Coevolution of Intelligent Agents using Cartesian Genetic Programming

Gul Muhammad Khan
Electronics Department
University of York
York, YO10 5DD,UK
gk502@ohm.york.ac.uk

Julian F. Miller
Electronics Department
University of York
York, YO10 5DD,UK
jfm7@ohm.york.ac.uk

David M. Halliday
Electronics Department
University of York
York, YO10 5DD,UK
dh20@ohm.york.ac.uk

ABSTRACT

A coevolutionary competitive learning environment for two antagonistic agents is presented. The agents are controlled by a new kind of computational network based on a compartmentalised model of neurons. The genetic basis of neurons is an important [27] and neglected aspect of previous approaches. Accordingly, we have defined a collection of chromosomes representing various aspects of the neuron: soma, dendrites and axon branches, and synaptic connections. Chromosomes are represented and evolved using a form of genetic programming (GP) known as Cartesian GP. The network formed by running the chromosomal programs, has a highly dynamic morphology in which neurons grow, and die, and neurite branches together with synaptic connections form and change in response to environmental interactions. The idea of this paper is to demonstrate the importance of the genetic transfer of learned experience and life time learning. The learning is a consequence of the complex dynamics produced as a result of interaction (coevolution) between two intelligent agents. Our results show that both agents exhibit interesting learning capabilities.

Categories and Subject Descriptors

I.2.2 [**ARTIFICIAL INTELLIGENCE**]: Automatic Programming—*Program synthesis*; I.2.6 [**ARTIFICIAL INTELLIGENCE**]: Learning—*Connectionism and neural nets*

General Terms

Algorithms, Design, Performance

Keywords

Genetic Programming, Co-evolution, Brain, Artificial Neural Networks

1. INTRODUCTION

In this paper we present a coevolutionary competitive learning environment in which two agents struggle to achieve tasks and survive in a predator-prey relationship [26]. These agents are controlled by a new kind of computational network inspired by the biological neurons. Agents have a health which is related to their actions and encounters in a grid-based environment. The first agent increases its health (and consequently its fitness) by avoiding deleterious and attaining beneficial encounters. While the opposing agent increases its health solely by direct antagonistic encounters with the first agent. Paredis describes this type of fitness as 'life-time fitness evaluation' and discusses how this 'arms race' provides a strong driving force toward complexity [25].

The computational network possesed by each agent is based on a compartmentalised model of neural functions based on neuroscience. In this model we have idealised seven neural functions which we have encoded as chromosomes (see section 4.4). These represent various aspects of the neuron: soma, dendrites and axon branches, and synaptic connections. The collection of chromosome (forming the genotype) is encoded and evolved using a well known GP technique, Cartesian Genetic Programming (CGP) [9, 10]. The neurons are placed in a two dimensional grid. Neurite branches are allowed to grow and shrink, and communicate with each other via synapses. Dendrites [8], synaptic dynamics [7] and synaptic communication have been included to enhance the capabilities of the computational network. The network we described has the potential virtue that it is autonomous in the sense that when the compartmentalised chromosomal programs are run a network of neurons, neurites and synapses grows in response to its own internal dynamics and the agents environmental experiences. Stanley and Miikkulainen acheive complexification by the incremental elaboration of solutions through adding new neural structures (i.e. neurons and connections) [24]. However our computational network can 'complexify' itself in a way inspired by the way the brain complexifies itself without any change in genetic code.

One of the difficulties one faces at the outset in attempting to create a dynamic computational model inspired by neuroscience is that as it stands, the internal dynamics of biological neurons are too complicated to be modeled in a machine learning technique. However, we took the view that the biology of neurons (i.e. their gross morphology and connectivity) *is* sufficiently well understood [22], [20] to allow us to identify essential sub-systems (and their inputs and outputs) that we must attempt to evolve in order to achieve a computational equivalent. Conventional models of neural networks do not consider the genetics of neurons and the development (as in biology) of a mature network during

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

learning. Instead, they are dominated by a static connectionist view of the brain. However, Genetic Programming (GP) offers the capability to represent neural programs and the transfer of genetic changes from generation to generation. GP has been shown to be able to solve problems of this type in the absence of a fixed model [3] and often these solutions are not fragile and show unexpected emergent behaviours such as self-assembly and self-repairability [11], [12] which are natural properties of living systems. Thus GP, in principle, provides us with a means to represent complex neuron 'engines' that can be evolved to exhibit the properties of real neural systems, without the restrictions of a theoretical model of these systems.

2. CARTESIAN GP

Cartesian Genetic Programming (CGP) was developed from the work of Miller and Thomson [9, 10] for the evolutionary design of feed forward digital circuits. In CGP programs are represented by directed acyclic graphs. Graphs have advantages in that they allow implicit re-use of subgraphs. In its original form CGP used a rectangular grid of computational nodes (in which nodes were not allowed to take their inputs from a node in the same column). However, later work relaxed this restriction by always choosing the number of rows to be one (as used in this paper). The genotype in CGP has a fixed length. The genes are integers which encode the function and connections of each node in the directed graph. However, the phenotype is obtained via following referenced links in the graph and this can mean that some genes are not referenced in the path from program inputs to outputs. This results in a bounded phenotype of variable length. As a consequence there can be non-coding genes that have no influence on the phenotype, leading to a neutral effect on genotype fitness. The characteristics of this type genotypic redundancy have been investigated in detail [10, 13, 14, 15, 16] and found to be extremely beneficial to the evolutionary process on the problems studied.

Each node in the directed graph represents a particular function and is encoded by a number of genes. The first gene encodes the function that the node represents, and the remaining genes encode where the node takes its inputs from. The nodes take their inputs from either the output of a previous node or from a program input (terminal). The number of inputs that a node has is dictated by the number of inputs that are required by the function it represents.

Recent work has introduced module acquisition and evolution into CGP [17] and shown that these techniques are more scalable on harder problems. However the work presented in this paper doesn't yet utilize these methods. In addition a form of of CGP in which there are separate chromosomes encoding independent output however sharing modules has been introduced and shown to improve problem solving ability considerably [18].

3. ESSENTIAL NEURAL ASPECTS

This section describes the neuron model that we have incorporated into the network, along with the biological inspiration. Neurons are the main cells responsible for information processing in the brain. They produce adaptibility, learning and intelligent behavior because of their specialized biophysical structure. Neurons have specialized extensions called dendrites and axons [19]. Dendrites bring information to the cell body and axons take information away from the cell body. Neurons communicate with each other through electrochemical processes called synapses. They take inputs from the neighbouring neurons and decide whether to transfer this information in a forward direction, by firing an action potential when the cumulative affect of the inputs is greater than the firing threshold. Neurons have a number of dendrites and a single axon. Dendrites have a branching tree-like structure. Axons have branches at the end to communicate with other neurons in their vicinity.

Neurons receive signals at the dendrite branches. The signals are processed locally due to interactions between neighbouring dendrite branches, and is further processed along the dendrite due to leaky channels (reduction in signal magnitude) and voltage gated channels (amplification of the signal). The soma receives all the signals from dendrites and decides whether to fire an action potential or not. If the soma fires, the action potential is transferred to the axon[21]. The axon takes the signal and transfers it to all the neighbouring neurons through its branches and synaptic connections. Neurons are highly dynamic: new branches may be produced in the axon and dendrites, old branches may vanish, branches grow and shrink, new neurons may be produced and old neurons may die (see section 4.4). We have idealized this behaviour of neuron in terms of seven processing compartments (see section 4.4):

- Local interaction among the branches of the same dendrite.
- Production of new branches, removal of branches, branch growth.
- Processing signals received from dendrites at soma, and deciding whether to fire an action potential.
- Creation or destruction of neurons, and modulation of the firing rate.
- Transfer of potential through axon branches to the neighbouring dendrite branches.
- Updating the weights (and consequently the capability to make a synapse) of neighbouring dendrite branches and the axon branch.
- Axon branch growth, possibility of new branches, or removal of branches.

4. CGP COMPUTATIONAL NETWORK

The CGP Computational Network (CGPCN) is organized in such a way that neurons are placed randomly in a two dimensional grid. The number of neurons are specified by the user. Each neuron is initially allocated a random number of dendrites, dendrite branches and axon branches. Neurons take information through dendrite branches and transfer it through axon branches to the neighbouring neurons. The dynamics of the network can change during this process, the branches may grow or shrink and move from one grid point to another, can produce new branches, and can disappear, the neurons may die or produce new neurons. Axon branches transfer information only to the dendrite branches in their proximity.

A *Statefactor* is used as a parameter to reduce the computational burden, by keeping some of the neurons and branches inactive for a number of cycles. When the statefactor is zero the neurons and branches are considered to be active and their corresponding program is run. The value of the *Statefactor* is affected by genetic processes. The network consists of:

- Neurons with a number of dendrites, with each dendrite having a number of branches and an axon having a number of axon branches.

- A genotype representing the genetic code of the neurons. Each genotype consists of seven chromosomes (see Section-4.4), each representing a digital circuitry. These chromosomes in turn represent the functionality of different parts of the neuron.

4.1 Information Processing in the Network

Information processing in the network starts by selecting the list of active neurons in the network and process them in a random sequence. The processing of neural components is carried out in time-slices so as to emulate parallel processing. Each neuron take the signal from the dendrites by running the dendritic electrical processing programs (encoded in the genotype). The signals from dendrites are averaged and applied to soma program along with the soma potential. The soma program is run to get the final value of soma potential, which decides whether a neuron fires an action potential or not. If so, the signal is transferred to other neurons through axosynaptic branches. The same process is repeated in all neurons. After each cycle of neural network the potential and state factor of the soma and the branches are reduced by certain factor. This provides a sense of time and makes inactive branches and neuron to move towards activity step by step. After five cycles of network or one step of the agent, the health and weights of neurons and branches are reduced by certain factor, in order to get rid of unimportant neurons and branches.

4.2 Evolutionary Strategy

The evolutionary strategy utilised is of the form $1 + \lambda$, with λ set to 4 [15], i.e. one parent with 4 offspring (population size 5). The parent, or elite, is preserved unaltered, whilst the offspring are generated by mutation of the parent. The best chromosome is always promoted to the next generation. However, if there is was a tie for best fitness then an offspring is chosen in preference to the parent. This ensures that the benefits of neutral genetic drift are utilized [13].

4.3 Cartesian Genetic Program (Chromosome)

The CGP function nodes used here consists of multiplexer-like operations [13] with three inputs per node. The operations on chromosomes are of two types: Scalar Processing and Vector Processing. In the scalar case, the inputs and outputs are integers. In the vector case, the inputs are arranged in the form of an array. The number of integers per vector is variable, in this way CGP can handle an arbitrary number of inputs.

4.4 CGP Model of Neuron

This neuron-inspired model consists of seven main processes:

-Electrical Processing in Dendrite
-Life Cycle of Dendrite Branch
-Electrical Processing in Soma
-Life Cycle of Soma
-Electrical Processing in Axo-Synaptic Branch
-Weight Processing in Axo-Synaptic Branch
-Life Cycle of Axo-Synapse Branch

Each of these processes are individually represented by a CGP chromosome. A detailed explanation of the processes follows.

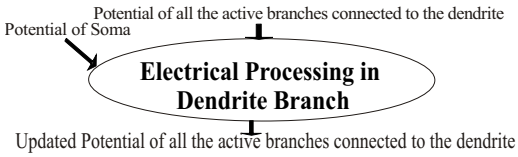

Figure 1: Electrical processing in the dendrite

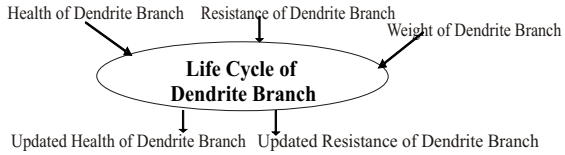

Figure 2: Life cycle of dendrite branch

4.4.1 Electrical Processing in Dendrite

This chromosome handles the interaction between potentials of dendrite branches. Figure 1 shows the inputs and outputs to the Electrical Processing in dendrite chromosome. Input consists of potentials of all the active branches connected to the dendrite and the soma potential. Since there are many dendrite branch potentials and one soma potential, we increase the importance of the soma potential by creating multiple entries (in this case 10) of it (in the input vector) before applying. This CGP program produces the updated values of the dendrite branch potentials as output. The potential of each branch is processed by adding weighted values of *Resistance*, *Health*, and *Weight* of the branch. The *Statefactor* of branches are adjusted based on the updated value of branch potential. If any of the branch is active, its life cycle CGP program is run, otherwise continue processing the other dendrites.

4.4.2 Life Cycle of Dendrite Branch

This chromosome shows the CGP algorithm for the life cycle of dendrite branches. Figure 2 shows inputs and outputs of the chromosome. Variation in *Resistance* of dendrite branches is used to decide whether it will grow, shrink, or stay at its current location. The updated value of dendrite branch *Health* decides whether to produce offspring, to die, or remain as it was with an updated *Health* value. Producing offspring results in a new branch at the same grid point connected to the same dendrite.

4.4.3 Electrical Processing in Soma

This chromsome is responsible for determining the final value of soma potential after receiving signals from all the dendrites. All the dendrites potentials are averaged, which in turn are the average of potentials of active branches attached to them. This average potential along with the soma potential is applied as input to the Electrical processing in Soma chromosome as shown in Figure 3.

The chromosome produces an updated value of the soma potential as output, which is further processed with a weighted summation of *Health* and *Weight* of the soma. The processed potential of the soma is then compared with the threshold potential of the soma, and a decision is made whether to fire an action potential or not. If the soma fires

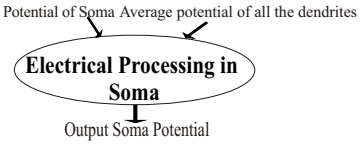

Figure 3: Electrical processing in soma

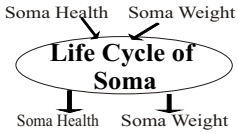

Figure 4: Soma life cycle

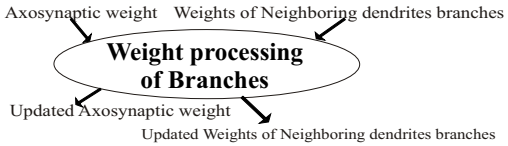

Figure 6: Axo-synaptic branch weight processing

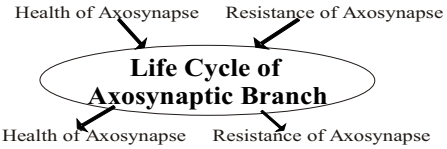

Figure 7: Axo-synaptic branch life cycle

it is kept inactive (refractory period) for a few cycles by changing its *Statefactor*, the soma life cycle chromosome is run, and the firing potential is sent to the other neurons by running the axosynapse electrical processing chromosome. The threshold potential of soma is also adjusted to a new value if the soma fires.

4.4.4 Soma Life Cycle

Figure 4 shows inputs and outputs of the soma life cycle chromosome. This chromosome is intended to evaluate the life cycle of neuron. This chromosome produces updated values of *Health* and *Weight* of the soma as output. The updated value of the soma *Health* decides whether the soma should produce offspring, should die or continue as it is. If it produces offspring, then a new neuron is introduced into the network with a random number of dendrites and branches at the same location.

4.4.5 Electrical Processing in Axo-Synaptic Branch

The potential from the soma is transferred to other neurons through axon branches. Both the axon and the synapse are considered as a single entity with combined properties. Figure 5 shows the inputs and outputs to the chromosome responsible for the electrical processing in axosynaptic branch. As mentioned before, the soma potential is biased (see section 4.4.1). The chromosome produces the updated values of dendrite branch potentials and the axo-synaptic potential as output. The axo-synaptic potential is then processed as a weighted summation of *Health*, *Weight* and *Resistance* of the axon branch. The axo-synaptic branch weight processing program (see figure 6) is run after the above process and the processed axo-synaptic potential is assigned to the dendrite branch having the highest updated *Weight*. The *Statefactor* of the axosynaptic branch is also updated. If the axo-synaptic branch is active its life cycle program is executed.

4.4.6 Axo-synaptic Branch Weight Processing

The weight of axon branches affects its capability to modulate and transfer the information (signal) efficiently. The weights are responsible for modulating the signal. They affect almost all the neural processes either by virtue of being an input to a chromosomal program or as a factor in post processing of signals.

Figure 6 shows the inputs and the outputs to the axosynaptic weight processing chromosome. The CGP program encoded in this chromosome takes as input the *Weights* of the axo-synapse and the neighbouring dendrite branches and produces their updated values as output.

4.4.7 Axo-Synaptic Branch Life Cycle

The role of this chromosome is similar to dendrite branch life cycle chromosome. Figure 7 shows the inputs and outputs of axosynaptic branch life cycle chromosome. It takes *Health* and *Resistance* of the axon branch as input, and produces the corresponding updated values as output.

The updated values of *Resistance* are used to decide whether the axon branch should grow, shrink, or stay at its current location. The *Health* of the axon branch decides whether the branch will die, produce offspring, or merely continue with an updated value of health.

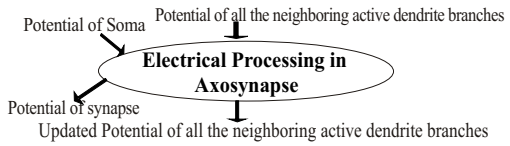

Figure 5: Electrical processing in axosynaptic branch

4.5 Inputs and Outputs

The inputs are applied through virtual axon branches by using axosynaptic electrical processing chromosomes. These branches are distributed in the network in a similar way to the axon branches of neurons. When inputs are applied to the system, the program encoded in the axo-synaptic electrical branch chromosome is executed, and the resulting signal is transfered to its neighbouring active dendrite branches. Similarly we have output virtual neurons which read the signal from the system through virtual dendrite branches. These virtual dendrite branches are distributed across the network. These branches are updated by the axo-synaptic chromosomes of neurons in the same way as other dendrite branches. The output from the output neuron is taken without further processing.

5. EXPERIMENTAL SETUP

5.1 Competitive Learning Scenario

The two agents live in a two dimensional grid (10x10) containing a number of pits (ten). Each agent has a home square. On one square there is a quantity of gold (see Fig. 9). This problem is a variant of an agent-based learning task, called Wumpus World, studied in Artificial Intelligence [23].

The positions of the pits and the gold is fixed but randomly assigned. The job of the first agent is to obtain the gold as many times as it can during its life time, while avoiding the pits and the second agent. The second agent's task is to catch the first agent as many times as it can. The second agent's task is more difficult than the first agent as the target for the second agent is mobile while the target for the first agent (the gold) is static.

During its life time, every time the second agent catches the first agent it becomes more difficult for it to catch it again, as the first agent learns, and tries to find a path to the gold that avoids the second agent. In order to avoid the imbalance in the difficulty of tasks for both agents, we introduced pits which only affect the life of the first agent. The first agent is weakened whenever it passes through squares containing pits. It always starts from a special square called home that is at the top left corner of the grid world. The second agent's home is at a corner of the grid that is diagonally opposite. Both the agents perceive a breeze in squares adjacent to the pits and a smell in the squares adjacent to each other directly not diagonally. The agents also perceive a glitter while passing close to the gold. Also, the agents will receive different signals while passing through these locations from different directions. So they have to learn to cope with, not only the breeze or the smell, but also with the direction of the breeze or the smell, and make a decision to move accordingly. All the locations (other than home) which are safe provide no signal. As the pits and the gold only directly affect the first agent, the second agent has to differentiate between all these signals (thus for it there is more noise in the environment) and try to identify the presence of the first agent in order to catch it. The two agents continue their journey as long as their health remains above zero (see later for more details).

It is important to appreciate how difficult this problem is. The two agents start with a few neurons with a random number of dendrites and branches, and with random connections. So, firstly, evolution must find a series of programs that build a computational network that is capable of solving the task while maintaining a stable network (i.e. not losing all the neurons or branches etc.). Secondly, it must find a way of processing infrequent environmental signals and differentiate among them. Thirdly, it must navigate in this environment using some form of memory or knowledge about the meaning of the signals, whether they are beneficial or deleterious. Fourthly, it must confer goal-driven behaviour on two virtual agents while increasing their life span.

Every time the second agent catches the first agent, both are reallocated to their home squares, to start their job again. Similarly, if the first agent gets the gold it is reallocated to its home square. However, in this case the second agent remains at its previous location. In order to do well the first agent has to remember how to find the gold again while avoiding pits and the second agent.

Both the agents are assigned with an initial health of 100 units. If the first agent is caught by the second agent its health is reduced by 60%, if caught by a pit its health is reduced by 10 units, if it gets the gold its health is increased by 60%. The second agent is not affected by anything except that its health is increased by 60% everytime it catches the first agent. For each single move the healths of the either agents are reduced by 1 unit. The fitness of the agents is accumulated over their lifetimes (while each agent's health is non-zero) in the following way: For each move, the fitness of the agent is increased by one. Everytime the agent obtains the gold, its fitness is increased by 1000. The fitness function for the second agent works as follows: For each move, the fitness of the agent is increased by one. Everytime the second agent catches the first agent, its fitness is increased by 1000.

When the experiment starts, both the agents take their inputs from the corresponding grid squares where they are located. This input is applied to their CGPCNs through virtual axosynapses. The networks are then run for five cycles. During this process they update the potentials of the virtual dendrite branches acting as the output of the networks. These updated potentials are averaged, and used to decide the direction of movement for the corresponding agents. The same process is repeated for the next grid square for both the agents until they die. The agents stop their journey if either their health becomes zero, or all their neurons die, or all the dendrite or axon branches die.

Each of five first agent population members are tested against the best performing second agent genotype from the previous generation. Similarly each of the five second agent population members are tested against the best performing first agent genotype from the previous generation. The initial random network is same for both the first and the second agent. It is the genotypes in each generation which cause different networks to form. The best first and second agent genotypes are selected as the parents for the new population.

The idea behind these experiments is two fold. Firstly we want to demonstrate that agents learn during their life time (i.e. post evolution) thus demonstrating that evolution has evolved the ability to learn. Secondly we wish to demonstrate that the genetic memory (learned experiences) obtained by an agent during its life time can be transferred to next generation through the genetic code.

5.2 CGP Computational Network Setup

The CGPCN is arranged in the following manner for this experiment. The network space where neurons and branches are located is arranged in the form of a 3x4 grid. Inputs and outputs are applied at five different random locations. Initial number of neurons is 5. Maximum number of dendrites is 5. Maximum number of dendrite and axon branches is 5. Maximum branch *StateFactor* is 7. Maximum soma *StateFactor* is 3. Mutation rate is 5%. Maximum number of nodes per chromosome is 100.

5.3 Results and Analysis

Figure 8 shows how the fitness of the agents, in one particular evolutionary run, change over 1,250 generations. It is evident that there are increases and decreases in fitness at different stages for each agent corresponding to when either of them has the upper hand.

The y-ordinate divided by 1000 gives the number of times an agent achieves its goals. The agents exhibit different kinds of dynamic behavior due to the interactions between neurons. Initially, the agents do not know anything about the gold, pits, and the signals that indicate the presence of these objects nearby. As they evolve, the agents develop their own memory of life experiences and this is encoded genetically. Each agent starts with a fixed randomly assigned

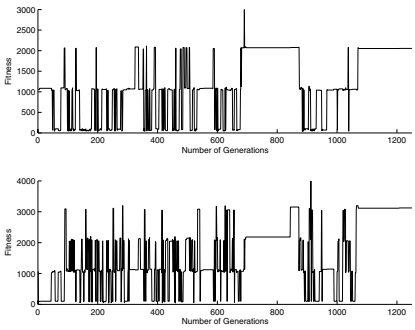

Figure 8: Fitness of the agents with evolutionary generations

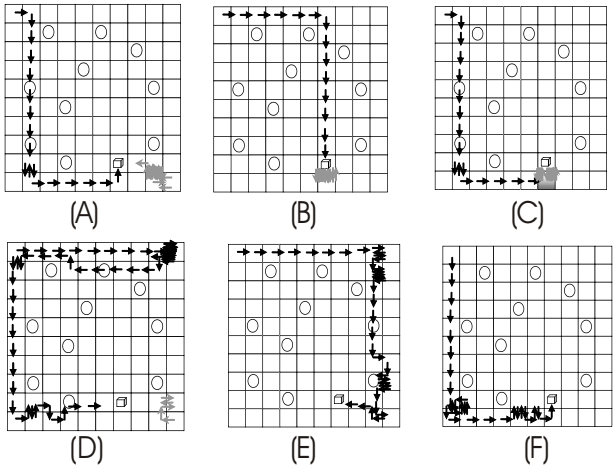

Figure 9: Different Paths Followed by a pair of Agents

neural structure that develops during the agent's lifetime making it capable of achieving the goal. Figure 8 shows that initially, the fitness of the agents varies a lot, but as the number of generation increases the agents become more skillful and the variations in the fitness reduce. The network is organized in such a way that one agents benefits at the expense of the other. There are also some points along the fitness graph when the fitness of both the agents goes down, this occurs when they are both unable to acheive their goals. Often however, even in the very next generation they find a way to achieve their goals. It is interesting to observe that just before generation 700, both agents become reasonably skillful. The first agent obtains gold twice (and on one occasion 3 times), while at the same time the second agent catches the first agent twice (and later three times). This is followed by agents having fluctuating fortunes. Following that, at around generation 1,100 we see both the agents acheiving their goals, with the second agent being the more successful. Further fitness improvements occur at later generations (not shown). The extra difficulty of the task faced by the second agent generally causes it to adopt more complex strategies and it is often more skillful than the first agent (i.e. it catches it more often than the first agent gets the gold).

In further experimental analysis we looked at the behaviour of the agents in detail. To do this we gave the agents an initial health of 300 rather than the 100 (used during evolution). This resulted in the first agent getting the gold five times (see figure 9) while second agent caught it just once. In the six figures A-F in Fig. 9 we see the movements of the two agents. The first agents movements are recorded with black arrows and the second agents by grey arrows. Arrows indicating the direction of movement. Squares 0 and 99 are the homes of the first and second agent respectively. The gold is located on square 86. The circles show the presence of pits. In Fig. 9A both agents begin with randomly assigned initial networks of neurons which become mature networks by running the seven CGP programs. The first agent takes a fairly direct route toward the gold, encountering two pits along the way (Fig. 9A). Note that, although avoiding pits is desirable it is not necessary, as the agent is only weakened. The second agent spends a great deal of time on squares 98 and 88 and moves toward the gold just after the first agent obtains it (and is subsequently, immediately placed back on its home square). After this, (Fig. 9B) interestingly, the first agent now takes a completely different route to the gold (in fact, the minimum path) and avoids all the pits. Meanwhile, the second agent just lurks near the gold, spending all its time on squares 96 and 86. We generally found that the second agent exhibited this kind of behaviour. This is a good strategy for it, as the first agent (if any good) will eventually come close to the gold and this gives the second agent a chance of catching it. On this occasion, the second agent is unlucky and jumps away from the gold at the same time that the first agent obtains it. Fig. 9C shows the first agent following an identical path to the gold as it did at the start. This is very surprising, as its neural network has changed during its lifetime and its experiences. This strongly suggests that it has somehow encoded a map of its environment (and we surmise that encountering pits is very useful, as unlike encountering a breeze the agent doesn't have to make inferences about the location of the pit). However, when it arrives near the gold (square 96) it is attacked by the second agent and is reallocated to its home square. Its subsequent behaviour is interesting (Fig. 9D). It follows a very different, meandering, path to the gold. It spends time alternating between squares 8 and 9, before turning back and arriving home again, only to set off down the left hand side, in the direction of the gold. The behaviour of the second agent is odd. Having been replaced to its home square (99) after attacking the first agent, it moves around briefly in the bottom right four squares before its CGP computational network dies. This illustrates a interesting, but puzzling, phenomenon that we observed with other evolved agents. Often, the agents's CGPCN dies when the health of the agent is a small number and becomes active (with many branches and synapses) when it has a high value of health. This is puzzling, since the health of an agent is not detectable to the agent's brain, as it is never supplied as an input to the CGPCN! In figures 9E and 9F we see the subsequent behaviour of the first agent, where it succesfully

obtains the gold again. The results suggest that the first agent produces an internal map of its environment early in their evolutionary history. It is interesting to note that after the first agent is attacked by the second its brain is strongly affected so that it follows a totally different path. However, when it doesn't find any gold, it returns to its home square and after that tries the same path that led to its attack by the second agent (with some minor variations), eventually finding the gold. In the subsequent tries it tries to take the shortest paths to the gold. Even after the events shown in figure 9F we found that when the first agent is reallocated to its starting position, it again followed a short path to get to the gold, but unfortunately when it reached the gold its network died.

We also examined the behaviour of this agent in a number of other situations. We removed all pits and found that the first agent moved around the environment, aparently at random, and was unable to find the gold. This strongly suggests that the agent uses environmental cues (i.e. pits) to navigate. We also moved the second agent to square 56 (and disabled its ability to move from this square). This lies directly on the path of the first agent in Fig. 9B. We found that the behaviour of the first agent was identical to its previous behaviour except that it was caught by the second agent (on square 56) and didn't avoid that square when it encountered the smell signal on square 46. This shows that this agents network building program has not yet given it a general response of avoiding the second agent. But this affected its network and caused it to follow a totally different path to avoid the second agent. The agents used in these experiments came from 220th generation. At this stage in evolution the useful behaviours have not yet been encoded in the genotype so that the first agent doesn't fully respond to the presence of the second agent and the degree to which it influences its health.

During an agent's lifetime the structural development and activity of the CGPCN changes considerably. We examined the variations in health (Fig. 10) and network morphology during the lives of the two agents whose behaviour was shown in Figure 9. Figures 11 and 12 shows respectively, the variation of numbers of neurons, axon and dendrite branches during the first and second agents life. The healths of the agents fluctuate according to their experiences. The rises shows the number of times the first agent obtains gold. The drops are of two kinds: the smaller drops shows the time when agent was fell into a pit, and larger drop shows when it was attacked by the second agent. The linear decreases are caused by the decrease in health with each step of the agent. The health of both the agents decrease continuously and only rise when they achieve their goal. When the second agent catches the first agent its health is increased 60%. It can be seen how the health of the first agent drops rapidly immediately after it is attacked by the second agent. Shortly after this, rather strangely, all the neurons in the second agent's network dies. Figure 11 and 12 shows some interesting behaviour. Close examination shows that the behaviour of the networks are strongly correlated with the variations in agents' health. We observed this behaviour over different evolutionary runs of the network. The structure of the network follows the rises and drops in the agents' health. This is puzzling since the network only receives the signals from the beneficial and deleterious events, however its structure changes in a way related to the importance (in terms

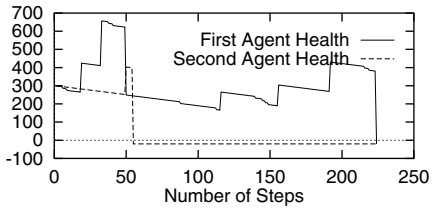

Figure 10: Variation in Agents Health as it makes moves (steps) in its environment

of health) of the signal it receives. In most of the cases once the first agent is caught by the second agent it was never able to get the gold again during its lifetime because the interaction with second agent affected its network by such a lot that it killed some neurons. In order to test the validity of this argument we tested the system by increasing its initial health as demonstrated above in figure 9. It was only through doing this that we found out that in this case the agent was able to get the gold again after being caught by the second agent. It was lucky that the second agent died soon after it caught the first agent. It is evident that the network of the second agent was about to die (see figure 11) before it caught the first agent, but when it caught the first agent it 'boosted' its brain and made it able to live a bit longer. Note, each agent step corresponds to five cycles of the network. We also tested the system by increasing and decreasing the initial healths of both the agents. But as the value of health increases it diminishes the influence of deleterious and beneficial encounters. And as it decreases, it causes the deleterious effects to outweigh the beneficial effects and the agents lose sight of their goals. So an initial life of 100 was considered as the best choice for the evolutionary experiments. Traditional artificial neural networks work on the principle of updating weights to find an aproximate solution to a problem. The static weighted connections hold the learned behaviour. Unfortunately, this can mean that even for a slightly changed problem the network needs to be retrained to maintain a good performance. Whereas in our case, the CGPCN builds the network and any learned behaviour through experience of the environment, so it is, in a sense, self-training. This environmental responsiveness, however comes at a cost. Since the networks are time-dependent it is possible for all neuron components to dwindle away.

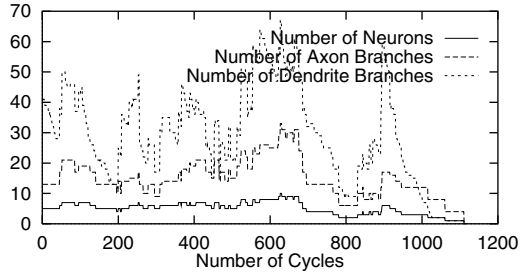

Figure 11: Variation of neurons and neural branches with network update cycles (5 per step) during the first agents life

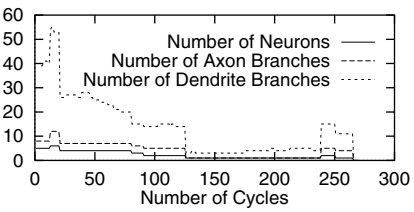

Figure 12: Variation of neurons and neural branches with network update cycles (5 per step) during the second agents life

6. CONCLUSION

We have described co-evolutionary competitive learning environment in which the neuron-inspired computational networks of two antagonistic agents grow and change in response to their behaviour, interactions with each other, and the environment. We found that the agents can learn from their experiences and in some cases appear to build a kind of map of their environment. We used a technique called Cartesian Genetic Programming to encode and evolve seven computational functions inspired by the function of the neuron. In future work, we plan to evaluate this approach in richer and more complicated environments. The eventual aim is to see if it is possible to evolve a general capability for learning.

7. REFERENCES

[1] Koch, C., Segev, I. (2000), "The Role of Single Neurons in Information Processing", Nature Neuroscience Supplement, 3, 1171-1177

[2] Stuart, G., Spruston, N., Hausser, M., eds. (2001), "Dendrites", Oxford University Press.

[3] Koza, J. R. (1992), Genetic Programming: "On the programming of computers by means of natural selection", Genetic Programming II: Automatic Discovery of Reusable Subprograms,(1994), MIT Press

[4] Kleim, J.A., Napper, R.M.A., Swain, R.A., Armstrong, K.E., Jones, T.A., Greenough, W.T.(1998) "Selective synaptic plasticity in the cerebellar cortex of the rat following complex motor learning", Neurobiol. Learn. Mem. 69 , 274-289.

[5] Terje, Lmo.(2003), "The discovery of long-term potentiation", Philos. Trans. Roy. Soc. Lond. B,Biol Sci 358 (1432): 617-20.

[6] Roberts, P.D., Bell, C.C.(2002), "Spike-timing dependent synaptic plasticity in biological systems", Biological Cybernetics, 87, 392-403.

[7] Graham, B., (2002), "Multiple Forms of Activity-Dependent Plasticity Enhance Information Transfer at a Dynamic Synapse", J.R. Dorronsoro (Ed.): ICANN 2002, LNCS 2415, 4550, Springer-Verlag.

[8] Panchev, C., Wermter, S., and Chen, H. (2002), "Spike-Timing Dependent Competitive Learning of Integrate-and-Fire Neurons with Active Dendrites", J.R. Dorronsoro (Ed.): ICANN 2002, LNCS 2415, 896901, Springer.

[9] Miller, J. F., Thomson, P., and Fogarty, T. C. (1997), "Designing electronic circuits using evolutionary algorithms. arithmetic circuits: a case study. *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*", Wiley, 105-131.

[10] Miller, J. F. and Thomson, P. (2000), "Cartesian genetic programming", in Proc. of EuroGP, LNCS, Vol. 1802, 121-132.

[11] Miller, J. F. (2004), "Evolving a self-repairing, self-regulating, French flag organism", in Proc of GECCO, Deb. K. et al (Eds.), LNCS, Vol. 3102, Springer-Verlag, 129-139.

[12] Miller, J. F. (2003), "Evolving Developmental Programs for Adaptation, Morphogenesis and Self-Repair", In Proc Eur. Conf. on Advances in Artif. Life, LNAI, Vol. 2801, 289-298, Springer.

[13] Miller, J. F., Vassilev, V. K., and Job, D. (2000), "Principles in the Evolutionary Design of Digital Circuits-Part I. *Genetic Programming*", 1:1/2, 7-35.

[14] Vassilev, V. K. and Miller, J. F. (2000). "The advantages of landscape neutrality in digital circuit evolution", In Proc. of ICES, LNCS, Vol. 1801, 252-263, Springer.

[15] Yu, T. and Miller, J. (2001), "Neutrality and the evolvability of Boolean function landscape", In Proc. of EuroGP, LNCS, Vol. 2038, 204-217, Springer.

[16] Yu, T. and Miller, J. (2002), "Finding needles in haystacks is not hard with neutrality", In Proc. of the 5th EuroGP, 13-25, Springer-Verlag.

[17] Walker, J. A., and Miller, J. F.(2004), "Evolution and Acquisition of Modules in Cartesian Genetic Programming", Proc. of EuroGP, Vol. 3003, LNCS, 187-197, Springer.

[18] Walker, J. A., Miller, J. F., and Cavill, R.(2006), "A multi-chromosome approach to standard and embedded cartesian genetic programming", In: Proc. of GECCO, Vol. 1, ACM Press, 903-910.

[19] Kandel E. R., Schwartz, J. H., and Jessell (2000), "Principles of Neural Science", McGraw-Hill. 4th Edition, 67-70.

[20] Shepherd, G. M. (1990), "The synaptic organization of the brain", Oxford Press.

[21] Michael S.G., Richard B.I., George R.M. (1998), "Cognitive NeuroScience, The Biology of the Mind", W.W.Norton & Company,

[22] Alberts B., et al. (2002), "Molecular Biology of the Cell", Garland Science, 3rd Edition.

[23] Russell, S., and Norvig, P. (1995), "Artificial Intelligence, A Modern Approach", Prentice Hall.

[24] Stanley, K., and Miikkulainen, R. (2004), "Competitive coevolution through evolutionary complexification", J. of AI Research, 21:63–100.

[25] Paredis, Jan. (1995), "Coevolutionary Computation",Artificial Life, Vol. 2, 4, 355-375.

[26] Hillis, W. (1990), "Co-evolving parasites improve simulated evolution as an optimization procedure", Physica D, 42,228-234.

[27] Marcus G. (2004), "The Birth of the Mind", Basic Books, 165.