A New Metric for Evaluating Genetic Optimization of Neural Networks

Jaime Dávila
Hampshire College
School of Cognitive Science
Amherst MA 01002
jdavila@hampshire.edu

Abstract- In recent years researchers have used genetic algorithm techniques to evolve neural network topologies. Although these researchers have had the same end result in mind (namely, the evolution of topologies that are better able to solve a particular problem), the approaches they used varied greatly. Random selection of a genome coding scheme can easily result in sub-optimal genetic performance, since the efficiency of different evolutionary operations depends on how they affect schemata being processed in the population. In addition, the computational complexity involved in creating and evaluating neural networks usually does not allow for repetition of genetic experiments under different genome coding.

In this paper I present an evaluation method that uses schema theory to aid the design of genetic codings for NN topology optimization. Furthermore, this methodology can help determine optimal balances between different evolutionary operators depending on the characteristics of the coding scheme. The methodology is tested on two GA-NN hybrid systems: one for natural language processing, and another for robot navigation.

1 Introduction

In recent years researchers have used genetic algorithm techniques to evolve neural network topologies. Although these researchers have had the same end result in mind (namely, the evolution of topologies that are better able to solve a particular problem), the approaches they used varied greatly.

De Garis (1996), for example, evolved NN by having a series of "growth commands" give instructions on how to grow connections among nodes. Each node in the network processed signals that told it how to extend its synapses. When two different synapses reached each other, a new node was formed. The genetic algorithm was responsible for evolving the sequence of growth commands that controlled how the network developed.

Fullmer and Miikkulainen (1991) developed a GA coding system where pieces of a genotype went unused, imitating biological DNA processing. Only information stored between a Start marker and an End marker was used to generate networks. The amount of correctly configured Start-End markers defined how many hidden nodes the network would have. In addition, information between these Start-End markers defined how the nodes were connected to each other. The meaning conveyed by each position in the used part of the genome depended on its distance from its corresponding Start symbol.

Figure 1: example of coding used by Fullmer and Miikkulainen.

For example, the genome shown in figure 1 would generate two nodes, one for string **"S,a,1,b,5,a,-2,E"** and another for string "S,b,0,a,3,E", which wraps around the end of the genome. Node **"a"** had an initial activation of 1 (because of substring "S,a,1"), is connected to node **"b"** with a weight of 5 (because of substring "b, 5"), and to itself with a weight of -2 (because of substring **"a, -2"**). Node b had an initial activation of 0 (because of substring "S,b,0") and a connection to node a with a weight of 3 (because of substring "a,3"). The network evolved by this process was used to control a virtual creature's movements in a square field, avoiding "bad" objects and coming into contact with "good" objects. The GA continued to run until a network that could solve the problem evolved. The number of generations needed until this network was found varied between 7 and 304 for objects that could be identified before hitting them, and between 15 and 414 generations when recognizing the object required traveling around it looking for a characteristic view.

Kitano (1994) used GA to evolve a sequence of graph generation rules, as opposed to directly coding network topology. Each genome defined a sequence of rules used to rewrite an element of the graph. When these rules were applied until only terminal symbols remained, the graph defined a connectivity matrix which was then used to configure a NN. For example, if we were developing a network with two nodes, a genome might code rules [S → AB][A → 01][B → 10]. When these three rules are applied we end up with a 2*2 matrix than defines the connectivity between the two nodes in the network.

Given that the approaches presented above are a very quick glance at the options available, it is obvious that researchers have to choose a representation scheme from a vast number of possibilities. Random selection of a genome coding scheme can easily result in sub-optimal genetic performance, since the efficiency of different evolutionary operations depends on how they affect schemata being processed in the population. In addition, the computational complexity involved in creating and evaluating neural networks usually does not allow for repetition of genetic experiments under different genome coding.

This paper is an attempt at developing a metric that can be used to evaluate different evolutionary computation coding schemes.

2 Evolving NN parameters

A review of the NN-GA literature quickly reveals that researchers have chosen to evolve different aspects of NN configurations. Concentrating on topology, there are several factors one might choose to optimize, such as number of nodes to use, how many hidden layers to have, how many nodes to have in each hidden layer, or even if a layered approach should be used or not.

At the lowest level of abstraction, we can see topology optimization as a process that determines if a connection should exist between nodes x_A and x_B, where A and B range across all nodes that exist in a network. If having a connection between x_A and x_B is useful for the task being solved, schema theory suggests that individuals in the population will benefit from having such a connection present, all other things being equal (Holland [1975]). For the same reason, any evolutionary operator that disrupts the existence of useful traits could decrease offspring fitness. If we assume that the typical evolutionary operations of crossover and mutation are to be applied, we can analyze genome coding in light of how it affects the way in which evolution progresses.

3 Low-level topology definition

If we view evolutionary computations as a process to define connections between any two nodes as their main result, then we can determine their ability to combine building blocks by estimating how likely it is for evolutionary operations to disrupt connection definitions; the less likely it is for connection definitions to be disturbed, the easier it is for the algorithm to combine building blocks present in the current population.

An operation like crossover can disrupt a connection definition every time a crossover point is selected between two genes that, taken together, define a connection between nodes of a network. Therefore, how likely it is for crossover to cause this disruption can be estimated by the distance between genes that combine to define any particular connection. If a particular connection is defined by alleles in genes g_i and g_j, then the bigger the distance between g_i and g_j, the bigger the chance that the connection will be disrupted by a crossover operation. Taking a sum of the distance between genes that can define a connection, and averaging over the number of connections, we obtain a total disruption index (TDI) of

$$\frac{\sum_{k=0}^{C}\sum_{i=0}^{N}\sum_{j=0}^{N} |i-j| * DC(k,i) * DC(k,j))}{C} \quad (1)$$

where C is the number of connections, N is the number of genes, P(x) returns the position of gene x in the chromosome, and DC(k, x) equals to a number between 0 and 1 which indicates what is the probability that gene x is involved in defining connection k.

4 Case Studies

The following section reports empirical testing of the mathematical formulation presented above for two GA-NN hybrid systems: one for natural language processing, and another for robot navigation.

4.1 Case Study #1: A Natural Language Processing System

This example is based on research originally presented in Dávila [1999a, 1999b, 1999c], which was used to evolve NN topologies for natural language processing. As an overview, a network is asked to receive a sentence one word at a time, and to incrementally build a description of the sentence in its output nodes. For example, if the sentence “the boy ran in the park” is entered, the network should respond by indicating that “the boy” is a noun phrase, and it acts as the agent of verb “ran”. The network should also indicate that “in the park” is a prepositional phrase modifying the verb “ran”.

Entering a word into the network amounts to activating a single node that represents the given word at the input layer, and at the same time activating those semantic nodes that reflect the meaning of the word being entered. For example, to enter the word “john”, a node that represents that word is activated, as well as nodes that indicate that the word being entered is a proper noun, singular, concrete, and human. In addition, an ID node is set to a value that would allow the network to distinguish “john” from other words that might have the same semantic identity, such as “mary”.

The language used in this research is composed of ten nouns: boy, girl, john, mary, horse, duck, car, boat, park,

river. Available semantic nodes are: human, animal, or mechanical (three mutually exclusive nodes); animate or inanimate (represented by one node, active if the noun is animate); proper (active if true, inactive otherwise); and one ID node.

In the original formulation for this problem, which I will call SYSTEM-A for the remainder of this paper, each network has 75 hidden nodes between the input and output layers. These 75 nodes are divided into N hidden layers, where N is a number between 1 and 30. The exact number of hidden layers is determined by the first gene of the corresponding genome. This position stores a random floating point number, with a value between 0 and 1. To determine how many hidden layers a network has, the value of this gene is multiplied by 30, and rounded to the next highest integer. If the result of this rounding up is 31, the network uses 30 hidden layers.

The number of hidden nodes in each of these hidden layers is also determined by the network's corresponding genome. The genome has 30 genes used to code the "relative worth" of each of the possible hidden layers. Once the number of hidden layers is determined to be N using the process described above, the N layers with the highest relative worth are identified. The 75 available hidden nodes are distributed among each of these N hidden layers according to each layer's worth relative to the sum of all N worth values.

The connections between layers are also determined by the network's genome. For each of the thirty possible layers, there is a gene that indicates where the layer takes its input from. Each of these genes stores a random floating point value between 0 and 1. To determine where each hidden layer takes its input from, its "takes-its-input from" gene value is multiplied by N+2 (where N is the number of hidden layers this network will have, as determined by the procedure outlined previously), and rounded to the nearest integer. The resulting number points to which layer this one takes its input from. We multiply by N+2 to allow a hidden layer to take its input from any of the N hidden layers, as well as either the input or the output layer. A value of 1 would mean the layer takes its input from the input layer. A value of N+2 would mean the layer takes its input form the output layer. For values between 2 and N+1, the layer would take its input from the layer with the $(N-1)^{th}$ highest relative worth.

Where each layer sends its output is determined in a similar way, using positions 62-91 of the genotype. Each of these genes stores a random floating point value between 0 and 1. To determine where each layer sends its output, its "sends-output-to" gene value is multiplied by N+1 and rounded to the nearest integer. The resulting number points to which other layer this one will send its output. We multiply by N+1 to allow for hidden layers sending its output to any of the N hidden layers, as well as to the output layer. A value of N+1 would mean the layer sends its output to the output layer. For values between 1 and N, the layer sends its output to the layer with the N^{th} highest relative worth. No layer sends its output back to the input layer.

Under SYSTEM-A, the existence of a connection k between nodes i & j depends on the number of layers that the hidden nodes are divided into (gene 30), which layers contain nodes i & j (genes 0-29), where the layer with node j takes its input from (a gene from among genes 31-60, depending on the values of j, gene 30, and the nodes/layer distribution determined by genes 0-29), and where the layer with node i sends its output to (a gene from among genes 61-90, again depending on the values of i, gene 30, and the nodes/layer distribution determined by genes 0-29).

As a comparison, the same type of network topology could be evolved by using the following coding scheme, which I will call SYSTEM-B. Each network still has 75 hidden nodes, but they are always divided into 30 hidden layers (that is, there is no gene used to determine how many hidden layers to use). The distribution of these 75 nodes into 30 layers is done based on 30 "relative worth" values in the genome. To determine how many node each layer has, the relative worth for a specific layer is divided by the sum of relative worth of all 30 layers, and then multiplied by 75.

The connections between layers are also determined by the network's genome. For each of the thirty layers, there is a gene that indicates where the layer takes its input from. Each of these genes stores a random floating point value between 0 and 1. To determine where each hidden layer takes its input from, its "takes-its-input from" gene value is multiplied by 32 and rounded to the nearest integer. The resulting number points to which layer this one takes its input from. A value of 1 would mean the layer takes its input from the input layer. A value of 32 would mean the layer takes its input form the output layer. For values between 2 and N+1, the layer would take its input from the $(N-1)^{th}$ hidden layer.

If we assume that DC(k,x) = 1 for all values of k and x, TDI(SYSTEM-A) = TDI(SYSTEM-B). In reality, though, DC(k,x) does not always return 1, and in fact it tends to return smaller values under SYSTEM-B than under SYSTEM-A. Notice, for example, that under SYSTEM-A the set of gene sequences that would allow node n to be in layer L has a higher cardinality than under SYSTEM-B, given that SYSTEM-B always has 30 hidden layers, while under SYSTEM-A the number of hidden layers is determined by a gene. This will, in turn, affect which genes are involved in defining a connection between two given nodes, which affects DC(k,x). Under standard random distributions, considering actual values for DC(k,x) would give TDI(SYSTEM-A) > TDI(SYSTEM-B). What this would mean is that it is easier for a particularly good schema to be disrupted by crossover under SYSTEM-A than under SYSTEM-B.

Aside from the effect of DC(k,x), of course, the actual positioning of the genes and how they map into phenome characteristics also has an effect on the disruption caused by crossover operations. Take, for example, a system with the same types of genes as SYSTEM-B, but where the position of the genes has been altered. Instead of having 30 worth values followed by 30 takes-input-from values and then 30 sends-output-to values, SYSTEM-C arranges genes so that

the worth, takes-input-from and sends-output-to genes for any one particular layer are in three consecutive positions (positions 3*L, 3*L+1, and 3*L+2, where L is the hidden layer number). Because a gene with a particular functionality will have the same effect on resulting phenomes regardless of its position in the genome, we can discard terms DC(k,i) and DC(k,j) in equation (1), and obtain

$$TDI'(\cdot) = \frac{\sum_{k=0}^{C}\sum_{i=0}^{N}\sum_{j=0}^{N}\left(\left|P(g_i) - P(g_j)\right|\right)}{C} \quad (2)$$

Under this definition, TDI'(SYSTEM-B) > TDI'(SYSTEM-C), since SYSTEM-C minimizes the distance between the relative-worth and takes-input-from genes . This means that crossover is less likely to disrupt a useful schema under SYSTEM-C.

To verify the effect these disruptions might have on solutions found by evolutionary computation, I performed the optimization of topologies for the natural language problem outlined previously under the three genome coding schemes discussed above. In order to better measure the effect of crossover, computations were performed with no mutation, and populations of 21 elements. Network fitness values were computed by taking the sum of square errors for all output nodes throughout the presentation of a language of 419 sentences. Training is performed on 20% of this same language. In order to verify consistent performance, all evolved networks were validated by performing 144 bootstrap validation runs per network (Weiss, Kulikowski 1991). Evolutionary runs were repeated 48 times, and the graphs presented here, although taken from particular runs, are typical of results throughout all runs.

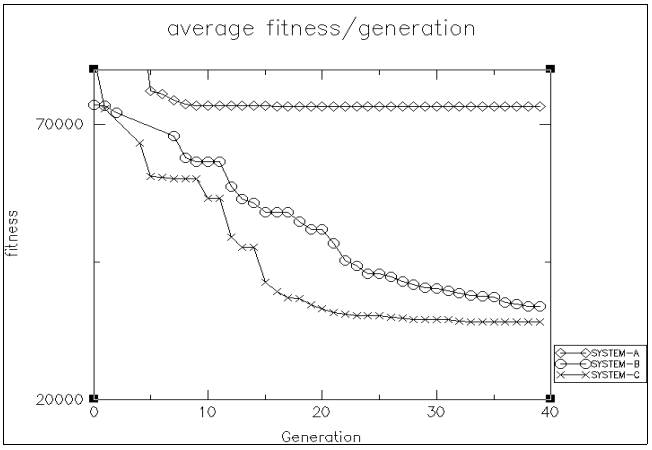

Figure 2: Average fitness for topology optimization runs under different genome coding systems.

Figure 2 shows the results through 40 generations for the three methods. Coding schemes SYSTEM-B and SYSTEM-C prove to be much better than SYSTEM-A. Since all systems were seeded with the same random number generator, this is probably caused by a high disruption index for SYSTEM-A, which does not allow the evolutionary algorithm to build upon good building blocks present in the original population. Both SYSTEM-B and SYSTEM-C reach comparable average fitness values after 40 generations, but SYSTEM-C reaches that value in little over 50% the number of generations required by SYSTEM-B. This is caused by an even lower TDI for SYSTEM-C, which allows the evolutionary computation to make the best use of building blocks present in the original population.

Figure 2 can also be used as an example of what the TDI computation reveals about the relationship between crossover and mutation for different coding schemes. The computation carried out for SYSTEM-C has a lower chance of building block disruption by the crossover operation. This means that evolution will depend heavily on operations other than crossover in order to introduce new phenome characteristics into the population. In the absence of any such other operation, crossover will continue to exploit good solutions, but will perform little exploration. This can be seen in figure 2 by the fact that SYSTEM-C reaches the vicinity of the final average value fairly quickly, but then seems to fail to find anything significantly better. The need for evolutionary exploration in a system like SYSTEM-C can be provided by operations such as mutation. To corroborate this, the experiments described above were repeated using a mutation rate of 1%. Results for those runs are presented in figure 3.

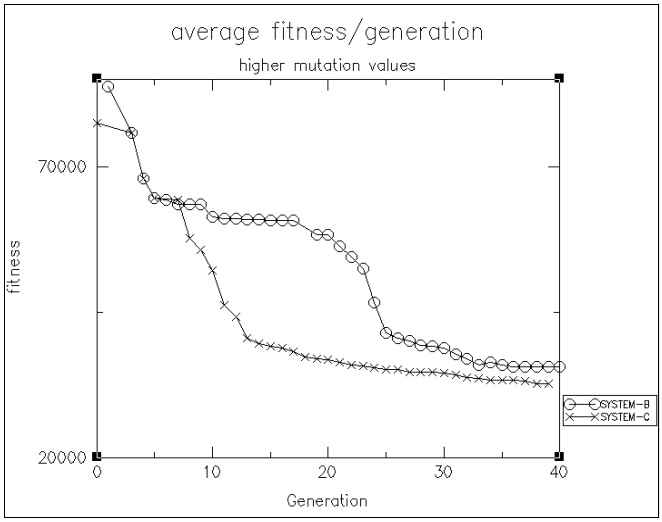

Figure 3: average fitness values for runs with 1% mutation.

As can be seen above, SYSTEM-C still reaches better values for average fitness faster than SYSTEM-B, but now has enough mutation to allow it to explore options not present in the original population, and improve on previous results towards the end of the genetic run. Notice also that the plateau experienced around generations 5-10 when running without mutation is now avoided.

4.2 Case study #2: A Robot Navigation System

To further analyze the effect of schema disruptions on the performance of evolutionary design of NN, I have developed a NN evolution system where the number of genes participating in the definition of a single connection can be configured to a number between 1 and 1000. Details of the task being solved can be found in (Davila, 2000). Briefly, a robot exists inside a cubic world with dimensions 10*10*10. A single cube appears at the top level of the world each second. Each of these boxes has a random floating point worth value between 0 and 1 All cubes fall one vertical unit per second. The robot located at the bottom of the cube can move in any of four directions (left, right, forward, or backwards) a single step each second. Where the robot moves is determined by a single output of a NN controlling it. The distance between the robot and any cube at ground level is computed, and the worth of the cube is divided by said distance, creating a closeness measure . For each second the simulation runs, this closeness measure is added to the robot s fitness. This process iterates for 500 seconds. The fitness of a NN is 500 minus the score attained by the robot it controls. A screen capture of the running process is shown in figure 4. Brighter falling cubes have a higher worth value.

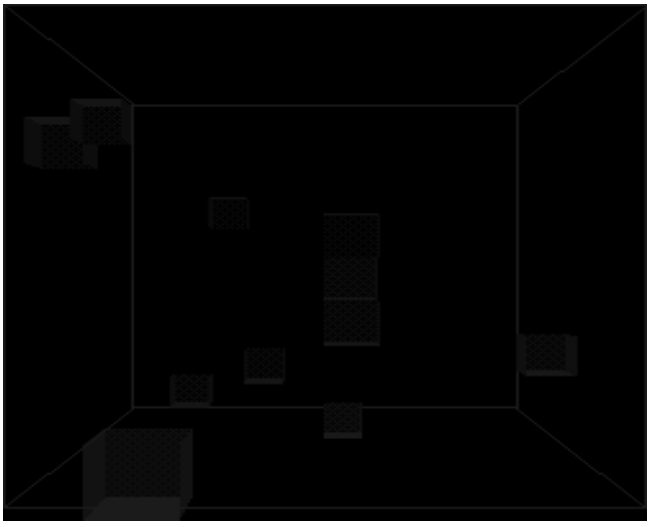

Figure 4: Cubic World screen capture.

A GA controls which set of sensor inputs is received by the NN. In the simplest case, a positive value on the nth gene means the nth input sensor is connected to the NN. The connection weight between this input and the hidden layer are determined by genes (1000 + (n-1)*H + 1) to (1000 + n*H + 1), where H is the number of hidden nodes to be used. This value for H is determined by calculating the maximum amount of hidden nodes that can be used while not ending with more that a predetermined number of total connections. This method for determining should have low values of TDI. Given that the existence of a connection is determined by only one gene, schema disruptions should be low.

The same evolutionary process can be repeated with the same number of total genes while incorporating different number of genes into the definition of each connection. The existence of input connection C is determined by looking at P participants, where P is chosen before the evolutionary run begins. The sum of the alleles in genes {C%SIZE, (C+I)%SIZE, (C+2I)%SIZE, ...(C+PI)%SIZE} is computed, where SIZE is the number of possible input connections and I is equal to SIZE divided by P. This produces a number between 0 and P. Dividing this number by P we obtain a number between 0 and 1. This number is then multiplied by 2^P, producing a number between 0 and 2^P,. The existence of connection C is determined by looking at the Bt^h bit of the binary representation of this last number, where B = C/I.

Given this coding scheme, the higher the number of participants, the higher the chances that evolution will be disrupted by crossover destruction of schemata. Typical average fitness values for two different runs are presented in figure 5.

Figure 5: average fitness for robots when using mutation rate of 5%.

Of particular interest is the fact that, although both runs fall into plateaus, the ones for the runs with 25 participants are fewer and smaller. This is caused by a higher TDI value introducing more genetic diversity to the population. This can be further understood when evolving topologies with 25 participants under smaller mutation rates. Results of such a run are shown in figure 6. Notice that the average fitness after 50 generations, under lower mutation rates, is very close to that for only 1 participant and a higher mutation rate.

Figure 6:average fitness for runs with 25 participants.

5 On the issue of Topology definition Methods

The methodology presented here is obviously based on a system's ability to define particular connections between 2 nodes. This should not be understood to mean that the individual connection level is the preferred representation scheme for all genome coding systems. For example, the TDI of a system that maps each gene to a definition of a particular connections of a network is TDI(.) = 0, but a genome for the language processing problem presented in this paper would need to have more than 17000 positions. The low TDI value obtained would be offset by the size of the problem space, and the impossibility of searching through any significant part of it. In addition, if a NN designer has determined that the best way to approach a problem is to have a system that stores internal states, then using a layered network is required, and a coding scheme that works at that level might be preferred. Even in those cases, where the coding does not work with individual connections, analyzing the scheme results useful. When crossover disrupts the connection(s) between two different layers, the magnitude of the change this causes increases with the number of connections between the layers involved.

In addition, there are cases where a researcher might choose to use a coding scheme with higher TDI values. For example, coding scheme SYSTEM-A has a higher TDI value that the other two systems presented here, even if we where to take DC(k,x) to return the same value $\forall$ k,x. At the same time, SYSTEM-A is the only coding scheme of the three that directly codes for the number for hidden layers to be used. If this is a desired property for a coding scheme, then a system with a higher TDI value might be called for. In general, a NN designer might choose to use a system with a high TDI value because of other foreseen advantages. TDI values give a good indication, though, of how likely it is for building blocks to be disrupted, so that an educated choice can be made.

7 Future Research

One of the most important aspects of the formulation presented here is that of the function DC(k,x). For genome coding schemes that would be close to identical otherwise (such as SYSTEM-A and SYSTEM-B), DC(.) can produce dramatic differences, as seen in figure 1. A complete mathematical description of this function, which is particular for each coding scheme, would allow to make better predictions regarding the feasibility of different coding schemes, and provide guidance regarding the balance between crossover and mutation.

In addition, extended empirical verification of the model will allow to better understand the effect of using different operations during evolution for different representational models. For example, SYSTEM-C might benefit from even higher mutation values than used here, and this property should be related to actual values for TDI. This, combined with a better understanding of function DC(.), will allow for better comparisons between TDI values and disruptions observed in empirical tests.

8 Conclusion

This paper has presented a new way of evaluating coding schemes for genetic algorithms used to optimize neural network topologies. The methodology is based on the likelihood that crossover operations can effectively construct on useful building blocks present in initial populations. By doing so it estimates how easy it is for an evolutionary model to find good solutions to the problem at hand, while at the same time providing information regarding how to best balance exploration with exploitation in a genetic system. This methodology results useful in designing a system that can effectively perform hard natural language processing tasks, and can in theory be extended to any GA/NN system.

Acknowledgements

I would like to thank the School of Cognitive Science of Hampshire College and the Department of Computer Science at City College of New York for the networks of Sun Ultra-stations where the empirical results presented here were obtained.

References

de Garis, H., (1996) CAM-BRAIN: The Evolutionary Engineering of a Billion Neuron Artificial Brain by 2001 Which Grows/Evolves at Electronic Speeds Inside a Cellular Automata Machine (CAM), *Lecture Notes in Computer Science – Towards Evolvable Hardware, Vol. 1062,* Springer Verlag, pp. 76-98.

Dávila, J., (1999a) Exploring the Relationship Between Neural Network Topology and Optimal Training Set by Means of Genetic Algorithms. *International Conference on Neural Networks and Genetic Algorithms*, Springer-Verlag, pp. 307-311.

Dávila, J. (1999b) Genetic Optimization of NN Topologies for the Task of Natural Language Processing. *International Joint Conference on Neural Networks*, Washington, D.C., 1999.

Dávila, J. (1999c) Genetic Optimization of Neural Network Configurations for Natural Language Learning. Doctoral Dissertation, Department of Computer Science, Graduate School and University Center, City University of New York.

Davila, J. (2000) Genetic Evolution of Sensors and Topology for a Neurally Controlled Robot. To appear in the *Proccedings of the GECCO-2000 Workshops*, 2000

Fullmer, B., Miikkulainen, R., (1991) Using marker-based genetic encoding of neural networks to evolve finite-state behavior. *Proceedings of the first European Conference on Artificial Life.* Paris, pp. 253-262.

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

Kitano, H., (1994) Designing Neural Networks using Genetic Algorithm with Graph Generation System, *Complex Systems, 4*, pp. 461-476.

Romaniuk, S. (1994) Applying crossover operators to automatic neural network construction, *Proceedings of the First IEEE Conference on Evolutionary Computation*, IEEE, New York, NY. pp. 750-752.

Weiss, S., Kulikowski, C. (1991) *Computer Systems that Learn. Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufmann. San Mateo, California.