The Age of Analog Networks

Claudio Mattiussi, Daniel Marbach, Peter Dürr, and Dario Floreano

■ *A large class of systems of biological and technological relevance can be described as analog networks, that is, collections of dynamic devices interconnected by links of varying strength. Some examples of analog networks are genetic regulatory networks, metabolic networks, neural networks, analog electronic circuits, and control systems. Analog networks are typically complex systems that include nonlinear feedback loops and possess temporal dynamics at different time scales. Both the synthesis and reverse engineering of analog networks are recognized as knowledge-intensive activities, for which few systematic techniques exist. In this paper we will discuss the general relevance of the analog network concept and describe an evolutionary approach to the automatic synthesis and the reverse engineering of analog networks. The proposed approach is called analog genetic encoding (AGE) and realizes an implicit genetic encoding of analog networks. AGE permits the evolution of human-competitive solutions to real-world analog network design and identification problems. This is illustrated by some examples of application to the design of electronic circuits, control systems, learning neural architectures, and the reverse engineering of biological networks.*

Many systems of primary interest in biology and in engineering can be seen as *analog networks*. An analog network (figure 1) is composed of a collection of nodes representing devices and a collection of directed or undirected links connecting the devices and having a connection strength represented by a numerical value. To illustrate the idea of analog networks and explain the practical importance of their automated synthesis and reverse engineering, we consider three examples: analog electronic circuits, artificial neural networks, and genetic regulatory networks.

An analog electronic circuit is a collection of interconnected electronic devices such as transistors, diodes, capacitors, and resistors (figure 2). The purpose of an analog electronic circuit is the production and processing of electrical signals whose amplitude can vary continuously in time. This is opposed to digital circuits, which process signals whose amplitude is discretized. Despite a steady trend towards the substitution of analog with digital signal processing, analog circuits maintain a crucial role in electronic design. For example, in many applications, analog electronic circuits are required in order to connect digital circuits to continuous input and output signals; these analog circuits have a profound impact on overall system performance. There is therefore a well-founded interest in the automation of the design of analog electronic circuits. However, analog design has proved much more difficult to automate than digital design. To understand the nature of this difficulty, one must consider that the function realized by an electronic circuit is determined by two aspects: its *topology* and its *sizing*.

The topology of a circuit refers to the nature of the devices that compose the circuit and how they are connected together. The sizing of a circuit refers to the values of the numerical parameters that characterize the devices and links. An example of a numerical parameter of a device is the capacitance of a capacitor. As mentioned above, the numerical parameter associated with a link corresponds to the interaction strength between the devices it connects. In the case of electronic circuits, the interaction strength between two devices is inversely proportional to the resistance between them. A zero resistance value corresponds to a direct connection and realizes the maximum connection strength. An infinite resistance corresponds to the absence of a link connecting the two devices. All other resistance values correspond to the presence of a link with a resistor between the devices and realize intermediate connection strengths. In digital circuits, only the two extreme values of connection strength are used to connect the devices (for example, logic gates) that constitute the circuit. In analog electronic circuits, in contrast, a large variety of connection strengths are typically required to achieve the intended functionality. This means that, compared to a digital designer, an analog designer must take into account a

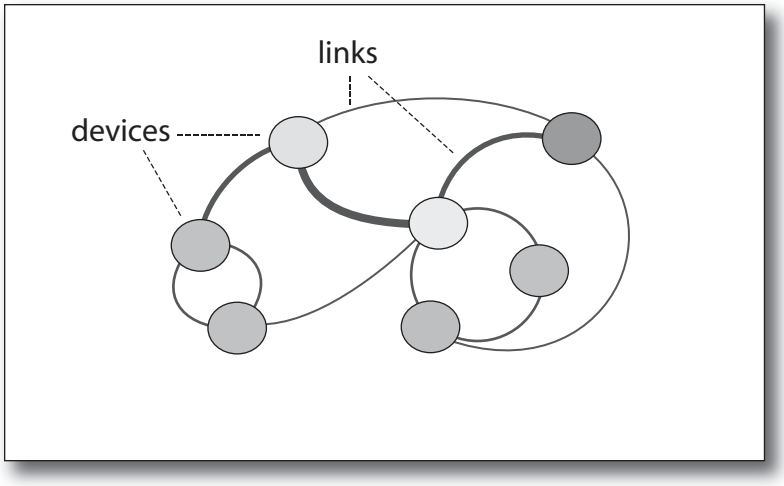


Figure 1. Analog Network.

An analog network is a collection of devices—represented here by the circles—connected by links of varying strength—represented here as lines of varying thickness between the devices.

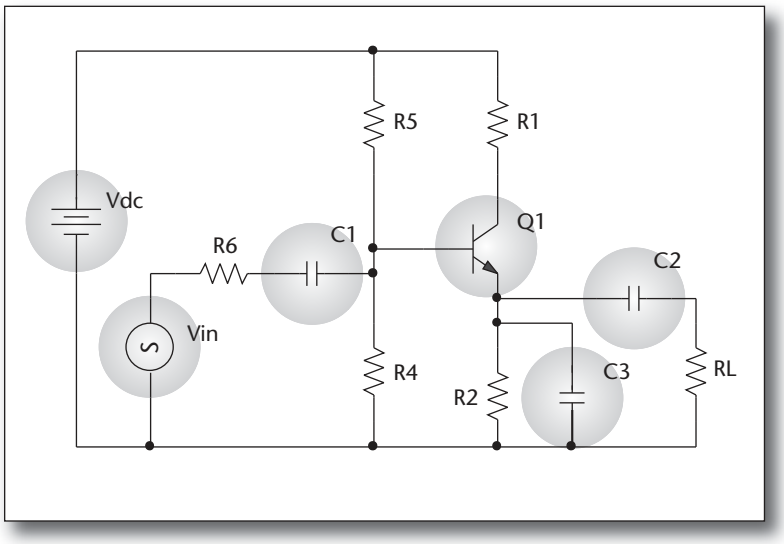


Figure 2. A Schematic Drawing Showing an Example of an Analog Electronic Circuit.

The circuit can be seen as a network of nonresistive devices (represented here with a shaded background) connected by resistive links.

much larger variety of possible interactions between a larger variety of parameterized devices that can potentially compose a circuit.

A similar situation holds in the field of artificial neural networks (ANNs). An ANN is a collection of interconnected nodes called *artificial neurons*, which are loosely inspired by models of biological neurons. Each artificial neuron realizes an input-output function that depends on a set of numerical parameters called *weights* and, possibly, some additional parameters (figure 3). Similarly to electronic circuits, the purpose of ANNs is the production and processing of signals. For example, an ANN might take as input the signal generated by a set of biometric sensors and be required to produce as output an estimate of the level of sleepiness of the person wearing the sensors. ANNs find application in numerous areas of great practical importance such as pattern recognition for medical applications, robot learning, and industrial process control. The difference between ANNs and analog electronic circuits is that, in ANNs, the links between devices are directed, and that artificial neurons are typically abstract computational devices implemented on a computer rather than actual physical devices. Several kinds of artificial neuron devices can be defined, including excitatory, inhibitory, and neuromodulatory, with linear or nonlinear input-output relationship, and with or without internal memory.

The functionality of an ANN, like that of an electronic circuit, is determined by its topology (also called its *architecture*)—that is, by the kind of artificial neurons that compose the network and their connectivity—and by its sizing—that is, the values of the parameters of the network. Apart from the directed nature of the ANN links, the weights of an ANN play a role that is analogous to the role of the reciprocal of the resistance between the devices that compose an electronic circuit. They define the strength of the directed connection between the outputs and the inputs of the neurons that compose the network. A null weight corresponds to the absence of a connection; larger weights correspond to stronger connections between an output and an input. For reasons similar to those mentioned for analog electronic circuits, the design of ANNs is a difficult task. To reduce this difficulty, the typical approach is to consider the networks as composed of just one kind of neuron and to assign a fixed topology, thereby limiting the design to the choice of the network sizing. However, these limitations considerably reduce the learning power of ANNs (Baum 1989).

Genetic regulatory networks (GRNs), which are the basic control and computational systems of biological cells (Bray 1995), are another example of analog networks. They are composed of a collection of interacting genes. A gene can be loosely defined as a fragment of the genome of a cell that can be activated to initiate the production of molecules such as RNA and proteins. The rate of this production activity is denoted as the *level of expression* of a gene. The rate of expression of a gene can be controlled by molecules produced externally of the cell and by the molecules produced by the genes themselves. Thus, genes can be seen as devices whose inputs are the concentrations of the molecules that can influence the activation of the

gene and whose output is the level of expression of the gene. Abstracting all the molecular details of the production of the molecules and of their interaction with the genome, one can represent the collection of interacting genes as a network whose nodes are the genes. The influence of the level of activation of a gene on that of another gene is represented by a numerical value associated with a link connecting the two nodes representing the genes (figure 4) (Stormo and Zhao 2007). Thus, in this simplified view, the topology of a GRN is given by the kind of genes that compose the network and their connectivity, and the sizing is defined by the strength of the interaction between the genes and possibly other numerical parameters defining the dynamics of the genes.

To understand and control the working of a cell, it is necessary to construct a model of its GRN, or of subnetworks that control some specific functions. Practical applications of GRN modeling include the understanding of genetically related diseases and drug design. The model can be inferred from the data resulting from the observation (for example, using DNA microarray chips) of the levels of activation of the genes in various circumstances, which can include the perturbation of the network, of the external inputs, and of the signals exchanged by the nodes. The inference of the topology and sizing of a GRN from these observations corresponds to a process of *reverse engineering* of the network (figure 5). Given the complexity of GRNs this process typically requires the help of a computer-based reverse-engineering tool.

Reverse engineering has a long history in traditional engineering disciplines (Ljung 1999)—for example, to understand a competitor's products—but has only recently become popular in biology (Bolouri and Davidson 2002). Indeed, biologists have typically applied a forward engineering approach, where the components of a system are studied in great detail and models are built bottom-up, based on an understanding of the individual parts and mechanisms (Lazebnik 2002). This approach becomes increasingly difficult when applied to complex analog networks such as gene regulatory networks. Substantial advances in experimental technology are currently boosting research in reverse engineering of biological networks. Moreover, the advent of *genetic engineering* and *synthetic biology* (Endy 2005) opened the way to the alteration of existing functionality and to the integration of new functionalities in GRNs. This corresponds to an activity of design of GRNs with complexity and potential impact analogous to that described above for analog electronic circuits and ANNs.

Other examples, such as cellular metabolic networks and control systems, can be described according to the same basic analog network scheme that applies to analog electronic circuits, ANNs, and GRNs. The preceding examples suggest that, in general, the synthesis and reverse engineering of analog networks are complex problems of great practical relevance, for which few automatic techniques exist. The identification of the common structure of an analog network in these systems means that we can adopt a common approach to the automation of their synthesis and reverse engineering.

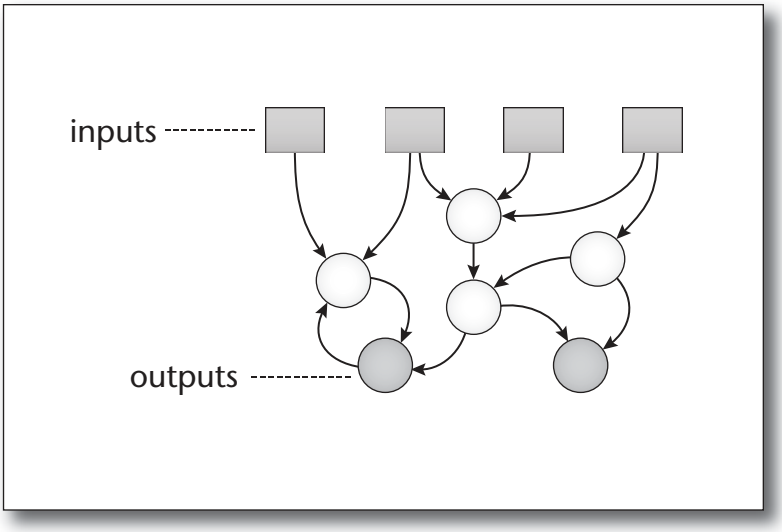


Figure 3. An Example of an Artificial Neural Network.

The network can be seen as a collection of nodes connected by directed links whose strength is represented by numerical values called weights.

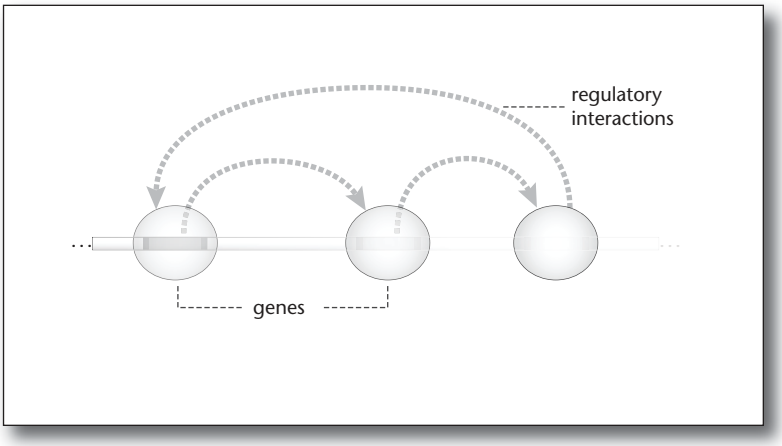


Figure 4. A Schematic Representation of a Genetic Regulatory Network.

The genes correspond to the nodes of a network and interact through links of varying strength, which represent the regulatory interactions between the genes.

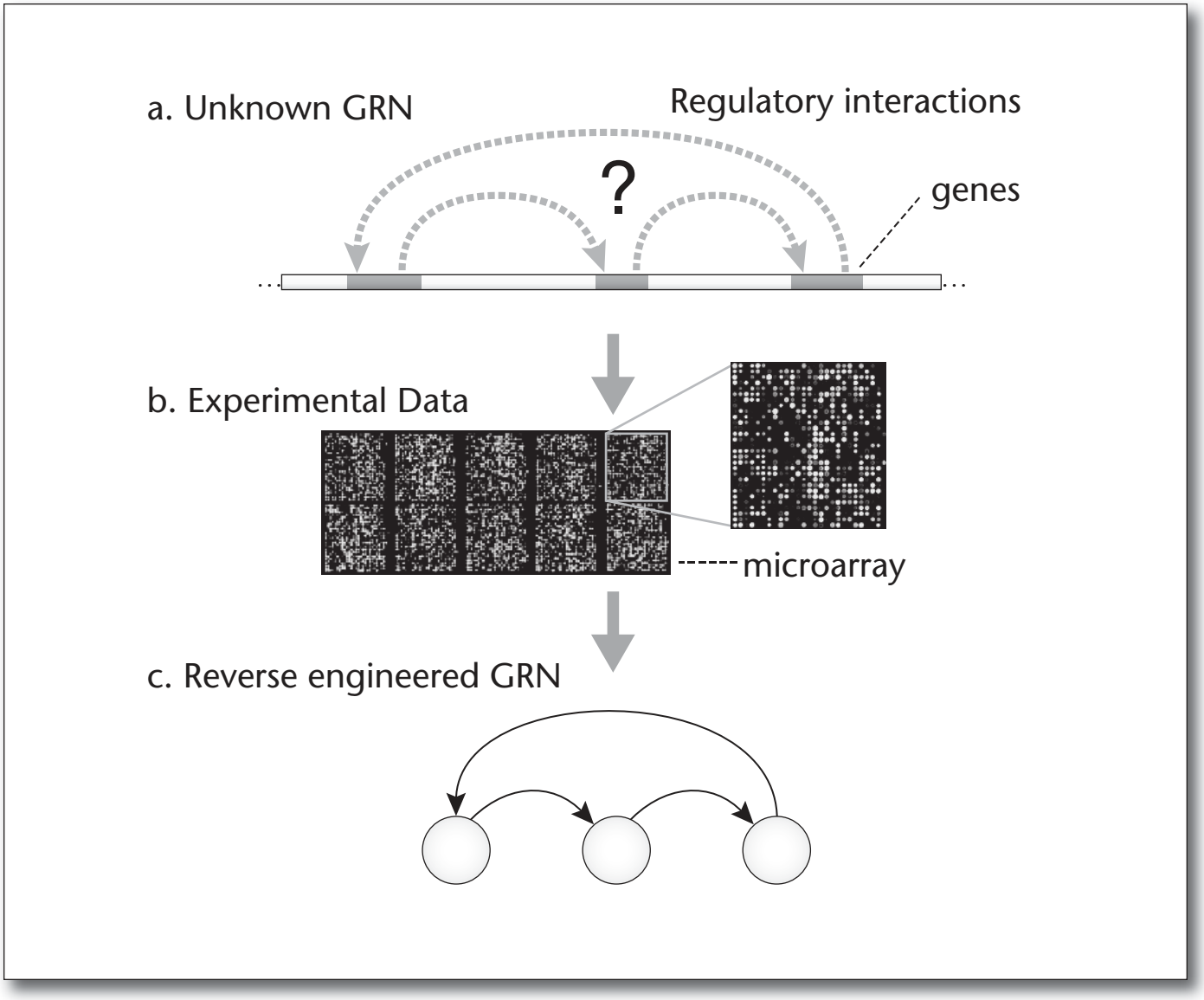


Figure 5. A Schematic Representation of the Process of Reverse Engineering for a Genetic Regulatory Network.

Automatic Synthesis and Reverse Engineering of Analog Networks

The effective automation of the design and reverse engineering of analog networks requires a search algorithm capable of exploring a solution space composed of analog networks of arbitrary topology and sizing. This means that the search algorithm must be endowed with (1) a way to represent the devices that enter the network, their connectivity, and the values of the network parameters and (2) a way to generate new tentative solutions using the information represented by the networks examined in the past search history. Evolutionary algorithms appear to be good candidates for this task, due to the flexibility of the representation and their potential for exploring the search space (see the Evolutionary Algorithms sidebar for a short description).

The simplest approach to the genetic representation of analog networks is the explicit encoding of all the devices, links, and parameters of the network. This approach, known as *direct encoding*, has the advantage of leading to genomes that are very easy to decode into the corresponding analog network (Yao 1999; Stanley and Miikkulainen 2002; Zebulum, Vellasco, and Pacheco 2000). The drawback is the rapid growth of the length and complexity of the genome with the size of the network, due to the necessity of encoding explicitly all the network connections and their strength, which affects the evolvability.

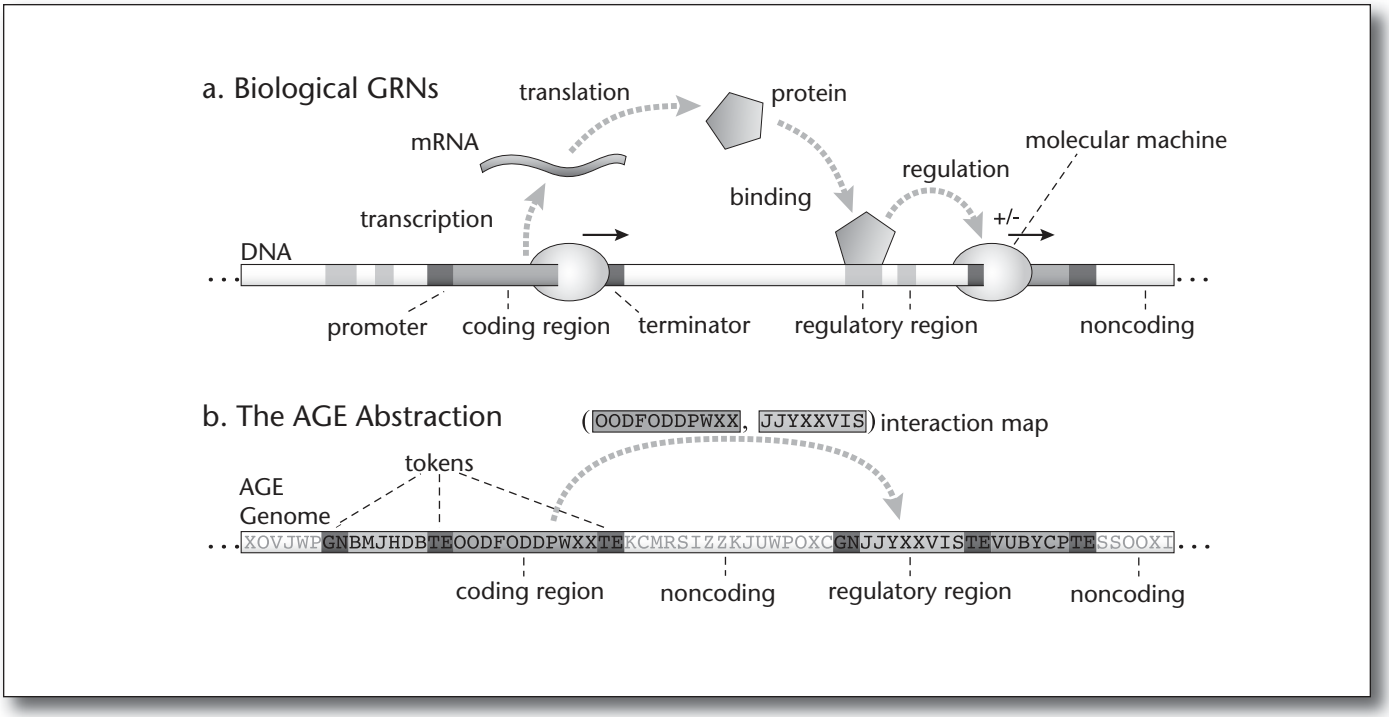


Figure 6. AGE Abstracts the Mechanism of Interaction between Genes Observed in Biological GRNs.

a. In biological GRNs, the link between genes is realized by molecules that are synthesized from the coding region of one gene and interact with the regulatory region of another gene. *b*. AGE abstracts this mechanism with an interaction map that transforms the coding and regulatory regions into a numerical value that represents the strength of the link.

An alternative and potentially very compact genetic representation of analog networks can be obtained using a strategy called *developmental encoding*. An example of developmental encoding is constituted by *genetic programming* (Gruau 1994, Miller and Thomson 2000, Koza et al. 2003). Genetic programming uses as genetic representation a sequence of instructions that can be used to build the network by performing a series of successive alterations of the network topology and sizing, starting from an elementary network called an embryo. One of the challenges of genetic programming is the necessity for the user to define a suitable set of network-modifying instructions. This set must be rich enough that all possible networks of interest can be produced. At the same time, one must craft the instructions and the mutation operators so as to ensure that only valid network-building programs are generated during the search. This means that for each different kind of analog network one must face the nontrivial task of designing a suitable set of network-modifying instructions. Note, however, that genetic programming is a powerful general-purpose approach to the representation and evolution of many kinds of complex structures and is not especially targeted to analog networks. Thus, it cannot be expected to be particularly suited to the evolution of analog networks.

In this article, we describe a new approach to the genetic representation and artificial evolution of analog networks and illustrate it with some examples taken from the domains considered in the introduction. This new approach is called *analog genetic encoding* (AGE) and realizes a compact and highly evolvable *implicit encoding* of analog networks. AGE was developed having in mind the whole class of analog networks. Thus, contrary to existing techniques for the analysis and synthesis of analog networks, which typically require a substantial effort of adaptation to each specific kind of analog network, AGE can be applied effortlessly to any kind of analog network. This means in particular that it is possible to realize a general automatic synthesis and analysis tool for this whole class of networks, and that the whole class can benefit from the advances and experience gained with each specific kind of network. As explained in more detail below, besides the advantage constituted by its generality, AGE has the characteristic of being very effective as a synthesis and analysis tool, producing results that are comparable or better than those produced by the existing specialized techniques.

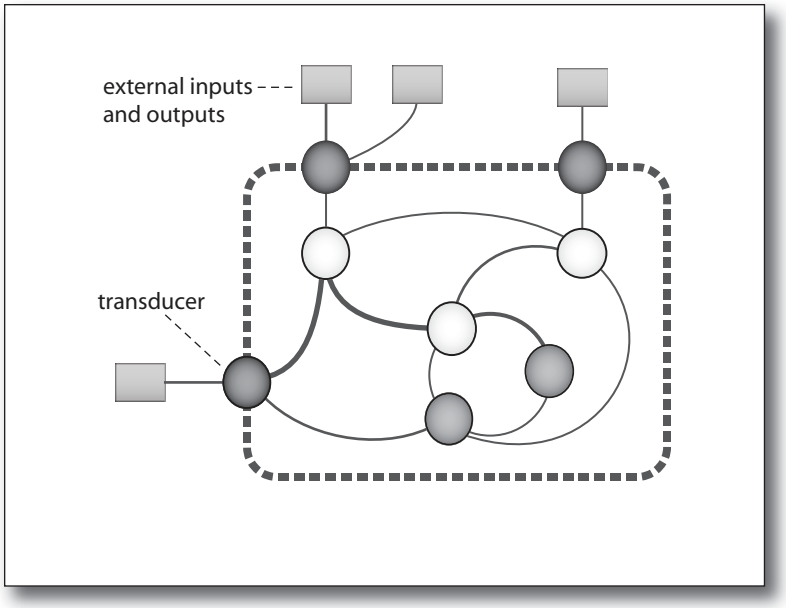


Figure 7. The Connection between the Evolved Analog Network and Inputs and Outputs.

In AGE, the connection between the evolved analog network and the predefined external inputs and outputs is realized using specialized devices called transducers, which can connect to the devices of the evolved network and to the external inputs and outputs.

Analog Genetic Encoding

AGE is loosely inspired by the working of biological GRNs. In biological GRNs, the interactions between the genes is not explicitly encoded in the genome but follows implicitly from the physical and chemical environment in which the genome is immersed. The activation of a biological gene depends on the interaction of molecules present in the vicinity of the gene with parts of the gene called *regulatory regions* (figure 6a). These are sequences of characters from the genetic alphabet to which the molecules can bind to promote or hinder the working of specialized molecular machinery that is in charge of expressing the gene. The expression of the gene corresponds to the scanning of another sequence of genetic characters, called *coding region*, in order to synthesize the molecules that are the products of the gene activation. The start and end of the coding region of a gene are marked by special sequences of characters from the genetic alphabet, called *promoter* and *terminator* regions. The molecules produced by a gene can in turn interact with the regulatory regions of other genes and influence their activation.

AGE abstracts and extends these GRN concepts to obtain a genetic representation that applies to generic analog networks. The AGE genome is composed of sequences of characters from a suitable alphabet, for example, the uppercase ASCII set. As in GRNs, devices are represented in the AGE genome by assemblies composed of two kinds of sequences of characters (figure 6b). The first kind of sequences is called *token*. Tokens play the role of markers and delimiters analogous to that played in biological GRNs by promoter and terminator regions. The second kind of sequences plays a role analogous to that played in biological GRNs by regulatory and coding regions. The strength of the interaction between two devices is implicitly determined by the second kind of sequence through a function called the *interaction map*. The interaction map takes as arguments two sequences of characters and produces a numeric value representing the strength of the interaction between two devices. In summary, decoding the AGE genome involves the identification of valid devices (which must be correctly delimited by the corresponding tokens) and the subsequent application of the interaction map to all pairs of coding and regulatory sequences. The interaction strength between two sequences may be zero, in which case there is no regulatory link between the two devices. Hence, the size of the decoded network is given by the number of devices in the genome and the topology and sizing follow from the computed interaction strengths. Further details on the encoding and the interaction map and a description of the method used to represent parameter values can be found in our previous work (Mattiussi 2005; Mattiussi and Floreano 2007; Mattiussi, Dürr, and Floreano 2007; Dürr, Mattiussi, and Floreano 2006; Marbach, Mattiussi, and Floreano 2007).

In general, an analog network performs its function by taking input signals from a predefined set of external input and delivering output signals to a predefined set of external outputs. For example, an analog electronic circuit may be required to amplify the signal produced by a signal source and deliver the amplified signal to a loudspeaker. To let evolution establish the connections between the evolved analog network and the external input and outputs, AGE uses a special kind of device called a transducer (figure 7). A transducer is a device that can be connected by the interaction map both to devices of the evolved circuit and to the external inputs and outputs. This establishes a bridge between the external inputs and outputs and the evolved analog network.

Some Properties of AGE

Let us briefly review some of the properties of AGE as a representation for analog networks in an evolutionary process.

First, AGE permits an easy adaptation of the evolutionary environment to arbitrary kinds of analog networks. To set up an evolutionary run, the user

of AGE needs to define just the external inputs and outputs and the characteristics of the various kinds of devices that can appear in the network. There is no need to tailor the genetic operators to the particular type of analog network of interest.

Mimicking biological GRNs, AGE encodes the interaction between the devices that form the network implicitly. This has the advantage of reducing the number of elements that must be encoded in the genome with respect to direct encodings. For example, the resistors appearing in the circuit represented in figure 2 will not explicitly appear as devices in an AGE encoding of this circuit. An important feature of the implicit encoding is that a single mutation can have several effects on the network structure. This may provide an advantage in terms of evolvability at the initial steps of the search by letting the evolution probe simultaneously the effect of many interactions. However, this may constitute a problem at later stages of the search, because a single mutation that simultaneously perturbs many interactions may hinder their separate optimization. To mitigate this potential difficulty, the device interaction map has been defined so as to allow a single sequence of characters to determine several noninterfering interactions with several distinct devices. Moreover, it is possible to implement a mechanism that lets evolution selectively silence some interactions. For further details on these points see Mattiussi (2005).

Most artificial genetic encodings constrain the genomes to maintain a fixed structure and admit a small number of genetic operators in order to remain decodable. In AGE, the sequences that define the interaction between devices can have variable length, and the interaction map that is used to establish the connection between devices is defined so as to apply to sequences of arbitrary length. The assemblies of sequences of characters that represent a device can be located anywhere in the genome and can be spaced by stretches of noncoding genetic characters. In this way the structure of the genome is not unduly constrained and tolerates a large class of genetic operators, which can alter both the topology and the sizing of the encoded network. In particular, the AGE genome permits the insertion, deletion, and substitution of single characters and the insertion, deletion, duplication, and transposition of whole genome fragments. All these genetic mutations are known to occur in biological genomes and to be instrumental to their evolution. In particular, gene duplication and the insertion of fragments of genome of foreign organisms are deemed to be crucial mechanisms for the evolutionary increase of complexity of GRNs (Shapiro 2005). Finally, the interaction map is defined so as to be highly redundant, so that many different pairs of character sequences produce the same numeric value. Thus, many mutations have no effect, resulting potentially in a high neutrality in the search space (see the Evolutionary Algorithms sidebar).

With AGE, the number of devices that compose the evolved networks is free to either increase or decrease during evolution. Genetic operations such as genome duplication can increase the number of devices, but operations such as substitution and deletion of genome fragments can disrupt the structure representing a device and thus remove it from the encoded analog network. In the experiments described below we have often observed that a compact network realizing the required functionality is obtained through intermediate stages involving larger networks. Presumably, the larger networks realize an easier evolutionary path towards the solution network, which, once produced, is simplified by evolution in order to increase its robustness to genetic mutations. Another important feature of AGE is that it does not assign the connections between the external inputs and outputs and the evolved analog network but lets evolution establish them. Thus, the search process is free to discover the subset of external inputs and outputs that are actually needed to produce the required functionality. The identification of the subset of inputs and outputs that are actually relevant to the task corresponds to a process of automatic feature selection.

Although it does not require it, AGE permits the incorporation of expert knowledge into the search process. One way to do this is to seed the initial population with networks that are known to solve a problem or subproblem that is similar to the goal of the search. Another way is to include in the set of devices that can be used in the network, subnetworks that are known to be useful in the context of the given problem.

Applications and Examples

We now present a series of examples of synthesis and reverse engineering of analog networks using AGE. We will focus the discussion on the characteristics and relevance of the problems considered and on the properties of the evolved networks. Detailed information about the working of the evolved networks, the computational effort required to obtain the solutions, and comparisons with alternative techniques can be found in the original papers cited in the context of each example. For example, Mattiussi and Floreano (2007) compare the computational effort, the circuit performance, and the circuit complexity resulting from the application of AGE and genetic programming to three analog electronic design problems, whereas Dürr, Mattiussi, and Floreano (2006) compares the computational effort and generalization properties resulting from the application of AGE

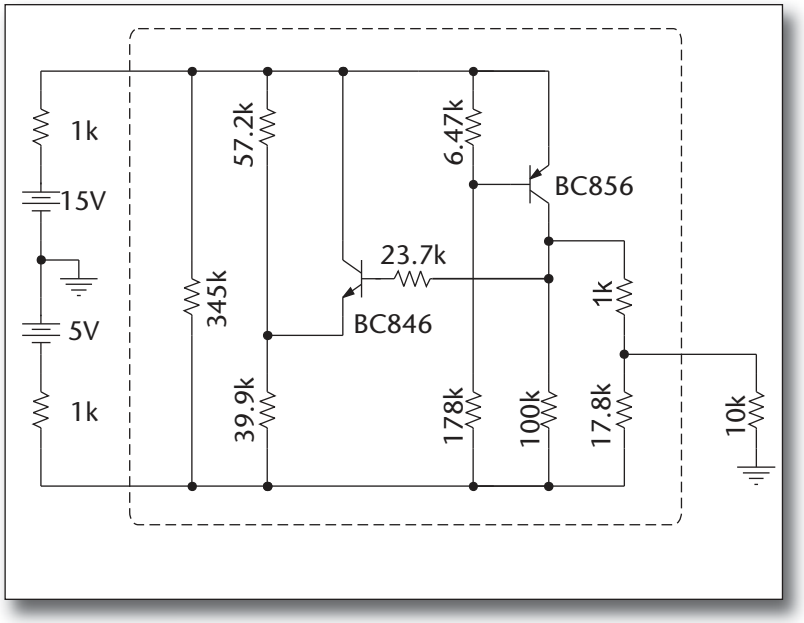


Figure 8. An Example of a Temperature-Sensing Circuit Automatically Designed Using AGE.

The predefined external devices are drawn outside of the dashed line.

and three other techniques for the automatic synthesis of ANNs to the design of a neural controller for the pole balancing problem described below. Here we note just that the required computational effort and the quality of the results compare well with those of the best existing techniques for the automatic synthesis and reverse engineering of analog networks. The experiments described below did not incorporate expert knowledge in the search and started from a population of individuals whose genomes contained a small collection of randomly generated device descriptors.

Synthesis of Electronic Circuits

To exemplify the automatic design of analog electronic circuits using AGE. we consider the synthesis of a temperature sensing circuit (Mattiussi and Floreano 2007). This is a problem of considerable practical importance, as temperature management is a critical issue in many applications and integrated temperature sensors are embedded in a growing number of systems.

To set up the search process, we have assigned two voltage supplies and a load resistor as predefined external devices (the devices drawn outside of the dashed box in figure 8). We have also assigned two types of bipolar transistors of opposite polarity as the set of devices that can be used to build the circuit. The goal of the synthesis is a circuit producing across the 10kΩ external resistor an output voltage that is proportional to the circuit temperature T in the range 0 degrees Celsius $\leq T \leq$ 100 degrees Celsius, with null output voltage for T = 0 degrees Celsius and an output voltage of 10V for T = 100 degrees Celsius. The quality of a circuit is defined as the sum of the squared discrepancies between the actual and desired output of the circuit for 21 equispaced values of temperature in the range of interest. To evaluate the quality of the circuits produced during the evolutionary search we have used the circuit simulator SPICE, which is available in the public domain (Vladimirescu 1994).

Figure 8 shows an example of a circuit found by the evolutionary algorithm. The relationship between temperature and output voltage of the circuit is shown in figure 9 and matches closely the desired linear relationship in the whole temperature range, with just a small deviation at T = 0 degrees Celsius.

Synthesis of Neural Controllers

To illustrate the application of AGE to the design of ANNs we describe now the synthesis of a neural controller (Dürr, Mattiussi, and Floreano 2006). The objective of the synthesis is a controller solving the double pole balancing problem without velocity information (figure 10). This is a standard benchmark problem used in the synthesis of neurocontrollers (Gruau 1994; Stanley and Miikkulainen 2002). Despite its apparent simplicity, it is a challenging problem that is related to interesting real-world applications such as the control of rocket stability.

As neuron devices for the AGE synthesis, we have chosen the dynamical neuron model described in Beer (1995). The reason for this choice is that, with suitable topology and sizing, a network of these devices can approximate arbitrarily well the trajectories of any smooth dynamic system for a finite interval of time. This ensures that if there is a smooth dynamic system that can solve the problem, the space of ANNs having these devices as nodes also contains a solution. To evaluate the quality of the solution, we computed the time that the evolved controller can keep the cart within given limits starting from given initial conditions. In addition, controllers that succeed in stabilizing the system for a certain time are evaluated with a set of different initial conditions in order to test their ability to generalize.

Figure 11 shows an example of ANN synthesized with AGE. Solutions found by AGE are typically very compact, display excellent generalization properties, and are obtained with a small number of evaluations when compared with other state-of-the art methods for the synthesis of neurocontrollers (Dürr, Mattiussi, and Floreano 2006).

Synthesis of Learning Neural Architectures

Most biological organisms are able to cope with complex and partially unpredictable environments. There is an obvious interest in the development of artificial agents capable to operate with the flexibility of biological organisms. Such agents could find application, for example, as autonomous robots for household and industrial tasks.

To date, using sets of coordinated prewired behavioral strategies, it has been possible to design autonomous agents such as floor cleaners and lawn mowers that are capable of dealing with tasks and environments of limited complexity. More flexible agents could be obtained using a learning mechanism capable of improving the performance of the agent based on the agent's past experience of interaction with the environment. An approach that looks particularly promising for the realization of this idea is based on the integration of a *value system* in the control system of the agent (Friston et al. 1994, Pfeifer and Scheier 2001). The value system would link the learning to the consequences of the agent's behavior judged according to its

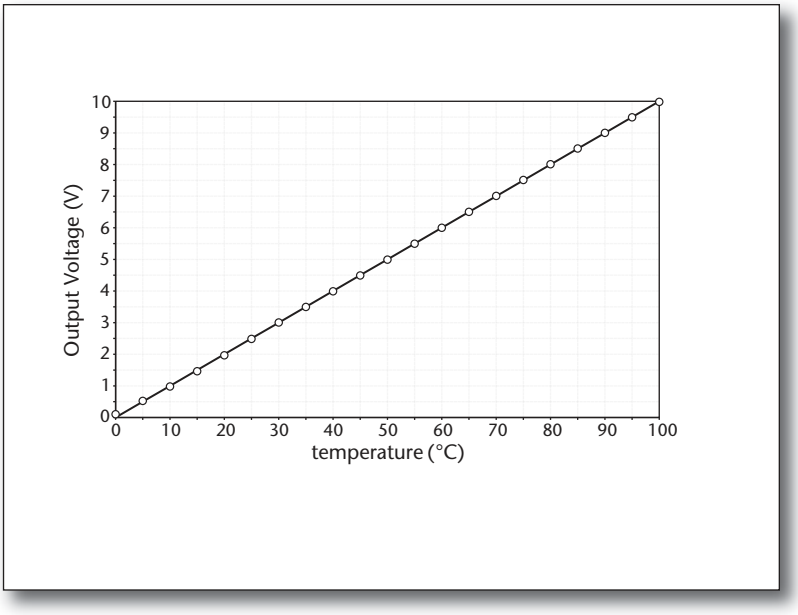


Figure 9. Relationship between Temperature and Output Voltage.

The circles show the output voltage of the evolved circuit shown in figure 8 for the set of circuit temperatures at which the circuit was tested during evolution. The background line represents the desired relationship.

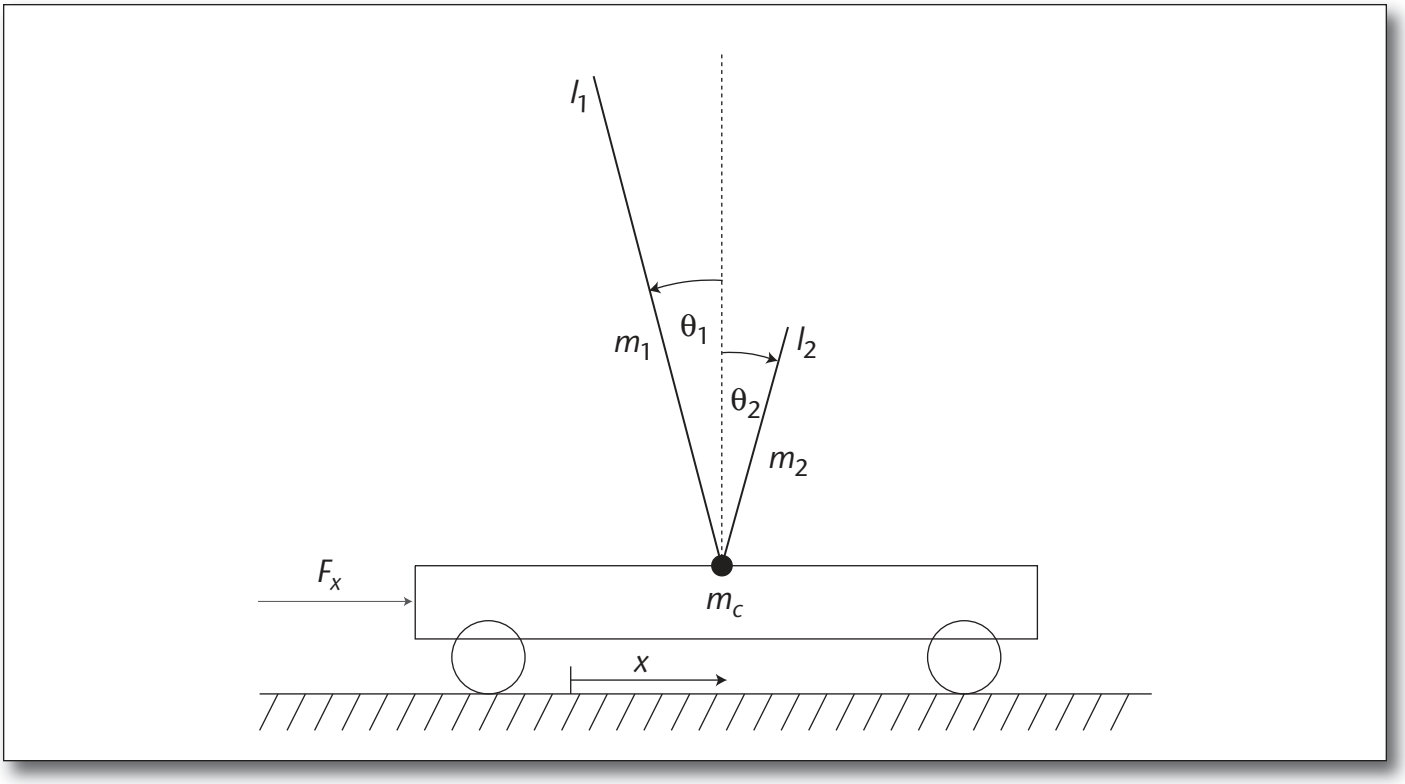


Figure 10. The Mechanical Setup of the Double Pole Balancing Problem without Velocity Information.

The setup consists of a cart with mass m_c and one degree of freedom x. Two poles of different lengths l_1 and l_2 and masses m_1 and m_2 are mounted on the cart with a rotational joint. The joint angles $_1$ and $_2$ as well as the position of the cart x are fed to a controller, which computes a force F_x that is applied to the cart. The controller has to stabilize the system in order to keep the joint angles and the position of the cart within given limits.

Evolutionary Algorithms

Evolutionary algorithms are a class of population-based stochastic search algorithms inspired by the process of Darwinian evolution (Fogel, Owens, and Walsh 1966; Holland 1975). An evolutionary algorithm maintains a collection (*population*) of tentative solutions called *individuals*. The goal of the search is defined through a user-defined measure of the quality of the individuals. The algorithm is required to find in the search space an individual with maximum quality or—in a more realistic engineering perspective—an individual satisfying a predefined criterion of quality. Typically, an initial population is generated by randomly sampling the search space. At each step of the algorithm the quality of the individuals that form the population is evaluated and a subset of the population (the parents) is selected for reproduction. The selection mechanism establishes a positive correlation between the quality of an individual and the number of new individuals (offspring) that it produces. The reproduction is carried out with some stochastic mutation and recombination of the parents in order to explore new regions the search space and combine the information carried by each parent. The probability distribution with which the offspring are generated in the search space is called the current *exploratory distribution* (Toussaint 2003). In so-called generational evolutionary algorithms, the old population is replaced by a new population composed of the newly formed individuals.

One peculiarity of evolutionary algorithms with respect to other stochastic search algorithms such as simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983) is that in evolutionary algorithms the individuals are represented in the form of a *genome*, which defines their *genotype*. The genotype representation is, in general, distinct from the form in which the individuals are represented in order to assess their quality. To evaluate the quality of an individual, its genotype must be decoded into a different representation called the *phenotype*. The mutations and recombinations applied during the reproduction, however, are applied to the genotype using a collection of probabilistically applied *genetic operators* that alter the genotype. This decouples the space in which the exploratory distribution is defined (the genotype space) from that in which the quality of the individuals is assessed (the phenotype space). Note, however, that the exploratory distribution in genotype space induces a corresponding exploratory distribution in the phenotype space.

An interesting consequence of the decoupling produced by the genotype-phenotype distinction arises when many different genotypes correspond to phenotypes having approximately the same quality. If the properties of the exploratory distributions allow the traversal of this set of genotypes, the set is called a *neutral network*. The existence of neutral networks permits continuing the exploration of the search space by random genetic drift when the current exploratory distribution does not provide access to individuals of improved quality with respect to the existing population. The probability of a stalling of the search is thus reduced because the drift on the neutral network can eventually give access to regions of the search space that contain individuals of higher quality.

Readers familiar with Monte Carlo methods (Robert and Casella 2004) might have found many resonating aspects in this description of evolutionary algorithms. In fact, evolutionary algorithms can be seen as a particular class of population Monte Carlo methods (Cappe et al. 2004) and the description of the former can be rephrased using the terminology of the latter. The individuals of evolutionary algorithms correspond to the particles of population Monte Carlo methods. The exploratory distribution in the phenotype space of evolutionary algorithms corresponds to the proposal distribution of population Monte Carlo methods (Robert and Casella 2004). When evolutionary algorithms use so-called *proportional selection*, the probability of reproduction of the individuals corresponds to the normalized importance weights of population Monte Carlo methods. The selection mechanism of evolutionary computation corresponds to the resampling mechanism of population Monte Carlo methods. The reproduction with mutation of the parents in evolutionary algorithms corresponds to the generation of new particles by sampling the proposal distribution of the resampled particles in population Monte Carlo methods. Finally, the genotype-phenotype distinction and the definition of the mutation in the genotype space can be interpreted as a particular way—specific to evolutionary algorithms—to define and parameterize the proposal distribution.

When population-based search methods are capable to maintain a sufficient population diversity (Mattiussi, Waibel, and Floreano 2004), the search space is explored in parallel by the individuals of the population. Typically, the quality of each individual can be evaluated independently from that of the other individuals of the population. This means that it is very easy to parallelize the evaluation of the quality of individuals, which is in general the most computationally expensive part of the search. Due to technological limitations, the increase in performance of single processors "came to a grinding halt" some years ago (Butler 2007). Consequently, the current trend in processor design is to increase the number of processing cores integrated on each chip and thus increase the global parallel processing capability rather than the processing power of each core. In this technological scenario, the easy parallelizability of population-based search methods like evolutionary algorithms is an additional bonus of these methods.

intended function. The learning would be activated to increase the probability of execution in a given context of actions that have produced favorable consequences in that context and to decrease the probability of execution of actions that led to unfavorable consequences.

The challenge in applying value-based learning to agents operating in realistic environments is to define a system architecture capable of estimating accurately contexts and values and to link them to the activation of the elements of a suitable repertoire of actions. A popular approach is to use a predefined set of actions controlled by a predefined system structure such as the actor-critic architecture inspired by the machine-learning technique of reinforcement learning (Sutton and Barto 1998). This approach is typically implemented using ANNs with hand-designed fixed structure, which realize the sensory preprocessing, the value system, and the action-selection mechanism. This puts on the designer most of the burden of guessing the correct structure of the value and action-selection systems for the problem at hand.

There is evidence that in biological organisms, evolved value-based learning systems are realized through the use of *neuromodulation* (Bailey et al. 2000). Specialized neuromodulatory neurons in the brain control activity-dependent plastic changes in the strength of the connections between other neurons. Applying this idea in the context of the automatic synthesis of ANNs, it is possible to bypass the difficulty of crafting the system architecture in value-based learning systems. It is very easy to set up AGE for the synthesis of neuromodulatory ANNs. To this end, it is sufficient to include in the set of neuron devices a neuron model that realizes a parametrized activity-dependent Hebbian learning (Niv et al. 2002) and a *neuromodulatory neuron* model that can connect to the standard neurons and modulate their learning (figure 12).

We have applied this approach to the synthesis of a learning neural architecture in a simulated foraging experiment (Soltoggio et al. 2007). In this experiment a simulated bee can collect nectar by landing on a field that contains two kinds of flowers (figure 13). The amount of nectar delivered by the flowers changes stochastically during the lifetime of the simulated bee. To maximize the amount of collected nectar the behavioral strategy must thus be adapted to the prevailing yield statistics of each flower. Using AGE we were able to evolve networks capable of maximizing the total amount of collected nectar in various scenarios. Figure 14 shows an example of a successfully evolved neuromodulatory architecture. The value-based learning strategy implemented by this and other ANNs evolved with AGE is quite general and can generalize to scenarios different from those used to assess the quality of the solution during evolution, outperforming the results obtained with hand-designed value-based learning architectures (Soltoggio et al. 2007).

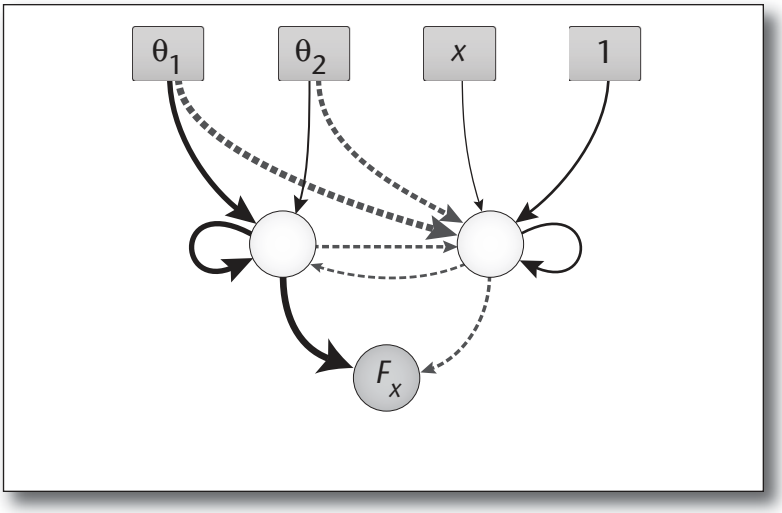


Figure 11. An Example of ANN Synthesized with AGE to Solve the Double Pole Balancing Problem.

Continuous arrows correspond to links with positive weight and dashed arrows to negative weights. The input denoted by 1 provides a fixed-value input (bias) to the network.

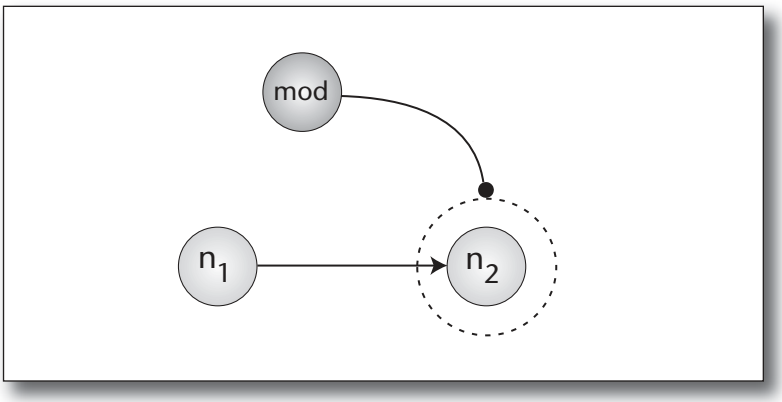


Figure 12. Neuromodulatory Neurons Permit the Implementation of a Value-Based Learning Mechanism in ANNs.

The basic learning mechanism changes the weight of the link between the two neurons n_1 and n_2 according to their activity. The neuromodulatory neuron, *mod*, modulates the basic learning mechanism and permits the synthesis of networks where the learning is activated only in particular circumstances.

Reverse Engineering of GRNs

In the previous sections, we have exemplified the application of AGE to the synthesis of different analog networks with prespecified functionalities. We shall now discuss the inverse problem, that is, unraveling an existing network with unknown topology and sizing given some data collected

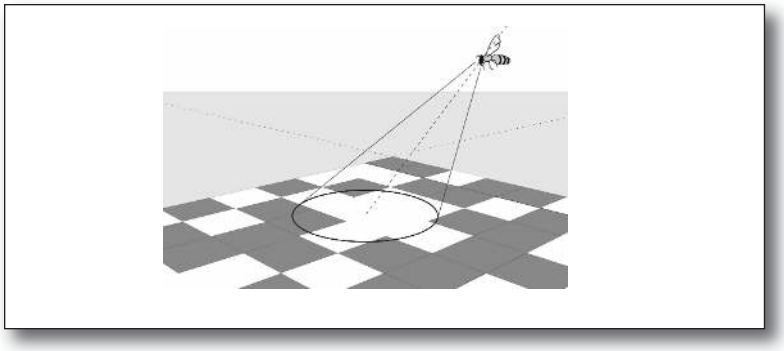


Figure 13. A Simulated Foraging Experiment.

In the experiment of synthesis of learning neural architectures, a simulated bee with a simple vision system flies over a field containing patches of blue and yellow flowers, represented here as dark and light squares. The quality of the behavior of the simulated bee is judged from the amount of nectar that the bee is able to collect in a series of landings.

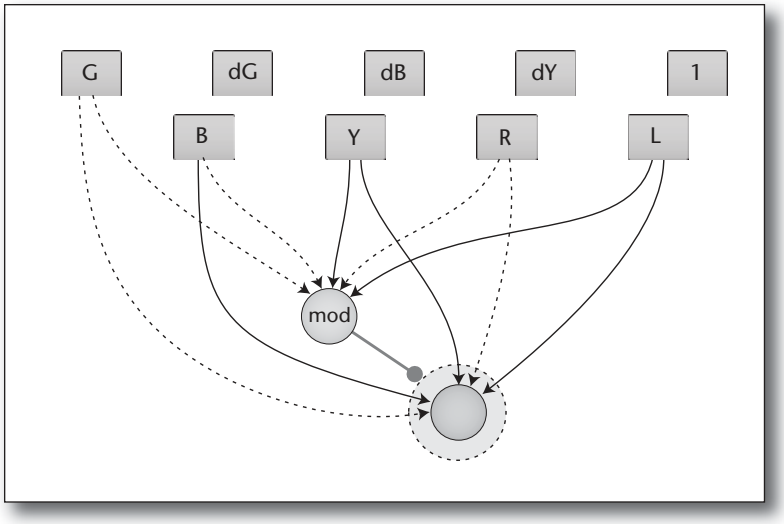


Figure 14. An Example of Neuromodulatory ANN Evolved with AGE, Which Solves the Foraging Task of Figure 13.

The *G*, *B*, and *Y* squares represent the color inputs of the visual system. The *R* square represents the reward input that gives the amount of nectar found on the flower on which the simulated bee has landed. The landing is signaled by the activation of the input *L*. The square denoted by 1 represents a fixed value input (bias), and the *dG*, *dB*, *dY* inputs represent the memory of the color observed just before landing. The *dG*, *dB*, and *dY* input signals were defined because they were used in experiments with hand-designed neural architectures for the same scenario (Niv et al. 2002). The figure shows that these inputs were found by the algorithm to be not necessary to solve the task and were thus left unconnected at the end of the evolutionary search. In the network shown here, the neuromodulatory *mod* neuron modulates the learning of all the connections between the inputs and the output neuron.

from observations of the network activity. This is the *reverse engineering* problem discussed in the introduction (see figure 5).

Setting up a reverse engineering experiment with AGE is straightforward. As for the synthesis of analog networks, the user first specifies the types of devices that can appear in the network. In the experiments discussed here, the devices are genes, and their dynamics are described by a standard phenomenological model of gene regulation, the so-called sigmoid model (Bolouri and Davidson 2002). Next, the AGE user has to define a measure of the quality of the evolved networks. Whereas in a synthesis experiment this measure is related to the desired network functionality, in reverse engineering it corresponds to the quality of the match between the gene expression data derived by simulating the evolved network and the gene expression data observed on the target network.

As a test case, we chose a nine-gene subnetwork of the SOS pathway of the bacterium *Escherichia Coli*. Using synthetic gene expression data from simulated gene perturbation experiments, we have successfully reverse-engineered both topology and sizing of this target network with high accuracy (Marbach, Mattiussi, and Floreano 2007). The evolved network that displays the best match with the gene expression data is shown in figure 15. The figure shows that AGE was able to determine 79 of the 81 possible links of the target network and to estimate correctly their enhancing or inhibitory nature. The discrepancy between the reverse-engineered network and the target network consists of one missing link (false negative) and one incorrectly identified link (false positive). Note that although in this example we have considered a network composed of just one type of device, it is possible to use AGE to reverse-engineer heterogeneous networks consisting of several different device types, such as gene-protein networks.

Discussion and Conclusion

Analog networks can realize complex functionalities using compact networks of relatively simple devices. This property follows from the possibility of effectively exploiting the rich nonlinear dynamics that can be generated by the interaction between the individual devices that form the network. When implemented in silicon, analog networks can realize impressive computational feats using a very small amount of power (Mead 1989). The price to pay for the remarkable compactness and efficiency of analog networks is in general a severe limitation of their programmability with respect to systems where the interactions and dynamics are constrained in order to achieve programmability (Conrad 1988). This is not a fatal limitation for analog networks, because there is a wealth of applications for which programmability is not required whereas power consumption and device count are critical factors.

The examples described in the previous section show that AGE is a powerful method for the synthesis and reverse engineering of analog networks,

which has the unique property of being easily adapted to various kinds of problems. In all cases where the criterion is applicable and a comparison is possible, AGE either outperforms or produces results comparable to those obtained with the best domain-specific methods for the automatic synthesis and reverse engineering of analog networks documented in the literature, from the point of view of the functionality of the networks, their size, and the required computational effort.

So far, the widespread use of analog networks has been limited by the difficulty of their synthesis by hand and by the dearth of effective tools for their automatic synthesis. Using evolutionary search algorithms based on AGE it is possible to overcome this design difficulty by automating the synthesis of analog networks. Even if AGE was conceived to optimize the representation and evolution of analog networks, the computational power required to perform the synthesis is in general considerable, and became available only recently. Thus, ironically, the digital computers that led to the demise of analog computers (Karplus 1958) give us now the resources for a renaissance of analog computation, in a new age of analog networks.

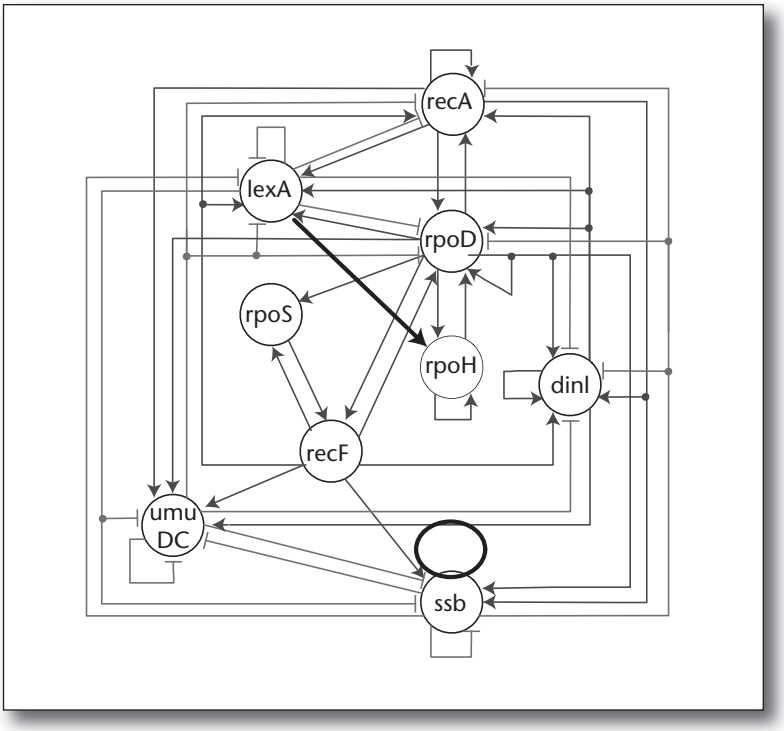


Figure 15. Topology of the E. coli SOS Network That Was Reverse-Engineered Using AGE.

The nodes of the network represent the genes. Arrows represent interactions that enhance gene expression. T ends denote interactions that inhibit gene expression. The network has been correctly reconstructed except for one missing link (thick arrow) and one incorrectly identified link (thick oval near the *ssb* node).

Acknowledgments

Many thanks to Simon Harding for reading and commenting on the manuscript. This work was supported by the Swiss National Science Foundation, grant no. 200021-112060.

References

Bailey, C. H.; Giustetto, M.; Huang, Y.-Y.; Hawkins, R. D.; and Kandel, E. R. 2000. Is Heterosynaptic Modulation Essential for Stabilizing Hebbian Plasticity and Memory? *Nature Reviews Neuroscience* 1(1): 11–20.

Baum, E. B. 1989. A Proposal for More Powerful Learning Algorithms. *Neural Computation* 1: 201–207.

Beer, R. 1995. On the Dynamics of Small Continuous-Time Recurrent Neural Networks. *Adaptive Behavior* 3(4): 469–509.

Bolouri, H., and Davidson, E. H. 2002. Modeling Transcriptional Regulatory Networks. *BioEssays* 24(12): 1118–1129.

Bray, D. 1995. Protein Molecules as Computational Elements in Living Cells. *Nature* 376: 307–312.

Butler, D. 2007. The Petaflop Challenge. *Nature* 448(7149): 6–7.

Cappe, O.; Guillin, A.; Marin, J. M.; and Robert, C. P. 2004. Population Monte Carlo. *Journal of Computational and Graphical Statistics* 13(4): 907–929.

Conrad, M. 1988. The Price of Programmability. In *The Universal Turing Machine: A Fifty Year Survey*, ed. R. Herken, 285–307. Oxford: Oxford University Press.

Dürr, P.; Mattiussi, C.; and Floreano, D. 2006. Neuroevolution with Analog Genetic Encoding. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN IX)*, 671–680. Berlin: Springer-Verlag.

Endy, D. 2005. Foundations for Engineering Biology. *Nature* 438(7067): 449–453.

Fogel, L. J.; Owens, A. J.; and Walsh, M. J. 1966. *Artificial Intelligence through Simulated Evolution*. New York: Wiley.

Friston, K. J.; Tononi, G.; Reeke, Jr., G. N.; Sporns, O.; and Edelman, G. M. 1994. Value-dependent Selection in the Brain: Simulation in a Synthetic Neural Model. *Neuroscience* 59(2): 229–243.

Gruau, F. 1994. Automatic Definition of Modular Neural Networks. *Adaptive Behaviour* 3(2): 151–183.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA: The MIT Press.

Karplus, W. J. 1958. *Analog Simulation: Solution of Field Problems*. New York: McGraw-Hill.

Kirkpatrick, S.; Gelatt, C. D.; and Vecchi, M. P. 1983. Optimization by Simulated Annealing. *Science* 220(4598): 671–680.

Koza, J. R.; Keane, M. A.; Streeter, M. J.; Mydlowec, W.; Yu, J.; and Lanza, G. 2003. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Norwell, MA: Kluwer.

Lazebnik, Y. 2002. Can a Biologist Fix a Radio?—Or, What

I Learned while Studying Apoptosis. *Cancer Cell* 2(3): 179–182.

Ljung, L. 1999. *System Identification: Theory for the User*. 2nd ed. Upper Saddle River, NJ: Prentice Hall.

Marbach, D.; Mattiussi, C.; and Floreano, D. 2007. Biomimetic Evolutionary Reverse Engineering of Genetic Regulatory Networks. In *Proceedings of the 5th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBIO 2007)*, 155–165. Berlin: Springer-Verlag.

Mattiussi, C. 2005. Evolutionary Synthesis of Analog Networks. Ph.D. Dissertation, EPFL, Lausanne.

Mattiussi, C., and Floreano, D. 2007. Analog Genetic Encoding for the Evolution of Circuits and Networks. *IEEE Transaction on Evolutionary Computation*. 11(5): 596–607.

Mattiussi, C.; Dürr, P.; and Floreano, D. 2007. Center of Mass Encoding: A Self-Adaptive Representation with Adjustable Redundancy for Real-Valued Parameters. In *Proceedings of the 2007 Conference on Genetic and Evolutionary Computation (GECCO 2007)*, 1304–1311. New York: ACM Press.

Mattiussi, C.; Waibel, M.; and Floreano, D. 2004. Measures of Diversity for Populations and Distances between Individuals with Highly Reorganizable Genomes. *Evolutionary Computation* 12(4): 495–515.

Mead, C. 1989. *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley.

Miller, J. F., and Thomson, P. 2000. Cartesian Genetic Programming. In *Proceedings of EuroGP2000*, 121–132. Berlin: Springer-Verlag.

Niv, Y.; Joel, D.; Meilijson, I.; and Ruppin, E. 2002. Evolution of Reinforcement Learning in Uncertain Environments: A Simple Explanation for Complex Foraging Behaviors. *Adaptive Behavior* 10(1): 5–24.

Pfeifer, R., and Scheier, C. 2001. *Understanding Intelligence*. Cambridge, MA: MIT Press.

Robert, C. P., and Casella, G. 2004. *Monte Carlo Statistical Methods*. 2nd ed. New York: Springer-Verlag.

Shapiro, J. 2005. A 21st Century View of Evolution: Genome System Architecture, Repetitive DNA, and Natural Genetic Engineering. *Gene* 345(1): 91–100.

Soltoggio, A.; Dürr, P.; Mattiussi, C.; and Floreano, D. 2007. Evolving Neuromodulatory Topologies for Reinforcement Learning-like Problems. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC 2007),* 2471–2478. Berlin: Springer-Verlag

Stanley, K., and Miikkulainen, R. 2002. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation* 10(2): 99–127.

Stormo, G. D., and Zhao, Y. 2007. Putting Numbers on the Network Connections. *BioEssays* 29(8): 717–721.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press.

Toussaint, M. 2003. The Evolution of Genetic Representations and Modular Adaptation. Ph.D. Dissertation, Institut fur Neuroinformatik, Ruhr-Universität Bochum, Germany.

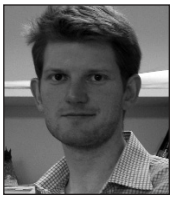
Vladimirescu, A. 1994. *The SPICE Book*. New York: Wiley.

Yao, X. 1999. Evolving Artificial Neural Networks. *Proceedings of the Institute of Electrical and Electronics Engineers (IEEE)* 87(9): 1423–1447.

Zebulum, R.; Vellasco, M.; and Pacheco, M. 2000. Variable Length Representation in Evolutionary Electronics. *Evolutionary Computation* 8(1): 93–120.

Claudio Mattiussi (claudio.mattiussi @epfl.ch) is a senior researcher at the Laboratory of Intelligent Systems, Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland. He conducts work on evolutionary computation, neural networks, and machine learning. His research interests include bioinspired artificial intelligence, systems biology, probabilistic engineering, evolutionary robotics, and the numerical formulation of physical field problems.

Daniel Marbach (daniel.marbach@ epfl.ch) is a Ph.D. student at the Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland, in the Laboratory of Intelligent Systems. He is currently developing novel approaches for the modeling and reverse engineering of gene regulatory networks. Other research interests include systems biology, evolutionary computation, bioinspired artificial intelligence, and evolutionary robotics.



Peter Dürr (peter.duerr@epfl.ch) is a Ph.D. student at the Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland, in the Laboratory of Intelligent Systems. His current research focus is the evolution of learning architectures in artificial neural networks. Other research interests include evolutionary computation, reinforcement learning, bioinspired robotics, and evolutionary biology.

Dario Floreano (dario.floreano@epfl. ch) is an associate professor in the School of Engineering at the Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland, where he is director of the Laboratory of Intelligent Systems. His research goals consist in extracting the principles of biological evolution and organization to develop artificial intelligence and robots that display novel or better functionalities. He wrote and edited several books in robotics and AI, the most recent one with Claudio Mattiussi, *Bioinspired Artificial Intelligence*. He is on the editorial board of 10 international journals, senior member of several professional societies, and cofounder of the International Society for Artificial Life.