GENETIC PROGRAMMING

Building Artificial Nervous Systems with Genetically Programmed Neural Network Modules

Hugo de Garis

CADEPS Artificial Intelligence and Artificial Life Research Unit,
Universite Libre de Bruxelles, Ave F.D. Roosevelt 50,
C.P. 194/7, B-1050, Brussels, Belgium, Europe.
tel + 32 2 650 2779, fax + 32 2 650 2715,
email CADEPS@BBRNSF11.BITNET

&

Center for Artificial Intelligence, George Mason University,
4400 University Drive, Fairfax, Virginia, VA 22030, USA.
tel + 1 703 764 6328, fax + 1 703 323 2630
email HUGODEG@AIC.GMU.EDU

Contents

1. Introduction
2. Neural Networks
3. The Genetic Algorithm
4. GenNets
5. Genetic Programming (GP)
6. Example 1 : "Pointer"
7. Example 2 : "Walker"
8. Example 3 : "LIZZY"
9. Discussion & Future Ideas
10. References

1. Introduction

One of the author's interests is studying future computing technologies and then considering the conceptual problems they will pose. The major concern in this regard is called the "Complexity Problem", i.e. how will computer scientists over the next twenty years or so cope with technologies which will provide them with the ability to build machines containing roughly the same number of artificial neurons as there are in the human brain. In case this seems a bit far fetched, consider four of these future technologies, namely Wafer Scale Integration (WSI), Molecular Electronics, Nanotechnology, and Quantum Computing. WSI is already with us. It is the process of using the full surface of a silicon crystal to contain one huge VLSI circuit. It is estimated by the mid 1990s[1] that it will be possible to place several million artificial neurons on a WSI circuit. Molecular Electronics[2] is the attempt to use molecules as computational devices, thus increasing computing speeds and allowing machines to be built with an Avogadro number of components. Nanotechnology is even more ambitious[3,4] , aiming at nothing less than mechanical chemistry, i.e. building nanoscopic assemblers capable of picking up an atom here and putting it there. We already know that a nanotechnology is possible, because we have the existence proof of biochemistry. Nanoscopic assemblers

could build any substance, including copies of themselves. Quantum computing promises to use subatomic scale (quantum) phenomena in order to compute[5], thus allowing the construction of machines with the order of 10 to the power 30 components.

On the assumption that most of these technologies will be well developed within a human generation, how are future computer scientists to cope with the truly gargantuan complexity of systems containing the above mentioned 10 to the power 30 components? This problem is not new. The biological world is confronted with this massive design problem every day. The solution taken by nature of course is Darwinian evolution, using techniques and strategies such as genetics, sex, death, reproduction, mutation etc. The traditional approach to computer architecture, i.e. explicit pre-planned design of machines, will probably become increasingly impossible. It is likely that machine designers of the future will probably be forced to take an evolutionary approach as well. Components will be treated more and more as black boxes, whose internal workings are considered to be too complex to be understood and analyzed. Instead, all that will matter will be the performances of the black boxes. By coding the structures of components in a linear "chromosome" like way, successful structures (i.e. those which perform well on some behavioural test) may see their corresponding chromosomes survive into the next generation with a higher probability. These linear structure codes (e.g. DNA in the biological world), can then be mutated by making random changes to them. If by chance an improvement in performance results, the mutated chromosome will gradually squeeze out its rivals in the population. Over many generations, the average performance level of the population will increase.

This chapter is based on the above evolutionary philosophy. It is concerned with the evolution of neural network modules which control some process e.g. stick legs which are taught to walk, or an artificial creature with a repertoire of behaviours. Before launching into a description as to how this is done, readers who are rather unfamiliar with either the Genetic Algorithm (GA)[6] or Neural Networks[7], are given a brief overview of both these topics. This is followed by a description as to how the GA can be used to evolve Neural Network modules (called GenNets) which have a desired behaviour. By putting GenNets together into more complex structures, one can build hierarchical control systems or artificial nervous systems etc.

Three examples of these ideas are presented in relative detail, based on some of the authors papers. The final sections provide ideas as to how Genetic Programming can play an important part in the new field of Artificial Life[8] by showing how artificial nervous systems can be constructed and how these ideas might be put directly into silicon and into robots.

2. Neural Networks

For the last 20 years, the dominant paradigm in Artificial Intelligence has been Symbolism, i.e. using symbols to represent concepts. This approach has only been moderately successful, which probably explains why so many AI researchers abandoned the Symbolist camp for the Neuronist camp (alias "Parallel Distributed Processing", "Connectionism" or "Neural Networks"), after the Neuronist revival in the 80s. This was certainly true for the author, who always had the feeling when interpreting the symbols he was using that somehow he was cheating. He and not the machine was giving the symbols their interpretation. At the back of his mind was always the impression that what he was doing was not "real" AI. Real AI to him was trying to model neural systems, and secretly he was reading neurophysiology, hoping to get into brain simulation in one form or another. After the first International Neural Networks Conference[9] was held, and I got hold of the proceedings, there was no turning back. Similar experiences have probably occured to many people because suddenly there were thousands of neuronists and six monthly neural network world conferences.

In the light of the Complexity Problem introduced in section 1, one soon realizes that one does not have to put many artificial neurons together to create a complicated whole, whose dynamics and analysis quickly transcend the scope of present day techniques. It seems probable that the intelligence shown by biological creatures such as mammals, is due to the enormous complexity of their brains, i.e. literally billions of neurons. If seems also not unreasonable that to produce machines with comparable intelligence levels, we may have to give them an equivalent number of artificial neurons, and to connect them together in ways similar to nature. But how are we ever to know how to do that, when we are talking about a trillion neurons in the human brain, with an astronomical number of possible connections?

Considering these weighty questions led the author to the idea that a marriage of artificial neural networks with artificial evolution would be a good thing - the basic idea being that one might be able to design functional neural networks without fully understanding their dynamics and connections. This is after all, the way nature has designed its brains. The author's view was that an artificially evolved neural net (which came to be called a GenNet) should be considered a black box. What was important was not HOW it functioned (although it is always nice to have a analytical understanding of the dynamics) but that it functioned at all. The art of designing GenNets and putting them together (an undertaking called Genetic Programming) was thus to be highly empirical, a "lets try it and see" method. To insist that one should have a theoretical justification for ones approach before proceeding, seemed to be unnecessarily restrictive, and would only hinder the rate of progress in designing artificial neural nets and artificial nervous systems.

There are other pragmatic arguments to justify this empirical approach. The author's belief is that technological progress with conventional computer design in terms of speed and memory capacities will allow neural net simulations of a complexity well beyond what the mathematicians are capable of analysing. One can expect the simulations to hit upon new qualitative phenomena which the analysts can subsequently attempt to understand. This chapter is essentially such an investigation. Several such qualitative phenomena have been discovered, as will be explained in later sections.

The above paragraphs have been more a justification for, rather than an explanation of, Genetic Programming. Since GP is a new approach, simply explaining what GP is without attempting to justify why it is, would seem to provide an incomplete picture of what GP is all about. Further justification of the approach, are the results already obtained and which will be discussed below. This section, on neural networks, makes no pretence at being complete. There are plenty of good text books now on the basic principles of neural nets[7]. However enough needs to be explained in order for the reader to understand how the Genetic Algorithm is applied to the evolution of neural nets. This minimum now follows.

Artificial neural networks are constructed by connecting artificial neurons together, whose function consists essentially of accepting incoming signals, weighting them appropriately and then emitting an output signal to other neurons, depending on the weighted sum of the incoming signals. FIG.1 shows the basic idea.

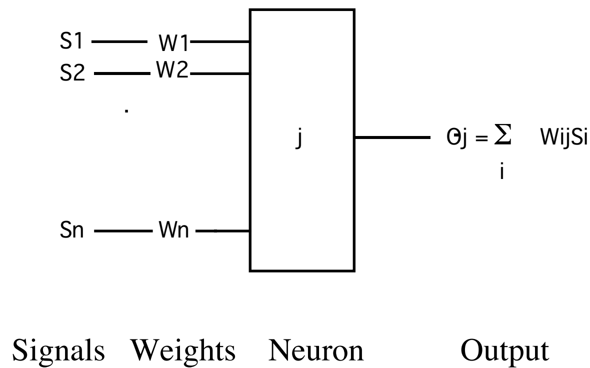
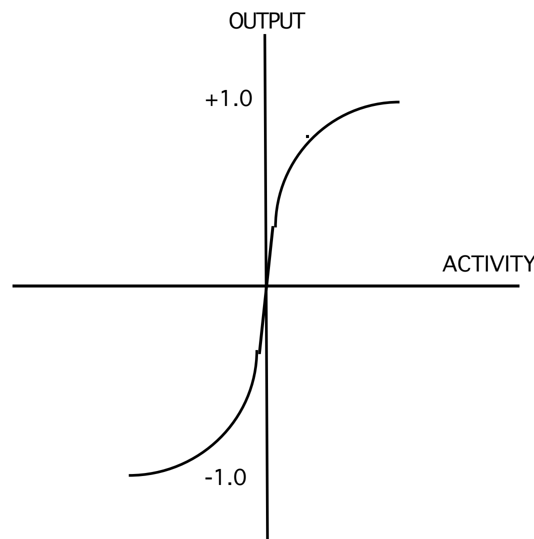


FIG.1 AN ARTIFICIAL NEURON

The incoming signals have strengths Si. Each incoming signal Si is weighted by a factor Wi, so that the total incoming signal strength is the dot product of the signal vector {S1, S2, S3, ...} and the weight vector {W1, W2, W3, ...}. This dot product is then usually fed through a non linear sigmoid function, shown in FIG 2.



OUTPUT = -1 + (2/(1 + exp(-ACTIVITY)))

FIG. 2 THE NEURAL OUTPUT FUNCTION

There are binary versions of the above ideas, where the input signals are binary, the weights are reals and the output is binary, depending on the sign of the dot product. For further details, see any of the neuronist textbooks[7]. However, for GP, the incoming signals are assumed to be reals, as are the weights and output signals. The sigmoid function is shown in FIG.2

Now that the functioning of a single neuron has been discussed, we need to consider how a group of neurons are usually connected together to form networks. After all, the very title of this section is Neural Networks. Broadly speaking there are two major categories of networks used by neuronists, known as "layered feed forward" and "fully connected". Both types have been used in the author's GP experiments. FIG. 3 shows a layered feed forward network. The use of the word "layered" is obvious. The most usual number of layers is 3, namely an input layer, a hidden layer and an output layer, as shown in the figure. The word feedforward is used because no output signals

are fed back to previous layers. For given weights, a given input vector applied to the input layer will result in a definite output vector. Learning algorithms exist which can modify the weights allowing a desired output vector for a given input vector. The best known such algorithm is called the "backpropagation" or "backprop" algorithm[7].

The other common network type is called "fully connected" as shown also in FIG. 3. The dynamics of fully connected networks are a lot more complicated, which explains why many neuronist theorists have prefered to work with feedforward networks. As we shall see, the level of complexity of the dynamics of the network is usually irrelevant to the Genetic Algorithm, which responds only to the quality of the results of the dynamics and not to how the dynamics are produced. A fully connected network obviously includes feedback and the outputs depend not only on the initial inputs but on the history of the internal signal strengths. This greater complexity may be a disadvantage in terms of its analysability, but its greater number of degrees of freedom may actually be useful for the functioning of GenNets, e.g. a greater robustness, or giving the GA a larger search space to play with.

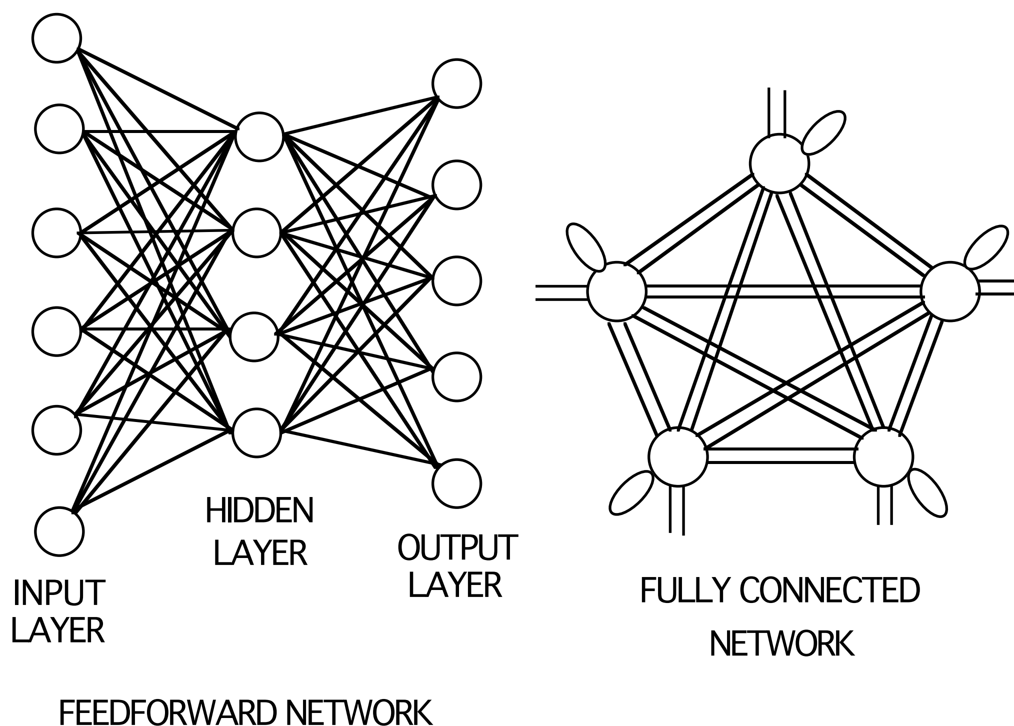


FIG. 3 NEURAL NETWORK ARCHITECTURES

Imagine as little as a dozen such artificial neurons all connected together, where the outputs of some of the neurons are used as parameter values in certain calculations. If the results of these calculations are fed back as external inputs to some of the neurons, and the process is repeated for hundreds of cycles or more, one has an extremely complicated dynamical system. However, a computer simulation of such a system is quite within the state of the art, even on relatively modest work stations.

In order to be able to evolve the behaviour of such a system described in the above paragraph, one needs to understand a little about the Genetic Algorithm, which is the technique used to control the artificial evolution of this behaviour. Hence a brief description of the Genetic Algorithm follows next.

3. The Genetic Algorithm

The Genetic Algorithm (GA), as has already been mentioned, is a form of artificial evolution. But evolution of what? There are a few key concepts which need to be introduced to be able to understand how the GA can be employed to evolve neural network behaviours. Probably the most important idea is that of a linear mapping of the structure of the system one is dealing with, into a string of symbols, usually a binary string. For example, imagine you were trying to find the numerical values of 100 parameters used to specify some complicated nonlinear control process. You could convert these values into binary form and concatenate the results into a long binary string. Alternatively, you could do the reverse and take an "arbitrary" binary string and extract (or map) the 100 numerical values from it.

Imagine forming a population of 50 of these binary strings, simply by using a random number generator to decide the binary value at each bit position. Each of these bit strings (of appropriate length to agree with the numerical representation chosen) will specify the values of the 100 parameters. Imagine now that you have a means of measuring the quality or performance level (or "fitness", in Darwinian terms) of your (control) process. The next step in the discussion is to actually measure the performance levels of each process as specified by its corresponding bit string. These bit strings are called "chromosomes" by GA specialists, for reasons which will soon become apparent. Let us now reproduce these chromosomes in proportion to their fitnesses, i.e. chromosomes with high scores relative to the other members of the population will reproduce themselves more.

If we now assume that the initial population of chromosomes is to be replaced by their offspring, and that the population size is fixed, then we will inevitably have a competition for survival of the chromosomes amongst themselves into the next generation. Poor fitness chromosomes may not even survive. We have a Darwinian "Survival of the Fittest" situation. To increase the closeness of the analogy between biological chromosomes and our binary chromosomes, let us introduce what the GA specialists call "genetic operators" such as mutation, crossover etc. and apply them with appropriate probabilities to the offspring.

Mutation in this case, is simply flipping a bit, with a given (usually very small) probability. Crossover is (usually) taking two chromosomes, cutting them both at (the same) two positions along the length of the bit string, and then exchanging the two portions thus cut out, as shown in FIG. 4

0101010101010 10101010101 010101010101

1111000011110 00011110000 111100001111

0101010101010 00011110000 010101010101

1111000011110 10101010101 111100001111

FIG.4 CROSSOVER OF CHROMOSOMAL PAIRS

If by chance, the application of one or more of these genetic operators causes the fitness of the resulting chromosome to be slightly superior to that of the other members of the population, it will have more offspring in the following generation. Over many generations, the average fitness of the population increases. This increase in fitness is due to favourable mutations and the possible combination of more than one favourable mutation into a single offspring due to the effect of crossover. Crossover is synonymous with sex. Biological evolution hit on the efficacy of sex very early in its history. With crossover, two separate favourable mutations, one in each of two parents, can be combined in the same chromosome of the offspring, thus accelerating significantly the adaptation rate (which is vital for species survival when environments are rapidly changing).

In order to understand some of the details mentioned in later sections, some further GA concepts need to be introduced. One concerns the techniques used to select the number of offspring for each member of the current generation. One of the simplest techniques is called "roulette wheel", which slices up a pie (or roulette wheel) with as many sectors as members of the population (assumed fixed over the whole evolution). The angle subtended by the sector (slice of pie) is proportional to the fitness value. High fitness chromosomes will have slices of pie with large sector angles. To select the next generation (before the genetic operators are applied), one simply "spins" the roulette wheel a number of times equal to the number of chromosomes in the population. If the roulette "ball" lands in a given sector (as simulated with a random number generator), the corresponding chromosome is selected. Since the probability of landing in a given sector is proportional to its angle (and hence fitness), the fitter chromosomes are more likely to survive into the next generation, hence the Darwinian "Survival of the Fittest", and hence the very title of the algorithm, the Genetic Algorithm.

Another concept is called scaling, which is used to linearly transform the fitness scores before selection of the next generation. This is often done for two reasons. One is to avoid premature convergence. Imagine that one chromosome had a fitness value well above the others in the early generations. It would quickly squeeze out the other members of the population if the fitness scores were not transformed. So choose a scaling such that the largest (scaled) fitness is some small factor (e.g. 2) times the average (scaled) fitness. Similarly, if there were no scaling, towards the end of the evolution the average (unscaled) fitness value often approaches the largest (unscaled) fitness, because the GA is hitting up against the natural (local) "optimum behaviour" of the system being evolved. If the fitness scores differ by too little, no real selection pressure remains, and the search for superior chromosomes becomes increasingly random. To continue the selective pressure, the increasingly small differences between the unscaled average fitness and the largest fitness are magnified by using the same linear scaling employed to prevent premature convergence. This technique is used when evolving GenNets.

There are many forms of crossover in the GA literature[6]. One fairly simple form is as follows. Randomly pair off the chromosomes selected for the next generation and apply crossover to each pair with a user specified probability (i.e. the "crossprob" value, usually about 0.6). If (so-called "two point") crossover is to be applied, each chromosome is cut at two random positions (as in FIG. 4) and the inner portions of the two chromosomes swapped.

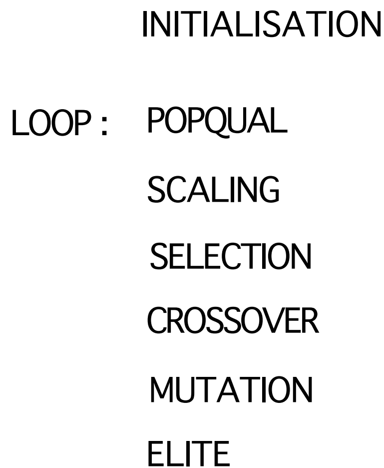


FIG. 5 TYPICAL CYCLE OF THE GENETIC ALGORITHM

A fairly typical cycle of the GA algorithm is shown in FIG. 5, where "Initialisation" usually means the creation of random binary strings. "Popqual" translates the binary strings into the structures whose performances are to be measured and measures the fitnesses of those performances. Often the chromosome with the highest fitness is saved at this point, to be inserted directly into the next generation once the genetic operators have been applied. "Scaling" linearly scales the fitness values before selection occurs. "Selection" is using a technique such as "roulette wheel" to select the next generation from the present generation of chromosomes. "Crossover" swaps over portions of chromosome pairs. "Mutation" flips the bits of each chromosome with a low probability (typically 0.001 or so) per bit. "Elite" simply inserts the best chromosome met so far (as stored in the "Popqual" phase) directly into the next generation. This can be useful, because the stochastic nature of the GA can sometimes eliminate high fitness chromosomes. The "elitist" strategy prevents this.

The above sketch of the GA is far from complete. However it contains the essence of the subject. For a more detailed analysis, particularly a mathematical justification as to why crossover works so well, see the several texts available[6], especially[10]. The GA is a flourishing research domain in its own right, and is as much a mix of empirical exploration and mathematical analysis as are neural networks.

Now that the essential principles of both neural networks and the GA have been presented, a description as to how the GA can be used to evolve behaviours in neural nets, can now follow.

4. GenNets

A GenNet (i.e. a Genetically Programmed Neural Net) is a neural net which has been evolved by the GA to perform some function or behaviour, where the fitness is the quality measure of the behaviour. For example, imagine we are trying to evolve a GenNet which controls the angles of a pair of stick legs over time, such that the stick legs move to the right across a computer screen. The fitness might simply be the distance covered over a given number of cycles. Most of the work the author has done so far has been involved with fully connected GenNets. Remember the GA is indifferent to the complexity of the dynamics of the GenNet, and this complexity might prove useful in the evolution of interesting behaviour.

This section will be devoted essentially to representing the structure of a GenNet by its corresponding bit string chromosome. The representation chosen by the author is quite simple. Label the N neurons in the GenNet from 1 to N. (The number of neurons in the GenNet is a user specified parameter, typically about a dozen neurons per GenNet). If the GenNet contains N neurons, there will be N*N interconnections (including a connection looping directly back to its own neuron as shown in FIG. 3). Each connection is specified by its sign (where a positive value represents an excitory synapse, and a negative value represents an inhibitory synapse), and its weight value. The user decides the number (P) of binary places used to specify the magnitude of the weights. (Typically this number P is 6, 7, or 8). The weights were chosen to have a modulus always less than 1.0 Assuming one bit for the sign (a 0 indicating a positive weight, a 1 a negative weight), the number of bits needed to specify the signed weight per connection is (P + 1). Hence for N*N connections, the number of bits is N*N*(P+1).

The signed weight connecting the ith (where i = 0 -> N-1) to the jth neuron is the (i*N + j)th group of (P+1) bits in the chromosome (reading from left to right). From this representation, one can construct a GenNet from its corresponding chromosome. In the "Popqual" phase, the GenNet is "built" and "run", i.e. appropriate initial (user chosen) signal values are input to the input neurons (the input layer of a layered feedforward network, or simply to a subset of the fully connected neurons which are said to be input neurons).

GenNets are assumed to be "clocked" (or synchronous), i.e. between two ticks of an imaginary clock (a period called a "cycle"), all the neurons calculate their outputs from their inputs. These outputs become the inputs for the next cycle. There may be several hundred cycles in a typical GenNet behavioural fitness measurement. The outputs (from the output layer in a feedforward net, or from a subset of fully connected neurons which are designated the output neurons) are used to control or specify some process or behaviour. Three concrete examples will follow. The function or behaviour specified by this output (whether time dependent or independent) is measured for its fitness or quality, and this fitness is then used to select the next generation of GenNet chromosomes.

5. Genetic Programming (GP)

Now that the techniques used for building a GenNet are understood, we can begin thinking about how to build networks of networks, i.e. how to use GenNets to control other GenNets, thus allowing hierarchical, modular construction of GenNet circuits. Genetic Programming consists of two major phases. The first is to evolve a family of separate GenNets, each with its own behaviour. The second is to put these components together such that the whole functions as desired. In Genetic Programming, the GenNets are of two kinds, either behavioural (i.e. functional) or control. Control GenNets send their outputs to the inputs of behavioural GenNets and are used to control or direct them. This type of control is also of two kinds, either direct or indirect. Examples of both types of control will be shown in the following sections.

Direct control is when the signal output values are fed directly into the inputs of the receiving neurons (usually with intervening weights of value +1.0). The receiving GenNets need not necessarily be behavioural. They may be control GenNets as well. (Perhaps "logic" GenNets might be a more appropriate term). Indirect control may occur when an incoming signal has a value greater than a specified threshold. Transcending this threshold can act as a trigger to switch off one behavioural GenNet and to switch on another. Just how this occurs, and how the output signal values of the GenNet which switches off influence the input values of the GenNet which switches on, will depend on the application. For a concrete example, see section 8, which describes an artificial creature called "LIZZY".

For direct control, the evolution of the behavioural and control GenNets can take place in two phases. Firstly the behavioural GenNets are evolved. Once they perform as well as desired, their signs and weights are frozen. Then the control GenNets are evolved, with the outputs of the control GenNets connected to the inputs of the behavioural GenNets (whose weights are frozen). The weights of the control GenNets are evolved such that the control is "optimum" according to what the "control/behavioural-GenNet-complex" is supposed to do. See section 6 for a concrete example of direct control. Once the control GenNets have been evolved, their weights can be frozen and the control/behavioural GenNets can be considered a unit. Other such units can be combined and controlled by further control GenNets. This process can be extended indefinitely, thus allowing the construction of hierarchical, modular GenNet control systems or as the author prefers like to look at them, artificial nervous systems. Note that there is still plenty of unexplored potential in this idea. The three examples which follow really only scratch the surface of what might be possible with Genetic Programming.

6. Example 1 : "Pointer"

"Pointer" is an example of time independent evolution, meaning that the outputs of GenNets are not fed back to the inputs. The inputs are conventionally clamped (fixed) until the outputs stabilize. FIG. 6 shows the setup used in this example. It is the two-eye two-joint robot arm positioning (or pointing) problem. The aim of the task is to move the robot arm from its vertical start position X to the goal position Y. J1 and J2 are the joints, E1 and E2 are the eye positions, and JA1, JA2, EA1 and EA2 are the joint and eye angles of the point Y.

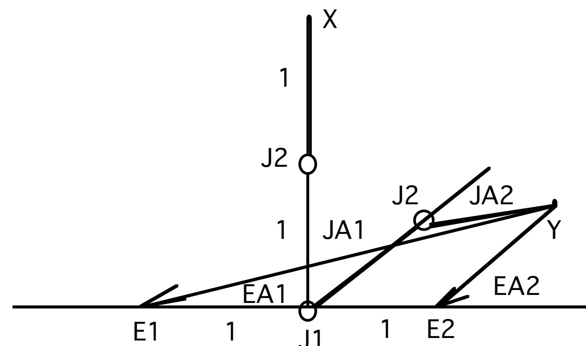


FIG. 6 SETUP FOR THE "POINTER" EXPERIMENT

"Pointer" is an example of the modular hierarchical approach of Genetic Programming. Two different GenNet modules will be evolved. The first, called the "joint module", controls the angle JA that a given joint opens to, for an input control signal of a given strength - and the second, called the "control module", receives inputs EA1 and EA2 from the two eyes and sends control signals to the joints J1 and J2 to open to angles of JA1 and JA2.

FIG. 7 shows the basic circuit design that the GA uses to find the "joint" and "control" modules. The joint modules (two identical copies) are evolved first, and are later placed under the control of the control module. Each module (containing a user specified number of neurons) is fully connected, including connections from each neuron to itself. Between any two neurons are two connections in opposite direction, each with a corresponding (signed) weight. The input and output neurons also have "in" and "out" links but these have fixed weights of 1 unit. The outputs of the control module are the inputs of the joint modules as shown in FIG. 7.

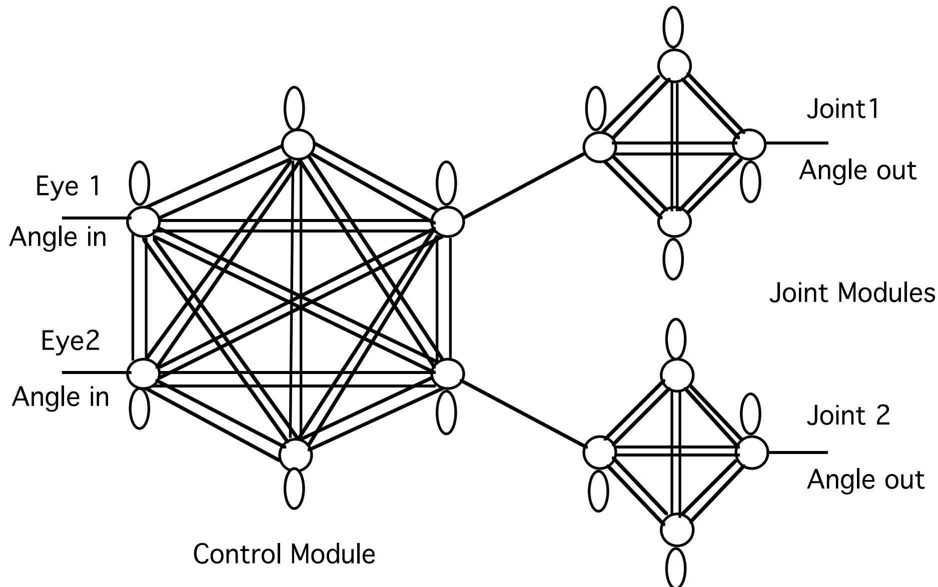


FIG. 7 GenNet MODULES FOR THE "POINTER" EXPERIMENT

The aim of the exercise is to use the GA to choose the values of the signs and weights of the various modules, such that the overall circuit performs as desired. Since this is done in a modular fashion, the weights of the joint module are found first. These weights are then frozen, and the weights of the control circuit found so that the arm moves as close as possible to any specified goal point Y.

With both weights and transfer functions restricted to the +1 to -1 range, output values stabilized (usually after about 50 cycles or so for 1% accuracy). In each cycle, the outputs are calculated from the inputs (which were calculated from the previous cycle). These outputs become the input values for the neurons that the outputs connect to. The GA is then used to choose the values of the weights, such that the actual output is as close as possible to the desired output.

The function of the joint module is effectively to act as a "halver" circuit, i.e. the output is to be half the value of the input. This may seem somewhat artificial, but the point of "pointer" is merely to illustrate the principles of modular hierarchical GenNet design (Genetic Programming). To evolve the "joint" module, 21 training input values ranging from +1 to -1 in steps of -0.1, were used. The desired output values thus ranged from 0.5 to -0.5, where 0.5 is interpreted to be half a turn of the joint, i.e. a joint angle of 180 degrees (a positive angle is clockwise).

The GA parameters used in this experiment were :- crossover probability 0.6, mutation rate 0.001, scaling factor 2.0, population 50, number of generations - several hundred. The fitness or quality measure used in the evolution of the joint module was the inverse of the sum of the squares of the differences between the desired and the actual output values.

FIG. 8 shows the set of training points Y used to evolve the control module. These points all lie within a radius of 2 units, because the length of each arm is 1 unit. For each of these 32 points, a pair of "eye angles", EA1 and EA2, is calculated and converted to a fraction of one half turn. These values are then used as inputs to the control circuit. The resulting 2 "joint angle" output values JA1 and JA2, are then used to calculate the actual position of the arm Y', using simple trigonometry. The quality of the chromosome which codes for the signs and weights of the control module is the inverse

of the sum of the squares of the distances between the 32 pairs of actual positions Y' and the corresponding desired positions Y.

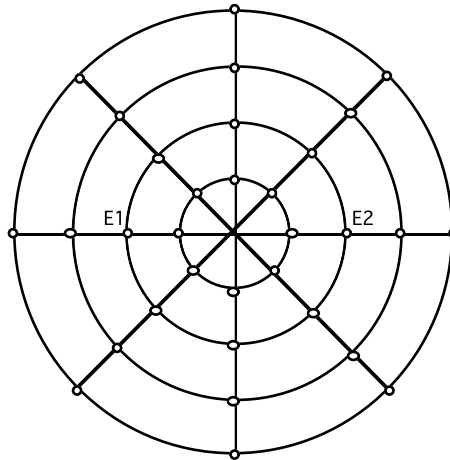


FIG. 8 TRAINING POINTS FOR THE "POINTER" EXPERIMENT

FIG. 9 shows an example solution for the 16 weight values of the joint module. With these weights (obtained after roughly 100 generations with the GA), the actual output values differed from the desired values by less than 1%. Similar solutions exist for the 36 weights of the control module, which also gave distance errors less than 1% of the length of an arm (1 unit). What was interesting was that every time a set of weights was found for each module, a different answer was obtained, yet each was fully functional. The impression is that there may be a large number of possible adequate solutions, which gives Genetic Programming a certain flexibility and power.

WEIGHTS		FROM NEURON			
		0	1	2	3
TO NEURON	0	-0.96875	0.84375	-0.875	0.90625
	1	-0.96875	-0.78125	-0.875	-0.78125
	2	-0.9375	-0.71875	-0.75	-0.28125
	3	-0.15625	-0.9375	0.3125	-0.65625

FIG. 9 WEIGHT VALUES OF JOINT MODULE

Also interesting in this result was the generalisation of vector mapping which occured. The training set consisted of 32 fixed points. However if two arbitrary eye angles were input, the corresponding two actual output joint angles were very close to the desired joint angles.

The above experiment is obviously static in its nature. In FIG. 7 there are no dynamics in terms of output values from the output neurons fed back to the inputs of the input neurons. However, what one does see clearly is the idea of hierarchical control,

where control GenNets are used to command functional GenNets. This idea will be explored more extensively in the "LIZZY"." experiment in section 8. The contents of this section are based on a paper by the author[11].

7. Example 2 : "Walker"

Having considered the time independent process in the above section, it is fascinating to question whether GenNets can be used to control time dependent systems, where the fitness would be the result of several hundred cycles of some essentially dynamic process in which feedback would feature. The great advantage of the GenNet approach is that even if the dynamics are complicated, the GA may be able to handle it (although there is no guarantee. See the final section on "evolvability"). The GA is usually indifferent to such complexity, because all it cares about is the fitness value. How this value is obtained is irrelevant to the GA. Good chromosomes survive, i.e. those which specify a high fitness performance will reproduce more offspring in the next generation.

As a vehicle to test this idea, the author chose to try to get a GenNet to send time dependent control signals to a pair of stick legs to teach them to walk. The set up is shown in FIG. 10.

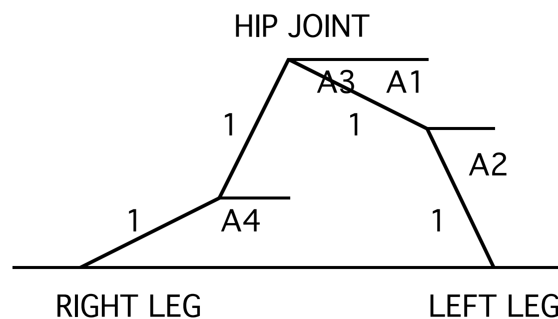


FIG. 10 SETUP FOR "WALKER" EXPERIMENT

The output values of the GenNet are interpreted to be the angular accelerations of the four components of the legs. Knowing the values of the angular accelerations (assumed constant over one cycle - where a cycle is the time period over which the neurons calculate (synchronously) their outputs from their inputs), and knowing the values of the angles and the angular velocities at the beginning of a cycle, one can calculate the values of the angles and the angular velocities at the end of that cycle. As input to the GenNet (control module) were chosen the angles and the angular velocities. FIG. 11 shows how this feedback works.

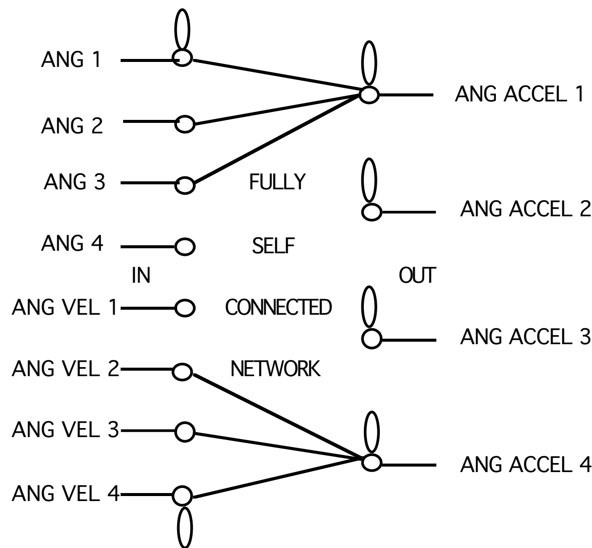


FIG. 11   THE "WALKER" GenNet

Knowing the angles, one can readily calculate the positions of the two "feet" of the stick legs. Whenever one of the feet becomes lower than the other, that foot is said to be "on the ground", and the distance (whether positive or negative) between the positions of the newly grounded foot and the previously grounded foot is calculated.

The aim of the exercise is to evolve a GenNet which makes the stick legs move as far as possible to the right in the user specified number of cycles, and cycle time. The GenNet used here consists of 12 neurons - 8 input neurons and 4 output neurons (no hidden neurons). The 8 input neurons have as inputs the values of the 4 angles and the 4 angular velocities. The input angles range from -1 to +1, where +1 means one half turn (i.e. 180 degrees). The initial (start) angles are chosen appropriately, as are the initial angular velocities (usually zero) which also range from -1 to +1, where +1 means a half turn per second. The activity of a neuron is calculated in the usual way, namely the sum of the products of its inputs and its weights, where weight values range from -1 to +1. The output of a neuron is calculated from this sum, using the antisymmetric sigmoid function of FIG. 2 The outputs of neurons are restricted to have absolute values of less than 1 so as to avoid the risk of explosive positive feedback. The chromosomes used have the same representation as described in the previous section.

The selection technique used was "roulette wheel" and the quality measure (fitness) used for selecting the next generation was (mostly) the total distance covered by the stick legs in the total time T, where (T = C*cycletime) for the user specified number C of cycles and cycletime. The fitness is thus the velocity of the stick legs moving to the right. Right distances are non negative. Stick legs which move to the left scored zero and were eliminated after the first generation.

Using the above parameter values, a series of experiments was undertaken. In the first experiment, no constraints were imposed on the motion of the stick legs (except for a selection to move right across the screen). The resulting motion was most unlifelike. It consisted of a curious mixture of windmilling of the legs and strange contortions of the hip and knee joints. However, it certainly moved well to the right, starting at given angles and angular velocities. As the distance covered increased, the speed of the motion increased as well and became more "efficient", e.g. windmilling was squashed to a "swimmers stroke". See FIG. 12 for snapshots.

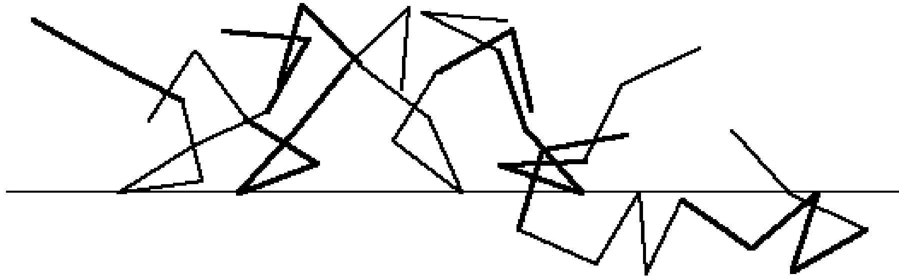


FIG. 12 SNAPSHOTS OF "WALKER" WITH NO CONSTRAINTS

In the second experiment, the stick legs had to move such that the hip joint remained above the floor (a line drawn on the screen). During the evolution, if the hip joint did hit the floor, evolution ceased, the total distance covered was frozen, and no further cycles were executed for that chromosome. After every cycle, the coordinates of the two feet were calculated and a check made to see if the hip joint did not lie below both feet. This time the evolution was slower, presumably because it was harder to find new weights which led to a motion satisfying the constraints. The resulting motion was almost as unlifelike as in the first experiment, again with windmilling and contortions, but at least the hip joint remained above the floor.

In the third experiment, a full set of constraints was imposed to ensure a lifelike walking motion (e.g. hip, knees and toes above the floor, knees and toes below the hip, knee angles less than 180 degrees etc). The result was that the stick legs moved so as to take as long a single step as possible, and "did the splits" with the two legs as extended as possible and the hip joint just above the floor. From this position, it was impossible to move any further. Evolution ceased. It looked as though the GA had found a "local maximum". This was a valuable lesson and focussed attention upon the important concept of "evolvability", i.e. the capacity for further evolution. A change in approach was needed.

This led to the serendipitous discovery of the concept of Behavioural Memory, i.e. the tendency of a behaviour evolved in an earlier phase to persist in a later phase. For example, in phase 1 evolve a GenNet for behaviour A. Take the weights and signs resulting from this evolution as initial values for a second phase which evolves behaviour B. One then notices that behaviour B contains a vestige of behaviour A. This form of "sequential evolution" can be very useful in Genetic Programming, and was used to teach the stick legs to walk, i.e. to move to the right in a step like manner, in 3 evolutionary phases.

In the first phase, a GenNet was evolved over a short time period (e.g. 100 cycles) which took the stick legs from an initial configuration of left leg in front of right leg to the reverse configuration. The angular velocities were all zero at the start and at the finish. By then allowing the resulting GenNet to run for a longer time period (e.g. 200 cycles) the resulting motion was "steplike", i.e. one foot moved in front of and behind the foot on the floor, but did not touch the floor. The quality measure (fitness) for this GenNet was the inverse of the sum of the squares of the differences between the desired and the actual output vector components (treating the final values as an 8 component state vector of angles and angular velocities).

This GenNet was then taken as input to a second (sequential evolutionary) phase which was to get the stick legs to take short steps. The fitness this time was the product of the number of net positive steps taken to the right, and the distance. The result was a definite stepping motion to the right but distance covered was very small. The resulting

GenNet was used in a third phase in which the fitness was simply the distance covered. This time the motion was not only a very definite stepping motion, but the strides taken were long. The stick legs were walking. See FIG. 13 for snapshots.

FIG. 13 SNAPSHOTS OF "WALKER" WITH FULL CONSTRAINTS

The above experiments were all performed in tens to a few hundred generations of the GA. Typical GA parameter values were - population size = 50, crossover probability 0.6, mutation probability 0.001, scaling factor = 2.0, cycletime = 0.03, number of cycles = 200 to 400. A video of the results of the above experiments has been made. To obtain a real feel for the evolution of the motion of the stick legs, one really needs to see it. The effect can be quite emotional, as proven by the reaction of audiences at several world conferences. It is really quite amazing that the angular coordination of 4 lines can be evolved to the point of causing stick legs to walk.

The success of WALKER had such a powerful effect on the author that he became convinced that it would be possible to use the basic idea that one can EVOLVE BEHAVIOURS (the essential idea of Genetic Programming) to build artificial nervous systems, by evolving many GenNet behaviours (one behaviour per GenNet) and then switching them on and off using control GenNets. A more concrete discussion of this exciting prospect follows in the next section.

8. Example 3 : "LIZZY"

This section reports on work still under way, so only partial results can be mentioned. The "LIZZY Project" is quite ambitious and hopes to make a considerable contribution to the emerging new field of Artificial Life[8]. This project aims to show that Genetic Programming can be used to build a (simulated) artificial nervous system using GenNets. This process is called "Brain Building" (or more accurately, "Nanobrain Building"). The term "nanobrain" is appropriate, considering that the human brain contains nearly a trillion neurons and that a nanobrain would therefore contain the order of hundreds of neurons. Simulating several hundred neurons is roughly the limit of present day technology, but this will change.

This section is concerned principally with the description of the simulation of just such a nanobrain. The point of the exercise is to show that this kind of thing can be done, and if it can be done successfully with a mere dozen or so GenNets as building blocks, it will later be possible to design artificial nervous systems with GenNets numbering in the hundreds, thousands and up. One can imagine in the near future, whole teams of human Genetic Programmers devoted to the task of building quite sophisticated nano(micro?) brains, capable of an elaborate behavioural range.

To show that this is no pipe dream, a concrete proposal will be presented showing how GenNets can be combined to form a functioning (simulated) artificial nervous system. The implementation of this proposal has not yet been completed (at the time of

writing), but progress so far inspires confidence that the project will be completed successfully. The "vehicle" chosen to illustrate this endeavour is shown in FIG. 14.

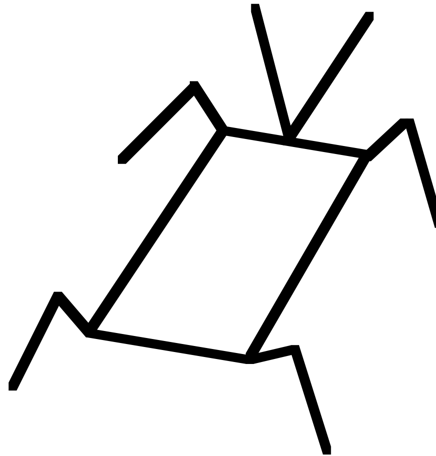


FIG. 14 "LIZZY", THE ARTIFICIAL LIZARD

This lizard-like creature (called LIZZY) consists of a rectangular wire-frame body, four two-part legs and a fixed antenna in the form of a V. LIZZY is capable of reacting to three kinds of creature in its environment, namely, mates, predators and prey. These three categories are represented by appropriate symbols on the simulator screen. Each category emits a sinusoidal signal of a characteristic frequency. The amplitudes of all these signals decrease inversely as a function of distance. Prey emit a high frequency, mates a middle frequency, and predators a low frequency. The antenna picks up the signal, and can detect its frequency and average strength. Once the signal strength becomes large enough (a value called the "attention" threshold), LIZZY executes an appropriate sequence of actions, depending on the frequency detection decision.

If the object is a prey, LIZZY rotates towards it, moves in the direction of the object, until the signal strength is a maximum, stops, and pecks at the prey like a hen (by pushing the front of its body up and down with its front legs), and after a while, wanders off. If the object is a predator, LIZZY rotates away from it, and flees until the signal strength is below attention threshold. If the object is a mate, LIZZY rotates towards it, moves in the direction of the object until the signal strength is a maximum, stops, and mates (by pushing the back of its body up and down with its back legs), and after a while, wanders off.

The above is merely a sketch of LIZZY's behavioural repertoire. In order to allow LIZZY to execute these behaviours, a detailed circuit of GenNets and their connections needs to be designed. FIG. 16 shows an initial attempt at designing such a "GenNet Circuit". There are 7 different motion GenNets, of which "random move" is the default option. When any of the other 6 motions is switched on, the random move is switched off. Provisionally each motion GenNet consists of 8 output neurons, whose output values (as in the WALKER GenNet) are interpreted as angular accelerations. The position of the upper (and lower) part of each leg is defined by the angle that the leg line takes on a cone whose axis of symmetry is equidistant from each of the XYZ axes defined by the body frame. The leg line is confined to rotate around the surface of this cone as shown in FIG. 15.

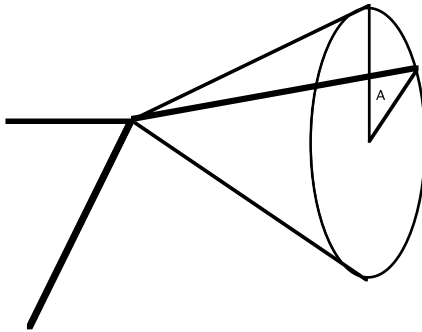


FIG. 15   CONICAL ROTATION OF "LIZZY's" LEGS

This approach was chosen so as to limit the number of degrees of freedom of the leg angles. If each leg part had two instead of one degree of freedom, and if one wanted to keep the GenNet fully connected, with 16 outputs, and 32 inputs (angles and angular velocities), i.e. 48 neurons per GenNet, hence 48 squared connections, the resulting chromosome would have been huge.

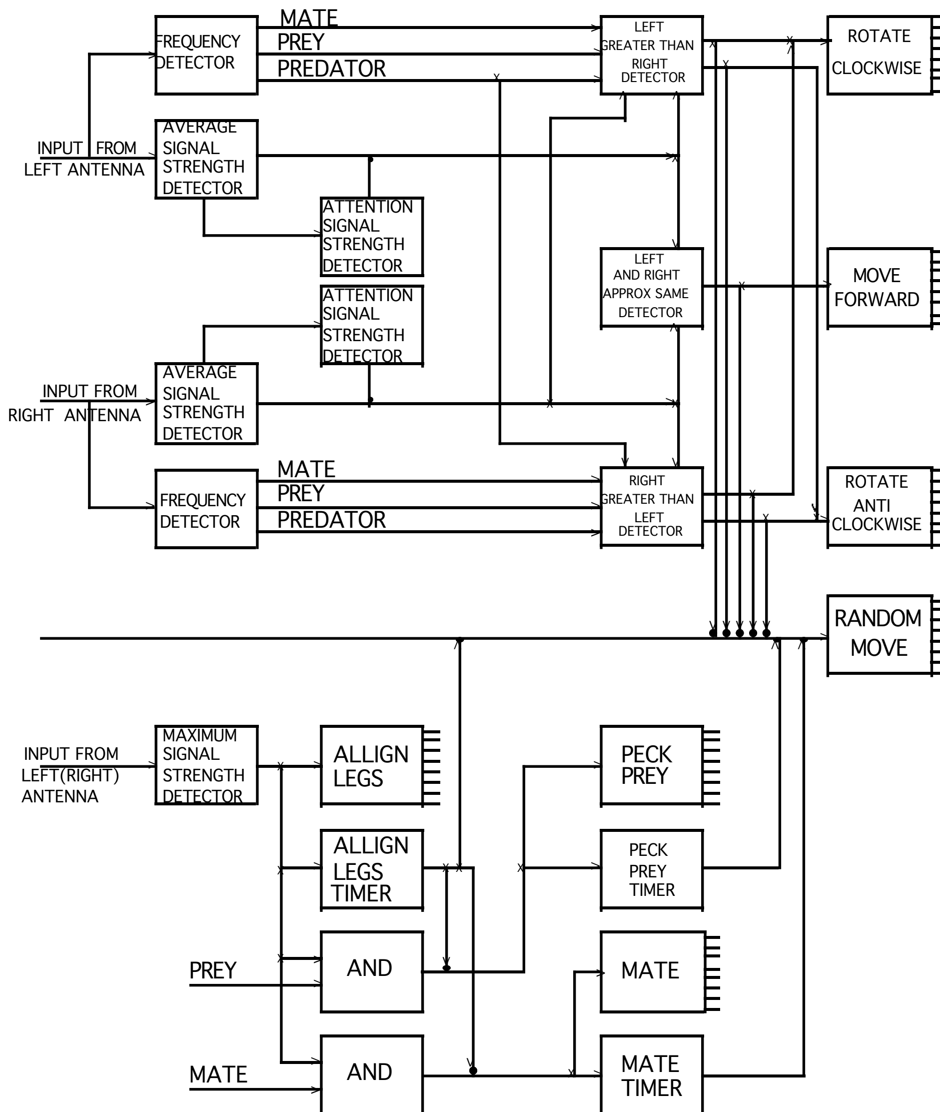


FIG. 16   THE "LIZZY" GenNet CIRCUIT

The power and the fun of GenNets is that one can evolve a motion without specifying in detail, how the motion is to be performed. For example, the fitness for the "move forward" GenNet will be a simple function of the total distance covered in a given number of cycles, (modified by a measure of its total rotation and its drift away from the desired straight ahead direction). Similar simple fitness functions can be defined for the two rotation GenNets. (In fact only one needs to be evolved. One can then simply swap body side connections, for symmetry reasons).

Understanding the GenNet circuit of FIG. 16 is fairly straightforward, except perhaps for the mechanism of rotating toward or away from the object. The antenna is fixed on the body frame of LIZZY. If the signal strength on the left antenna is larger than that on the right, and if the object is a prey or mate, then LIZZY is to turn towards it by rotating anticlockwise, or away from it by rotating clockwise if the object is a predator. Eventually the two signal strengths will be equidistant from the object. Once this happens, LIZZY moves forward.

The paragraphs below describe each of the GenNets used in FIG. 16 and give their quality (fitness) functions. Since this section reports on work still in progress, not all the GenNets have been completed at the time of writing. Those remaining "to-be-evolved" are indicated by (TBE).

<u>Detector GenNets</u>

* Frequency Detector (TBE)
* Average Signal Strength Detector (TBE)
* Maximum Signal Strength Detector (TBE)

The above detector GenNets have yet to be evolved. Initial attempts at frequency detection using GenNets have shown that this will not be a trivial business, indicating that Genetic Programming is still very much an art.

* Left Greater than Right Detector

This GenNet was evolved using supervised learning. It had two input neurons, 1 output neuron, and 7 hidden neurons, 10 in all. The desired output value was to be greater than 0.1 if the "left" input value was greater than the "right" input value by 0.1, and negative otherwise. Each input value ranged from -1 to +1 in steps of 0.05, giving 21*21 combinations. The fitness was defined as the inverse of the sum of the squares of the errors, e.g. if the inputs were (left 0.60) and (right 0.45) and the output was 0.045, then the error was (0.1 - 0.045) squared. With clamped input values, output values usually stabilize after a few dozen cycles. The output values were measured after a fixed number of cycles (e.g. 20). This GenNet worked fine with very high quality fitness values.

Note that whether the rotate clockwise or anticlockwise motion GenNet is to be switched on depends upon the category of the detected creature. One could either have an L>R detector per creature category or one could try to evolve a multifunction GenNet with extra input(s), to indicate prey/mate or predator. If the predator input value is high, and L>R, then the rotate clockwise GenNet should switch on. Experience shows that multifunction GenNets (which have different behaviours for different input "switching" values), are harder to evolve. Sometimes they work, sometimes not. (For example, the author was able to get Walker to move (with no constraints) either right or left across the screen depending upon whether the input value on an extra input "switching" neuron was fixed to +0.5 or -0.5). Again, GP remains an art at the moment. Mathematical criteria for evolvability do not yet exist unfortunately.

* Right Greater than Left Detector

This is a trivial modification of the above GenNet. One merely switches the input connections.

* Left and Right About Same Detector

This GenNet is similar to the two above. The input training values are the same (21*21 combinations). If the two input values differed by less than 0.1, then the output value was to be greater than 0.1, otherwise negative. The fitness was the same inverse of the sum of the squares of the errors. This GenNet evolved easily.

* Attention Signal Strength Detector

This GenNet takes the output of the average signal strength detector and checks if that average value lies above a given threshold. If so, it gives an output value greater than 0.1, negative otherwise. The training input values ranged in steps of 0.01 over the same +/-1 range (201 possibilities). If the input was greater than 0.6 (an arbitrary threshold value) then the output was supposed to be 0.1, otherwise negative. The same fitness definition was used. No problems here.

<u>Logic GenNets</u>

* AND Gate

This GenNet has two input, one output and 5 hidden neurons. An input value is interpreted to be "high" if it is greater than 0.1 If both are high, the output value is supposed to be greater than 0.1, otherwise negative. The input training values for each input neuron ranged over +/- 1 in steps of 0.1 (21*21 combinations). The fitness was as above.

<u>Effector (Behavioural/Motion) GenNet</u>s

* Straight Ahead Walk

This GenNet contained 24 neurons, (16 input, 8 output, no hidden). (How does one choose these numbers? There are no formulas. Once again, GP is an art). The input values are calculated from the output values as described in section 2. The motion definition of LIZZY chosen is as follows. This motion occurs in a cycle. The first step is to find the position and orientation of the body frame resulting from the completion of the previous cycle. The input (conical) angle values are used to position the legs relative to the body frame (where for computational convenience, the body frame is momentarily assumed to lie in the XY plane). LIZZY's body is then given the same orientation and position as at the end of the previous motion cycle.

The next step is to position the legs relative to the floor. LIZZY is lowered or raised vertically so that the lowest leg touches the floor (the XY plane) at the point F1. LIZZY is then rotated about F1 in a vertical plane which includes the second lowest foot until that foot touches the floor at F2. LIZZY is then rotated about the axis F1-F2 until the third lowest foot touches the floor.

To get LIZZY to walk, a further component of the motion definition was introduced. The front two feet were assumed to be "sticky" and the back two feet merely "slid" on the floor. Either the left or right front leg was defined as the "pushleg", i.e. the front leg on the floor which did the "pushing" of LIZZY. Whenever the current pushleg lifted off the floor, the other front leg became the pushleg and its position on the floor fixed. Hence the motion definition contains three components - positioning the legs

relative to the wire frame body, positioning the body and legs relative to the floor, and then displacing the legs on the floor.

This motion definition seems rather artificial with hindsight, but was only an initial attempt to illustrate the principle of GP nervous system design. The fitness function for getting LIZZY to move straight ahead was simply the component of the distance covered along the straight ahead direction. LIZZY did walk reasonably well. (See the video).

* Rotate Clockwise

The same GenNet architecture as the above was used. The motion definition differed slightly in that to get a rotation, the moment that one of the front legs became the pushleg, the orientation of the vector between the pushleg body corner position and the fixed pushleg "foot-on-the-floor" position was fixed. LIZZY had to rotate so as to keep this vector fixed as the orientation of the leg relative to the body changed. The fitness was defined to be 1.0 - cosine(angle rotated). LIZZY could do a 180 degree turn in about 120 cycles.

* Rotate Anticlockwise

As above, but anticlockwise. In order to measure the sign of the angle rotated, the sine rather than the cosine was used initially, and the cosine calculated from it. Rotations in the wrong direction were given zero fitness and eliminated after the first generation.

* Allign Legs (TBE)

It is debatable whether such a GenNet is needed. The idea is just to take LIZZY from any leg configuration and reorientate it to a standard position i.e. with the leg angles taking given (symmetric) values, in order to allow LIZZY to peck or mate. For further discussion on this point, see the next section.

* Timers

These GenNets are used to specify how long the pecking and mating are to go on. The output goes from a high value (e.g. >0.6) to a low value (e.g. <0.1) after a given number of cycles (e.g. 50 cycles). The fitness was the usual inverse of the sum of the squares of the errors between desired and actual output values.

* Peck

This GenNet was rather trivial to construct. The back legs were frozen in an elevated symmetrical position. The value from any one (and only one) of the angle inputs was fed to all four front leg components. This inevitably led to a pecking motion by LIZZY's head.

* Mate

This GenNet was very similar to the Peck GenNet. This time the front legs were fixed in a symmetric position, and the four back legs were fed one (and only one) of the values of the angle input neurons. This led inevitably led to a mating motion.

* Random Move

Any reasonable motion GenNet will do here, to get LIZZY to move randomly (e.g. a half evolved GenNet for rotation, or straight ahead motion).

9. Discussion and Future Ideas

As is obvious from the previous section, a lot remains to be done before LIZZY is fully functional. Not only are there GenNets which have not yet been evolved, LIZZY's GenNets have yet to be integrated into the LIZZY GenNet circuit. Initially, switching on and off GenNets will use a simple "reset", i.e. LIZZY's leg positions will be reset to those taken at the beginning of the evolution of the particular motion whose GenNet is being switched on. However, at a later stage, a form of "limit cycle" behaviour may be possible. Instead of evolving a behaviour from only one set of starting parameters, one evolves the same behaviour over an ensemble of sets, where the fitness is the average performance over the ensemble. If the GenNet performance evolves into a form of limit cycle, the behaviour may occur independently of the starting conditions. This is an interesting idea to pursue, but depending on the motion definition, may require considerable computing power or lots of time for the evolution. If "behavioural limit cycling" is possible, then transitions between behaviours can be smooth. No resets will be necessary because no matter what the final state of the legs are when one GenNet switches off, the new behaviour will be generated from that (starting) state by the next GenNet switching on.

A related question concerns whether GenNets can be evolved which have a form of "behavioural inertia", i.e. they continue to behave in the desired way even when the number of executed cycles goes beyond the number used to evolve the behaviour. If "behavioural limit cycling" is not occuring, this "behavioural inertia" cannot be taken for granted.

If the ideas discussed in section 8 prove to be successful, several steps will need to be taken in order to make Genetic Programming a generally applicable methodology to computer science in general and to artificial life in particular. There are at least two major directions of development for future research. One concerns the speed at which artificial evolution can be implemented. It would be nice if these techniques could be accelerated by putting them directly into hardware, e.g. VLSI accelerator chips for GenNet development. Such hardware versions of GenNet generators have been called "Darwin Machines"[11,12,13]. They will be an essential component in future large scale GP projects. One can imagine teams of (human) Genetic Programmers devoted to the task of evolving large numbers of GenNets to be later incorporated into increasingly complex artificial nervous systems. If several thousand GenNets need to be developed for example, then evolution speed will be essential. Traditional serial computers are hopelessly slow for this kind of work.

The second major direction for future GP research development is to put GenNets into real world insect robots, i.e. to Genetically Program them. The alternative to this second direction would be to exclusively simulate increasingly sophisticated nervous systems. It is the first option that the author intends to take, because if one spends a lot of time and effort in creating elaborate simulations, at the end of the day, one does not have anything as valuable as a functioning real world robot for a comparable effort. However, simulation will remain important. One does not build an insect robot without being reasonably sure that it will work. This assurance usually comes from a successful simulation.

The definition of motion given in the LIZZY section is not real world compatible. It will therefore be changed to one which is, namely moving one leg at a time, and adjusting the orientation of the body frame to each change. The GA can be used to decide the order of moving the legs. This motion definition is much closer to how one would get a real insect robot to move. One can then allow the robot to do its own real world evolution. If these robots can be made cheaply, many of them could evolve simultaneously, reporting their fitnesses to a central computer which selects the

chromosomes of the next generation and passes them back to the robots. This is the vision that the author has for Genetically Programmed insect robots for the near future.

Building insect robots like this would presumably mean storing GenNet weights in ROMs and connecting them up to control real world detectors and effectors such that the robot is given an artificial nervous system. However there is a problem with evolving GenNet robots, and that concerns the time needed to measure their fitnesses in the real world. The great advantage of computer simulation is computer nanosecond speed. However if one sends control (ROM) GenNet control signals to real insect robot effectors, they will perform at real world mechanical speeds, i.e. in milliseconds at best and more likely in seconds. Hence measuring the fitnesses of a population will inevitably take time. There are several approaches one can take to this problem. One approach is to simulate the dynamics of the robots as accurately as possible, to get a "ball park" estimate of the fitness, and then use the resulting GenNet weights as starting values in a real world evolution.

Either one robot could be used sequentially many times, where the results of each chromosome's (dynamical) "expression" would be stored for each generation in order to be able to select the next generation, or, many such robots could evolve in parallel, as mentioned above. As microtechnology is gradually replaced by nanotechnology, the size and cost of these robots may decrease dramatically. Nanorobots ("nanots") could be made in huge numbers and evolved in parallel.

There are other ideas which need to be considered in the future of Genetic Programming. The artificial nervous system discussed in section 8 was entirely "instinctual", i.e. it was incapable of learning. The LIZZY GenNet circuit of FIG. 16 is "prewired". There is no adaptation based on LIZZY's experience. Nervous systems are more efficient and useful to the bodies they serve if they are capable of improving their performance on the basis of previous experience. Future GenNets capable of learning will be essential. This is an important short term research goal of GP.

Another idea which needs to be explored is the possibility of getting the GA (rather than a human Genetic Programmer) to combine GenNets to build artificial nervous systems. This raises the question as to which GenNets should be combined and how. It is difficult enough to evolve individual GenNets, so evolving artificial nervous systems would be orders of magnitude more difficult and more computationally expensive. However, if it were done incrementally, some progress in this direction might be possible. Modification of already successful artificial nervous systems may lead to more successful variants and thus be amenable to GA treatment.

Another aspect of GP which needs serious consideration concerns the concept of evolvability, i.e. the capacity of a system to evolve rapidly enough to be interesting. From experience, the author knows that if the choice of representation of the dynamics of the system or the choice of the fitness function is inappropriate, evolution can be slow to infinitesimal. For example, if a tiny change in the starting conditions causes major changes in the fitness result, then the "fitness landscape" (in the parameter space of solutions) may be "spiky" instead of just "hilly", making the GA search virtually random and thus very slow. This may not be such a problem if the parameter space is "solution-dense", but if not, then isolated solution-spikes may be very hard to find. Criteria for evolvability is a research topic worthy of the GA community. GenNet chromosomes can be very long (thousands of bits), especially when artificial nervous systems are being built, because they may have to code for large numbers of parameter values. See section 8 for example. The dynamic behaviour resulting from many time dependent parameters may be very sensitive to changes in the initial values of these parameters or to the GenNet weights.

One final idea concerns the possibility that elaboration of the middle "logic" layer (i.e. between the detectors and the effectors) in artificial nervous systems may become

sophisticated enough to be capable of handling "symbolic reasoning". If one looks at the development of biological life on Earth, one sees that instinctual behaviour came first, and only later did symbol manipulation evolve. The nervous systems capable of manipulating symbols were essentially superstructures built on an instinctual behavioural base. It is not too difficult to imagine that more elaborate GenNet circuits capable of handling sequences of production rules (of the form "If A&B&C => D"), may form an elementary basis for symbolic representation and manipulation. If logic GenNets can be used in such a way, we may see the beginnings of a "Grand Synthesis" between traditional symbolic Artificial Intelligence, Genetic Algorithms, Neural Networks and Artificial Life. Many "Lifers" (Artificial Life researchers) hope that their bottom up behavioural approach and the top down disembodied symbolic approach of AI researchers will one day meet somewhere in the middle.

10. References

1. M. Rudnick and D. Hammerstrom, “An Interconnect Structure for Wafer Scale Neurocomputers”, in *Proceedings of the 1988 Connectionist Models Summer School* 1988, eds D. Touretzky, G. Hinton, T. Sejnowski, Morgan Kaufmann, 1989.

2. M.A. Reed, "Quantum Semiconductor Devices", in "*Molecular Electronic Devices*", F.L. Carter, R.E. Siatkowski, H. Wohltjen eds. North Holland, 1988.

3. K.E. Drexler, "*Engines of Creation : The Coming Era of Nanotechnology*", Doubleday, 1986.

4. C. Schneiker, "*Nano Technology with Feynman Machines : Scanning Tunneling Engineering and Artificial Life*", in 8.

5. R.P. Feynman, "Quantum Mechanical Computers", *Optics News*, pp 11-20, February 1985.

6. D.E. Goldberg, “*Genetic Algorithms in Search, Optimization, and Machine Learning*”, Addison-Wesley, 1989.

7. Rumelhart D.E. & McClelland J.L., "*Parallel Distributed Processing*", Vols 1 & 2, MIT Press, 1986.

8. C.G. Langton ed., "*Artificial Life*", Addison Wesley, 1989.

9. *Proceedings of the 1st International Conference on Neural Networks*, San Diego, 1987.

10. J. H. Holland, "*Adaptation in Natural and Artificial Systems*", Ann Arbor, Univ. of Michigan Press, 1975.

11. H. de Garis, “Genetic Programming : Modular Neural Evolution for Darwin Machines”, *Proceedings IJCNN90 WASH DC, International Joint Conference on Neural Networks*, January 1990, Washington DC.

12. H. de Garis, "Genetic Programming : Building Nanobrains with Genetically Programmed Neural Network Modules", *Proceedings IJCNN90 SanDiego, International Joint Conference on Neural Networks*, June1990, SanDiego.

13. H. de Garis, "Genetic Programming : Building Artificial Nervous Systems Using Genetically Programmed Neural Network Modules", *Proceedings 7th. Int. Conf. on Machine Learning*, Austin Texas, June 1990, Morgan Kaufmann, 1990.