Properties of Genetic Representations of Neural Architectures

Karthik Balakrishnan and Vasant Honavar *
Department of Computer Science
Iowa State University
Ames, IA 50011. U.S.A

Abstract

Genetic algorithms and related evolutionary techniques offer a promising approach for automatically exploring the design space of neural architectures for artificial intelligence and cognitive modeling. Central to this process of evolutionary design of neural architectures (EDNA) is the choice of the *representation scheme* that is used to *encode* a neural architecture in the form of a gene string (*genotype*) and to *decode* a genotype into the corresponding neural architecture (*phenotype*). The representation scheme used not only constrains the class of neural architectures that are representable (evolvable) in the system, but also determines the *efficiency* and the *time-space* complexity of the evolutionary design procedure as a whole. This paper identifies and discusses a set of properties that can be used to characterize different representations used in EDNA and to design or select representations with the necessary properties for particular classes of applications.

1 Introduction

Artificial Neural Networks (ANN) offer an attractive paradigm of computation for many applications (e.g., pattern recognition, system identification, cognitive modeling etc.) for a number of reasons including: potential for massively parallel computation, robustness in the presence of noise, resilience to the failure of components, amenability to adaptation and learning (through the modification of computational structures so as to change their behavior) etc. Practical applications of ANN require the choice of a suitable network topology and the processing functions computed by individual units. In addition, parameters of the network (typically the *weights*) are also specified either explicitly or implicitly through a learning algorithm. In the latter case, a set of examples that specify the network's desired input-output behavior are also provided, and exploited by the learning algorithm in determining the relevant parameters.

Though numerous learning algorithms (with varying abilities) have been formulated, designing the network architecture, is still a process of trial and error, relying on heuristics and past experience with similar applications. Since the performance of an ANN on a given application is critically dependent on the efficacy of the training algorithm used, which in turn is constrained by the choice of the network architecture, techniques for designing *good* (efficient, fast, robust) network architectures are of great interest. This entails searching the space of neural architectures. Since this space is large, *exhaustive search* is infeasible. *Heuristic search* techniques (e.g., constructive or generative learning algorithms) (Honavar & Uhr, 1993; Chen *et al.*, 1995) can often be used to incrementally build problem-specific architectures. *Genetic algorithms* (GA) (Holland, 1975; Goldberg, 1989; Koza, 1992) offer an attractive approach for efficiently searching vast, complex and deceptive problem spaces. The use of GA to search the space of neural architectures for near-optimal designs is therefore a natural extension of constructive algorithms for designing ANN. Several researchers have recently begun to investigate techniques for designing neural architectures using GAs (see (Balakrishnan & Honavar, 1995b) for a bibliography). The focus of this paper is on the characterization of genetic representations used in EDNA systems.

This paper is organized as follows: The rest of this section briefly summarizes the process of EDNA. Section 2 introduces several properties of genetic representations of neural architectures; and Section 3

*This research was partially supported by the National Science Foundation through the grant IRI-9409580 to Vasant Honavar. The authors would like to thank Rajesh Parekh for his helpful comments on the paper.

illustrates the characterization of properties of a prototypical EDNA system and concludes with a summary and directions for future research.

1.1 Evolutionary Design of Neural Architectures

The elements of EDNA are shown in Figure 1.

Evolutionary algorithms (Holland, 1975; Goldberg, 1989) are models of processes that appear to be at

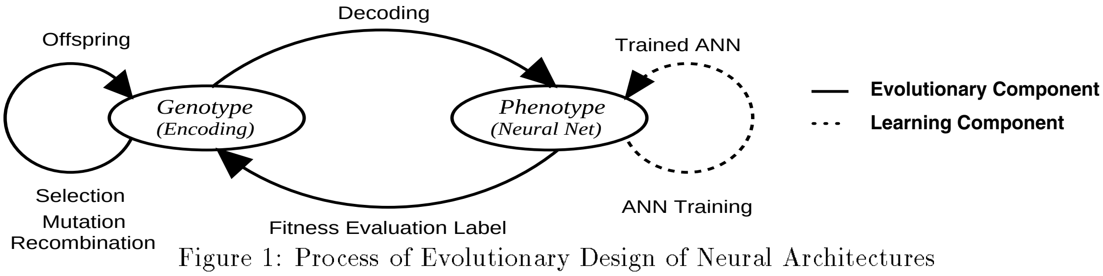

Figure 1: Process of Evolutionary Design of Neural Architectures

work in biological evolution. Such systems work with populations of *genotypes*, where a genotype can be thought of as an arrangement (possibly a string) of *genes*. Each gene takes on values, called *alleles*, from a suitably defined domain of values. Each genotype typically encodes for one, although possibly several, *phenotypes* (which correspond to elements of the space of candidate solutions). In EDNA, genotypes are *encodings* of corresponding ANN (phenotypes). The components of the genetic code are the genes which may represent numeric values, or even complex symbol structures that can be translated into ANN by a suitable *decoding* process. The process of decoding can be very simple or fairly involved, depending on the representation (encoding and decoding mechanisms) used. The resulting phenotype (ANN) is evaluated on the given task (or a set of tasks) with respect to suitable task-specific performance measures or user-specified design constraints (e.g., size, cost etc.). This evaluation of the phenotype assigns a *fitness label* to the corresponding genotype (see Figure 1).

The evolutionary procedure works by preferentially selecting genotypes for reproduction based on their fitness and introducing variety into the population through genetic operators such as mutation, crossover and inversion. If this works, over many generations, the population gradually evolves towards genotypes that correspond to high fitness phenotypes. This general procedure (perhaps with minor variations) is at the heart of most EDNA systems.

1.2 An Example of EDNA

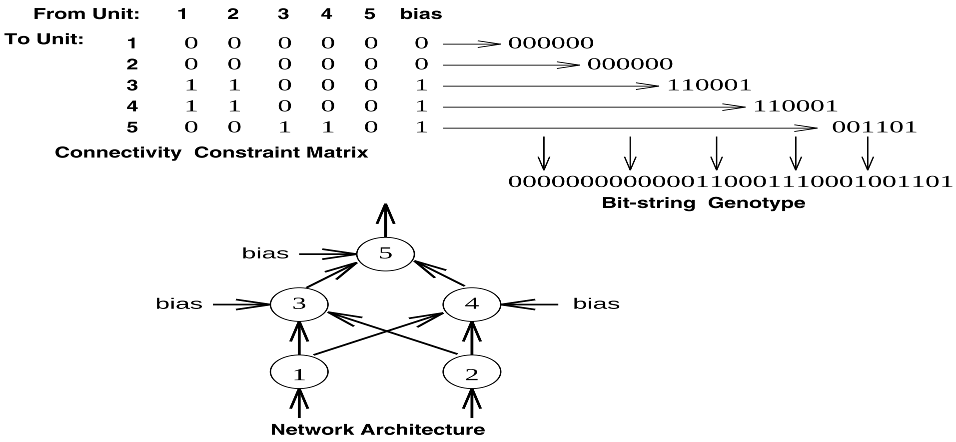

Figure 2: An example of EDNA: Miller et al. 1989

Miller et al. (1989) propose a system in which the topology of a network with N units is represented by a *connectivity constraint matrix*, C of dimension $N \times (N+1)$ wherein, the first N columns specify the constraints on the connections between the N units, and the final column codes for the connection that corresponds to the *threshold* of each unit. Each entry C_{ij}, of the connectivity constraint matrix indicates the nature of the constraint on the connection from unit j to unit i (or the constraint on the threshold bias of unit i if $j = N + 1$); a 0 indicates the *absence* and a 1 indicates the *presence* of a *trainable* connection between the corresponding units. A *genotype* is constructed by concatenating the rows of the matrix, to yield a bit-string of length $N \times (N + 1)$. This is shown in Figure 2.

The fitness of the genotype is evaluated as follows: First, the genotype is *decoded* into the corresponding ANN. All feedback connections are ignored even though they can be specified in the genotype. Thus, this system evolves purely *feed-forward* networks. Next, all the connections in the network are set to small random values and trained for a fixed number of epochs using *generalized delta rule* on a given set of training examples. The *total sum squared error* ($\mathcal{E}$) of the network, at the end of the training phase, is used as the *fitness* measure, with *low* values of $\mathcal{E}$ corresponding to better performance and hence a higher fitness label for the corresponding genotype.

The system maintains a population of such genotypes (bit-strings), and uses a *fitness-proportionate* selection scheme for choosing parents for reproduction. The genetic operator *crossover* swaps rows between parents while *mutation* randomly flips bits in the genotype with some low, pre-specified probability. Miller et al. (1989) report results obtained on the evolution of ANN for *XOR, four-quadrant* and *pattern-copying* problems.

2 Genetic Representations of Neural Architectures

Central to the EDNA procedure, as outlined above, is the choice of a *genetic representation* (i.e., an encoding scheme and the corresponding decoding scheme). The choice of the representation is critical since it dictates the class of neural architectures that could possibly evolve in the system. Further, the genetic operators for the system are defined based (largely) on the representation chosen. These factors contribute directly or indirectly to the *efficiency* (with respect to time, compactness of the resulting ANN, etc.) of the evolutionary procedure. Thus, a careful characterization of the properties of genetic representations as they relate to the performance of EDNA systems (and more generally, evolutionary algorithms) is a necessary venture.

Some authors have defined *properties* of genetic representations for evolving ANN. However, most such characterizations have been restricted to a specification of the properties of the encoding scheme without considering in detail, the associated decoding process (Collins & Jefferson, 1990; Gruau, 1994). A closer look at several of the EDNA systems strongly argues for the need for a more complete and precise characterization of the properties of genetic representations, taking both encoding as well as decoding processes into account. This paper is an attempt in that direction.

This section identifies and defines some of the key properties of genetic representations of neural architectures. We expect these definitions to get more refined as we examine a larger variety of EDNA systems more closely. Since space does not permit a detailed exposition here, the interested reader is referred to (Balakrishnan & Honavar, 1995a) for an analysis of properties of several EDNA systems.

2.1 Properties of Genetic Representations

We begin by stating some useful definitions.

(D1) $\underline{\mathcal{G}}$ – the space of genotypes representable in a chosen encoding scheme. $\mathcal{G}$ may be explicitly enumerated or implicitly specified using a grammar Γ whose language $L(\Gamma) = \mathcal{G}$.

(D2) $\underline{p = \mathcal{D}(g, \mathcal{E}_D)}$ – where $\mathcal{D}$ is the decoding function that produces the phenotype p corresponding to the genotype g possibly under the influence of the environment $\mathcal{E}_D$ (for e.g. the environment may set parameters of the decoding function). A value of λ for $\mathcal{E}_D$ denotes the lack of direct interaction between the decoding process and the environment. Further, $\mathcal{D}$ may be *stochastic*, with an underlying probability distribution over the space of phenotypes.

(D3) $\underline{p_2 = \mathcal{L}(p_1, \mathcal{E}_L)}$ – Learning procedure, generates phenotype p_2 from phenotype p_1 under the influence of environment $\mathcal{E}_L$. The environment may provide the training examples, set the free parameters (e.g., the learning rate used by the algorithm) etc. We will use $\mathcal{L} = \lambda$ to denote the absence of any form of learning in the system

(D4) $\underline{\mathcal{P}}$ – the space of all phenotypes that can be constructed (in principle) given a particular genetic representation scheme: $(\forall p \in \mathcal{P})(\exists g \in \mathcal{G})\ [(p_1 = \mathcal{D}(g, \mathcal{E}_D)) \wedge (p = \mathcal{L}(p_1, \mathcal{E}_L))]$

(D5) $\underline{\mathcal{S}}$ – the set of *solution networks*, i.e., neural architectures or phenotypes (a subset of $\mathcal{P}$) that satisfy the desired performance criterion (as measured by the fitness function π) in a given environment $\mathcal{E}_\pi$.
If an EDNA system with a particular representation $\mathcal{R}$ is to successfully find solutions, even in principle, $\mathcal{S} \subseteq \mathcal{P}$, or, at the very least, $\mathcal{S} \cap \mathcal{P} \neq \emptyset$. In other words, there must be at least one solution network that can be constructed given the chosen representation $\mathcal{R}$.

(D6) $\underline{\mathcal{A}}$ – the set of *acceptable* neural architectures. $\mathcal{A}$ may in general, be different from $\mathcal{P}$. However, it must be the case that $\mathcal{A} \cap \mathcal{S} \neq \emptyset$ if a particular EDNA system is to be useful in practice.

We now identify some properties of genetic representations of neural architectures. Unless otherwise specified, we will assume the following definitions are with respect to an *a-priori* fixed choice of $\mathcal{E}_D, \mathcal{L}$ and $\mathcal{E}_L$.

1. **Completeness:** A representation $\mathcal{R}$ is *complete* if every neural architecture in the solution set can be constructed (in principle) in the system. Formally, the following two statements are equivalent definitions of completeness.
   - $(\forall s \in \mathcal{S})(\exists g \in \mathcal{G})[(p_1 = \mathcal{D}(g, \mathcal{E}_D)) \wedge (s = \mathcal{L}(p_1, \mathcal{E}_L))]$
   - $\mathcal{S} \subseteq \mathcal{P}$

2. **Closure:** A representation $\mathcal{R}$ is *completely closed* if *every* genotype decodes to an acceptable phenotype. The following two assertions are both equivalent definitions of closure.
   - $(\forall g \in \mathcal{G})[(p_1 = \mathcal{D}(g, \mathcal{E}_D)) \wedge (\mathcal{L}(p_1, \mathcal{E}_L) \in \mathcal{A})]$
   - $\mathcal{P} \subseteq \mathcal{A}$

   A representation that is not closed can be transformed into a closed system by *constraining* the decoding function appropriately. Additionally, if the genetic operators are designed to have the property of closure, then one can envision *constrained closure* wherein all genotypes do not correspond to acceptable phenotypes, however, closure is guaranteed since the system *never* generates the invalid genotypes (see (Balakrishnan & Honavar, 1995a) for details).

3. **Compactness:** Suppose two genotypes g_1 and g_2, both decode to the same phenotype p, then g_1 is said to be more *compact* than g_2 if g_1 occupies less *space* than g_2:
   - $(p_1 = \mathcal{D}(g_1, \mathcal{E}_D)) \wedge (\mathcal{L}(p_1, \mathcal{E}_L) = p) \wedge (p_2 = \mathcal{D}(g_2, \mathcal{E}_D)) \wedge (\mathcal{L}(p_2, \mathcal{E}_L) = p) \wedge \mid g_1 \mid < \mid g_2 \mid$

   where $\mid g \mid$ denotes the size of storage for genotype g.

   This definition corresponds to *topological-compactness* defined by Gruau (94). His definition of *functional-compactness* – which compares the genotype sizes of two phenotypes that exhibit the same behavior, can be expressed in our framework (for solution networks) as
   - $(p_1 = \mathcal{D}(g_1, \mathcal{E}_D)) \wedge (\mathcal{L}(p_1, \mathcal{E}_L) \in \mathcal{S}) \wedge (p_2 = \mathcal{D}(g_2, \mathcal{E}_D)) \wedge (\mathcal{L}(p_2, \mathcal{E}_L) \in \mathcal{S}) \wedge \mid g_1 \mid < \mid g_2 \mid$

4. **Scalability:** Several notions of scalability are of interest. For the time being, we will restrict our attention to the *change* in the *size* of the phenotype (as measured for example, in terms of the number of units, connections, or modules). This change in the size of the phenotype manifests itself as a change in the *size* of the encoding (space needed to store the genotype), and a corresponding change in *decoding time*. We can characterize the relationship in terms of the *asymptotic order of growth* notation commonly used in analyzing computer algorithms — $O(\cdot)$.

   For instance, let $n_{N,C} \in \mathcal{A}$ be a network (phenotype) with N units and C connections (the actual connectivity pattern does not really matter in this example). We say that the representation is $O(K)$–*size-scalable with respect to units* if the addition of *one* unit to the phenotype $n_{N,C}$ requires an increase in the size of the corresponding genotype by $O(K)$, where K is some function of N and C. Size-scalability of encodings with respect to connections, modules, etc. can be similarly defined.

The representation is said to be $O(K)-$ *time-scalable with respect to units* if the time taken for decoding the genotype for $n_{N+1,C}$ exceeds that used for $n_{N,C}$ by no more than $O(K)$. Similarly, time-scalability with respect to the number of connections, modules, etc. can also be defined.

5. **Multiplicity:** A representation $\mathcal{R}$ is said to exhibit *genotypic multiplicity* if multiple genotypes decode to an identical phenotype. In other words, the decoding function is a many to one mapping from the space of genotypes to the corresponding phenotypic space.
   • $(\exists n \in \mathcal{P})\ (|\ \{g \in \mathcal{G}\ |(p = \mathcal{D}(g, \mathcal{E}_D)) \wedge (n = \mathcal{L}(p, \mathcal{E}_L))\}\ | > 1)$
   Genotypic multiplicity may result from a variety of sources including, the encoding and decoding mechanisms.

   A representation $\mathcal{R}$ is said to exhibit *phenotypic multiplicity* if different instances of the same genotype can decode to different phenotypes. In other words, the decoding function is a one to many mapping of genotypes into phenotypes.
   • $(\exists g_1, g_2 \in \mathcal{G})[(p_1 = \mathcal{D}(g_1, \mathcal{E}_D)) \wedge (n_1 = \mathcal{L}(p_1, \mathcal{E}_L))) \wedge (p_2 = \mathcal{D}(g_2, \mathcal{E}_D)) \wedge (n_2 = \mathcal{L}(p_2, \mathcal{E}_L))) \wedge (g_1 = g_2) \wedge (n_1 \neq n_2)]$
   Phenotypic multiplicity may result from several factors including the effects of the environment, learning, or stochastic aspects of the decoding process.

6. **Ontogenetic Plasticity:** A representation $\mathcal{R}$ exhibits *ontogenetic plasticity* if the determination of the phenotype corresponding to a given genotype, is influenced by the environment. This may happen as a result of either environment-sensitive developmental processes (in which case $\mathcal{E}_D \neq \lambda$), or learning processes (in which case $\mathcal{L} \neq \lambda$).

7. **Modularity:** Gruau's (94) notion of *modularity* is as follows: Suppose a *network* n_1 includes several instances of a subnetwork n_2 then the encoding (genotype) of n_1 is *modular* if it codes for n_2 only once (with instructions to copy it which would be understood by the decoding process). Modularity is closely tied to the existence of *organized structure* or regularity in the phenotype that can be concisely expressed in a genotype in a form that can be used by the decoding process. Other notions of modularity having to do with *functional* modules, *recursively-defined* modules etc. are worth exploring (Balakrishnan & Honavar, 1995a).

8. **Redundancy:** Redundancy can manifest itself at various levels and in different forms in an EDNA system. Redundancy often contributes to the robustness of the system in the face of failure of components or processes. For instance, if the reproduction and/or decoding processes are error-prone, an EDNA system can benefit from *genotypic redundancy* (wherein the genotype contains redundant genes) or *decoding redundancy* (wherein the decoding process reads the genotype more than once). If the phenotype is prone to failure of components (units, connections, sub-networks), an EDNA system can benefit from *phenotypic redundancy.* It is worth noting that genotypic redundancy does not necessarily imply phenotypic redundancy and vice versa (depending on the nature of the decoding process). This simply reiterates the importance of examining the entire representation (encoding as well as decoding) when addressing properties of EDNA systems. Also note that there are many ways to realize both genotypic as well as phenotypic redundancy: by replication of identical components (structural redundancy) or by replication of functionally identical units, or by building in modules or processes that can dynamically restructure themselves when faced with failure of components etc. (von Neumann, 1956).

9. **Complexity:** Complexity is perhaps one of the most important properties of any EDNA system. It is difficult to characterize satisfactorily using any single definition. It is probably best to approach this using several different notions of complexity including: various useful notions of *structural complexity* of genotypes, *decoding complexity*, *computational* (*space/time*) *complexity* of each of the components of an EDNA system (including decoding of genotypes, fitness evaluation, reproduction, etc.); and perhaps even other measures inspired by *information theory.*

3 Discussion

The main objective of this paper is to identify properties of genetic representations of neural architectures that are relevant in an operationally useful characterization of different EDNA systems. The following table illustrates such a characterization of the EDNA system proposed by Miller et al. (1989).

Property	Satisfied	Comments
Completeness	$\surd$	With respect to the set of feed-forward ANN only. Cannot evolve recurrent ANN.
Closure	$\times$	Since phenotype can have *isolated* units, no *paths* from input to output, etc.
Compactness	$\times$	Topological compactness cannot be defined since each phenotype has a unique representation. Functional compactness is possible.
Space Scalability	$\surd$	$O(N)$ with respect to units, where N=NumberOfUnits in the phenotype. Independent of change in number of connections.
Time Scalability	$\surd$	$O(N)$ with respect to units. Independent of change in number of connections.
Multiplicity	$\times$	Each phenotype has a unique representation.
Ontogenetic Plasticity	partly	Decoding process is fixed ($\mathcal{E}_D = \lambda$), however, ($\mathcal{L} \neq \lambda$). Uses generalized delta rule for training ANN.
Modularity	$\times$	Genotype specifies individual connections not modules.
Genotypic Redundancy	$\times$	No redundancy in genotype – one gene for each connection.
Phenotypic Redundancy	$\times$	Cannot directly incorporate redundancy. Redundant units and modules are possible, connections are not.
Space Complexity	$\surd$	Dictated by size of the genotypes required for the task.
Time Complexity	$\surd$	Dictated by the use of GA and back-propagation training.

Table 1: Properties of the EDNA system proposed by Miller et al. (1989)

It is our hope that a careful analysis of properties of genetic representations would help identify good choices of genetic representations in different applications. Suppose we have to choose an EDNA system for the design of ANN controllers for robots that have to operate in hazardous, and largely a-priori unknown environments. Examples of such applications include exploration of unknown terrains, nuclear waste cleanup, etc. The task environment and user-specified design constraints should govern the choice of EDNA systems. Since robots in such environments are required to plan and execute sequences of actions (where each action in a sequence may be dependent on previous actions performed as well as the sensory inputs), a *recurrent* ANN is probably needed. Further, if the system is to be used to design robots capable of functioning in different (and largely a-priori unknown) environments, it would benefit from *ontogenetic plasticity* (that would allow the environment to shape the resulting ANN). The hazardous nature (e.g., in nuclear waste cleanup) or remoteness of the environment (e.g., in the case of robots used to explore distant planets) makes it desirable that the design be robust in face of component failures etc., which calls for *phenotypic redundancy* of some form. In addition, implementation technology and cost considerations might impose additional constraints on the design of the ANN controller. For instance, hardware realization using current VLSI technology would benefit from locally connected, modular networks built from simple processors. Extended periods of autonomous operation might require designs that are efficient in terms of power consumption, etc.

Thus, the application domain and design constraints translate to a number of identifiable properties of genetic representations of neural architectures. Therefore, a systematic study of properties of genetic representations in the context of specific classes of applications will help identify the relative strengths and weaknesses of different EDNA systems. Exploration of the properties identified and defined in this paper, is a tentative step in this direction. Future work needs to further refine and extend this characterization to address the tradeoffs between various loci for plasticity, redundancy, modularity etc. Analyzing properties of genetic operators and characterizing their impact on the evolutionary design of neural architectures is another interesting research direction.

References

Balakrishnan, K., & Honavar, V. (1995a). *Analysis of Genetic Representations and Operators in the Evolutionary Design of Neural Architectures.* (in preparation).

Balakrishnan, K., & Honavar, V. (Januaryb 1995). *Evolutionary Design of Neural Architectures – A Preliminary Taxonomy and Guide to Literature.* Tech. rept. CS TR95-01. Department of Computer Science, Iowa State University, Ames.

Chen, C., Parekh, R., Yang, J., Balakrishnan, K., & Honavar, V. (1995). *Analysis of Decision Boundaries Generated by Constructive Neural Network Learning Algorithms.* (submitted for review to WCNN'95).

Collins, R., & Jefferson, D. (1990). An Artificial Neural Network Representation for Artificial Organisms. *Pages 259–263 of: Proceedings of the Conference on Parallel Problem Solving from Nature.*

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, Mass.: Addison-Wesley.

Gruau, Frederic. (1994). *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm.* Ph.D. thesis, Ecole Normale Supérieure de Lyon.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press.

Honavar, V., & Uhr, L. (1993). Generative Learning Structures for Generalized Connectionist Networks. *Information Sciences*, **70**(1-2), 75–108.

Koza, J. R. (1992). *Genetic Programming: On the programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, Mass.

Miller, G.F., Todd, P.M., & Hegde, S.U. (1989). Designing Neural Networks Using Genetic Algorithms. *Pages 379–384 of:* Schaffer, J.D. (ed), *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann.

von Neumann, J. (1956). Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. *Pages 43–98 of: Automata Studies.* Princeton University Press.