On the Artificial Evolution of Neural Graph Grammars

Martin H. Luerssen (luer0001@infoeng.flinders.edu.au)
School of Informatics & Engineering
The Flinders University of South Australia
GPO Box 2100 Adelaide 5001 Australia

David M. W. Powers (powers@computer.org)
School of Informatics & Engineering
The Flinders University of South Australia
GPO Box 2100 Adelaide 5001 Australia

Abstract

Artificial neural networks and other connectionist models of computation are frequently credited with biological plausibility. Since biological systems are products of Darwinian evolution, network optimisation by artificial evolutions has considerable appeal. However, the computational expense of this can become prohibitive unless an emphasis is placed on modularity and reuse. Gene expression holds the answer to this. Genes are translated into proteins that self-organize into phenotypic traits such as the brain, with feedback loops controlling the further expression of genes. In this paper we present a generalization of this mechanism, a context-free graph grammar that describes a graph of finite-state automata. The graph is generated by replacing hyperedges with subgraphs of automata and other hyperedges according to a set of hypergraph productions. These automata need not be homogeneous, e.g. they may correspond to different types of neurons, reflecting the diversity of neurons in the brain. Desirable hypergraph productions are retrieved from a population of productions, which evolve by mutation of existing productions and subsequent selection against a user-defined criterion.

Introduction

Automata networks are networks of finite-state automata that update their respective states according to the states of their neighbours (Tchuente, 1988). Neural networks and cellular automata are specific instances of this connectionist model of computation and are distinguished by the types of finite-state automata and interconnection graph chosen. Unlike the serial architecture of modern computers, the scalability of automata networks is not reliant on a continued shrinkage of circuitry. Automata networks adhere to the same principle of distributed parallelism characterizing the human brain, yet while the brain seems slow and large compared to a computer, few would dispute its exceptional capabilities. The evident viability of this principle, however, has not translated into widespread implementation. Serial machines are easy to program – the step by step transformation from problem to solution is often intuitive. Conversely, handcrafting an automata network for a specific application can be exceedingly difficult.

The popularity of neural networks over other automata networks is principally due to the effective learning algorithms that can be and have been devised for them. Gradient search based techniques such as backpropagation (Rumelhart, Hinton & Williams, 1986) are the most widely employed of these, but necessitate finding a differentiable objective function that corresponds to the problem at hand. For weight learning in a multilayer perceptron with continuous threshold functions this is a straightforward exercise. For the majority of other learning problems, however, it is not. Even with a perceptron, we therefore encounter some limitations.

Weights are meant to model the synapses between biological neurons. Weight learning hence reflects synaptic change, which is a major source of functional change in the brain – but not the only. Adult neurogenesis, i.e. the perpetual incorporation of new neurons into neural structures, has been long conjectured and shown in models to play a crucial role in brain plasticity and learning (Cecchi et al., 2002). The classic example is the seasonal death and regrowth of neurons in canaries, which renew their yearly repertoire of songs in this manner (Alvarez-Buylla, Ling & Nottebohm 1992). Experimental evidence of adult neurogenesis in the human brain has also been reported (Eriksson et al. 1998).

Weight learning is not the only possible – or even necessary – form of adaptation in the perceptron either. In fact, the success of weight learning is often dependent on the number and arrangement of automata and edges within the network, which is not in any way adjusted by backpropagation. A range of heuristic algorithms have been suggested that attach or prune components of the network as warranted (Alpaydim, 1994), but none can claim to have an immediate biological equivalent.

Evolutionary Optimisation

Nervous systems are shaped not only by their environment but also by genetic factors, which are the product of evolution. Darwin's theory of evolution explains the adaptation of species by the principle of natural selection, which favours those species that are fittest, i.e. most adapted to their environment, and consequently most successful at reproducing (Darwin, 1859). The notion of universal

Darwinism (Dawkins, 1983) asserts that the same characteristics that make life susceptible to evolutionary change can also be found in other systems. Plotkin (1993) has proposed the g-t-r heuristic as the modus operandi of evolution, comprising the three phases of generation, testing, and regeneration.

Numerous evolutionary algorithms have been devised that exploit this heuristic for the optimisation of functions. Common to all of these is the notion of increasing the overall fitness of a population of 'species', say, different parameter choices, by replacing poor species with improved ones (Bäck, 1996). A population of offspring is generated from the existing population by means of applying mutation and/or recombination to selected multi-sets of species (see Figure 1). Those that are selected form a new population; the cycle is repeated until a species is deemed sufficiently fit as measured on some user-defined criterion. Evolutionary algorithms have been successfully and extensively applied towards the optimisation of automata networks, cellular automata (Mitchell, Crutchfield, & Das, 1997) as well as neural networks (Yao, 1999).

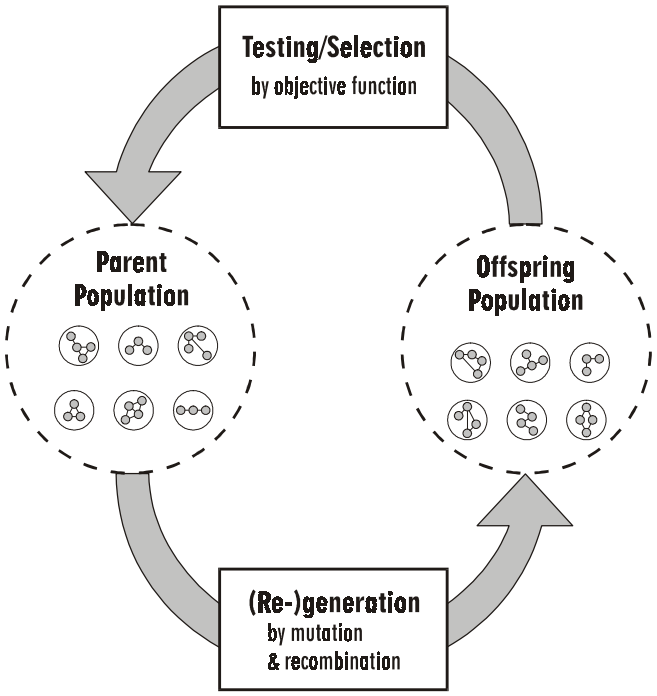

Figure 1: *Evolutionary algorithms generate an offspring population of candidates from an existing population; the fittest candidates are subsequently selected as parents for the next generation.*

While an evolutionary algorithm will always converge upon a globally optimal solution given sufficiently large mutations, it may take a good deal longer than the user can afford. The number of samples needed to estimate the space of all possible solutions increases exponentially with the number of parameters constituting each solution. This 'curse of dimensionality' plagues most statistical learning problems and is not easily overcome. A noteworthy example is the Golem Project (Lipson & Pollack, 2000), a distributed computing project utilizing the processing units of over 30000 Internet users to evolve robots. Because of (or despite of) the considerable resources being applied, it could be shown that robots beyond a certain level of complexity were impractical to evolve. As more components were added to the robot, the parameter permutations grew so large that the solution space became intractable.

Genes & Modules

Genetics has extended evolutionary theory by the concept of heredity, with genes acting as transfer units. The genes of an organism constitute its genotype, or genome, which is encoded into several chromosomes of DNA. Natural selection applies to the phenotype, which is an expression of the genotype within a given environment. Genes composed of DNA are transcribed into RNA, translated into polypeptides, and then processed into proteins which self-organize into phenotypic traits (Futuyma, 1998). Complex feedback loops control the further expression of genes.

Genetic algorithms set themselves apart from other evolutionary algorithms by acknowledging this genotype-phenotype distinction. The intricacies of molecular genetics are typically omitted, however, and a far simpler interpretation function translates what usually amounts to a binary string into a construct whose fitness can be assessed. With the simplest approach, a direct encoding scheme, each gene describes a specific detail of the phenotype, such as the value of a particular weight in the network. According to the building block hypothesis, gene sequences that confer above-average fitness (so-called building blocks) become increasingly dominant in subsequent generations and form instances of larger sequences that confer even greater fitness (Holland, 1992).

For a gene sequence to be reusable in different contexts, modularity is implied. A system can be understood as modular if it can be described in terms of parameters subsets – modules – that are more tightly coupled internally, i.e. between parameters of a single subset, than externally, i.e. between parameters of different subsets (Simon, 1996). Since a subset that is statistically independent of its context can be optimised on its own, a modular system becomes far simpler to optimise on the whole.

But while modularity can account for much of the effectiveness of genetic algorithms, most genetic algorithms do little to facilitate and exploit modularity. In a typical genetic algorithm the proximity of genes is unrelated to any dependencies between them, and crossover operators indiscriminately create and destroy modules. Moreover, an evolved solution with multiple instances of the same module, i.e. a solution that exhibits regularity, cannot be represented in a compact manner. This is specifically relevant when representing large constructs such as the human brain, which contains about 10^{11} neurons with an average of 10^5 connections each, but has to be encoded in a fraction of the $2x10^9$ base pairs that constitute the human genome. A direct encoding scheme is clearly unsuitable for this.

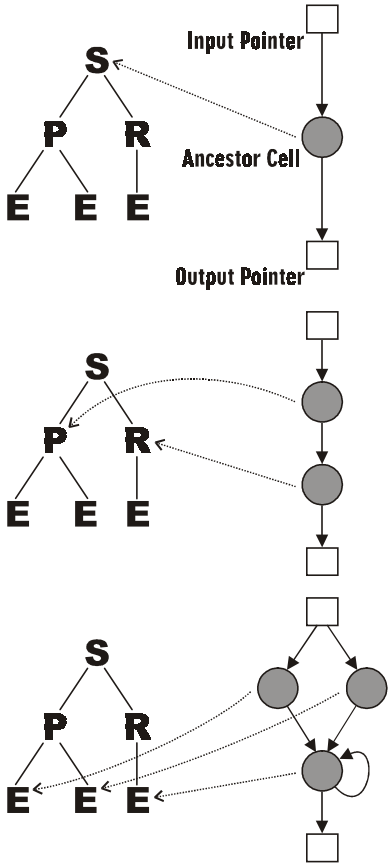

Figure 2: *Network construction according to cellular encoding. The left half of this figure depicts a tree of graph rewriting instructions (S = sequential division; P = parallel division; R = recurrent link; E = end program); the right half is the developing network. Dotted arrows point to the instruction that applies to the cell at each step.*

Algorithmic Encoding

The algorithmic complexity of a structure is the minimum length of an algorithm that generates it (Li & Vitányi, 1993). While algorithmic complexity is an incomputable measure in itself, it highlights the advantage of algorithmically describing a structure: Any regular structure can be coded by a more compact algorithm. The human genome is therefore best understood as an algorithm, not a blueprint. Each of our cells contains a copy of our genome, which governs our development. Cells vary in functionality depending on the genes they express. Kennedy (1998) simulated the evolution of cells in which such mechanisms operate. Our goal is to determine an algorithmic description for automata networks along these same lines.

Lindenmayer (1968) introduced L-systems in an attempt to describe the development (ontogenesis) of linear and branching structures in plants. L-systems are parallel string rewriting systems that rewrite a starting string into a new string by applying a set of production rules to all symbols of the string in parallel. Kitano (1990) and Boers & Sprinkhuizen-Kuyper (2001) employed L-systems to describe neural networks, necessitating a translation of the generated string into a network. Gruau (1994) proposed an alternative approach, called cellular encoding, which represents transformations to cells as nodes on a binary tree, as shown in Figure 2. Different subtrees reflect the different transformations applied to different cells following cell division. Subtrees can be swapped with those of other genomes, facilitating reuse of 'useful' subnetworks, analogous to the automatic reuse of functions in genetic programming (Koza, 1994). The reuse of subnetworks within the same network, however, requires definition of an operator that references other parts of the tree. This effectively adds cycles to a tree and raises a fundamental question – why not begin with a graph in the first place?

Gene expression in biological organisms sees a gene modulate the activity of itself and other genes. In a quantized model of this, each gene can be interpreted as an instruction that executes and then triggers other genes. The genotype thus becomes a directed graph of instructions (genes) that is continuously traversed to construct the phenotype. Our proposal is to evolve such graphs to construct automata networks. More formally, we intend for a context-free graph rewriting system, where each gene maps to a graph to be extended by other genes.

Method

Edges in a graph usually have arity two, that is, they connect two vertices. A hyperedge connects several vertices, and a graph with hyperedges is hence called a hypergraph. Whereas in a graph the vertices are often regarded as objects and edges as relationships between these objects, in a hypergraph it is quite intuitive to reverse these roles. This suits us well. We can define hyperedges as incomplete segments of the graph that are connected by vertices. The graph can grow by replacing labelled hyperedges with subhypergraphs according to a set of hypergraph productions (Habel, 1992). The choice of productions constitutes the genome that defines the output of this hyperedge replacement system.

A graph generated by the hyperedge replacement system becomes an automata network by adding the appropriate semantics to vertices and edges. Our decision to evolve automata networks rather than neural networks, as other systems have done, arises from the common misperception of neural networks as networks of homogeneous threshold units. Biological neurons typically belong to many categories with greatly differing computational properties, with some, e.g. starburst cells (Euler, Detwiler, & Denk, 2002), extending well beyond the classical threshold model.

Heterogeneous networks also offer advantages with respect to implementation. Certain operations can be accomplished far more speedily if done natively on a serial computer, rather than as a simulation of a neural network. For instance, the number of weights and hence training time of a multilayer perceptron classifying large images is often larger than the classification difficulty would warrant. Instead of resampling the images to a smaller size before being passed to the network, however, we may define a 'resampling neuron' (representative perhaps of a neural cluster) that is part of the network as any other neuron. Consequently, the evolutionary algorithm need only

optimise this graph of neurons, not several paradigmatically distinct stages of a hybrid architecture.

Neighbouring processors of different types, however, may not be able to communicate unless a uniform communication protocol exists for all processors. We instead propose to only allow compatible (but not necessarily identical) processors to become neighbours, effectively mimicking the existence of distinct neurotransmitter pathways found in the brain. For this purpose, we implemented a type verification system that requires each hyperedge to define the processor types by which it can be connected to other hyperedges. Hyperedges cannot be added to a subhypergraph unless they fit into the existing type context. Thus, only a single static check at the time of the conception of a subhypergraph is needed – but more on this later.

Cell Expression

For convenience, let us define a *cell* to correspond to the typed hyperedge and the associated production. A cell and its components are shown in Figure 3. Each cell comprises an array of typed *input references* to external processors that the components of this cell can access. The cell also includes one or more of its own processors that can access these references. Cells may additionally define a subhypergraph, or *cell graph*, by which the cell is replaced. An array of typed *output references* grants other cells access to the processors of this cell and the cells of the cell graph. We can broadly classify cells into *non-terminals* that define a cell graph, and *terminals* that only consist of a processor. In the latter case, the cell merely functions as a processor's interface to other processors.

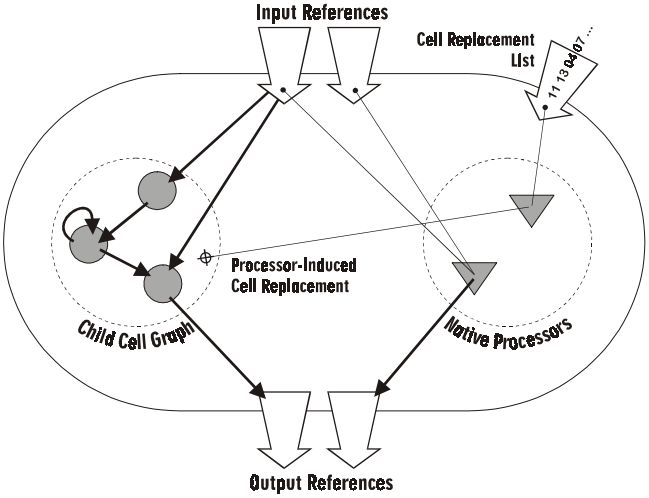

Figure 3: *A cell and its principal components (see text for description).*

During cell expression, non-terminal cells are replaced by graphs of other cells, which, if non-terminal, are replaced by further graphs. Starting from any non-terminal cell, a diverging pathway through a network of cell replacements is followed, until all remaining cells are terminals and the graph is completed (see Figure 4 for a depiction of this process). Having attempted to justify graph replacement on the basis of gene expression, this is where the relationship is strongest. It is not the genes that an organism holds that define it as much as the genes that it expresses. In our model the set of all cells represents the gene pool of all organisms. This pool is dispersed among different organisms in a distributed system like the biosphere. Modelling the gene pool as a single entity, however, gives us a duplication-free, easily assessable representation. Species, or networks of processing units, are differentiated by their start positions, i.e. start cells, within this gene pool.

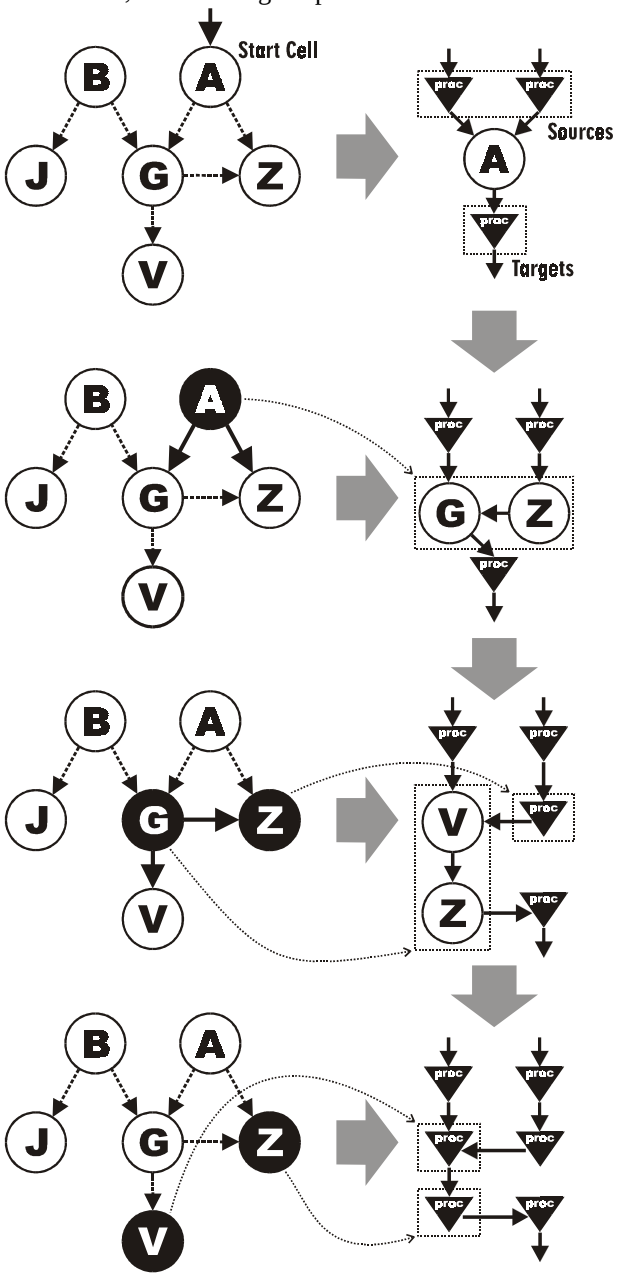

Figure 4: *Automata networks are constructed by cell replacement. Transitions between cells are depicted in the left half, the resulting network of processors in the right half. Starting with cell A, each replaced cell generates a subgraph of processors and/or cells that are also replaced (dotted arrows designate a cell's replacement graph).*

Cell Evolution

Given a distribution of pattern-label relationships, e.g. faces and associated names, we want to find the network that best approximates the mapping between pattern (input to the network) and label (output of the network). Not all cells represent candidates for such networks; some lead to subnetworks employable as building blocks by others. Unlike these, however, candidate networks must have an interface that can be linked into an array of external processors that provide and collect inputs and outputs. All processors, including those of the network, are then updated for several cycles until a stable output is generated. A performance rating for the network is computed by comparing the obtained outputs with the expected outputs.

The network is then re-grown from its start cell, but this time with a mutation in one of the cells that were expressed during this process. If the resulting network performs better than the earlier one, it replaces it. That is, a new cell is generated with the mutation and added to our set of cells that constitute the gene pool. If the original cell corresponded to a building block shared by other candidate networks, the start cell of the tested network and all subsequently expressed cells that refer to the new cell must be changed, as also shown in Figure 5.

This is an expensive process and there is a (physical) limit on how many different cells there can be, effectively an information limit on the gene pool. If this limit is exceeded, mutation cannot create additional new cells. Only if a cell is not referenced by any other cell will it be removed to make room for new cells. The choice of cells being mutated clearly has an effect of the efficiency of this system, and we are still seeking criteria for guiding this choice. Our model so far requires a cell to pass the number of references to it, as well as its performance rating (originating in the objective function), to the cells to which it refers. The rating of a cell is always the best rating it has received. Extensively referenced cells with high ratings are less likely to be mutated, thereby focusing any change upon less utilized cells where improvements are more likely and less expensive to achieve.

Mutations are applied to the cell graph defined by each cell. Several different kinds of topological mutation have been implemented, including addition, deletion or substitution of cells, as well as reshuffling of references between cells. For the context-free graph grammar not to violate any type constraints, each mutation must retain the type correctness of the graph to which it applies. It also needs to ensure that all references are defined, so as to avoid processors trying to access non-existent data. A feed-forward network topology can be enforced by assigning cells an order in which they are connected.

Topological mutations are the only means of change – no recombination (crossover) operator is modelled, because cell substitution is equivalent to subtree-swapping in genetic programming and therefore all that is needed. We have also not considered weight optimisation, as it can be well argued that competitive learning or backpropagation techniques are much more effective for this purpose than evolution, and can be easily integrated into the respective processors that constitute the network.

The discussed model of cell expression allows for recursion, which can be beneficial in attaining a network representation that is modular on several scales. However, it also requires a mechanism for termination, as the network may otherwise grow indefinitely. In our model the decision of replacing a cell has therefore been delegated to a *developmental processor* that is part of each cell. It can initiate cell replacement based on the state of the cell or components thereof. At present, its sole purpose is to tally recursions based on a list of previous cell replacements it has received from the corresponding processor of the cell that created it. If the number of recursions is greater than an evolved limit, the cell will not be replaced. This could leave some references between processors undefined – a gap in the graph. For this reason, all non-terminal cells are derivatives of terminal cells, where the cell graph overrides any existing output references to native processors. Network development can therefore be terminated at any stage, as all cells are possible terminals.

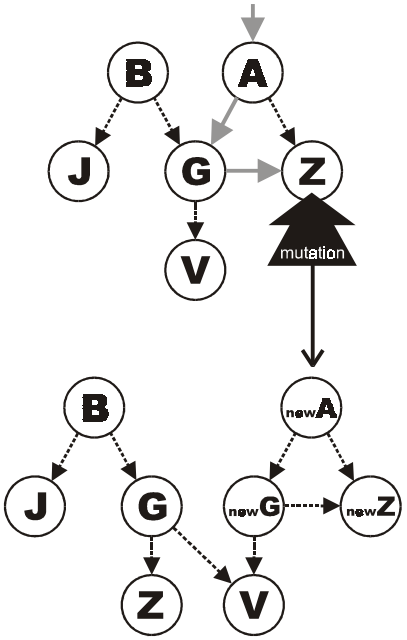

Figure 5: *Mutations are applied at any stage during graph rewriting. In this case, cell Z is mutated when triggered by cell G, which is triggered by cell A. If the resulting processor network is superior to the original network, Z will be replaced by a new Z with this mutation, and G and A replaced by a new G and A that refer to the new Z and G. The original Z and G must be retained, since B may perform worse with the new Z.*

Conclusion

In this paper we have proposed a graph rewriting system that can construct automata networks. Our goal is to attain a modular, hence efficient, representation for evolving the topology of these networks. The productions of a collective graph grammar, the equivalent of a gene pool, are shared among and evolved according to all the networks of a population. Experiments will have to show whether our approach leads to the expected improvements in

convergence, especially on large-scale networks needed for such tasks as image classification. An open question that requires further research is how to best evaluate the usefulness of new productions and keep the gene pool maximally diverse. We also plan to extend our model from growing networks to maintaining and adapting their topology in changing conditions. With network ontogenesis controlled by components of the network itself, the groundwork for this has been laid.

References

Alpaydin, E. (1994). GAL: Networks that grow when they learn and shrink when they forget. *International Journal of Pattern Recognition and Artificial Intelligence, 8,* 391-414.

Alvarez-Buylla, A., Ling, C. Y., & Nottebohm, F. (1992). High vocal center growth and its relation to neurogenesis, neuronal replacement and song acquisition in juvenile canaries. *Journal of Neurobiology, 23,* 396-406.

Bäck, T. (1996). *Evolutionary algorithms in theory and practice*. New York: Oxford University Press.

Boers, E. J. W. & Sprinkhuizen-Kuyper, I. G. (2001). Combined biological metaphors. In M.J.Patel, V. Honavar, & K. Balakrishnan (Eds.), *Advances in the evolutionary synthesis of intelligent agents* (pp. 153-183). Cambridge, MA: MIT Press.

Cecchi, G. A., Petreanu, L. T., Alvarez-Buylla, A., & Magnasco, M. O. (2001). Unsupervised learning and adaptation in a model of adult neurogenesis. *Journal of Computational Neuroscience, 11,* 175-182.

Darwin, C. (1859). *On the origin of species by means of natural selection*. London: Murray.

Dawkins, R. (1983). Universal darwinism. In D.S.Bendall (Ed.), *Evolution from molecules to man* (pp. 403-425). Cambridge, MA: Cambridge University Press.

Euler, T., Detwiler, P. B., & Denk, W. (2002). Directionally selective calcium signals in dendrites of starburst amacrine cells. *Nature, 418,* 845-852.

Eriksson, P. S., Perfilieva, E., Bjork-Eriksson, T., Alborn, A. M., Nordborg, C., Peterson, D. A. et al. (1998). Neurogenesis in the adult human hippocampus. *Nature Medicine, 4,* 1313-1317.

Futuyma, D. J. (1998). *Evolutionary biology*. (3rd ed.) Sunderland, MA: Sinauer Associates, Inc.

Gruau, F. (1994). *Neural network synthesis using cellular encoding and the genetic algorithm.* Doctoral dissertation, l'Ecole Normale Supérieure de Lyon.

Holland, J. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Cambridge, MA: MIT Press.

Kennedy, P. J. (1998). *Simulation of the evolution of single celled organisms with genome, metabolism and time-varying phenotype.* Doctoral dissertation, University of Technology, Sydney.

Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation systems. *Complex Systems, 4,* 461-476.

Koza, J. R. (1994). *Genetic programming II: automatic discovery of reusable programs*. Cambridge, MA: MIT Press.

Habel, A. (1992). *Hyperedge replacement: grammars and languages*. Berlin: Springer-Verlag.

Li, M. & Vitányi, P. (1993). *An introduction to Kolmogorov complexity and its applications*. New York: Springer-Verlag.

Lindenmayer, A. (1968). Mathematical models for cellular interaction in development, parts I and II. *Journal of Theoretical Biology, 18,* 280-315.

Lipson, H. & Pollack, J. (2000). *The Golem project*. Retrieved May 26, 2003, from Brandeis University, CS Dept. Web site: http://demo.cs.brandeis.edu/golem/

Mitchell, M., Crutchfield, J. P., & Das, R. (1997). Evolving cellular automata to perform computations. In T.Bäck, D. B. Fogel, & Z. Michalewicz (Eds.), *Handbook of evolutionary computation*. Bristol: Oxford University Press.

Plotkin, H. C. (1993). *Darwin machines and the nature of knowledge*. Cambridge, MA: Harvard University Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D.E.Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: explorations in the microstructure of cognition, volume 1: foundations* (pp. 318-362). Cambridge, MA: MIT Press.

Simon, H. A. (1996). *The sciences of the artificial*. (3rd ed.) Cambridge, MA: MIT Press.

Tchuente, M. (1988). Computation on finite networks of automata. In C.Choffrut (Ed.), *Automata networks* (pp. 53-67). Berlin: Springer-Verlag.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE, 87,* 1423-1447.