A Developmental Model of Neural Computation 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

The brain has long been seen as a powerful analogy from which novel computational techniques could be devised. However, most artificial neural network approaches have ignored the genetic basis of neural functions. In this paper we describe a radically different approach. We have devised a compartmental model of a neuron as a collection of seven chromosomes encoding distinct computational functions representing aspects of real neurons. This model allows neurons, dendrites, and axon branches to grow, die and change while solving a computational problem. This also causes the synaptic morphology to change and affect the information processing. Since the appropriate computational equivalent functions of neural computation are unknown, we have used a form of genetic programming known as Cartesian Genetic Programming (CGP) to obtain these functions. We have evaluated the learning potential of this system in the context of solving a well known agent based learning scenario, known as wumpus world and obtained promising results.

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, Brain, Artificial Neural Networks

1. INTRODUCTION

In this work, a type of developmental brain-inspired computational network is presented and evaluated. It is based on the idea of evolving programs that build a computational neural structure.

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

In spite of the success of Artificial Neural Networks (ANNs), there are many aspects of biological neural systems that have been largely ignored. Marcus argues convincingly about the importance of development in the understanding of the brain, "mechanisms that build brains are just extensions of those that build the body" [32]. Despite this, there are virtually no evolved artificial developmental neural approaches in the research literature. The only work we could find was that of Rust et al who evolved the parameters in rule-based developmental neural model to grow dendritic trees and artificial retina [31], [30] and Federici who evolved developmental spiking neural networks [33].

There is now abundant evidence that sub-processes of neurons are highly time-dependent so that many structures are in a constant state of being re-built and changed [26]. In addition, memory is not a static process and the location and mechanisms responsible for remembered information is in constant (though, largely gradual) change. The act of remembering is a process of reconstructing and changing the original structure that was associated with the original event [27].

The physical topology of the neural structures in the brain is constantly changing and is an integral part of its learning capability [21]. Koch argues that "Dendritic trees enhance computational power" [1]. Dendrites themselves should not be regarded as passive entities that simply collect and pass synaptic inputs to the soma. In most cases they shape and integrate these signals in complex ways[2]. Neurons communicate through synapses. Synapses are not simply the point of connection, they change the strength and shape of the signal either for short time [6], [8] or long time [7]. Thus, the physical architecture of the neuron is important. Inspired by this our network is allowed to change its morphology. The branches can self-prune [4], [5], and can produce new branches to get an optimized network that depend on the complexity of the problem. In the model, a neuron consists of a soma, dendrites [10], and axons with branches and dynamic synapses [9] and synaptic communication. Neurons are placed in a grid to give branches a sense of virtual proximity. Branches are allowed to grow and shrink, and communicate with each other.

The internal dynamics of the neuron is too complicated to be modeled using conventional programming design techniques. However, our view is that the biology of neurons *is* sufficiently well understood [24], [22] so that we can identify essential sub-systems that we must attempt to evolve in order to achieve a computational equivalent.

Here, we used a Genetic Programming (GP) approach to this problem. 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 [13], [14] 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.

In this paper we used Cartesian Genetic Programming [12] to construct the computational network. Each neuron is considered as a computational device with each subprocessing part represented by a chromosome. The genotype of a neuron consists of a collection of chromosomes representing the sub-components of the neuron. The chromosomes are subjected to evolution until the desired intelligent behaviour is obtained.

2. CARTESIAN GP

Cartesian Genetic Programming (CGP) was developed from the work of Miller and Thomson [11, 12] 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 [12, 15, 16, 17, 18] and found to be very 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 [19] 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 CGP in which there are separate chromosomes encoding independent output however sharing modules has been introduced, and shown to improve problem solving ability considerably [20].

3. NEURON MODEL USED

This section describes the neuron model 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 [21]. 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 if the cumulative affect of the inputs is greater than the firing threshold. Neurons have a number of dendrites and a single axon. Each dendrite can have a branching tree-like structure. Axons also 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[23]. 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 the behaviour of neuron in terms of seven processing compartments (see section 4.4):

- Local interaction among the neighbouring branches of the same dendrite.
- Production of new branches, removing old branches, branch growth.
- Processing signals received from dendrites at the 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 old 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 so that they are only aware of their spatial neighbours. 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 also changes 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 electrical processing in dendrites. 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 for a neuron whether to fire an action potential or not. If an action potential is fired 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. So it will provide a sense of real time voltage and make 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 ride of unimportant neurons and branches.

4.2 Evolutionary Strategy

The evolutionary strategy utilised is of the form $1 + \lambda$, with λ set to 4 [17], 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 two or more are equally good then newest(genetically) is chosen[15].

4.3 CGP Chromosome

The CGP function nodes used here consists of multiplexer-like operations [15]. The number of inputs per node is three. 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.

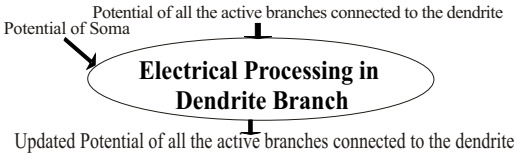

Figure 1: Electrical processing in the dendrite

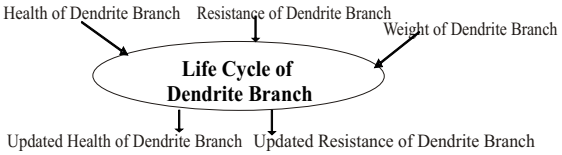

Figure 2: Life cycle of dendrite branch

4.4 CGP Model of Neuron

This model is inspired from the biological neuron. It 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.

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.

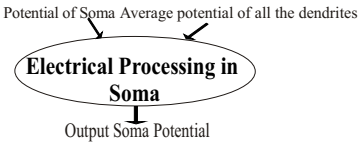

Figure 3: Electrical processing in soma

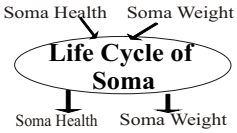

Figure 4: Soma life cycle

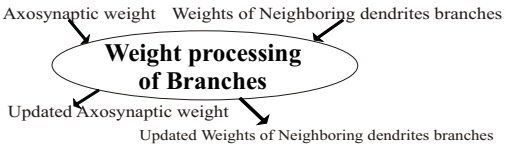

Figure 6: Axo-synaptic branch weight processing

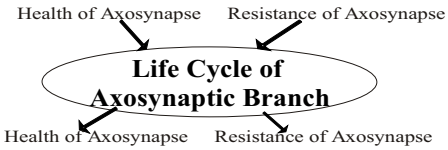

Figure 7: Axo-synaptic branch life cycle

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 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.

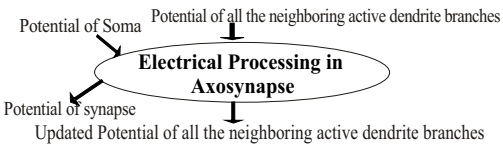

Figure 5: Electrical processing in axosynaptic branch

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.

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. These branches are part of input neurons, which are virtual neurons. They take the input from the environment and transfer it through virtual axosynapse with out processing it. 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 Wumpus World

The Wumpus World is an agent-based learning problem used as a testbed for various learning techniques in Artificial Intelligence [25]. It consists of a two dimensional grid containing a number of pits, a wumpus (monster) and an agent[25]. The agent always starts from a unique square (home) in a corner of the grid. The agent's task is to avoid the wumpus, find the gold, return to home. The agent can perceive a breeze in squares adjacent to the pits and a stench in the squares adjacent to the wumpus.

In most environments there is a way for the agent to safely retrieve the gold. In some environments, the agent must choose between going home empty-handed, taking a chance that could lead to death, or finding the gold. In our experiments we have slightly adapted the rules of the wumpus world. Namely, the agent encountering pits or wumpus is only weakened (thus reducing its life), it is not killed. These changes are introduced to facilitate the capacity of the agent to learn. The CGPCN learns everything from scratch and builds its own memory (including the meaning of signals, pits and the wumpus).

It is important to appreciate how difficult this problem is. The agents starts with a few neurons with random connections. So, firstly, evolution must find a series of programs that build a computational network that is stable (doesn't lose all neurons or branches etc.). Secondly, it must find a way of processing infrequent environmental signals. Thirdly, it must navigate in this environment using some form of memory. Fourthly, it must confer goal-driven behaviour on the agent. This makes the wumpus world problem a challenging problem.

The wumpus world used here is a two dimensional grid (10x10), having ten pits, one wumpus and the gold. The location of wumpus, gold and pits is chosen randomly. In the square containing wumpus, gold, pits, and in directly (not diagonally) adjacent square the agent can perceive stench, glitter (near gold) and breeze respectively. Also we have arranged it so that the agent will receive input signals of different magnitudes depending on the direction that the agent perceives the signal. The agent detects that it is on the home square via a special input signal. All the locations which are safe will provide no signal. The agent is assigned an initial life of 200 units. If agent is caught by wumpus its life is reduced by 60%, if caught by a pit its life is reduced by 10 units, if it gets the gold its life is increased by 50 units, on arriving home its life is over. For each single move the agent's life is reduced by 1 unit. The fitness of an agent is accumulated (while its life is greater than zero) during its lifetime in the following way:

- For each move, the fitness of the agent is increased by one.
- If the agent returns home without the gold, its fitness is increased by 200.
- If the agent obtains the gold, its fitness is increased by 1000.
- If the agent returns home with the gold, its fitness is increased by 2000.

When the experiment starts, the agent takes its input from the grid square. This input is applied to the computational network of the agent through virtual axosynapses. The network is then run for five cycles. During this process it updates the potentials of the virtual dendrite branches which act as output of the system. After the above process the updated potentials of virtual dendrite branches are noted and averaged. The value of this average potential decides the direction of movement for the agent. The same process is then repeated for the next grid square. The agent is terminated if either its life become zero, or all its neurons die, or all the dendrite or axon branches die, or if the agent return home.

The network is tested on five different genotypes to produce different agent behaviors. The best agent genotype is selected as the parent for a new population.

5.2 CGPCN Setup

The CGPCN is arranged in the following manner for this experiment. The network 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

The agent performance is determind by its capability to solve three kinds of tasks. The first task for the agent is to learn how to come back to home, the second task is to find the gold, and the third and the final task is to bring the gold back to home. During this process the agent has to avoid pits and wumpus in order to increase its life span. It also needs to sustain the neural network to solve these problems. Performances of the agents from independent evolutionary runs with different initial population, but with the same wumpus world is given in Table 1.

The first column of table 1 gives the run number. The second column shows the number of generation taken by the agent to learn how to come back to home, the third column shows the number of generations taken to find the gold and the last column shows the number of generation taken by the agent to bring the gold back to home. Table 2 shows the performance of different evolutionary runs starting from the same initial population but where each run is evaluated on a different Wumpus World.

The fitness function was devised in such a way that it initially forces the agent to sustain its network and so increase its life span. So in the initial generations evolution tries to build a stable and sustainable computational network. Once this is acheived, evolution starts to produce agents that firstly learn how to come back home, then learn how to find the gold and finally they bring the gold back home. This is evident from results in table 1 and 2.

Run	Home without Gold	Gold obtained	Home with Gold
1	-	5	12
2	5	14	267
3	1	95	120
4	-	3	649
5	-	3	119
6	-	7	186
7	1	8	88
8	-	2	14
9	15	18	304
10	9	14	987
11	-	2	93
12	5	19	762
13	-	3	256
14	-	3	380
15	5	10	280
16	3	4	1108
17	2	20	674
18	1	20	98
19	2	5	35
20	3	6	98

Table 1: The number of generations taken to perform various tasks in a fixed Wumpus World in independent evolutionary runs.

At the start, the agent does not know anything about gold, home, wumpus, or pits and the signals that indicate the presence of these objects nearby. As the system is evolved the genetic code in the agent allows a computational structure to be built that holds a memory of the meaning of these signals during the agents life cycle. This knowledge is built from the initial genetic code when it is run on the initial randomly wired network. When different runs of the experiment are examined, it is found that in all cases the agent learns to avoid the wumpus and pits and tries to get the gold.

Conventional artificial neural networks work on the principle of updating weights to aproximate the solution to a problem. So for even a slightly changed nature of problem you need to retrain the network to find the new weights which allow the changed problem to be solved. Whereas, in the case of CGPCN, we are evolving programs that build and change continuously a computational network within an external environment. Naturally, we are interested in whether these evolved programs could build a network that could lead to successful agent behaviour in different wumpus worlds (i.e. behaviour that is general and not specific to a particular wumpus world). To test this we took the programs for the best evolved agent on a particular Wumpus World and tested its performance on other wumpus worlds that had been generated at random. The performance of this agent is presented in Table 3.

Wumpus World	Home	Gold	Home and Gold
1	-	5	12
2	11	28	49
3	-	6	1079
4	7	79	1051
5	-	9	1769
6	3	6	161
7	3	4	379
8	2	20	92
9	-	2	738
10	2	41	203

Table 2: The number of generations taken by the evolved computational networks to perform the tasks on different Wumpus Worlds starting from the with same initial population.

Wumpus World	Fitness
1	3103*
2	256
3	86
4	70
5	3051*
6	258
7	1068
8	1068
9	1170
10	1169

Table 3: The fitness achieved by the best evolved agent on different wumpus world environments

The first column gives the wumpus world identifier. The second column shows the agent fitness on each wumpus world. As explained earlier the fitness shows whether the agent came back to home without gold, with the gold, or whether it got the gold but did not go home. If the fitness of agent is above 3000 it shows that it got the gold and brought it to home. If the agent fitness is above 1000 it means that it got the gold, and if the fitness of the agent is above 200 it means that it came back to home without any gold. From Table 3 it is evident that in two of the cases when the gold is placed at the same location, but pits and wumpus are located at different location, the agent was able to get the gold and bring it to home (marked with an asterisk). In other cases, sometimes the agent gets the gold but is unable to find its way to home, and sometimes it cannot find the gold and returns home empty handed. There are also cases when the agent cannot find the gold or get home. Further observations during the experiment revealed that the irrespective of the new environment the agent always first looks for the gold at the place where the gold was when it was evolved. This is very interesting as it shows that the evolved genetic codes are able to build a computational network that holds information (how to find the gold).

5.4 Development of network over the lifetime of the agent

While solving wumpus world the CGPCN changes sunstantially in its structure both in term of presence and absence of neurons and branches and whether they are active or inactive. Figure 8 shows the variation in the life of the agent while it is solving a task (getting gold and bringing it back home). Close examination of this reveals that there are mainly slow continuous changes but also there are sudden changes in the agent's life. If the agent does not acheive any tasks its life continually decreases step by step. When it falls into a pit its life is decreased by ten units. There is a large

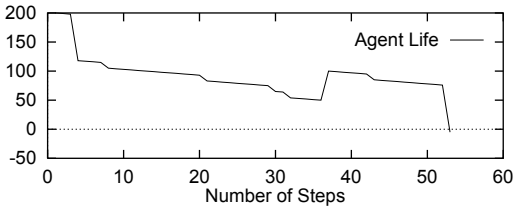

Figure 8: Agent life

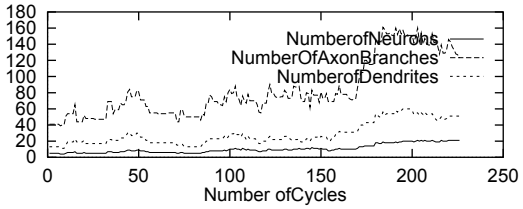

Figure 9: Agent Network Statistics

decrease in life when the agent encounters the wumpus. In this particular case the agent falls into a pit five times and meets the wumpus once. When the agent gets the gold one can see a sudden increase in its life (by 50units). Finally when agent return home its life is terminated as shown in the righthand corner of the graph (Figure 8). This particular case is selected just for demonstration. Figures 9 shows the variation in number of neurons, dendrite branches and axon branches during this agent's life. Figures 10 shows the variation in the number of active neurons, dendrite branches and axon branches at different stages of life of the agent. Figures 9 and 10 shows interesting network dynamics. The network dynamics is quite random at the start, but later it stabilizes. Once a sustainable network is obtained then the agent tries to find its goal. The agent take less time to get to its second goal (home) than the first goal (gold), because now it has a sustainable network and all it needs to do is to get the gold back to home. In a way a single run of network shows all the learning that is done during evolution. e.g. getting a sustainable network and solving the problem. Also, a study of the network behaviour over a number of evolutionary runs, reveals that the agents avoid the rest of the environment and go directly to the place where the gold is located. This is reminiscent of instinctive behaviour.

6. CONCLUSION

We have presented a method for evolving programs that construct a dynamic computatational network inspired by the biological brain. We have evaluated this approach on a classic AI problem called wumpus world. Results are very promising and indicate that this system might be capable of developing an ability to learn continuously within a task environment. We found that a network tested on a different wumpus world preserves the sustainability of the network and the avoidance of pits and the wumpus. It is not possible, at present to compare directly the effectivness of this approach with other artificial neural networks as others have not worked in the wumpus world scenario.

In future work, we plan to examine whether the dynamic computational network is able to solve problems faster and more accurately merely by obtaining repeated experience of its task environment (post evolution). Also, we plan to investigate whether the computational networks can be trained to solve a sequence of problems without forgetting how to solve earlier problems (conventional artificial neural networks suffer from a phenomenon known as 'catastrophic forgetting' [28][29])

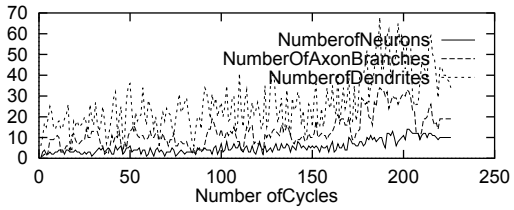

Figure 10: Agent Network Activity

7. REFERENCES

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

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

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

[4] Van Ooyen A., and J. van Pelt. (1994), "Activity-dependent outgrowth of neurons and overshoot phenomena in developing neural networks", Journal of Theoretical Biology vol. 167, pp. 27-43.

[5] Becerra, J.A., Duro, R.J., Santos, J., (2002), "Self pruning gaussian synapse networks for behaviour based robots", J.R. Dorronsoro (Ed.): ICANN 2002, LNCS 2415, pp. 837843, Springer-Verlag Berlin Heidelberg.

[6] 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: pp. 274-289.

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

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

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

[10] 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, pp. 896901, Springer-Verlag Berlin Heidelberg.

[11] 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, pages 105–131.

[12] Miller, J. F. and Thomson, P. (2000), "Cartesian genetic programming", in Proc. of the 3rd European Conf. on Genetic Programming. LNCS, Vol. 1802, pp.121-132.

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

[14] Miller, J. F. (2003), "Evolving Developmental Programs for Adaptation, Morphogenesis and Self-Repair", In: Proceedings of the 7th European Conference on Advances in Artificial Life. LNAI, Vol. 2801, pp. 289-298.

[15] 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, pages 7 –35.

[16] Vassilev, V. K. and Miller, J. F. (2000). The advantages of landscape neutrality in digital circuit evolution. In *Proceedings of the Third International Conference on Evolvable Systems: From Biology to Hardware*, pages 252-263, Springer-Verlag, Vol. 1801.

[17] Yu, T. and Miller, J. (2001). Neutrality and the evolvability of Boolean function landscape. In *Proceedings of the Fourth European Conference on Genetic Programming*, pages 204–217, Springer-Verlag.

[18] Yu, T. and Miller, J. (2002). Finding needles in haystacks is not hard with neutrality. In *Proceedings of the Fifth European Conference on Genetic Programming*, pages 13–25, Springer-Verlag.

[19] Walker, J. A., and Miller, J. F.(2004), "Evolution and Acquisition of Modules in Cartesian Genetic Programming", Proc. of the 7th EuroGP, Vol. 3003, LNCS, pages: 187-197, Springer.

[20] Walker, J. A., Miller, J. F., and Cavill, R.(2006), "A multi-chromosome approach to standard and embedded cartesian genetic programming", In: Proceedings of the 8th annual conference on Genetic and evolutionary computation, Vol. 1, ACM Press, pages: 903-910.

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

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

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

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

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

[26] Smythies, J.R.,(2002), "The Dynamic Neuron", Bradford.

[27] Rose, S.,(2003), "The Making of Memory: From Molecules to Mind", Vintage.

[28] McCloskey, M., and Cohen, N.J.,(1989), "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem", The Psychology of Learning and Motivation, Vol. 24, pages: 109-165

[29] Ratcliff, R. (1990), "Connectionist Models of Recognition and Memory:Constraints Imposed by Learning and Forgetting Functions", Psychological Review, Vol. 97, pages: 205-308

[30] Rust, A., Adams, R., George, S., and Bolouri, H. (1997), "Designing development rules for artificial evolution", In Proceedings of 3rd International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA97).

[31] Rust, A., Adams, R., George, S., and Bolouri, H. (1996), "Artificial evolution: Modelling the development of the retina", Technical Memorandum ERDC/1996/0015, ERDC, University of Hertfordshire, UK, September 1996.

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

[33] Federici D. (2005), "Evolving Developing Spiking Neural Networks", In Proceedings of CEC 2005 IEEE Congress on Evolutionary Computation pages: 543-550