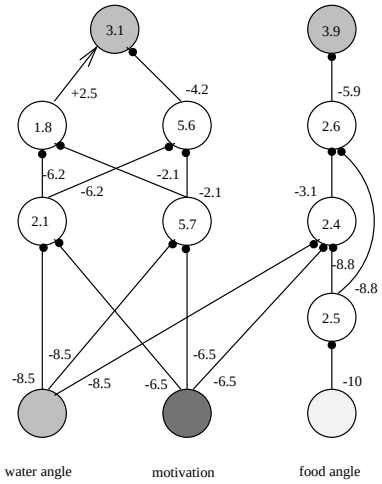

Figure 11. Control architecture (connection weights and unit biases) of an animat capable of reaching the food zone when hungry (motivation unit set to 1) and the water zone when thirsty (motivation unit set to 0) in the work of Cangelosi et al. The two output units binarily encode the four possible motor actions (11 = go forward; 10 = turn left; 01 = turn right; 00 = do nothing) (after [36]).

This scheme has been used to evolve animats living in a bidimensional environment containing two separate randomly-placed zones, one containing food and the other water. Each animat's task is to reach the food zone and to remain there as long as it is hungry, and to reach the water zone and to remain there as long as it is thirsty. Each animat is equipped with a

sensory system that allows it to perceive the direction and the distance of the center of the food zone, and the direction and the distance of the center of the water zone. Motivational units detect whether or not the animat is in a state of hunger or of thirst. An animat is supposed to remain hungry or thirsty until it has spent ten time steps in the food or water zones; thereafter, it shifts to the opposite state. Finally, each animat has a motor system that allows it to go one step forward in the facing direction, to turn 90 degrees left, to turn 90 degrees right, or to do nothing.

A typical run of the evolutionary process is characterized by three periods. The population of the first period is essentially made up of animats that are able to reach the food zone but not the water zone. Their neural control architecture does not include any motivational unit and uses only food sensors in its functional sensory motor pathways. During the second period, a water sensory motor pathway begins to form and motivational units begin to be integrated into the control architecture. Animats capable of reaching the food zone when hungry and the water zone when thirsty appear during the third period, after roughly 300 generations. As exemplified on Figure 11, their control architecture tends to include a single motivational unit and two modules, one for food and one for water. Moreover, these modules tend to use direction information only, architectures that encode both direction and distance, or distance only, having been eliminated from the population in previous generations.

Similar approaches, but with no cell division or migration, are described in [37] and [38]. In the latter, the neural development is influenced by both the genes and the environment, because a neuron is allowed to grow its branching axon only if the neuron's activation variability — which depends upon the variability of the environmental stimulation to the network — exceeds a genetically specified threshold.

4.2 Vaario

Vaario's approach [39, 40, 41] explicitly takes into account environmental effects on the development of neural networks and, like Boers and Kuiper's approach, is inspired by Lindenmayer's systems. However, instead of using linear character strings, it makes use of abstract objects which typically represent artificial cells — each characterized by a set of attributes and a set of production rules to execute. In this model, each cell is actively checking the environment and, on the basis of the corresponding information, executes one or more of its production rules. Cell attributes mostly refer to the concentrations of various chemical elements and enzymes. Production rules are characterized by the set of conditions which must be fulfilled for them to be executable and by the kind of actions they trigger. They are divided into four types:

- *cytoplasm rules*, interpreting the genetic code and modifying the internal state of a cell;
- *membrane rules*, modifying the internal state of a cell according to the interactions between the cell and its environment;
- rules creating a cell;
- rules deleting a cell.

In particular, these rules are used to model various morphogenetic processes, such as cell division, cell fate, axon and dendrite growth, axon guidance and target recognition, cell death, elimination of connections, anatomical plasticity and synaptical plasticity (Figure 12).

For example, the process of axon and dendrite growth depends on the presence of obstacles and of target cells in the environment. Connections bounce against obstacles and climb the

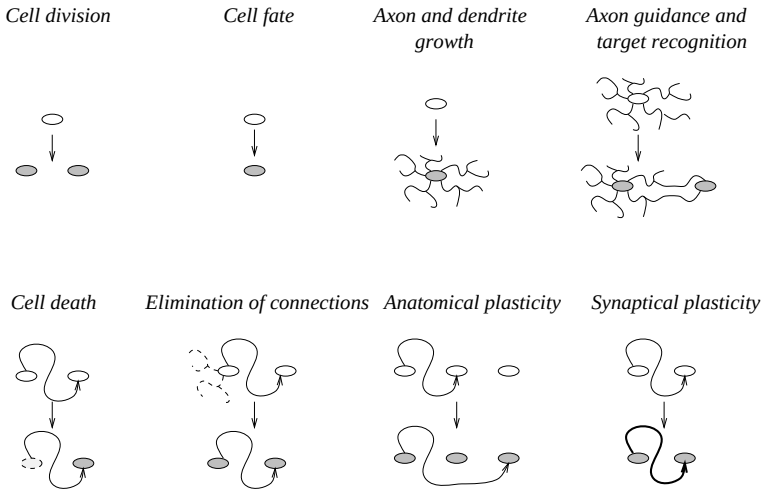

Figure 12. Some morphogenetic processes in Vaario's model (after [39]).

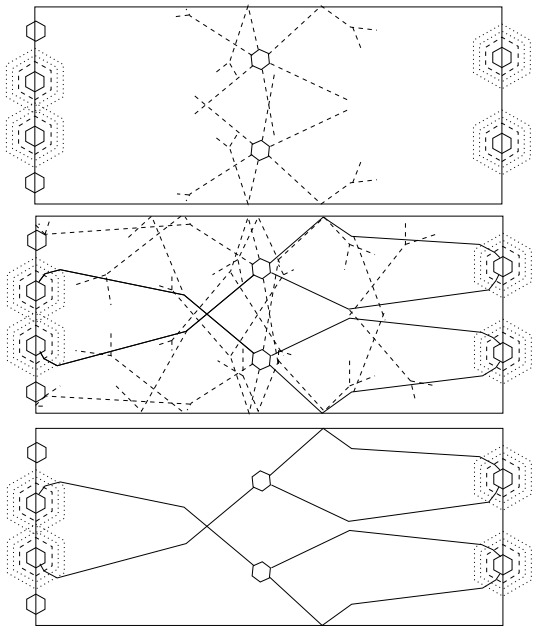

Figure 13. Three developmental stages displayed by Vaario's animat: initial growth (top), initial withdrawal (middle) and final withdrawal (bottom) (after [40]).

gradient fields of the chemical substances emitted by target cells. When a connection finally reaches a target cell, it creates a synaptic connection and stops growing. Moreover, those connections unable to find any target neuron gradually withdraw.

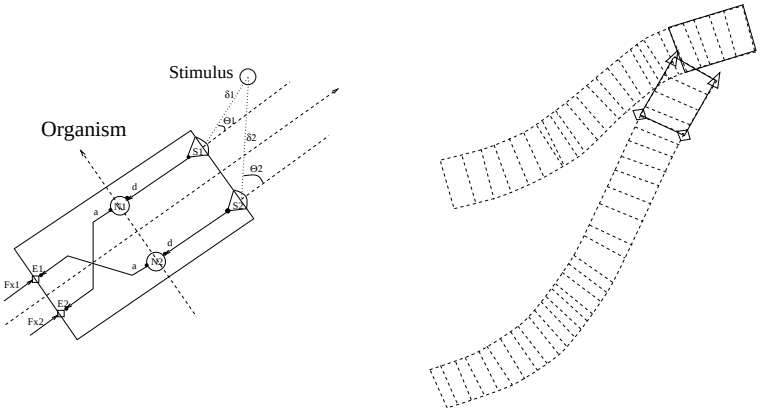

Figure 14. The developed nervous system and the tracking behavior of Vaario's animat (after [40]).

In the current implementation of Vaario's model, the genotype of an animat is not encoded as a bit string, but as a symbolic representation that describes various characteristics of the sensors, of the actuators and of the initial neuron, in a nervous system to be developed [39]. This representation also allows recombination and mutation operations. Thus, several characteristics — like the time to branch, the branching angle and the type of target cells involved in connection growth, or the numbers, positions and properties of the animat's sensors and actuators — are genetically determined.

Figure 13 shows the development of the nervous system of an animat with two sensors (right), which allow the animat to perceive a given stimulus, and four actuators (left), which allow the animat to move. The cell positions and the targeting labels (i.e. which neuron will be connected to which sensors and actuators) have been specified explicitly. Figure 14 shows what kind of neural network can be evolved in order to generate a tracking behavior. The signal generated by each sensor is a genetically coded function of distance and angle of the stimulus. Likewise, each actuator generates a force which depends on the incoming signal in a genetically determined manner.

5 Morphological development

The two research efforts to be described in this section involve the simultaneous development and evolution of both an animat's control architecture and its morphology. The work of Dellaert and Beer implements a genetic regulatory network and various biological processes. The work of Sims is less biologically-grounded, but it allows for a modular approach, as in the previously described models of Boers and Kuiper, and of Gruau.

5.1 Dellaert and Beer

The developmental model of Dellaert and Beer [42] is concerned with the development of a whole organism, not just a neural network. It is inspired from Kaufmann's work [43] and relies

upon a genetic regulatory network whose binary elements each correspond to the presence (or absence) of a specific gene product or to the expression (or the non-expression) of some gene. According to the updating rule and connectivity of each element, the state of the network — which corresponds to the pattern of gene expression in a given cell — may change over time but will, ultimately, settle in a fixed point or a limit cycle. Such a dynamical process is used to model a cell cycle: in particular, a cell division occurs when the cell's regulatory network settles in a steady state, with a specific element being set to a predetermined value. When this occurs, the pattern of gene expression of the parent cell is passed on to the next generation, and a subset of genetic elements is used to determine the final differentiation of the two daughter cells.

Within such a framework, the morphology of an animat is a two-dimensional square consisting of cells of various types, each having a copy of the same Boolean network that constitutes the animat's genotype. However, the state of the network, corresponding to the pattern of gene expression in a particular cell, may be different in each cell, according to the cell's initial state and to the various influences experienced up to the present time. The physical extent of each cell is represented as a two-dimensional square element that can divide in either of two directions, vertical or horizontal. When division occurs, it takes place in such a way that the longest dimension of the parent cell is halved and that the two daughter cells together occupy the same space as the original cell.

Development starts out with a single square that represents the zygote. During development, the state of the regulatory network of each cell changes according to both the internal dynamical process mentioned above and the external influences provided by inter-cellular communications or by specific symmetry-breaking processes. For instance, the influence of neighboring cells is condensed into a so-called *neighborhood vector*, which is the logical OR of all the state vectors of these cells, and this neighborhood vector is combined with the cell's state vector to determine the next state. Likewise, a symmetry-breaking process occurs at the time of first cleavage, which switches a bit of the Boolean network state vector in one of the zygote's two daughter cells. Other symmetry-breaking processes cause the update of a cell's state vector to depend upon information on whether the cell is situated on the external surface of the animat or whether it borders the animat's horizontal mid-line.

In order to evaluate the morphogenetic potentialities of their encoding scheme, Dellaert and Beer have evolved a simple animat that roughly reproduces the relative positioning of sensors, actuators and control system that one would expect to find in a simple chemotactic agent (Figure 15). In particular, this animat exhibits bilateral symmetry, with sensors (cell-type 2) placed sideways at the front and actuators (cell-type 4) placed sideways at the back, and with a control structure made of *neural tissue* (cell-type 1) connecting them. Such an organization has been obtained by making the evolutionary process depend upon a fitness function that evaluates the discrepancies between the differentiation patterns of any developed animat and that of a hypothetical ideal chemotactic agent. In the case of the animat in Figure 15, these discrepancies have not been entirely eliminated, because two actuator cells are clearly out of place, in front of the animat.

Figure 16 describes the genome of this animat in two equivalent forms. It is a Boolean network with six nodes, each characterized by a specific update rule which sets the state of the corresponding node according to information contributed by two inputs. These inputs are also determined genetically and represent connections from other nodes (positive integers), connections from nodes in neighboring cells (negative integers in the range [-4, -1]) or influences of the external environment (-5) or of the mid-line (-6).

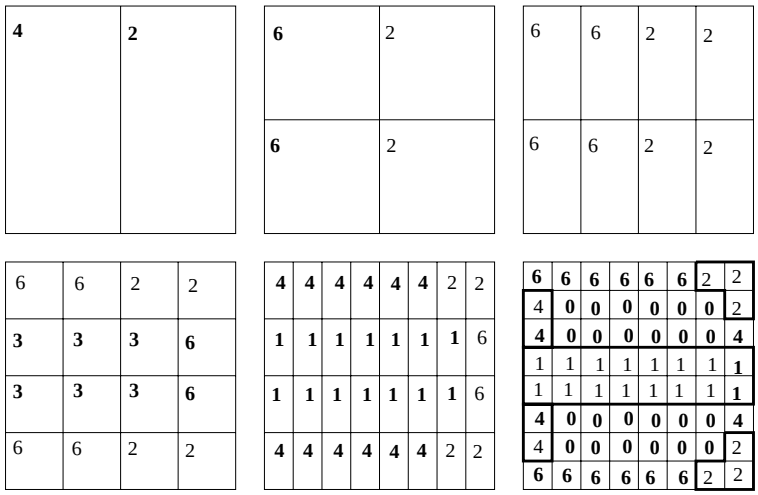

Figure 15. The first six consecutive stages of development of Dellaert and Beer's animat (after [42]).

a) node	0 1 0 1 0 0 1 1	inputs
1	0 0 1 0	3 -6
2	1 1 0 0	-2 -1
3	0 0 0 1	-5 5
4	1 1 0 1	4 4
5	0 1 1 0	6 -6
6	0 1 1 1	6 -1

b) node	Equivalent Boolean function
1	~3 AND mid
2	~(-1)
3	ext AND 5
4	~4 OR 4
5	6 XOR mid
6	6 OR -1

Figure 16. The genotype of Dellaert and Beer's animat (after [42]).

Thus, as shown in Figure 17, the update rule of node 1 in a given cell depends upon the state of node 3 in the same cell and upon the situation of this cell relative to the animat's mid-line: if the cell borders the mid-line, the value of bit -6 is 1, otherwise it is 0. Likewise, the update rule of node 3 depends upon the state of node 5 and upon the situation of this cell relative to the animat's external surface: if the cell is situated on this surface, the value of bit -5 is 1, otherwise it is 0. As Figure 17 also shows, the update rules of nodes 2 and 6 in a given cell depend on the state of node 1 in neighboring cells or, more precisely, on the state of bit 1 in the cell's neighborhood vector.

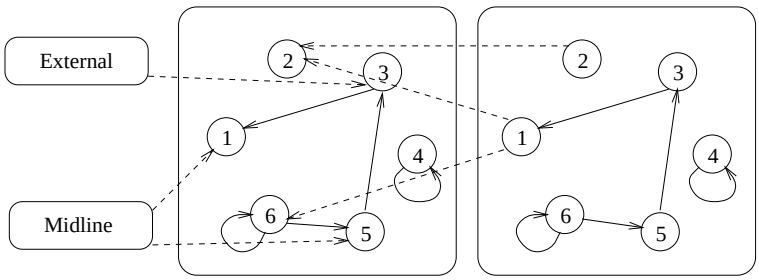

Figure 17. Intra-cellular and extracellular communications in Dellaert and Beer's animat (after [42]).

In this application, cellular division was dependent upon the state of node 4, whose updating rule ($\sim$ 4 OR 4) maintains this node in a permanently active state. Thus, a division occurred at every step, resulting in a maximum number of cells.

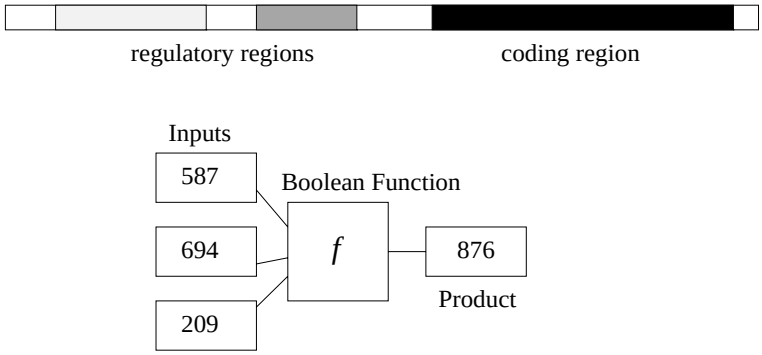

Figure 18. Dellaert and Beer's operon compared with a biological gene. It states, for example, that chemical 876 is produced iff chemical 587 is present and both chemicals 694 and 209 are absent in the cytoplasm (after [44]).

Various extensions of this model that involve a genome with a more complex connectivity, which can shrink or grow over successive generations, are described in [44] . Indeed, such a genome represents a set of *operons*, and every operon specifies (1) a gene product that is produced when the operon is active, (2) the other products that can regulate its expression, and (3) a Boolean function that specifies how the expression is regulated (Figure 18). In such a model, a special gene product serves to signal whether a cell should divide or not; other gene products serve to induce asymmetric cell divisions — because they are distributed to only one

daughter cell at division time; and still other gene products serve to signal that a given cell lies at the surface of the organism or on its mid-line.

A crucial extension of the previous model is the inclusion of a process of axonal growth, which depends upon a specific gene. Whenever this gene is expressed, the cell is checked to ascertain whether it expresses a certain *cell adhesion molecule* (CAM). If it does, an axon is sent out that searches for CAM's of the same type and grows in the direction where it finds the most. When the soma of a cell expressing a specific *trophic factor* is encountered, the growth process stops and a synaptic contact is made whose weight depends upon the amount of trophic factor available.

In [44] a hand-coded genome is provided that codes for the development of both the body and the control architecture of an animat exhibiting a simple *hate* behavior, as described by Braitenberg [45]. Using this hand-coded organism to seed the initial population for a genetic algorithm, Dellaert and Beer succeeded in producing a better performing animat after a few generations, in which the actuator regions were innervated in a different and stronger way than in the ancestor organism.

5.2 Sims

By using nested directed graphs, Sims [46, 47] also genetically encodes both the morphology and the control architecture of various animats. In this approach, a first-level directed graph of nodes and connections encodes the phenotype embodiment of a virtual animat as a hierarchy of three-dimensional rigid parts that can exhibit a recursive structure or can duplicate instances of the same appendage (Figure 19).

Each node in the graph contains information describing a rigid part, notably its physical shape and the way its motion relative to its parent part is constrained. Likewise, each connection contains information about the position of a child part relative to its parent. Nodes and connections also encode how many times a given node should generate a phenotype part when in a recursive cycle, and when and how tail- or hand-like components should be incorporated at the end of a chain of repeating units.

The control architecture of a given animat is encoded as second-level directed graphs of nodes and connections, included in each first-level node and describing the neural circuitry of the corresponding morphological unit, or belonging to an overall central control system. These second-level nodes describe either input sensors, internal neurons or output actuators. The second-level connections define the flow of signals between these nodes and allow the neurons and actuators within a morphological unit to receive signals from sensors or neurons in their parent or in their child units (Figure 20).

Each sensor is contained within a specific part of the body and measures either aspects of that part or aspects of the world relative to that part. Some sensors provide information about joint values, others react to physical contacts, and still others react to a global light-source. Internal neurons can perform diverse functions on their inputs — like *sum, product, divide, interpolate, memory, oscillate-wave, oscillate-saw*, etc. — to generate their output signals. As for actuators, each one exerts a muscle force on a specific degree of freedom of a specific joint.

Sims's approach allows virtual animats to evolve by optimizing for a specific behavior, such as swimming, walking or following [46]. Every animat is grown from its genetic description and then placed in a dynamically simulated virtual world, with which it interacts realistically, thus allowing its fitness to be assessed. For instance, Figure 20 shows the evolved genotype of a swimming animat. Its phenotype morphology includes four flippers shown in Figure 21, which are put into proper paddling motion by the phenotype control system of Figure 22.

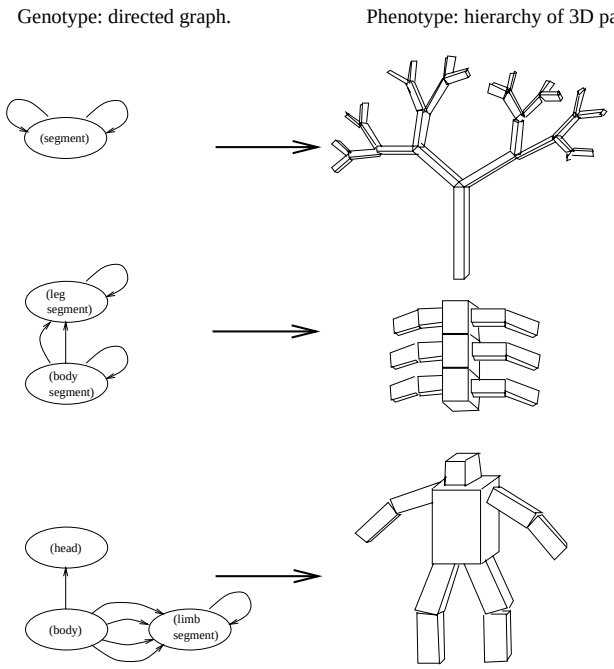

Figure 19. Designed examples of genotype first-level directed graphs and corresponding animat morphologies (after [46]).

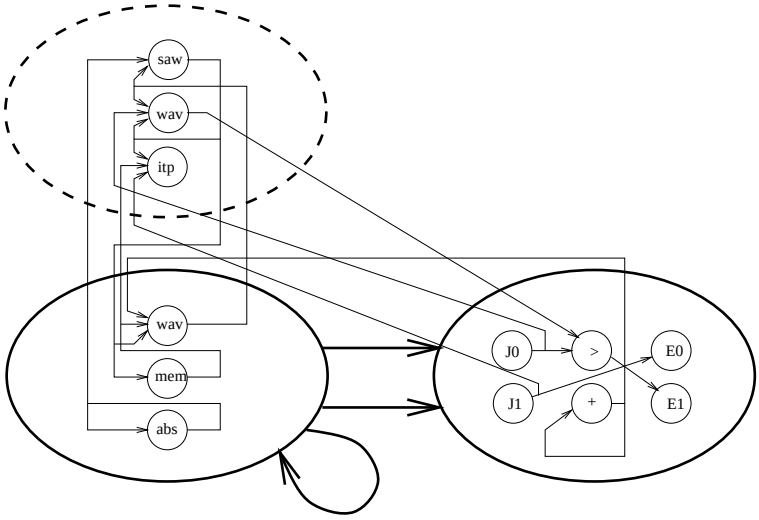

Figure 20. An evolved nested-graph genotype. The first-level graph in bold describes the animat's morphology. The second-level graph describes its neural circuitry. J0 and J1 are joint angle sensors; E0 and E1 are actuator outputs. The dashed node contains centralized neurons that are not associated with any part (after [46]).

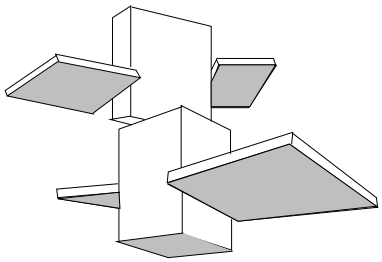

Figure 21. The phenotype morphology generated from the evolved genotype shown in Figure 20 (after [46]).

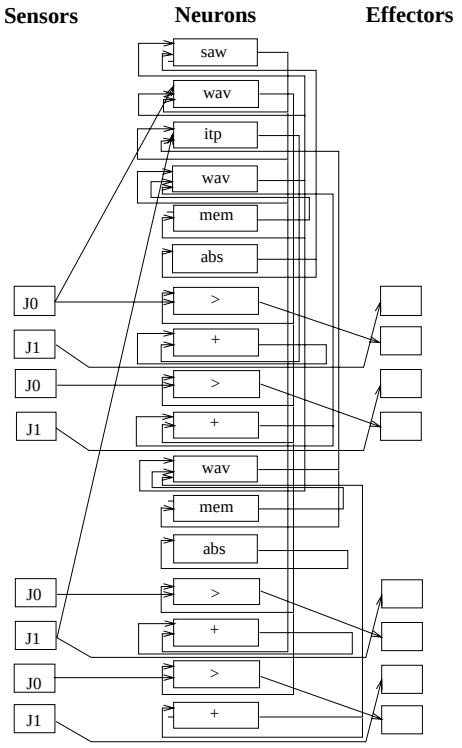

Figure 22. The phenotype control architecture generated from the evolved genotype shown in Figure 20. The actuator outputs of this architecture cause paddling motions in the four flippers of the morphology shown in Figure 21 (after [46]).

Likewise, [47] describes a system of co-evolving animats that compete in one-to-one contests for gaining control of a common resource. The results obtained prove that interesting and diverse strategies and counter-strategies are likely to emerge within such a framework.

6 Discussion

The introduction of a developmental process between the genotype and the phenotype of a given animat clearly has numerous consequences, which are yet to be fully understood and assessed. In particular, the corresponding genetic operators have very different effects according to where they are applied within the genotype. For instance, it is well known that a mutation occurring in the part of the genotype that is expressed early in the development will have more extensive consequences on the resulting phenotype than a mutation whose effect would be expressed during the latest stages of development. Intuitively, such a characteristic would be viewed as advantageous, because it seems likely to provide evolutionary algorithms with additional exploratory power. However, it turns out that, in traditional versions of evolutionary algorithms, it is already very difficult to adapt the corresponding genetic operators so that they can select and favor those building blocks useful in a given application. Thus, it seems highly likely that the acquisition of the corresponding empirical or theoretical knowledge will be much more difficult and lengthy for applications resorting to a developmental process.

Likewise, it seems likely that the use of developmental processes in conjunction with evolutionary algorithms will prove to be valuable in a more fundamental perspective. Indeed, such an approach obviously makes it possible to study how genetic information and environmental influences interact and complement each other during development — although such interactions have not been implemented in every model described here. In particular, this approach should help in specifying for which environment and for solving which kind of survival problem, Nature has been committed to inventing the process of development. In other words, it should help in assessing the adaptive value of this process and in specifying how it interacts with those of learning and evolution. A first step in such a direction has been made by Elman [48] who showed "the importance of starting small" as far as the learning abilities of a given neural network were concerned. Likewise, it might be interesting to speculate over what consequences are to be expected from letting a control architecture start developing (and, eventually, learning) within an environment that raises less challenging survival problems than those that a fully developed architecture would later face. It is probably not without reason that the most advanced animals spend a lot of time and energy in rearing their youngs in such protected conditions. It is also probably not without reason that they try to ensure such conditions are rich enough, through various means like, for instance, social interactions and play.

Insofar as a learning mechanism can also be submitted to evolution and development, two important implementation issues can be anticipated, which are related to the classical *structural* and *temporal* problems raised by reinforcement learning procedures [49]. Indeed such applications will resort either to a global learning procedure — according to which every connection in a given neural network will be adjusted during development — or to local learning rules that will involve only specific connections. Likewise, learning will be triggered at every developmental step or at specific developmental stages. Here again, numerous experiments will probably be needed before any heuristic will prove to be efficient in the corresponding implementation choices.

The research efforts that have been described in this paper relied basically on four different paradigms for modeling development:

- Rewriting rules;
- Axonal growth processes;
- Genetic regulatory networks;
- Nested directed graphs.

Every such paradigm implements an indirect encoding scheme that exhibits some specific advantages. For instance, rewriting rules allow compact encodings endowed with good scalability properties to be devised. As demonstrated by Gruau's work, such a solution is also suitable for generating modular architectures, thus providing an animat with the important functionality of problem decomposition. Sims's approach to modularity, although relying on a very different solution, also leads to impressive realizations. This suggests that modularity is a powerful feature for an encoding scheme and that it will probably be interesting to allow the axonal growth paradigm, for instance, to generate modular architectures. This should be easy because, at least in the applications that have been described here, genes do not specify the identity of the target neuron with which a given connection has to be established. Therefore, if a part of the genotype were replicated several times, each copy of the corresponding module would be connected to different neurons, depending upon its position within the overall architecture. As for the paradigm of genetic regulatory networks, it provides another interesting property: that of symmetry-breaking, whose effects on the resulting architectures are extremely difficult and tedious to code in a direct genotype-to-phenotype mapping.

Although it is somewhat premature to speculate about the relative merits of these different paradigms, it is clear that they are all capable of developing the control architecture of an animat. Therefore, they should prove useful in the future, at least in a purely engineering perspective, provided they turn out to be applicable to problems more difficult than those that have been solved here. Indeed, the solution to such simple problems usually required control architectures implementing mere *stimulus-response* mechanisms. Therefore, it would probably be extremely useful to test whether each of these approaches is capable of discovering more *cognitive* control architectures that would implement, for instance, some memory or planning abilities. In this perspective, it is interesting to note that the two research efforts to date that lead the farthest are those of Gruau and Sims — i.e., probably those that are the most distant from biological reality. This may be due to the fact that the natural mechanisms involved in the processes of evolution and development are still imperfectly understood or that these mechanisms are not used in an optimal context, for instance for selecting the behavioral characteristics most useful in a given environment.

It should also be noted that, although the applications described here have all involved simulated animats, some results have already been obtained that demonstrate that an evolutionary algorithm can be used to evolve a neural network controlling a real robot. Thus, Lewis et al. [50] succeeded in evolving a walking behavior, while Floreano and Mondada [51] did evolve motion and obstacle avoidance. Several other research efforts involving so-called *programmable hardware* [52, 53, 54, 55] or *cellular robotic systems* [56, 57, 58] will probably be extended in the future, so that they could add development to an evolutionary process involving a real robot. Such approaches may even foreseeably involve both the overall morphology and the control system of these robots. In particular, several experiments have already been carried out in which the morphology of the sensors co-evolves with the control architecture, both in simulation [59] and in hardware applications [60].

7 Conclusions

This paper has described six recent approaches that combine an evolutionary algorithm and a developmental process in order to automatically design a neural network controlling the behavior of an animat. Some of these approaches also involved a learning process. Although it is not yet possible to assess the relative merits of these research efforts — which are quite different from each other — there is good reason to think that they will ultimately prove helpful for automatically designing efficient control architectures linking perception to action in animats in general, and in real robots in particular. However, the corresponding progresses will very probably be slow.

In addition to their operational value, these approaches should provide a valuable contribution to theoretical biology in the future and enable a better understanding to be gained of the interactions between development, learning and evolution, i.e., the three main adaptive processes exhibited by natural systems.

References

[1] J.-A. Meyer and S. W. Wilson, editors. *From Animals to Animats. Proceedings of the First International Conference on Simulation of Adaptive Behavior*. The MIT Press/Bradford Books, Cambridge, MA, 1991.

[2] J.-A. Meyer, H. L. Roitblat, and S. W. Wilson, editors. *From Animals to Animats. Proceedings of the Second International Conference on Simulation of Adaptive Behavior*. The MIT Press/Bradford Books, Cambridge, MA, 1992.

[3] D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, editors. *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. The MIT Press/Bradford Books, Cambridge, MA, 1994.

[4] J.-A. Meyer and A. Guillot. From animals to animats: Everything you wanted to know about the simulation of adaptive behavior. Technical report, Groupe de Bioinformatique, ENS, Paris, 1990.

[5] J.-A. Meyer and A. Guillot. Simulation of adaptive behavior in animats: Review and prospects, in J.-A. Meyer and S. W. Wilson, editors, *From Animals to Animats. Proceedings of the First International Conference on Simulation of Adaptive Behavior*. The MIT Press/Bradford Books, Cambridge, MA, 1991.

[6] J.-A. Meyer and A. Guillot. From SAB90 to SAB94: Four years of animat research, in D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, editors, *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. The MIT Press/Bradford Books, Cambridge, MA, 1994.

[7] R. D. Beer, H. J. Chiel, and L. S. Sterling. A biological perspective on autonomous agent design. *Robotics and Autonomous Systems*, 6:169–186, 1990.

[8] D. Cliff, I. Harvey, and P. Husbands. Explorations in evolutionary robotics. *Adaptive Behavior*, 2(1):73–110, 1993.

[9] D. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(2), 1994.

[10] J. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press, 1975.

[11] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, 1992.

[12] L. Fogel, A. Owens, and M. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley & sons, New York., 1966.

[13] T. Bäck, F. Hoffmeister, and H. Schwefel. A survey of evolution strategies, in R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, San Mateo, CA, 1991.

[14] P. J. Angeline. Genetic Programming: A current snapshot, in A. V. Sebald and L. J. Fogel, editors, *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 224–232. World Scientific, Singapore, 1994.

[15] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[16] C. Fujiki and J. Dickinson. Using the Genetic Algorithm to generate LISP source code to solve the prisoner's dilemna, in J. J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[17] J. Koza. *Genetic Programming II: Automatic Discovery of Reusable Subprograms.* The MIT Press, 1994.

[18] J. Schaffer, D. Whitley, and L. Eschelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art, in D. Whitley and J. Schaffer, editors, *Combinations of Genetic Algorithms and Neural Networks.* IEEE Computer Society Press, 1992.

[19] D. J. Montana and L. Davies. Training feedforward neural networks using genetic algorithms, in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, volume 1, pages 762–767. Morgan Kaufmann, 1989.

[20] G. F. Miller, P. M. Todd, and S. U. Hedge. Designing neural networks using genetic algorithms, in *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann, 1989.

[21] S. Harp, T. Samad, and A. Guha. Towards the genetic synthesis of neural networks, in *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann, 1989.

[22] R. K. Belew, J. McInerney, and N. N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. Technical Report CS90-174, UCSD, 1990.

[23] P. Husbands, I. Harvey, D. Cliff, and G. Miller. The use of genetic algorithms for the development of sensorimotor control systems, in P. Gaussier and J. Nicoud, editors, *From Perception to Action. Proceedings of the PerAc'94 Conference*, pages 110–121. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[24] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4:461–476, 1990.

[25] F. Gruau. Cellular encoding of genetic neural networks. Technical Report 92-21, Laboratoire de l'Informatique du Parallélisme, ENS Lyon, May 1992.

[26] E. Boers and H. Kuiper. Biological metaphors and the design of modular artificial neural networks. Master's thesis, Leiden University, August 1992.

[27] A. Lindenmayer. Mathematical models for cellular interactions in development. Part I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.

[28] D. Pomerleau. ALVINN: An autonomous land vehicle in a neural network. Technical Report CMU-CS-89-107, Computer Science Departement, CMU, Pittsburg, PA, 1989.

[29] F. Gruau. Genetic systems of boolean neural networks with a cell rewriting developmental process, in D. Withley and J. Schaffer, editors, *Combination of Genetic Algorithms and Neural Networks*. IEEE Computer Society Press, 1992.

[30] F. Gruau and D. Withley. Adding learning to the cellular development of neural networks: Evolution and the Balwin effect. *Evolutionary Computation*, 1(3):213–233, 1993.

[31] P. Pratt. Evolving neural networks to control unstable dynamical systems, in A. V. Sebald and L. J. Fogel, editors, *Proceedings of the Third Annual Conference on Evolutionary Programming*. World Scientific, Singapore, 1994.

[32] F. Gruau. Automatic definition of modular neural networks. *Adaptive Behavior*, 3(2):151–184, 1994.

[33] R. Beer and J. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1):91–122, 1992.

[34] G. E. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex Systems*, 1:495–502, 1987.

[35] R. Belew. When both individuals and populations search: Adding simple learning to the genetic algorithm, in D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.

[36] A. Cangelosi, D. Parisi, and S. Nolfi. Cell division and migration in a 'genotype' for neural networks. *Network: computation in neural systems*, 1995. In press.

[37] S. Nolfi and D. Parisi. Growing neural networks. Technical Report PCIA-91-15, Institute of Psychology, Rome, 1991.

[38] S. Nolfi, O. Miglino, and D. Parisi. Phenotypic plasticity in evolving neural networks, in P. Gaussier and J. Nicoud, editors, *From Perception to Action. Proceedings of the PerAc'94 Conference*. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[39] J. Vaario. *An Emergent Modeling Method for Artificial Neurol Networks*. PhD thesis, University of Tokyo, August 1993.

[40] J. Vaario. From evolutionary computation to computational evolution. *Informatica*, 1994. To appear.

[41] J. Vaario. Modeling adaptive self-organization, in R. A. Brooks and P. Maes, editors, *Proceedings of the Fourth International Workshop on Artificial Life*, pages 313–324. The MIT Press/Bradford Books, Cambridge, MA, 1994.

[42] F. Dellaert and R. Beer. Toward an evolvable model of development for autonomous agent synthesis, in R. A. Brooks and P. Maes, editors, *Proceedings of the Fourth International Workshop on Artificial Life*. The MIT Press/Bradford Books, Cambridge, MA, 1994.

[43] S. Kaufmann. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22:437–467, 1969.

[44] F. Dellaert and R. Beer. Co-evolving body and brain in autonomous agents using a developmental model. Technical Report CES-94-16, Dpt of Computer Engineering and science Case Western Reserve University, Cleveland, OH., August 1994.

[45] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. The MIT Press, 1984.

[46] K. Sims. Evolving virtual creatures, in *Computer Graphics Proceedings, Annual Conference Series*, pages 15–23, 1994.

[47] K. Sims. Evolving 3D morphology and behavior by competition, in R. A. Brooks and P. Maes, editors, *Proceedings of the Fourth International Workshop on Artificial Life*. The MIT Press/Bradford Books, Cambridge, MA, 1994.

[48] J. Elman. Incremental learning, or the importance of starting small. Technical Report 9101, CRL, UCSD, CA, March 1991.

[49] L. Lin. *Reinforcement Learning for Robots Using Neural Networks*. Phd dissertation, School of Computer Science, CMU, Pittsburg, PA, 1993.

[50] M. Lewis, A. Fagg, and A. Solidum. Genetic programming approach to the construction of a neural network for control of a walking robot, in *Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, France, 1992.

[51] S. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot, in D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, editors, *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. The MIT Press/Bradford Books, Cambridge, MA, 1994.

[52] H. Hemmi, J. Mizoguchi, and K. Shimohara. Development and evolution of hardware behaviors, in R. A. Brooks and P. Maes, editors, *Proceedings of the Fourth International Workshop on Artificial Life*. The MIT Press/Bradford Books, Cambridge, MA, 1994.

[53] T. Higuchi, H. Iba, and B. Manderick. Applying evolvable hardware to autonomous agents, in Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Proceedings of the Third International Conference on Parallel Problem Solving from Nature*. Springer-Verlag, Berlin, Germany, 1994.

[54] P. Marchal, C. Piguet, D. Mange, A. Stauffer, and S. Durand. Embryological development on silicon, in R. A. Brooks and P. Maes, editors, *Proceedings of the Fourth International Workshop on Artificial Life*. The MIT Press/Bradford Books, Cambridge, MA, 1994.

[55] A. Thompson. Evolving electronic robot controllers that exploit hardware resources, in *Proceedings of the Third European Conference on Artificial Life*. Springer Verlag, 1995.

[56] G. Beni and J. Wang. Swarm intelligence in cellular robotic systems, robots and biological systems: Toward a new bionics, in Dario, Sansini, and Aebischer, editors, *Computer and systems Sciences*, volume 102. Springer Verlag, 1993.

[57] T. Fukuda, G. Iritani, T. Ueyama, and F. Arai. Self-organizing robotic systems. organization and evolution of group behavior in cellular robotic system, in P. Gaussier and J. Nicoud, editors, *From Perception to Action. Proceedings of the PerAc'94 Conference*, pages 24–35. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[58] T. Fukuda and T. Ueyama. *Cellular Robotics and Micro-Robotic Systems*. World Scientific, 1994.

[59] C. W. Reynolds. Evolution of corridor following behavior in a noisy world, in D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, editors, *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 402–410. The MIT Press/Bradford Books, Cambridge, MA, 1994.

[60] I. Harvey, P. Husbands, and D. Cliff. Seeing the light: Artificial evolution, real vision, in D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, editors, *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 392–401. The MIT Press/Bradford Books, Cambridge, MA, 1994.