Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

Evolution of Cartesian Genetic Programs for Development of Learning Neural Architecture

Gul Muhammad Khan gk502@nwfpuet.edu.pk
Electrical Engineering Department, NWFP UET Peshawar
Pakistan

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

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

Abstract

Although artificial neural networks have taken their inspiration from natural neurological systems they have largely ignored the *genetic* basis of neural functions. Indeed, evolutionary approaches have mainly assumed that neural learning is associated with the adjustment of synaptic weights. The goal of this paper is to use evolutionary approaches to find suitable computational functions that are analogous to natural subcomponents of biological neurons and demonstrate that intelligent behaviour can be produced as a result of this additional biological plausibility. Our model allows neurons, dendrites, and axon branches to grow or die so that synaptic morphology can change and affect information processing while solving a computational problem. The compartmental model of neuron consists of a collection of seven chromosomes encoding distinct computational functions inside neuron. Since the equivalent computational functions of neural components are very complex and in some cases unknown, we have used a form of genetic programming known as Cartesian Genetic Programming (CGP) to obtain these functions. We start with a small random network of soma, dendrites, and neurites that develops during problem solving by executing repeatedly the seven chromosomal programs that have been found by evolution. We have evaluated the learning potential of this system in the context of a well known single agent learning problem, known as Wumpus World. We also examined the harder problem of learning in a competitive environment for two antagonistic agents, in which both agents are controlled by independent CGP Computational Networks (CGPCN). Our results show that the agents exhibit interesting learning capabilities.

Keywords

Cartesian Genetic Programming (CGP), Artificial Neural Networks, Co-evolution, Generative and developmental approaches, Learning and memory.

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. In spite of the success of Artificial Neural Networks (ANNs), there are many aspects of biological neural systems that have been largely ignored. Clearly the developmental processes that construct the brain ought to be important for

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

an understanding of the brain, "mechanisms that build brains are just extensions of those that build the body" (Marcus (2004)). Despite this, there are very few evolved artificial developmental neural approaches in the research literature.

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 (Smythies (2002)). In addition, memory is not a static process (time invariant) and the location and mechanisms responsible for remembered information are in constant (though, largely gradual) change. Rose argues that the act of remembering is a process of reconstructing and changing the original structure that was associated with the original event (Rose (2003)). The physical topology of the neural structures in the brain is constantly changing and is an integral part of its learning capability (Kandel et al. (2000)) see pages 67-70). "Dendritic trees enhance computational power" and dendrites themselves should not be regarded as passive entities that simply collect and pass synaptic inputs to the soma (Koch and Segev (2000)). In most cases they shape and integrate these signals in complex ways (Stuart et al. (2001)). Neurons communicate through synapses. Synapses are not merely the point of connection, they can change the strength and shape of the signal either over short time scales (Kleim et al. (1998), Roberts and Bell (2002)) or long time scales (Terje (2003)). It is clear then that the physical architecture of the neuron is important. Inspired by this, we have developed an approach that allows the computational network to change its own morphology. Neurite branches can self-prune (Van Ooyen and Pelt (1994), Becerra et al. (2002)), and can produce new branches to arrive at a network whose structure and complexity is related to properties of the learning problem. In the model, a neuron consists of a soma, dendrites (Panchev et al. (2002)), axons with branches and dynamic synapses (Graham (2002)) and synaptic communication. Neurons are placed in a two dimensional grid to give branches a sense of virtual proximity. Branches are allowed to grow and shrink, and communication between axon branches and dendritic branches is allowed.

One of the difficulties in attempting to create a dynamic computational model inspired by neuroscience is that the internal dynamics of biological neurons are too complicated to be modeled with 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 to allow us to identify essential sub-systems that we must attempt to evolve in order to achieve a computational equivalent (Alberts et al. (2002), Shepherd (1990)). Conventional models of neural networks do not consider the genetics of neurons and the biological development of a mature network during 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. Thus, we argue that GP, in principle, provides us with a means to represent complex neuron 'engines' that can be evolved to exhibit computational analogues of real neural systems, without the restrictions imposed by requiring a theoretical model of these systems. Indeed, GP has been shown to be able to solve problems of this type (see Koza (1992)) and often these solutions are not fragile and show unexpected emergent behaviours such as self-assembly and self-repairability reminiscent of biological systems (Miller (2004)).

We have used a form of GP known as Cartesian Genetic Programming to construct our computational networks (Miller and Thomson (2000)). Each neuron is considered as a computational device with each sub-processing part represented by a chromosome. The genotype of a neuron consists of a collection of chromosomes representing the sub-components of the neuron. We evolve the programs encoded in the chromosomes until

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

the network that they construct exhibits the desired intelligent behaviour.

The capabilities of the networks for learning is tested on a well known artificial intelligence problem known as Wumpus World (Russell and Norvig (1995)). The application of genetic programming to Wumpus World has already been considered (Spector (1996); Spector and Luke (1996)).

It is worth mentioning that although we have tested the system in an artificial environment inspired by Wumpus World, the goal of the system is to evolve the capability of learning in general. We show that our networks build memory and intelligent behaviour in their lifetime. This approach has not been explored by others so we cannot directly compare our system with others (even those operating on Wumpus World). We show that the ability to build memory and exhibit intelligent behaviour is somehow encoded in the evolved programs and can be transmitted from generation to generation.

In addition, we examined and adapted the Wumpus World problem as a coevolutionary competitive learning environment in which two agents (predator and prey) struggle to achieve tasks and survive (Hillis (1990)). Each agent is independently controlled by an evolved CGP Computational Network. We allowed agents to have a quantity analogous to energy, which is increased or decreased as a consequence of their actions and encounters in the environment. The first agent increases its energy level (and consequently its fitness) by avoiding deleterious encounters, and attaining beneficial encounters. The opposing agent increases its energy level solely by direct antagonistic encounters with the first agent (which to it are beneficial). Paredis describes this type of fitness as 'life-time fitness evaluation' and discusses how this 'arms race' provides a strong driving force toward complexity (Paredis (1995)).

Nolfi and Floreano adopted this approach by co-evolving two competing populations of predator and prey robots in order to explore how lifetime learning allows evolved individuals to achieve behavioural generality, i.e. the ability to produce effective behavior in a variety of different circumstances (Nolfi and Floreano (1998)). Since both populations change across generations, predators and prey face ever-changing and potentially progressively more complex challenges.

Stanley and Miikkulainen's approach to achieve complexification is through the incremental elaboration of solutions by adding new neural structures (Stanley and Miikkulainen (2002),Stanley and Miikkulainen (2004)). They start with a simple neural structure and through a managed approach, develop it to produce a complex neural structure. In our model the computational network is able to 'complexify' itself. This was inspired by the fact that the brain complexifies itself without any change in genetic code.

Learning in brains occurs after evolution has finished (i.e. in the lifetime of the individual). In other words, evolution has created self-learning programs. These are programs which when run construct a system that is capable of learning by experience. We aim to do the same. We evolve programs which when run, continuously build and change a neural architecture in response to environmental interaction. We emphasize that no evolution takes place in the lifetime of the individual while it is learning.

The motivation behind this work is to explore the importance of neural development, neural architecture, physical proximity and the genetic code inside neurons in obtaining a capability for learning. The major contribution of this work is to demonstrate that a continuously developing system can learn and survive in a dynamic environment and its 'capability of learning' can be transferred from generation to generation through the genotype. In fact, if we are able to develop genotypes that develop into phenotypes capable of learning it will be an advance in the field of artificial intelligence

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

and machine learning.

Section 2 will provide some background studies and motivation for the model. Section 3 will discuss the key features and biological inspiration of the model. In section 4, we will explain the overall model of CGPCN. Section 5 will provide some analysis and results of the model tested in the Wumpus World environment. And finally section 6 will give some concluding remarks.

2 Background and Motivation

2.1 Cartesian Genetic Programming (CGP)

Cartesian Genetic Programming was developed from the work of Miller and Thomson for the evolutionary design of feed forward digital circuits (Miller et al. (1997); Miller and Thomson (2000)). 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 work). 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 and found to be extremely beneficial to the evolutionary process on the problems studied (Miller et al. (2000); Vassilev and Miller (2000); Yu and Miller (2001)). Indeed, it has been shown that evolutionary search proceeds most quickly when extraordinary levels of redundancy are present (i.e. when 95% of all genes are redundant), (Miller and Smith (2006)).

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 (its arity).

2.2 Neural development

Artificial neural networks are intended to mimic, in some sense, the computation seen in nervous systems. However many ANN models ignore the fact that the neurons in the nervous system are part of the phenotype which is derived from the genotype through a process called development (Kumar (2003)). The information specified in the genotype determines some aspects of nervous system, which specifies the rules for developing the nervous system based on environmental interaction during developmental phase. Natural organisms, however, not only posses nervous systems but also genetic information stored in the nucleus of their cells (genotype).

Development schemes are intended to increase the scalability of ANNs by having a minimum number of genes that define the properties of the network, instead of having a one to one relationship between genotype and phenotype.

Nolfi et al presented a model in which the genotype-phenotype mapping (i.e. ontogeny) takes place during the individual's lifetime and is influenced both by the geno-

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

type and by the external environment (Nolfi et al. (1994)). The two-dimensional neural networks adapt during their lifetime to different environments. The neurons had no dendrites, only upward growing axons. Connections between neurons happen when axons arrive in the vicinity of another neuron. The activity of a neuron affects whether its axon grows or not, in this way they linked lifetime 'electrical' activity with morphological structure.

Parisi and Nolfi argued that, if neural networks are viewed in the biological context of artificial life, they should be accompanied by genotypes which are part of a population and inherited from parents to offspring (Parisi (1997); Parisi and Nolfi (2001)). They produced a method of developing a genotype into traditional artificial neural network as the phenotype. Traditional neural networks are inspired by the brain in terms of its connectionism. They ignore the structural layout, their physical arrangement and the processing inside and between biological neurons.

Cangelosi proposed a similar neural development model, which starts with a single cell undergoing a process of cell division and migration (Cangelosi et al. (1994)). This cell division and migration continues until a collection of neurons arranged in two-dimensional space is developed. These neurons grow their axons to produce connections among each other to develop a neural network. The rules for cell division and migration are specified in genotype, for related approaches see (Dalaert and Beer (1994); Gruau (1994)). Again the ultimate goal of his work was to develop a traditional artificial neural network as phenotype from a small genotype.

Rust and Adams devised a developmental model coupled with a genetic algorithm to evolve parameters that grow into artificial neurons with biologically-realistic morphologies (Rust et al. (2000); Rust and Adams (1999)). They also investigated activity dependent mechanisms (Rust et al. (1997)) so that neural activity would influence growing morphologies. Although Rust and Adams showed that the technique was able to produce realistic and activity dependent morphology, they did not investigate the networks carrying out a function.

Quartz and Sejnowski have laid down a powerful manifesto for the importance of dynamic neural growth mechanisms in cognitive development (Quartz and Sejnowski (1997)) and Marcus has emphasized the importance of growing neural structures using a developmental approach "I want to build neural networks that grow, networks that show a good degree of self-organization even in the absence of experience" (Marcus (2001)).

Jakobi created an artificial genomic regulatory network (Jakobi (1995)). He used proteins to define neurons with excitatory or inhibitory dendrites. The individual cell divides and moves due to protein interactions with an artificial genome, causing a complete multicellular network to develop. After differentiation each cell grows dendrites following chemical sensitive growth cones to form connections between cells. This develops a complete conventional recurrent ANN, which is used to control a simulated Khepera robot for obstacle avoidance and corridor following.

Federici presented an indirect encoding scheme for development of a neuro-controller (Federici (2005)). The adaptive rules used were based on the correlation between post-synaptic electric activity and the local concentration of synaptic activity and refractory chemicals. Federici used two steps to produce the neuro-controllers: a growth program (implemented as a simple recurrent neural network) in a genotype to develop the whole multi-cellular network in the form of a phenotype and a translation step where cells are interpreted as spiking neurons.

Roggen et al. devised a hardware cellular model of developmental spiking ANNs

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

(Roggen et al. (2007)). Each cell can hold one of two types of fixed input weight neurons, excitatory or inhibitory each with one of 5 fixed possible connection arrangements to neighbouring neurons. In addition, each neuron has a fixed weight external connection. The neuron integrates the weighted input signals and when it exceeds a certain membrane threshold it fires. This is followed by a short refractory period. They have a leakage which decrements membrane potentials over time.

A number of researchers have studied the potential of Lindenmeyer systems (Lindenmeyer (1968)) for developing artificial neural networks and generative design. Boers and Kuiper have adapted L-systems to develop the architecture of artificial neural networks (ANNs) (numbers of neurons and their connections) (Boers and Kuiper (1992)). They evolved the rules of an L-system that generated feed-forward neural networks. They found that this method produced more modular neural networks that performed better than networks with a predefined structure. Hornby and Pollack evolved L-systems to construct complex three-dimensional robot morphologies and neural controllers (Hornby et al. (2003)). They showed that evolved generative designs had high degrees of modularity, regularity and hierarchy when compared to those evolved without a generative encoding. Also, the generative representation had significantly improved rates of progress at early stages of evolved design compared with directly evolved representations.

Deep belief nets present a probabalistic generative model composed of multiple layers of stochastic and latent variables to reduce the dimensionality of networks for high dimensional data sets (images). They consist of a layer of visible units that represent the data and a layer or layers of hidden units that learn features that capture higher-order correlations in the data. Deep belief nets are extensively used for generating and recognizing images, video sequences, and motion-capture data (Hinton and Osindero (2006); Sutskever and Hinton (2006); Taylor et al. (2006)). Although generative it is not a developmental approach as the network does not have a lifetime, also the architecture used is that of standard Artificial Neural Network with nodes and connections. It is generative in the sense that it is trained at various stages and layers.

Irel further enhances the algorithm by using an approach called Deep Spatio-Temporal Interference Network (DeSTIN). It is a scalable deep learning framework for capturing spatiotemporal dependencies in high dimensional data sets (Arel and Karnowski (2009)). It uses a state variable called the belief state that inherently captures both spatial and temporal information. Each node is allocated a predefined number of belief state variables that change in response to the stimuli thus making the network dynamic.

George and Hawkins presented a Hierarchical Bayesian model inspired by the architecture of visual cortex for invariant visual patterns recognition (George and Hawkins (2005)). This system was biologically more plausible than the earlier techniques and was tested in a range of scenarios exhibiting invariance across a range of transformations and robustness in the presence of noise.

Yamauchi and Beer have investigated as to whether a dynamic neural network provide an effective control mechanism that integrates the reactive, sequential and learning behavior in autonomous agents (Yamauchi and Beer (1994)). They have applied their approach to three different types of tasks including: Landmark recognition, one-dimensional navigation and sequence learning. They have evolved various parameters of continuous time recurrent neural network using a direct encoding scheme. As pointed out by the authors the primary difficulty with this approach is how it scales with increasingly complex problems, as in direct encodings, the size of genotype in-

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

creases in proportion to the size of phenotype. Increases in the size of genotypes means larger search spaces thus potentially increasing the cost in terms of time to obtain the desired results. The uniqueness of their algorithm lies in the fact that the agent controlled by this network can not only react to its environment, but also integrate information over time to modify its future behavior.

Floreano et al. explored behavioral robustness using the idea of synaptic plasticity (Urzelai and Floreano (2001); Blynel and Floreano (2002)). They evolved neurocontrollers that solve a light-switching task with no reward mechanism and subsequently investigated these controllers for their learning ability by providing four different types of environmental changes to them. These changes include: new sensory appearances, transfer from simulations to physical robots, transfer across different robotic platforms and re-arrangement of environmental layout. The plastic ANNs were able to overcome these four kinds of changes, contrary to a classic ANN with fixed weights. Floreano and Urzelai pointed out that evolving networks with synaptic plasticity can solve complex problems better than recurrent networks with fixed-weights (Floreano and Urzelai (2000)). Performance of plastic and recurrent networks have been tested in various domains giving mixed results with either of them performing better (for a review see Sebastian Risi and Stanley (2010)).

Soltoggio et al. suggested using a heterosynaptic rule to evolve plastic ANNs in a simple dynamic, reward based scenario (T-maze environment) (Soltoggio et al. (2008)). The reward is regularly swapped among the two ends of T-maze to modify the environmental condition thus making it dynamic. They have designed the fitness function to select the individuals that manage to adapt to the changing environment quickly. Although enabling heterosynaptic plasticity improved the performance of evolved controllers in the particular case studied, the learning capabilities of the proposed system were not investigated in general (i.e. over a wide range of conditions and problems).

Sebastian and Stanley extended the HyperNEAT encoding to evolve both synaptic weights and parameters of learning rules (Sebastian Risi and Stanley (2010)). They tested their method on the T-maze experiments as well as a foraging bee scenario. In both cases, they change the reward from time to time thus modifying the environmental conditions. Although they too used synaptic plasticity they found simple fitness demonstrated poor performance. They introduced the idea of selecting *novel* solutions rather than by fitness alone and showed that it enhanced the learning capability of the agents (they refer to this as *novelty search*).

Karl Sims used a graph based GP approach to evolve virtual robotic creatures. The morphology of these creatures and the neural systems for controlling the physical movement of their component body parts were both genetically determined (Sims (1994)). The genotypes were structured as directed graphs of nodes and connections. When a creature is synthesized from its genetic description, the neural components described within each part are generated along with the morphological structure.

Downing favours a higher abstraction level in neural development to avoid the complexities of axonal and dendritic growth while maintaining key aspects of cell signaling, competition and cooperation of neural topologies in nature (Downing (2007)). He developed a system and tested it on a simple movement control problem known as *starfish*. The task for the k-limbed animate is to move away from its starting point as far as possible in a limited time, producing encouraging preliminary results.

From the above discussion it is clear that all the techniques applied so far have tried to develop an artificial neural network which is a connectionist system. Genotypes were mostly used either to specify the rules of the development or the characteristics

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

of a specific network. The internal operations of each node was specified rather than evolved. Also the connections between neurons are considered as simple wires instead of complicated synaptic processes. Apart from the Cangelosi model, the rest of the models are developed before being applied, so the developmental process stops once the evaluation is started. This means that there is no development in real time. The model proposed by Cangelosi is notable in that the networks change their morphology in response to the number of sensors and the environmental activities.

The model we are proposing has taken its inspiration from biology. Our genotype defines the electrical and developmental processing inside neurons. The architecture we provided to our neuron has branching dendrites and axons. Neurons communicate with each other based on virtual proximity through a complex synaptic process defined in genotype. The genotype is evolved from generation to generation and the phenotype is developed from an initial small random structure during problem solving. The network continues to develop during runtime with neurons and branches growing and dying. The network increases and decreases its complexity based on the environmental conditions. The rules for development of network are not specified, they are evolved to get the desired results.

The next section will provide the key features of the model and its comparison with other models.

3 Key features and biological basis for the CGPCN model

This section draws together the key ideas from biology, Artificial Neural Networks (ANNs), neural development, and describes the main features of CGPCN model, along with the biological basis of these ideas. Table 1 lists all the properties of biological systems that are incorporated into CGPCN. Table 1 also shows the presence and absence of these properties in existing ANNs and neural development models(Kumar (2003).

Table 1 list the important properties of both biological and artificial neural systems. We discuss the meaning of each property below.

Neuron Structure: Biological neurons have dendrites and axon with branches. Each neuron has a single axon with a variable number of axon branches. It also has a variable number of dendrites and dendrite branches. Some published developmental neural models have these features. In our model, neurons have three key morphological components (see section 4.5):

-Dendrites with branches that receive and process inputs.

-A cell body which processes signals from dendrites.

-An axon which transfers signals to other neurons through axon branches.

ANNs have only nodes and connections, there is no concept of branches.

Interaction of Branches: In biological dendrites the signals from different branches actually interact with each other, whereas in ANNs and published neural development there is no concept of interaction between connections. We have adopted this feature and evolved the function for interaction between branches, as no precise mathematical model is available to approximate this function (see section 4.5 and also the appendix 3.11.1.3.1.). Although ANNs and neural development sometimes regard connections between neurons as dendritic, they are far from biological dendrites in the types of morphology that can exist (Kandel et al. (2000)).

Resistance: Branches in biological neurons have the property of electrical resistance. Resistance not only affects the strength of the signal but is also related to the length of the branch. There is no concept of branch resistance in the ANN literature, but there is a concept of length in some of the neural developmental models. In our model

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

Name	**ANNs**	**Neural development**	**Biology**	**CGPCN**
Neuron Structure	Node with connections	Node with axons and dendrites	Soma with dendrites, axon and dendrite branches	Soma with dendrites, axon and dendrite branches
Interaction of branches	No	No	Yes	Yes
Resistance	No	Yes/No	Yes	Yes
Health	No	No	Yes	Yes
Neural Activity	No	No	Yes	Yes
Synaptic Communication	No	No	Yes	Yes
Arrangement of Neurons	Fixed	Fixed	Arranged in space (Dynamic Morphology)	Arranged in Artificial space (Dynamic Morphology)
Spiking (Information processing)	Yes, but not all	Yes, but not all	Yes	Yes
Synaptic Plasticity	Yes	No	Yes	Yes
Developmental Plasticity	Yes	No	Yes	Yes
Arbitrary I/O	No	No	Yes	Yes
Learning Rule	Specified	Specified	Unspecified	Unspecified
Activity Dependent Morphology	No	Some	Yes	Yes

Table 1: List of all the properties of biological systems that are incorporated into CGPCN or are present in ANNs and neural development models.

neurite (dendrite or axon) branches have a quantity which we call resistance which is related to their length and which affects the strength of the signal that is transmitted. Resistance also plays an important role in branch growth during the developmental phase (see section 4.1 and also the appendix 3.11.1.3.1.).

Health: Biological neurons have property which one can loosely consider to be 'Health', since some neurons are weak and die, or may be diseased in some way. Generally speaking, biological neurons that are inactive for a long time tend to dwindle away. There is no concept of health of neurons or branches in ANNs or neural development models. However, we have adopted this property to allow the replication or death of neurons and neurites. Health also affects the signal processing inside neurons and neurites (see section 4.1 and also the appendix 3.11.1.3.1.).

Neural Activity: Neural activity refers to the degree to which neurons and neurites are active. In biology, not all neurons and neurites actively participate in every function that brain performs. In ANNs and neural development all neurons have to be processed before the function of the network can be assessed. This means that all the neurons are always active in ANNs. Following biology, in our model the network dynamics decides which neuron or neurite should be processed for a particular task. This is done by either making it active or changing the time for which it will be inactive. It is important to note that in our model this kind of neural activity refers to whether or not we run the internal processes of a neuron. Thus even if a neuron is not firing (in a refractory period) it can be active. Activity is used to keep neurons active for a number of cycles after it fires, also if it is not responding to the input signal it is kept inactive for a number of cycles. This is done partly to include the refractory period concept of neuron and partly to reduce the computational burden on the system, by keeping non-responsive neurons inactive for more time (see section 4.1 and the appendix 3.11.1.3.6.).

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

Synaptic Communication: Electrical signals are transferred from one neuron to another through synapses. Synapses are not just the point of contact, as considered in many published models of ANNs and neural development, they modulate the signal and provide a complex mechanism for signal transfer. We have evolved CGP programs to find the useful mechanisms to allow signal transfer across a synapse (see 4.5.1). Synapses in biological systems affect the potentials of the branches nearby through changes in the concentrations of chemicals (i.e. ions in space between neurons). Although synapses in the CGPCN are not chemical, when a synapse in the CGPCN is made, it causes the weights and potential values of the neighbouring branches to update (see the appendix 3.11.1.3.10. for further details). This is analogous to the chemical changes at synapses.

Arrangement of Neurons: The overall architecture of both ANNs and many neural developmental systems is fixed once developed, whereas biological neurons exist in space, interact with each other and move their branches from one place to other. We have adopted similar mechanism by placing neurons in Euclidean grid space, such that they have spatial proximity and this affects how they interact with each other. The axon and dendrite branches are also allowed to navigate over the Euclidean grid (see 4.5). This means that the morphology of the network is able to change during problem solving.

Spiking (Information processing): Biological neurons are decision making entities, they integrate the signal and fire. Some ANN models have adopted this in the form of spiking neurons. We also do this and allow our neurons to integrate the signal received from branches and fire an action potential.

Signals are passed from neuron to neuron via spikes (nerve impulses) in the biological brain, some of the ANN models (Spiking Neural networks) and Neural development models have used a spiking mechanism for signal processing. We have also adopted a similar mechanism in our model, as signals are transfered to other neurons only if the neuron fires (see 4.5.1 and also the appendix 3.11.1.3.6.).

Synaptic Plasticity: A key concept in understanding neural systems is plasticity. Plasticity means the ability to change or reform. Plasticity occurs at several levels in the nervous system. Much of the dynamic capabilities exhibited by neural systems results from synaptic plasticity, which refers to changes in synaptic transmission (Debanne et al. (2003)). This synaptic plasticity occurs at both post-synaptic and pre-synaptic levels. It can involve changes in the post-synaptic excitability (the probability of generating action potential in response to a fixed stimulus), which depends on the previous pattern of input (Gaiarsa et al. (2002)). The numbers of receptors (sites of neurotransmitter action) on the membrane can also be altered by synaptic activity (Frey and Morris (1997)). These processes can interact resulting in positive feedback effects, where some cells never fire and others may saturate at some maximal firing rate.

Synaptic plasticity is incorporated in the CGPCN by introducing three types of weights: 1) dendrite branch, 2) soma, and 3) axon branch (see 4.5.1 and also the appendix 3.11.1.3.3., 3.11.1.3.9. and 3.11.1.3.11. for details about how these affect dendrite branches, somae, and axon branches). These weights can be adjusted by genetic processes during the development of the network.

Changes in the dendrite branch *weight* are analogous to the amplifications of a signal along the dendrite branch (see London and Husser (2005)), whereas changes in the axon branch (or axo-synaptic) *weight* are analogous to changes at the pre-synaptic level and postsynaptic level (at synapse). Inclusion of a soma *weight* is justified by the observation that a fixed stimulus generates different responses in different neurons

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

(Traub (1977)). The weight of the soma has an impact on the electrical signal received at soma. The soma has a number of channels allowing ions to flow across membranes due to diffusion. The concentration of ions across the membrane affects the firing of the action potential at the soma. We have adopted this property in our model by having a soma weight which affects the final value of the soma potential (see the appendix 3.11.1.3.3.).

Synaptic plasticity in SNN models is primarily based on the work of Hebb (Hebb (1949)), the basic tenet of which is that repeated and persistent stimulation of a post-synaptic neuron by a pre-synaptic neuron results in a strengthening of the connection between the two cells. A recent development of this idea is the use of spike time dependent plasticity (STDP) rules for updating weights in SNN networks (Roberts and Bell (2002),Song et al. (2000)). *Weight* changes depend on the relative timing of pre- and postsynaptic spikes. This process has been documented in a range of neural systems, and a number of rules have been proposed. Weights can either increase, decrease or remain unchanged, such methods are reviewed in (Roberts and Bell (2002)). Although these schemes use an update rule depending on the firing time of only two neurons, they can result in interesting patterns of global behavior, including competition between synapses (Van Rossum et al. (2000)).

The **learning rule** is imposed in the ANNs and many neural development systems. In the model we propose we do not know what would be the best learning rules to use, so we allow these to be discovered through evolution. Indeed, we have adopted aspects of the above ideas in our CGPCN system by allowing the three different types of *weight* to change in response to firing of neurons. After every neural firing, life cycle and weight processing programs are run, this updates the soma weight and health (see later for details in CGP Neuron sections). These *weight* changes allow neurons and branches to become more active when they are involved in processing data. Synaptic weights are adjusted using a evolved program (see sections 4.5.1 and 4.5.2).

As explained in the next section, dendrite and axon branches, and soma which are more active are updated more frequently as their genetic code is run only when they are active. This interaction allows the patterns of activity propagating through the network to influence developmental and evolutionary processes. This interaction is known to be important for living neural systems, where sensory input and other environmental factors are important in shaping developmental aspects (see page 1125 onwards in (Kandel et al. (2000)). As in biology synapses that are active at the same time as the postsynaptic cell are strengthened, whereas those that are not synchronously active are eliminated (Nguy en (1996); Malenka (1994)). The elaboration, retraction, and remodeling of neural connections between neurons is activity-dependent because the ultimate patterns of connections can be disrupted by blockades of neuronal activity (Penn and Shatz (1999)). Signals received from the external environment drive the activity that shapes connections. Penn and Shatz demonstrated from their analysis that when synapses are active they are somehow protected from elimination, but when they are inactive they receive a withdrawal signal (Nguy en (1996); Penn and Shatz (1999)). Thus the pattern of activity is important in shaping the development of the system.

Developmental Plasticity: Neurons in biological systems are in constant state of change, their internal processes and morphology change all the time based on environmental signals. The development processes of the brain are affected a lot by external environmental signals. This phenomenon is called Developmental Plasticity. One form of developmental plasticity is synaptic pruning (Van Ooyen and Pelt (1994)). This process eliminates weaker synaptic contacts, but preserves and strengthens stronger con-

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

nections. More common experiences, which generate similar sensory inputs, determine which connections to keep and which to prune. More frequently activated connections are preserved. Neuronal death occurs through the process of apoptosis, in which inactive neurons become damaged and die. This plasticity enables the brain to adapt to its environment.

A form of developmental plasticity is incorporated in our model, branches can be pruned, and new branches can be formed. This process is under the control of a 'life cycle' chromosomes (described in detail in section 4.5 and for the dendrite branches in the appendix 3.11.1.3.3. and axo-synapses 3.11.1.3.11.) which determines whether new branches should be produced or branches need to be pruned. Every time a branch is active, a life cycle program is run to establish whether the branch should be removed or should continue to take part in processing, or whether a new daughter branch should be introduced into the network.

Starting from a randomly connected network, we allow branches to navigate (move from one grid square to other, and make new connections) in the environment, according to the rules that alter resistance explained in detail in section 4.5 and for the dendrites in the appendix 3.11.1.3.3. and for axo-synapses in the appendix 3.11.1.3.11.. An initial random connectivity pattern is used to avoid evolution spending extra time in finding connections in the early phase of neural development. So instead of developing the whole network from a single cell like the earlier neural development techniques, we start our network with a small number of neurons, dendrites, axon and dendrite branches arranged randomly (see the appendix 1.1.14.).

Arbitrary I/O: Biological neural systems can adapt to the change in number of sensory inputs. In almost all ANN and neural developmental models the number of inputs and outputs is predetermined and fixed. However, the architecture of our network changes all the time, so new input/output neurons can be introduced into CGPCN at runtime without affecting the overall operation of the network (see section 4).

Activity Dependent Morphology: There are few proposed models in which changes in levels of activity (in potentials or signals) between neurons leads to changes in neural morphology. This can occur in the model we propose in the following way. The network morphology changes in response to signaling. The electrical processing at each compartment (dendrite, soma or axon) decides whether to run its corresponding life cycle (developmental) program or not. These developmental programs decide on the morphology of each compartment (see 4.5.2 and also appendix sections 3.11.1.3.3., 3.11.1.3.9. and axo-synapses 3.11.1.3.11.).

In summary, we have idealized the behaviour of a neuron in terms of seven main processes. One to three are electrical processes, four to six are life cycle mechanisms, seven is weight processing:

1. Local interaction among neighbouring branches of the same dendrite.
2. Processing of signals received from dendrites at the soma, and deciding whether to fire an action potential.
3. Synaptic connections which transfer potential through axon branches to the neighbouring dendrite branches.
4. Dendrite branch growth and shrinkage. Production of new dendrite branches, removal of old branches.
5. Axon branch growth and shrinkage. Production of new axon branches, removal of old branches.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

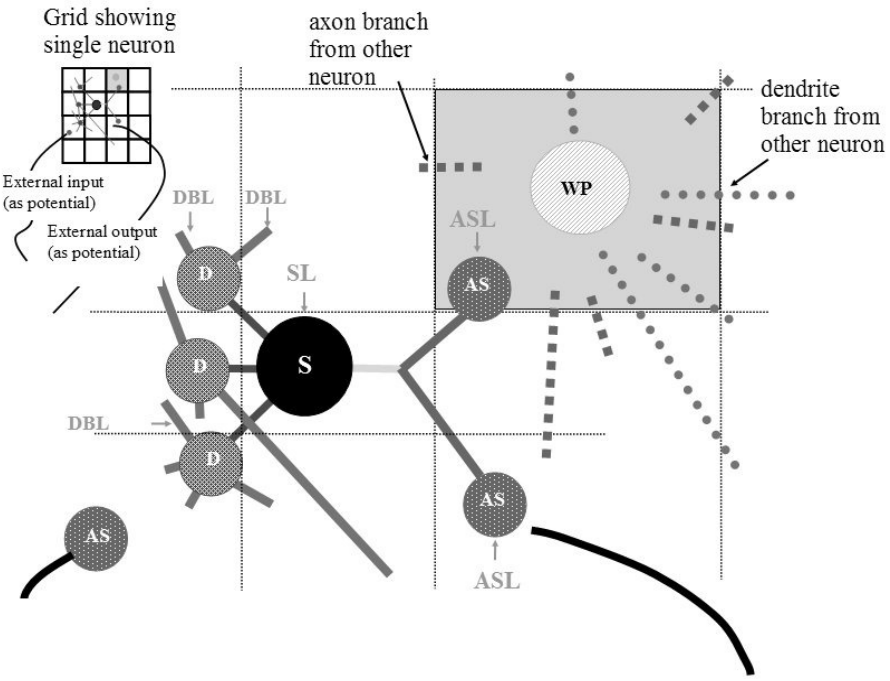

Figure 1: On the top left a grid is shown containing a single neuron. The rest of the figure is an exploded view of the neuron given. The neuron consists of seven evolved computational functions. Three are 'electrical' and process a simulated potential in the dendrite (D), soma (S) and axo-synapse branch (AS). Three more are developmental in nature and are responsible for the 'life cycle' of neural components (shown in grey). They decide whether dendrite branches (DBL), soma (SL) and axo-synaptic branches (ASL) should die, change, or replicate. The remaining evolved computational function (WP) adjusts synaptic and dendritic weights and is used to decide the transfer of potential from a firing neuron (dashed line emanating from soma) to a neighbouring neuron.

6. Creation or destruction of neurons.
7. Updating the synaptic weights (and consequently the capability to make synaptic connections) between axon branches and neighbouring dendrite branches.

Each aspect is represented with a separate chromosome (CGP program).

4 The CGP Computational Network (CGPCN)

This section describes in detail the structure of the CGPCN, along with the rules and evolutionary strategy used to run the system (further details are available in the appendix).

In the CGPCN neurons are placed randomly in a two dimensional spatial grid. They can only interact directly with their spatial neighbours (as shown in figure 1). Each neuron is initially allocated a random number of dendrites, dendrite branches, one axon and a random number of axon branches. Neurons receive information through dendrite branches, and transfer information through axon branches to neigh-

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

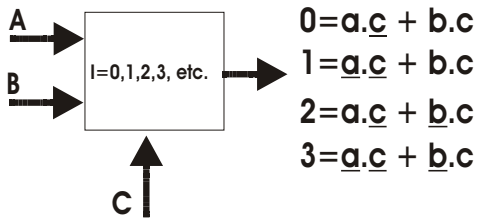

Figure 2: Multiplexer diagram, showing inputs A, B and C, and functions 0, 1, 2 and 3 . Figure also lists all the four possible functions that can be implemented by multiplexer.

bouring neurons. The dynamics of the network also changes, since branches may grow or shrink and move from one CGPCN grid point to another. They can produce new branches and can disappear, and neurons may die or produce new neurons. Axon branches transfer information only to dendrite branches in their proximity. Electrical potential is used for the internal processing of neurons and communication between neurons, and we represent it as an integer.

4.1 Health, Resistance, Weight and Statefactor

Four integer variables are incorporated into the CGPCN, representing either fundamental properties of the neurons (*health*, *resistance*, *weight*) or as an aid to computational efficiency (*statefactor*). The values of these variables are adjusted by the CGP programs. The *health* variable is used to govern replication and/or death of dendrites and connections. The *resistance* variable controls growth and/or shrinkage of dendrites and axons. The word *resistance* is used to demonstrate the attenuating effect on the signal due to the length of branches, so more resistance mean longer branches and vice versa. The *weight* is used in calculating the potentials in the network. Each soma has only two variables: *health* and *weight*. The *statefactor* is used as a parameter to reduce computational burden, by keeping some of the neurons and branches inactive for a number of cycles. Only when the *statefactor* is zero, are the neurons and branches are considered to be active and their corresponding programs are run. The value of the *statefactor* is affected indirectly by CGP programs. The bio-inspiration for the *statefactor* is the fact that not all neurons and/or dendrite branches in the brain are actively involved in each process.

4.2 Cartesian Genetic Program (Chromosome)

The CGP function nodes used here consist of multiplexer-like operations (Miller et al. (2000)). Each function node has three inputs and implements one of the four functions as shown in figure 2:

Here a, b and c are the inputs to the node (as shown in Figure 2). These functions are arithmetic with operations of addition (+), multiplication (.) and complementation (maximum number minus the number). All the inputs and outputs are 8-bit integers.

The four functions in figure 3 are the possible input combinations of a three input (two inputs and a control) multiplexer. Multiplexers can be considered as atomic in nature as they can be used to represent any logic function(Chen and Hurst (1982); Miller et al. (2000)) . As we are using arithmetic operations throughout in all the processing parts, instead of using bit wise ANDing and ORing we have transformed these

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

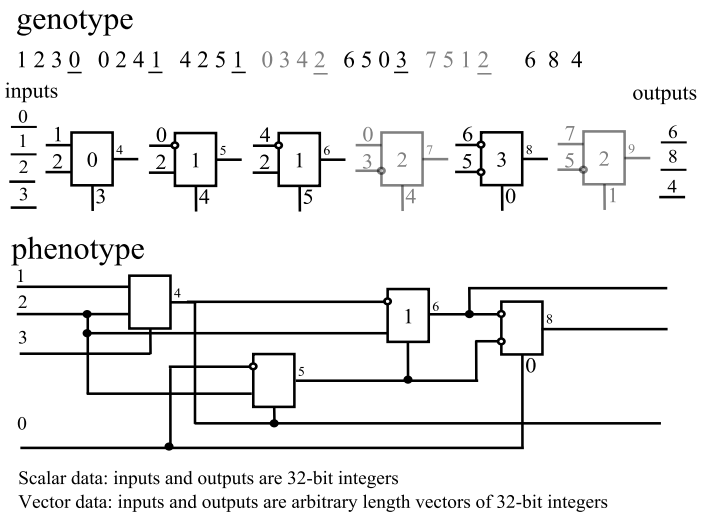

Figure 3: Structure of CGP chromosome. Showing a genotype for a 4 input, 3 output function and its decoded phenotype. Inputs and outputs can be either simple integers or an array of integers. Note nodes and genes in grey are unused and small open circles on inputs indicate inversion. The function type in the genotype is underlined. All the inputs and outputs of multiplexers are labeled. Labels on the inputs of the multiplexer shows where are they connected (i.e. they are addresses). Input to CGP is applied through the input lines as shown in the figure. The number of inputs (four in this case) and outputs (three in this case) to the CGP is defined by the user, which is different from the number of inputs per node (three in this case i.e. a, b and c.)

functions into equivalent arithmetic operations. So AND is replaced by multiplication and OR by addition. Complementation is obtained by subtracting the number from the maximum number.

Figure 3 shows the genotype and the corresponding phenotype obtained by connecting the nodes as specified in the genotype. The Figure also shows the inputs and outputs to the CGP. Output is taken from the nodes as specified in the genotype (6, 8, 4). In our case we have not specified the output in the genotype and have used a fixed pseudo random list of numbers to specify where the output should be taken from. We have done so to reduce the search space for the genotype to increase the speed of evolution.

Here the number of rows used is one as described earlier. Thus, the number of columns is equal to number of nodes. The maximum number of nodes are defined by the user. These nodes are not necessarily all connected. Inputs are applied to CGP chromosomes in two ways:

- Scalar
- Vector

In the former, the inputs and outputs are integers while in the latter inputs required by the chromosome are arranged in the form of an array, which is then divided into ten CGP input vectors. If the total number of inputs can not be divided into ten equal parts, then they are padded with zeros. This allows us to process an arbitrary number of

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

inputs by the CGP circuit chromosome simply by clocking through the elements of the vectors. In general CGP can not take variable number of inputs, so this method allows it to take variable number of inputs at run time. As the inputs are arranged in the form of vectors, and each vector can have arbitrary number of elements. This method adds some noise and this is more pronounced when the number of inputs are less than ten, as we pad it with zero when the number of inputs can not be divided into ten sub vectors. But as the number of inputs increases the effect of noise is reduced. In GP (including CGP) programs are assigned a fixed number of inputs. Since the number of dendrite and axo-synaptic inputs to evolved programs is variable it was necessary to devise a way of handling this situation. It was decided to present such inputs in the form of a fixed number of arbitrary length input vectors. The number of such input vectors used is ten, this is an arbitrary choice, but would be easily parameterized so that experiments could be carried out to find suitable values. Including it as a 'parameter' gene is also an option. The implication of dividing into vectors is that the elements of the vectors have to be clocked through the CGP chromosome since the functions set used do not operate on arbitrary length vectors. The issue of how best to deal with problems that require an arbitrary number of inputs is a complex one and there is no consensus in the GP research community about how to deal with this. Recently there has been some progress on the issue. Harding et al introduced the idea of having *functions* whose role was to return input values Harding et al. (2010). Each time one of these input functions is called in the genotype it returns the *next* input. It may be possible in the future to implement an input gathering strategy related to this idea.

4.3 Evolutionary Strategy

The evolutionary strategy utilized is of the form (1 + λ)-ES, with λ set to 4 (Yu and Miller (2001)), 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 chromosomes achieve the same highest fitness then the newest (genetically) is always chosen (Miller et al. (2000)).

The steps in the evolutionary cycle are as follows:

- Create a random population of five genotypes (Each genotype consists of seven chromosomes of neuron)
- Create a CGPCN with random number of dendrites and branch structures.
- An evolutionary generation consists of:
- For each genotype *C* in the population:

  Produce a copy of the random CGPCN

  Run the genotype on the CGPCN.

  Calculate fitness of the resulting CGPCN, *F(C)*
- From population select best *F(C)*; if two or more are equally best then pick newest of them (Miller et al. (2000))
- Create new population by mutation, keeping the promoted genotype unchanged

  Continue until either a maximum number of generations, or a solution is found

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

Thus, a mature network is produced by executing the program encoded in the genotype starting from the same initial random network of neurons, dendrites, and dendrite and axon branches.

The promoted genotype is mutated to produce four new genotypes (the offspring) in the following way:

- Calculate the number of genes to be mutated i.e.
  N_{bit} = N_g x μ/100; and
  N_g = (N_i +1) x N_n x N_{ch}
  Whereas
  - N_g = Number of Genes
  - μ = Mutation Rate
  - N_i = Number of Inputs per Node (3 in this case)
  - N_n = Number of Nodes per Chromosome
  - N_{ch} = Number of Chromosomes (7 in this case)
  - N_{bit} = Number of Bits to be mutated
- Select pseudo randomly, one gene at a time and mutate it pseudo randomly. Mutation of a gene means:
  - if it is a connection
    - replace with another valid connection.
  - if it is a function
    - replace with another valid function.

4.4 Inputs, Outputs and Information Processing in the Network

The external inputs (encoding a simulated potential) are applied to the CGPCN and presented to axo-synaptic electrical processing chromosomal branches as shown in figure 4. These are distributed in the network in a similar way to the axon branches of neurons. After this, the program encoded in the axo-synaptic electrical branch chromosome is executed, and the resulting signal is transferred to its neighbouring active dendrite branches. Similarly we have outputs which read the signal from the CGPCN through dendrite branches. These branches are updated by the axo-synaptic chromosomes of neurons in the same way as other dendrite branches and after five cycles the potentials produced are averaged and this value is used as the external output. Instead of having only one output branch at a specific location, we use five and take the average of them as the final output. We have done so to avoid evolution spending extra time in locating just one particular branch to communicate with. The thinking behind this was also related to the fact that 'outputs' (i.e. actuators) in living organisms are innervated by many neurons. The numbers of cycles (5) is a user defined parameter. It is more than one in order to provide enough time for signals to transfer from input to output. In principle evolution could be used to obtain the number of cycles however by using controlled parameter experiments it would be possible to investigate this parameters role in the effectiveness of the system.

Information processing in the network starts by selecting the list of active neurons in the network and processing them in a random sequence. Each neuron take the signal

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

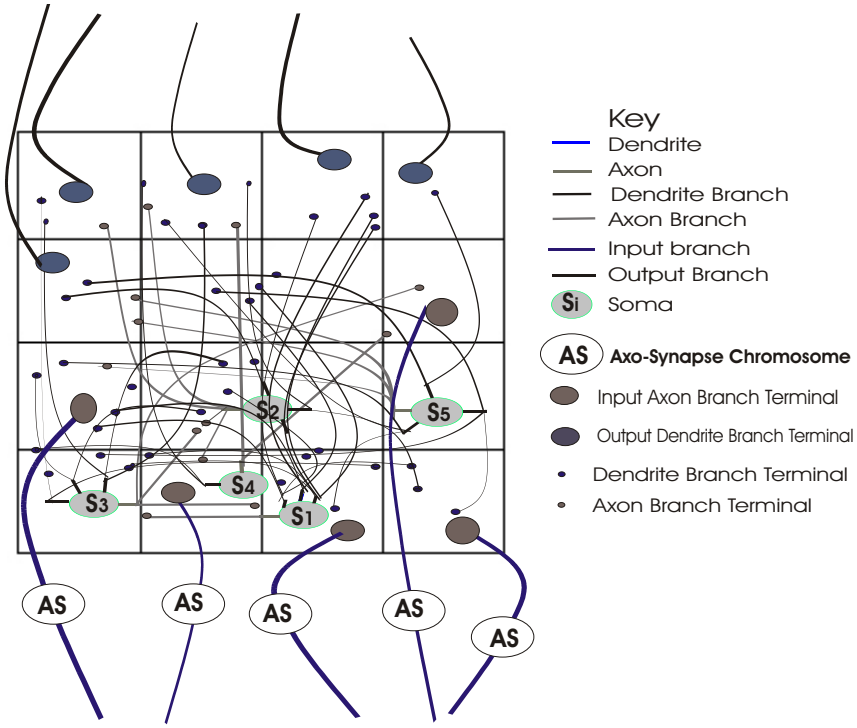

Figure 4: A schematic illustration of a 4×4 CGPCN grid. The grid contains five neurons, each neuron has a number of dendrites with dendrite branches, and an axon with axon branches. Inputs are applied at five random locations in the grid using input axo-synapse branches by running axo-synaptic CGP programs. Outputs are taken from five random locations through output dendrite branches. The figure shows the exact locations of neurons and branches as used in most of the experiments as an initial network. Each grid square represents one location, branches and soma are shown spaced for clarity. Each branch location is represented by where its terminal is located. Every location can have as many neurons and branches as the network produces, there is no imposed upper limit.

from the dendrites by running the electrical processing program in the dendrites. The signals from dendrites are averaged and applied to the 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 should fire an action potential or not. If the soma fires, an action potential is transferred to other neurons through axo-synaptic branches. The same process is repeated in all neurons. A description of the seven chromosomes is given in the next section.

4.5 CGP Model of Neuron

In our model neural functionality is divided into three major categories: electrical processing, life cycle and weight processing. These categories are described in detail below.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

4.5.1 Electrical Processing

The electrical processing part is responsible for signal processing inside neurons and communication between neurons. It consists of dendrite branch, soma, and axo-synaptic branch electrical chromosomes. Here instead of using the terminology of axon we have used 'axo-synapse' as in biology the axon functions largely as a lossless transmission line to transfer signal from one end to other. The only part which is responsible for signal processing is the synapse. This is why we have used the term axo-synapse because this serves the purpose of both axon and synapse.

The D chromosome handles the interaction of dendrite branches belonging to a dendrite. It takes active dendrite branch potentials and the soma potential as input and then updates their values. The *Statefactor* is decreased if the potential is increased by the update by a large amount and vice versa. If a branch is active (has its statefactor equal to zero), its life cycle program is run (DBL), otherwise it continues processing the other dendrites.

The chromosome responsible for the electrical behaviour of the soma, S, determines the final value of soma potential after receiving signals from all the dendrites. 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 it fires, it is kept inactive (refractory period) for a few cycles by changing its *statefactor*, the soma life cycle chromosome (SL) is run, and the firing potential is sent to the other neurons by running the program encoded in axo-synapse electrical chromosome (AS). The threshold potential of the soma is adjusted to a new value (maximum) if the soma fires.

The potential from the soma is transferred to other neurons through axon branches. The AS program updates neighbouring dendrite branch potentials and the axo-synaptic potential. The *statefactor* of the axo-synaptic branch is also updated. If the axo-synaptic branch is active its life cycle program (ASL) is executed.

After this the weight processing chromosome (WP) is run which updates the *Weights* of neighbouring (branches sharing same grid square) branches. The processed axo-synaptic potential is assigned to the dendrite branch having the *largest* updated *Weight*.

4.5.2 Life Cycle of Neuron

This part is responsible for replication or death of neurons and neurite branches and also the growth and migration of neurite branches. It consists of three life cycle chromosomes responsible for the neuron and neurite development.

The dendrite and axo-synaptic branch chromosomes update *Resistance* and *Health* of the branch. Changes in *Resistance* of a neurite branch are used to decide whether it will grow, shrink, or stay at its current location. The updated value of neurite branch *Health* decides whether to produce offspring, to die, or remain as it was with an updated *Health* value. If the updated *Health* is above a certain threshold it is allowed to produce offspring, and if below certain threshold, it is removed from the neurite. Producing offspring results in a new branch at the same CGPCN grid point connected to the same neurite (axon or dendrite).

The soma life cycle 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, die or continue as it is. If the updated *Health* is above certain threshold it is allowed to produce offspring and if below a certain threshold it is removed from the network along with its neurites. If it produces offspring, then a new neuron is introduced into the network with a random number of neurites at a different

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

random location.

Branches are produced at the same location because they are the offspring of the branch at that location and should be connected to the same dendrite. This follows what happens in biology, where branches originate on more major neurite components. The soma life cycle introduces a new neuron in the system. In neurogenesis the new neurons do not have morphologies similar to the parent neuron instead their morphology is acquired gradually in response to their individual neural environment. Further, the location of soma is not that important because the communication is done through neurite branches.

5 Applications of CGPCN

The capabilities of the CGPCN is tested in two different experimental scenarios. In the first case, we demonstrate its ability to learn during its lifetime and transfer this learning capability genetically from generation to generation, using the Wumpus World problem. The importance of the life cycle chromosomes is also demonstrated. In the second case, the network is tested in a coevolutionary environment with two agents learning to solve their tasks while interacting with each other.

5.1 Wumpus World Problem

Wumpus World, is a variant of Gregory Yobs "Hunt the Wumpus" game (Yob (1975)). It is an agent-based learning problem used as a testbed for various learning techniques in Artificial Intelligence (Russell and Norvig (1995)). The Wumpus World was originally presented by Michael Genesereth(Russell and Norvig (1995)) . It consists of a two dimensional grid containing a number of pits, a Wumpus (monster) and an agent (Russell and Norvig (1995)) as shown in figure 5.

An agent always starts from a unique square (home) in a corner of the grid. The agent's task is to avoid the Wumpus and the pits, find the gold, and return with it to home. The agent can perceive a breeze in squares adjacent to the pits, a stench in the squares adjacent to the Wumpus, and a glitter on the gold square. The agent can be armed with one or more arrows. So it can shoot the Wumpus by facing it. 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. Wumpus World can be presented in different environments (Yob (1975) used an environment that was a flattened dodecahedron), but the most common is the rectangular grid. Spector and Luke investigated the use of genetic programming to solve the Wumpus World problem (Spector (1996); Spector and Luke (1996)).

5.2 Experimental Setup

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 and the agent does not have any arrow to shoot the Wumpus. Also a signal called 'glitter' is observable on the squares adjacent to the gold. 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. Evolution must find a series of programs that build a computational network that is stable (that doesn't lose all neurons or branches

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

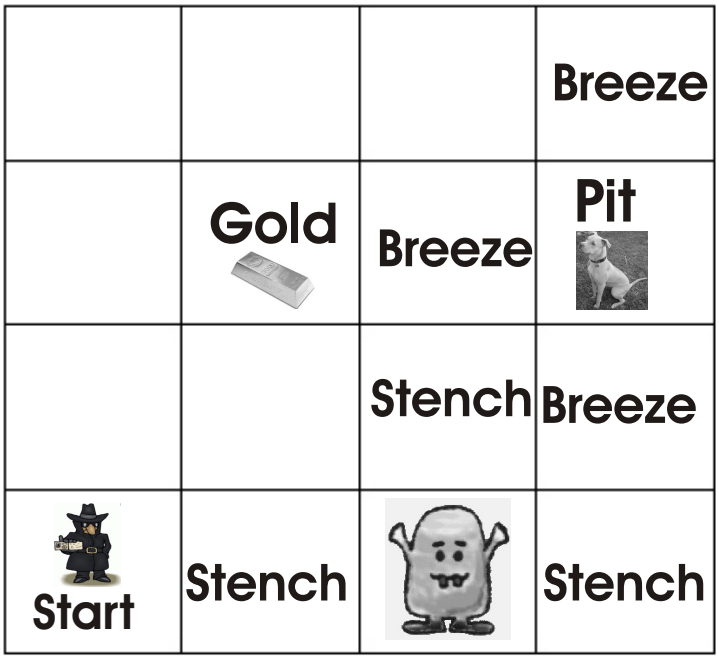

Figure 5: A two dimensional grid, showing Wumpus World environment, having one Wumpus, One agent on the bottom left corner square, one pit and gold (Russell and Norvig (1995))

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. We believe that this makes the Wumpus World problem a challenging problem.

It is worth considering how a non-recurrent ANN would perform on such a problem. Many squares in the environment provide no (zero) input so on such squares a network would give a static output signal. This signal would be interpreted as a movement. Thus it is easy to see that such networks could only change direction as a result of a change in an environmental signal from one square to another. While it is true that another machine learning approach could possibly take a sequence of moves leading to gold and reverse them leading back to home, it would be unlikely to do well in a randomly generated Wumpus World. We will see later that some of the evolved agents using the neural model described can both find a sequence of moves without changes in environmental inputs and solve the task in a different Wumpus World.

The Wumpus World we used is a two dimensional grid (10x10), having ten pits, one Wumpus and the gold. The location of the Wumpus, gold and pits is chosen randomly. In the square containing the Wumpus, gold, pits, and in a directly (not diagonally) adjacent square the agent can perceive stench, glitter and breeze, respectively. The input to the CGPCN is a potential of zero, if there is nothing on the square, a potential of 60 if encounters a pit, 120 if caught by Wumpus and 200 if arrives at the square containing the gold. The system is arranged in such a way that the agent will receive input signals of different magnitudes depending on the direction that the agent per-

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

ceives the signal. The agent's CGPCN can perceive the direction of a breeze through the magnitude of the breeze potential. We chose the following values to represent these directions: north of the pit is 40, east 50, south 70, and 80 for west. Similarly the stench signal has a direction which is represented by different magnitudes of potentials as follows: for north 100, for east 110, for south 130 and finally 140 for west of Wumpus. Also it receives 180, 190, 210 and 220 from north, east, south and west of gold on for glitter. An agent detects that it is on the home square via a maximum input signal (255 is the maximum value of potential). All the other locations which are safe provide no signal to the agent. The agent can always perceive *only one* signal on a grid square, even if there are more than one. The priority of the signals is in the following order: Wumpus, gold, pit, stench, glitter, breeze. The agent is assigned a quantity called energy, which has an initial value of 200 units. If an agent is caught by the Wumpus its energy level is reduced by 60%, if it encounters a pit its energy level is reduced by 10 units, if it gets the gold its energy level is increased by 50 units, and on arriving home the agents ceases to exist. For each single move the agent's energy level is reduced by 1 unit, so if the agent just oscillates in the environment and does not move around and acquire energy through solving tasks, it will run out of energy and die.

The fitness of an agent is calculated according to its ability to complete learning tasks. It is accumulated over the period that the agents energy level is greater than zero (or before it returns home). The fitness value, which is used in the evolutionary scheme, is calculated as follows:

While (Energy ≥ 0)

- For each move, the fitness is increased by one. This is done, to encourage the agents to have 'brain' that remains active and does not die.
- 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.

With each move the energy decrements but the fitness increments, thus evolution is searching for the shortest sequence of moves that leads to the largest fitness reward. In the case of the Wumpus World shown in Fig. 6 an ideal path (giving maximum fitness is shown). Note, all paths that lead to the gold without backtracking are the shortest paths. All such paths that avoid pits will receive the highest fitness.

When the experiment starts, the agent takes its input from the Wumpus World grid square in the form of potential representing the status of the square (home, gold, breeze, etc). This input is applied to the computational network of the agent through input axo-synapse branches distributed at five different locations in the CGPCN grid. This updates the potentials of the neighbouring dendrite branches in CGPCN grid connected to different neurons. The network is then run for five cycles (one step). A cycle is equivalent to selecting the active neurons and running all the processing inside them. To elaborate, this means selecting active neurons, processing them in a random sequence with each neuron receiving inputs from the active dendrite branches processing them and finally sending the processed signal to other neurons through active axon branches (as described in section 4.3, para-2, information processing). During this process the potentials of the output dendrite branches are updated which act as the output of the CGPCN. After every five cycles (one step), the updated potentials of all output dendrite branches are noted and averaged. The value of this average potential is used

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

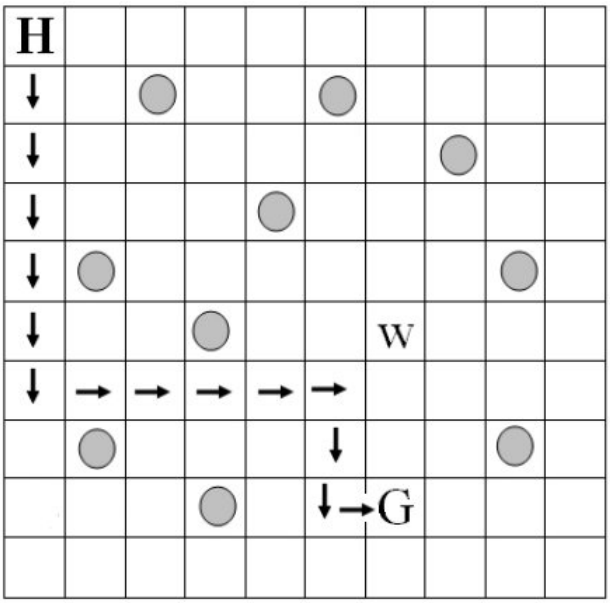

Figure 6: A fixed Wumpus World with a path that would receive the highest fitness

to decide the direction of movement for the agent (direction of next step in the environment). If there is more than one direction the potential is divided into as many ranges as possible movements. For instance, if two possible directions of movement exist, then it will take one direction if the potential is less than 128 and the other if greater. The same process is then repeated for the next Wumpus World grid square. The agent is terminated if either its energy level becomes zero, or all its neurons die, or all the dendrite or axon branches die, or if the agent return home.

5.2.1 CGPCN Setup

The CGPCN is arranged in the following manner for this experiment. The network is arranged in the form of a 3x4 CGPCN grid (Note: this should not be confused with the Wumpus World grid in which the experiment takes place). The grid is small in order to give opportunity for branches to communicate well, as the number of neurons and neurites are small. Inputs and outputs to the network are located at five different random CGPCN grid squares. The initial number of neurons is 5. The maximum number of dendrites is 5. The maximum number of dendrite branches and axon branches is 5 to start with. We provided a random initial structure to the network and allow it to obtain the proper structure during task environment. The maximum branch *statefactor* is 7 in order to reduce the computation time. The maximum soma *statefactor* is 3 because it needs to be inactive at least for two cycles after firing (refractory period). The mutation rate is 5%. The maximum number of nodes per chromosome is 100. We used 8-bits to represent potential, resistance, weight and health in this experiment, so the maximum value is 255. Also the node operation of CGP genetic code are 8-bit arithmetic operations as described earlier. Chromosome length is thus 400 integers (100 nodes * (No. of connections per node (3) + Node function))

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

Task.No	Task	Average No. of Generations	Minimum No. of Generations	Maximum No. of Generations
1	Home empty handed	4	1	15
2	Found the gold	13	2	95
3	Home with gold	139	12	304

Table 2: The average, maximum, and minimum number of generations taken to perform various tasks in a fixed Wumpus World in 15 independent evolutionary runs.

5.2.2 Results and Analysis

The agent performance is determined 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. The agent has to perform all three tasks in one World independently. However it obtains the largest fitness by completing the last task (No. 3 in table 2). Each agent population's performance is tested in the same (one) Wumpus World where gold, pits and Wumpus is located at the same place. During this process the agent has to avoid pits and the Wumpus in order to increase its life span. The life span is the period of time while an agents's energy level is above zero before it returns home. An agent needs to sustain the neural network to solve its tasks. Agent performance on these tasks is tested in independent evolutionary runs, with the same Wumpus World (shown in Fig. 6) and different initial populations. Table 2 shows the performance of the agent in fifteen independent evolutionary runs. From the table it is evident that the agent solves the first and second tasks more quickly than the last task, which is much more difficult than the first two.

Figure 7 shows the fitness graph of an agent while trained to solve a Wumpus World. Initially the fitness is smaller (learning how to come back to home), after a few generations (12) there is a sudden increase (above 1000) when the agent found the gold. The agent continues to improve its fitness by living longer and finding the gold. And finally at generation 280, the agent found a path that leads to home after getting the gold.

In order to test the difficulty level of this Wumpus world, we have generated ten thousands (10000) random initial genotypes and use the same initial random neural structure provided for the above experiments. We have obtained the following statistics about various tasks that the agent solves out of ten thousands random genotypes. Figure 8 shows the fitness of various randomly generated genotypes on the Wumpus world of Figure 6. From the figure it is evident that the agent was able to solve the first task (coming back to home) 523 times, second task (Finding the gold) 201 times, and finally the third task (Finding and bringing gold to home) only 3 times (fitness more than 3000) out of ten thousands randomly generated genotypes. This clearly demonstrates the difficulty level of the task investigated as it takes on average 3333 genotypes to find an agent that find the gold and returns it home. From Table 2 we find that the average number of genotypes required to do this is 557 (the 1+4 ES requires 4 new genotypes to be evaluated each generation and the initial generation requires 5).

The network is further tested by starting with the same initial population and different Wumpus Worlds, with pits, Wumpus and gold at different locations. In each run agents were evolved that could solve each task. Table 3 shows the performance

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

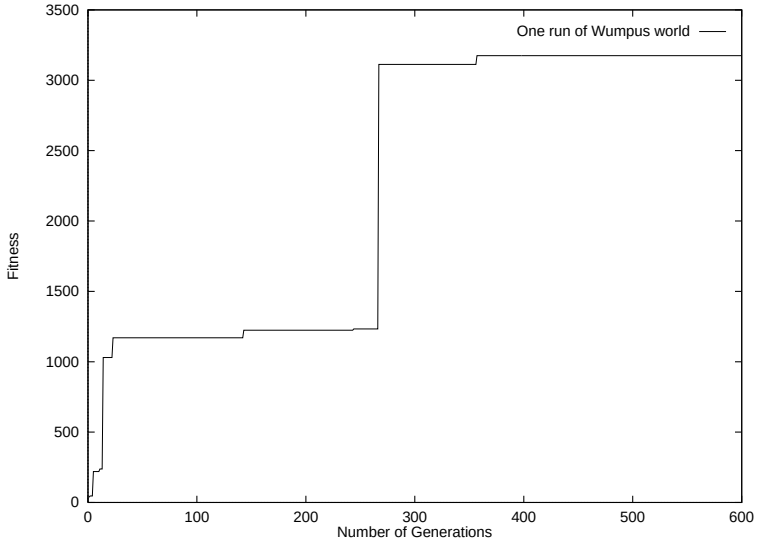

Figure 7: Fitness diagram showing the performance of the best agent over various generations during evolution.

Task.No	Task	Average No. of Generations	Minimum No. of Generations	Maximum No. of Generations
1	Home empty handed	5	2	11
2	Found the gold	20	2	79
3	Home with gold	530	12	1769

Table 3: The average, maximum, and minimum number of generations taken to perform various tasks on different Wumpus Worlds starting with the same initial population in ten independent evolutionary runs.

of the agent in ten independent evolutionary runs. From the table it is evident that the network took a similar number of generations as in the first case to solve the problem. Which shows that the initial randomly generated Wumpus World is of average difficulty.

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 (incremented with every move of agent). At the start the agent does not achieve any goals, the fitness is directly related to the time the agent spends in Wumpus World. For that time it has to maintain its network, so in the initial generations evolution tries to build a stable and sustainable computational network. Once this is achieved, evolution starts to produce agents that first learn how to come back home, then learn how to find the gold and finally they bring the gold back home.

At the start, the agent does not know anything about gold, home, Wumpus, pits and the signals that indicate the presence of these objects. As the system is evolved

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

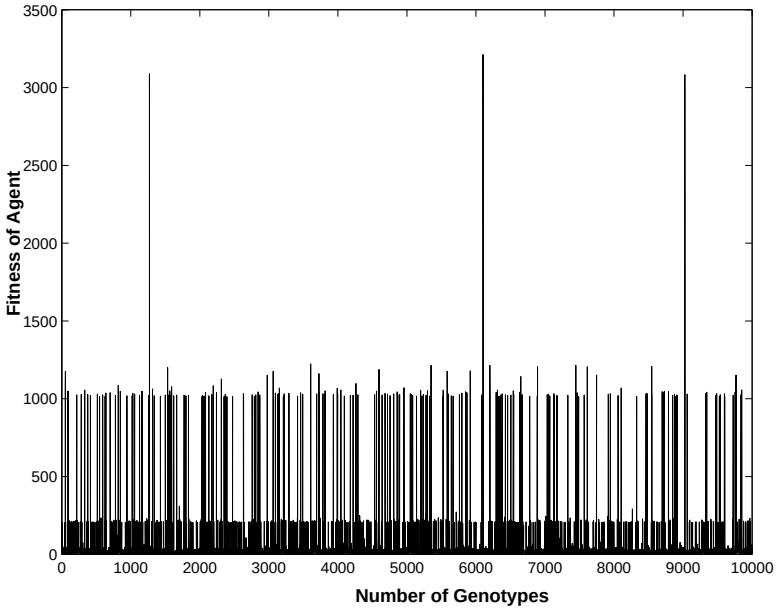

Figure 8: Fitness diagram showing the performance of ten thousands randomly generated genotypes on the Wumpus World of Figure 6

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 approximate the solution to a problem. For small changes in the nature of the problem, these networks generally need to be retrained to solve the changed problem (Cunningham et al. (2000)). However, in the 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

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

we took the programs of ten independent evolutionary runs for the best evolved agent trained on a particular Wumpus World and tested its performance on other Wumpus Worlds that had been generated at random.

From different tests we found that in cases when the gold is placed at the same location, but pits and Wumpus are located at different locations, the agent was always able to get the gold and bring it home. In other cases when the gold was not placed at the same location, sometimes the agent gets the gold (30% of time) but is unable to find its way to home, and sometimes it cannot find the gold (50% of time) and returns home empty handed. There are also cases when the agent could not find the gold or get back home (20% of time), demonstrating that we have not yet obtained the general problem solving behaviour and there is still some noise in the network. Further observations during these experiments revealed that, 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). It should be noted that the genotypes are evolved and evaluated with an initial randomly generated small network of neurons and connections. The networks that control the agent develop over time during the lifetime of the agent. For learning to happen solely as a result of evolution then it must come up with low-level rules operating on a simulated electrical potential on a network that is changing with time in such a way that the path of an agent was always the same.

5.2.3 Development of network over the lifetime of the agent

While solving the Wumpus World the CGPCN changes substantially in its structure, both in term of the presence and absence of neurons and branches, and whether they are active or inactive. Figure 9 shows the variation in the energy level of one agent while finding the gold and bringing it back home. When it falls into a pit its energy level is decreased by ten units. In this particular case the agent falls into a pit three times but is never caught by the Wumpus. There is a sudden increase in energy level of 50 units when the agent finds the gold (step 58). When the agent returns home we terminate its life by setting the energy level to zero. Figure 10 shows the variation in numbers of neurons, dendrite branches and axon branches during this agent's life. Figure 11 shows the variation in the number of active neurons, dendrite branches and axon branches at different stages in the life of this particular agent. Figure 12 shows the variation in neural structures and growth of the CGPCN at different stages. As the network develops, dendrites branches vanish, and new dendrite branches are introduced. After ten steps the network has gained three additional neurons. At twenty-five steps the network rather strangely consists of two independent sub-networks. The reason for this is not clear as the agent has until that point (see Figure 9) not encountered any hazards. However after that the numbers of neurons increases. It is evident from Figure 12 that network change substantially, earlier on the changes to topology look more random but later the network adopts a more stable topology. It is interesting to observe that the network starts with neurons having 2-4 dendrites and, after all the changes ends up with similar numbers.

Figures 10, 11, and 12 illustrate interesting network dynamics. Network dynamics appear random at the start, but soon appeared to get a sustainable structure as the agent performs the task. The agent take less time to get to its second goal (home) than the first goal (gold), this may be because it has developed a sustainable network that allows it to quickly achieve this. This suggests that the agent may produce a map of its

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

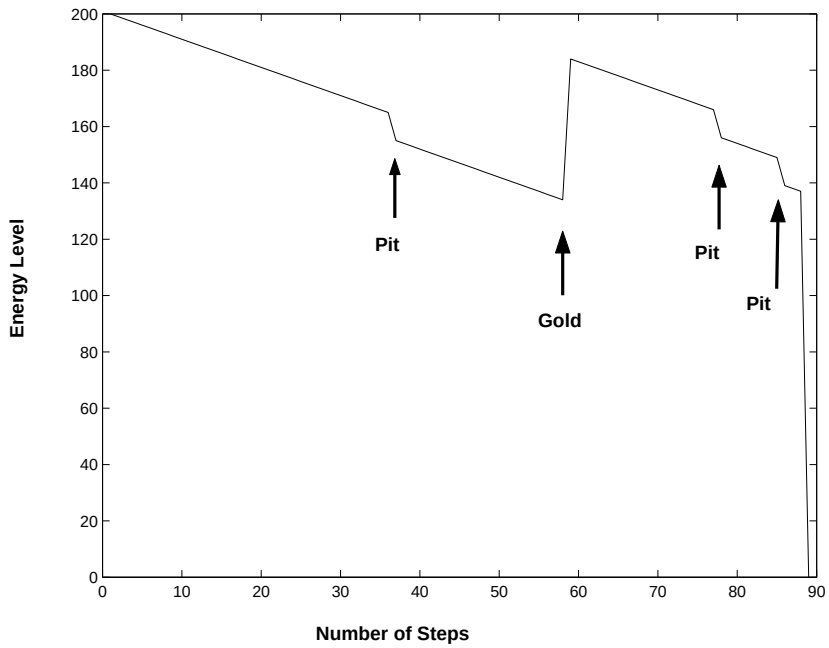

Figure 9: Variation in agent energy level shown against the number of completed steps (1 step $\equiv$ 5 cycles) while solving the Wumpus World problem. The energy shows a continual decrement of 1 per step, sudden drops occur when the agent encounters a pit (steps 36, 77, 85), and a sudden increase when it finds the gold (step 58). The energy is decreased to zero when the agent returns home.

environment during its developmental stage, which is used to find the return path. In general we observed that the return path to home is usually more direct than the path to the gold. If we imagine that the agent's movement is determined by the throw of dice, it could take an infinite time to get the gold and bring it back to home, as every empty square gives same signal, so it might end up oscillating between two squares all the time. As all the pits give the same signal it is difficult to differentiate one pit from another, thus making Wumpus World a difficult task to solve.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

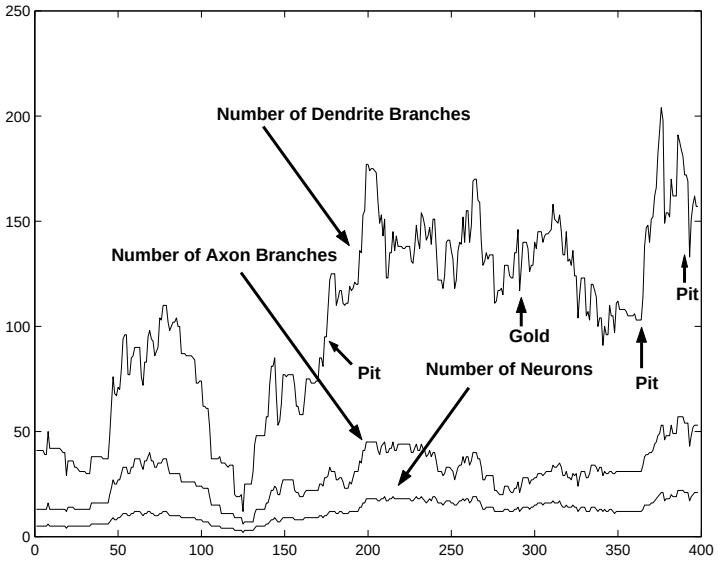

Figure 10: Variation in numbers of neurons, axon branches and dendrite branches at different stages of an agent's life while solving Wumpus World problem shown against the number of completed cycles. The numbers of these components fluctuate at the start, and then appear to stabilize later in the run, at around 200 cycles.

5.2.4 Testing the network without life cycle programs

We investigated the behaviour of evolved networks on a Wumpus World with different initial populations but a fixed morphology in the CGPCN. Fixed morphology means no life cycle (developmental) program is run inside neurons, dendrite branches and axon branches. Therefore the network structure does not change, and only electrical and weight processing chromosomes are run and evolved. We provided the agent with a random CGPCN structure having the following characteristics (as shown in figure 13):

- Initial Number of Neurons are 5.
- Neuron (N) 1 and 2 have four dendrites, N3 and N5 have three dendrites and N4 have one dendrite as shown in figure 13.
- Input axosynaptic branches are distributed at locations 5,2,5,2 and 4 in the CGPCN grid respectively.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

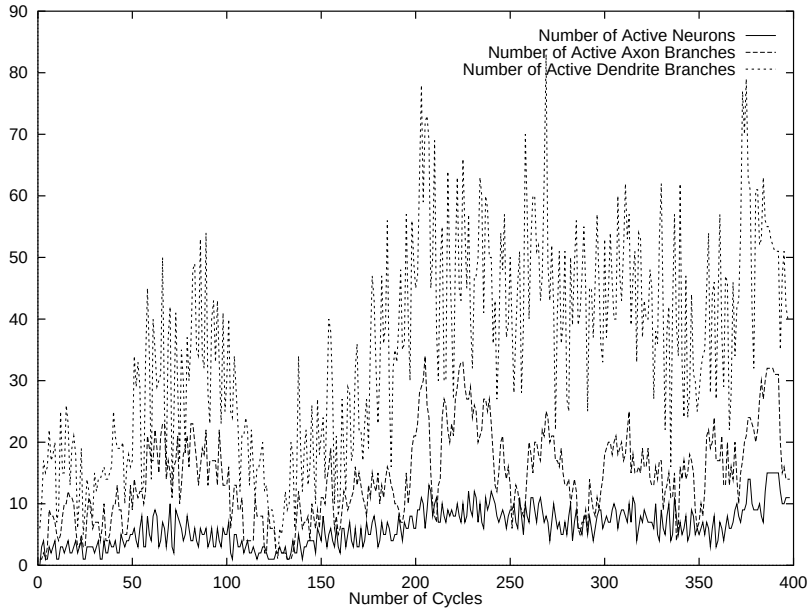

Figure 11: Variation in numbers of active (with state factor zero) neurons, axon branches and dendrite branches at different stages of an agent's life while solving Wumpus World problem.

- Output dendrite branches are distributed at 8,6,11,8 and 9 respectively.
- Neurons are provided with axon branches such that N1 has one, N2 four, N3 three, N4 two and N5 three axon branches.

These neurons are arranged in CGPCN grid as shown in figure 13 such that they have full connectivity. All neurons are connected in the path from input to output.

We ran fifteen (15) independent evolutionary runs for 500 generations each, table 4 shows the number of generations that the fixed CGPCN network took to complete various tasks.

The table shows that the life cycle (developmental programs) of the neural components is an important factor in the learning capability of the networks. In fifteen runs the agent was able to solve the complete task (bringing the gold to home) only four times (27% of all the runs). This compares markedly with Table 2 where all fifteen agents equipped with life cycle programs were able to bring the gold back home. The fixed networks that could bring gold back home (4) took on average 236 generations in comparison to an average of 139 (with 100% success) with life cycles. Chalup also proposes an incremental learning scheme which allows the development of the network during learning phase, would work much better than artificial imposed fixed network structure (Chalup (2001)).

5.2.5 Learning and Memory development in CGPCN

We performed further experiments on the Wumpus World to test the validity of the argument that the genotype of the agent obtained through evolution holds a memory of information in the Wumpus environment in its genetic structure.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

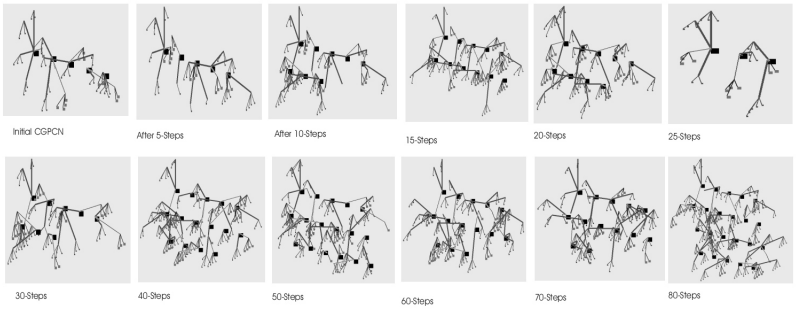

Figure 12: Structural changes in an agent's CGPCN network at different stages of Wumpus World shown against number of completed steps. The network has 5 neurons at the start, and 21 neurons after completing 80 steps. Black squares are somae, red thick lines are dendrites, yellowish green lines are axons, green lines are dendrite branches, and blue lines show axon branches. Note that this figure is not an exact indication of locations of the network components in the 12 square CGPCN grid, but is a schematic illustration to give an indication of the number of components in the network. Inputs and outputs are (not shown) are at fixed locations (see text)

We examined the learning behaviour of ten highly evolved agents in different scenarios to those in which they were trained. Instead of killing an agent after it had returned home with the gold, we always placed it on its home square immediately it had obtained the gold, and allowed it to continue to live and move around in the environment. In some cases we found that the agent immediately took a path toward the gold (50% of the cases). Some of these agents move around in the environment and then come back home, then go out again and visit different areas and come back to home, and finally end up oscillating near home (30% of the time). In other cases the agent leaves home but gets stuck somewhere in the corner of the Wumpus World, oscillating between squares until it dies (20% of the time). However, in most of the cases the agent tries to get the gold again, taking a shorter path to the gold over several attempts. This suggests that a map building ability (memory) is encoded in the genetic code of CGPCN and also that it possesses goal driven behaviour so that it can get the gold again and again. Some of the agents end up stuck in a corner exhibiting oscillatory behaviour. This oscillatory behaviour may be caused by the agent being in the unusual situation of being in a corner. This may not be encountered often during the experiences of different generations.

Here we examine a specific case where an evolved agent retrieved the gold three times, but finally ends up oscillating. The evolved agent in question solves the Wumpus World by following the path as shown in figure 14a toward the gold and the path shown in figure 14b to return home after getting gold. Figure 15 shows the characteristics of agent's movements. On the y-axis we have plotted the temporal sequence of the square numbers that the agent takes and on the x-axis the number of steps taken by the agent. This highlights the oscillatory behaviour of the agent. These oscillations occur when the agent is stuck in a corner, and spends time to get out. In the fourth attempt, the agent spends so long oscillating that it dies due to its energy level reaching zero. Figure 16 shows the variation in energy level of this agent, the figure illustrates similar features to that in figure 9.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

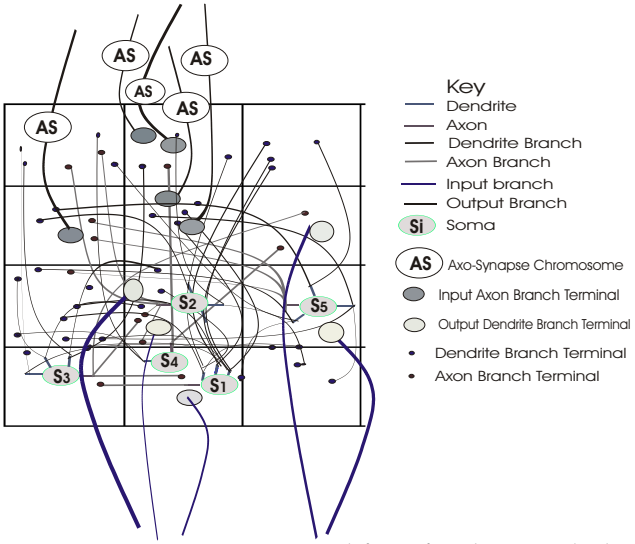

Figure 13: Neuron arrangement in CGPCN grid for a fixed network showing five neurons with different morphologies (dendrite and axon branches). Also showing the inputs and outputs to the network. The figure shows the exact locations of neurons and branches in the network. Not all connections are shown for clarity. Grid squares are numbered from left to right and top to bottom from 1 to 12.

Run.No	Found the gold	Home with gold
1	4	290
2	1	Never
3	2	Never
4	1	Never
5	3	Never
6	1	Never
7	1	334
8	1	Never
9	1	Never
10	1	245
11	1	Never
12	1	Never
13	1	Never
14	1	75
15	1	Never

Table 4: The number of generations taken by agents with fixed CGPCN network to perform various tasks in Wumpus World scenario starting with the different initial population in Fifteen (15) independent evolutionary runs.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

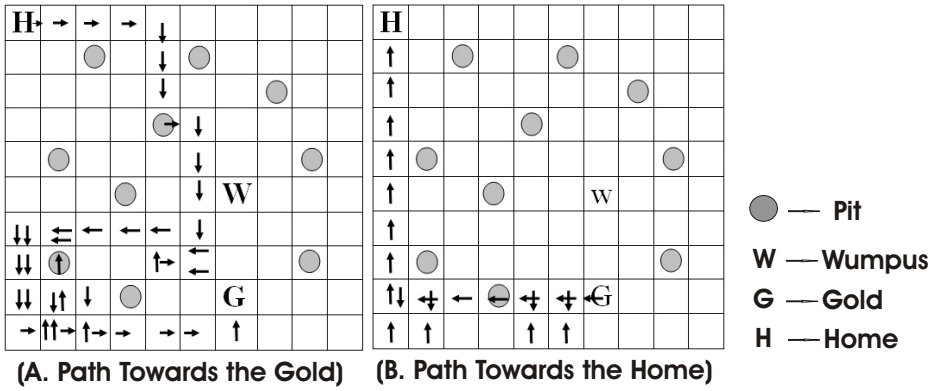

Figure 14: Path followed by the agent, starting from upper left corner (home square), A) from home toward the gold, B) from gold back to home.

It is worth noting that an agent that moves in a straight line (even at edges) is highly non-random. A random move at an edge would have 33% probability. Thus a straight path of 8-steps (as in figure 14b) returning home would have a probability of 0.0152%. So if it occurs, it is unlikely to be purely by chance. However, agent moves are of course, not randomly generated, and agents encountering the same environmental situation with similar internal state would be likely to act in a similar way. It is interesting and non-trivial that an agent whose network is highly dynamic can still carry out similar behaviour.

Now we examine the movements of the same agent. Figure 17 shows the different paths followed by the agent in four subsequent journeys. Figure 18 shows the variation in energy level, and figure 19 shows the movements of the agent in graphical form. Figure 17a is similar to Figure 14a, showing the first path followed by the agent toward the gold. The agent takes 135 steps to find the gold in the first of these journeys. Figure 19 highlights the oscillations of agent on the first journey. Figure 17b shows the path followed by the agent during its second journey (after being placed back home). On this occasion the agent took an almost straight path toward the gold with only slight oscillations along the way. It also encounters a pit once, as shown by the drop in energy level in figure 18. Figure 19 illustrates the reduced oscillatory behaviour during the second journey. It take agent 38 steps to complete the second path. Figure 17c shows the path followed by the agent on the third journey. The agent takes almost the same path as the second journey, but with more oscillations. It encounters two pits, but finally gets the gold, however finding the gold takes longer than on the second trip, taking 44 steps to complete its journey. In the fourth and final journey the agent follows a similar path to that of third, but with more oscillations and when it reaches a corner it gets stuck until its energy level decays to zero.

5.3 Competitive Learning Scenario

Coevolutionary computation is largely used in a competitive environment. In competitive coevolution, individual fitness is based on performance against an opponent. Thus fitness shows the relative strengths of solutions not the absolute strengths. These competing solutions create an "arms race" of increasingly better solutions (Dawkins and Krebs (1979); Rosin and Belew (1997); Van Valin (1973)). Feedback mechanisms

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

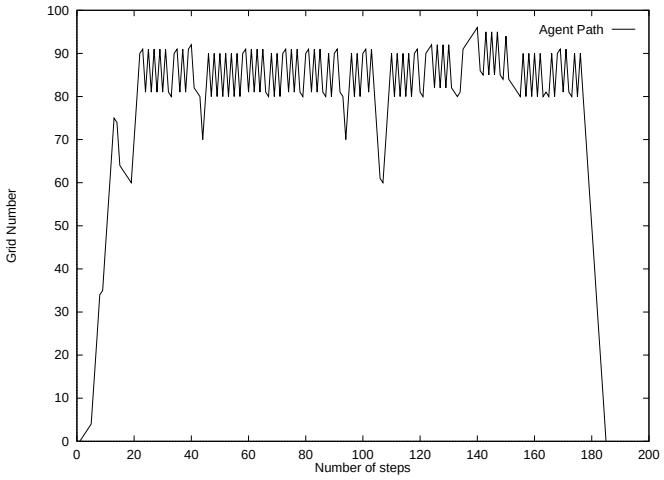

Figure 15: Path followed by the agent in graphical form, from home to gold and back to home, with grid points enumerated along rows from top left to bottom right (0 - 99). The graph highlights the oscillatory behaviour of the agent.

between individuals based on their selection produce a strong force toward complexity (Paredis (1995)). Competitive coevolution is traditionally used to evolve interactive behaviors which is difficult to evolve with an absolute fitness function. To encourage interesting and sophisticated strategies in competitive co-evolution every player's network should play against a high quality opposing player (Stanley and Miikkulainen (2004)).

When both agents are allowed to move around the environment the problem becomes dynamic and uncertain. We have performed a significant number of runs and experiments to test the capabilities of agents that are developed as result of evolution of their rules of development and environmental interactions in a dynamic environment.

5.3.1 Experimental setup

The performance of the CGPCN is also tested in a coevolutionary scenario, where *both* agent and Wumpus are provided with a CGPCN. The agent and Wumpus live in a two dimensional grid (10x10) containing ten pits (as shown in figure 20). In this scenario the Wumpus is also provided with a home square at the bottom right corner as shown in figure 20. This time we slightly modified the task for the agent. After the agent gets the gold, it is immediately placed back at its home square, and its task is to get the gold again. The agent has to get the gold as many times as it can during its life time while avoiding pits and Wumpus. The task of the Wumpus is to catch the agent as many times as it can. The job of the Wumpus is more difficult than the agent as the target for the Wumpus is mobile, while the target for the agent (the gold) is static. Both the Wumpus and the agent are placed back at their respective home squares each time the Wumpus catches the agent. The task of learning in the coevolutionary scenario is much harder than the single agent world as each time the Wumpus catches the agent it becomes more difficult for it to catch it again, as the agent learns and tries to find a path to the gold that avoids the Wumpus. Pits only affect the energy level of the

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

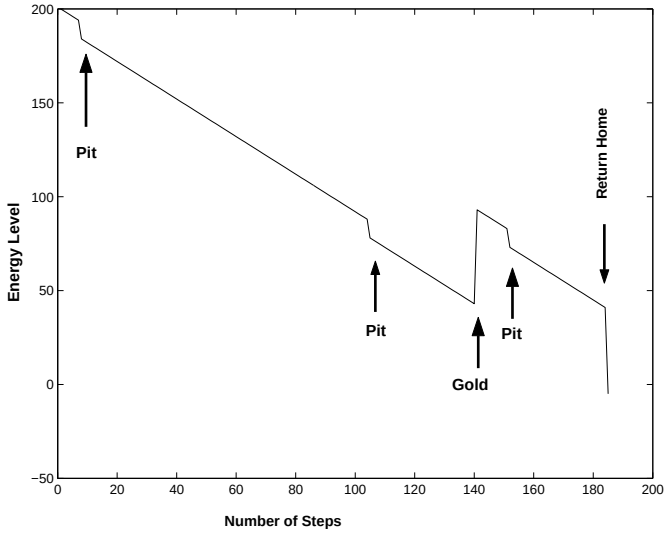

Figure 16: Variation in agent energy level shown against the number of completed steps during a journey from home to gold and back home. The energy shows a continual decrement of 1 per step, sudden drops when caught by a pit (steps 7, 104, 151), and a sudden increase when it finds the gold (step 140). The energy is decreased to zero when the agent returns home.

agent, the agent is weakened whenever it passes through squares containing pits. The Wumpus's home is diagonally opposite the agent's. Both the agent and the Wumpus perceive a breeze in squares adjacent to the pits and a smell in the squares adjacent to each other directly (not diagonally). They also perceive a glitter while passing close to the gold. They receive different signals while passing through these locations from different directions. 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 zero signal. As the pits and the gold only directly affect the agent, the Wumpus 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 agent in order to catch it. Both agent and Wumpus can perceive only one signal on a grid square. The priority of the signals for agent is in the following order: Wumpus, gold, pit, stench, glitter, breeze. Whereas for Wumpus it is in the following order: agent, gold, pit, agent shade, glitter, breeze. Both the agent and the Wumpus continue to exist as long as their energy level remains above zero (see next

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

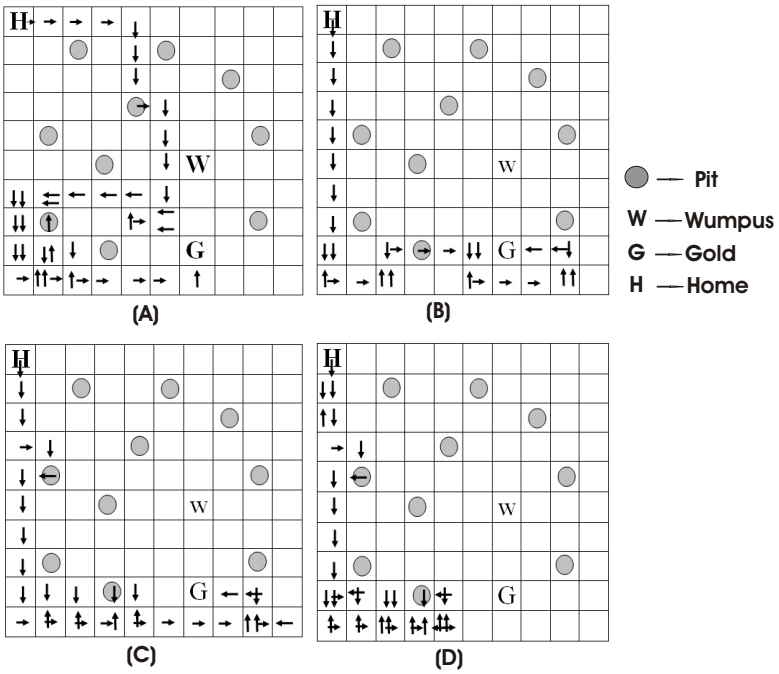

Figure 17: Different paths followed by the same agent in four cases from the home square towards the gold, a) first path toward the gold starting with an initial random CGPCN, b) Second path followed by the agent after bringing the gold home, note the comparatively shorter path followed, c) In the third case, the agent tried to find the shortest possible path toward the gold, ending up caught twice by a pit, resulting in a longer journey than the second case, and finally, d) Fourth case where it followed a similar path to that of case c, but ends up oscillating which causes the agent to die.

section for more details).

Both the agent and the Wumpus are assigned an initial energy level of 100 units. Both the agents have one life cycle i.e. they will be in the environment as long as their energy level is higher than zero. Both the agent and Wumpus move one square at a time. They do not have the option to stay in the same place. They always have to move in any of the possible directions directly (not diagonally). If the agent is caught by the Wumpus its energy level is reduced by 60%, if it falls into a pit its energy level is reduced by 10 units, if it gets the gold its energy level is increased by 60%. The Wumpus is not affected by anything except that its energy level is increased by 60% every time it catches the agent. For each single move both agent and Wumpus have their energy level reduced by 1 unit.

The fitness of both the agent and the Wumpus is accumulated over their lifetimes (while their energy level is non-zero) in the following way:

- For each move, the fitness of both the agent and Wumpus is increased by one.
- Every time the agent obtains the gold, its fitness is increased by 1000.
- Every time the Wumpus catches the agent, its fitness is increased by 1000.

Each of five agent population members are tested against the best performing

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

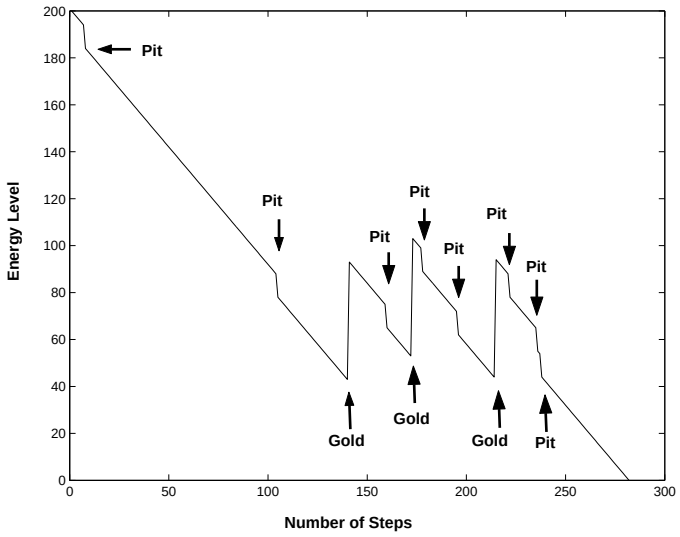

Figure 18: Variation in agent energy level during the four journeys illustrated in figure 17. The energy shows a continual decrement of 1 per step, sudden increases when the gold is retrieved (steps 40, 172, 214), and sudden drops when caught by a pit (steps 24, 104, 159, 178, 195, 221, 235, 237). In this case the agent dies from the gradual decline in energy with each step as it exhibits oscillatory behaviour.

Wumpus genotype from the previous generation. Similarly each of the five Wumpus population members are tested against the best performing agent genotype from the previous generation. The initial random network is the same for both the agent and the Wumpus. It is the genotypes in each generation which generate different networks and their associated functionality. The best agent and Wumpus genotypes are selected as the respective parents for the new population.

The idea behind these experiments is twofold. Firstly we want to demonstrate that agents learn during their life time (i.e. post evolution) while their energy level is above zero, thus demonstrating that evolution has evolved the ability to learn. Secondly we wish to demonstrate that the learning capability is obtained as a result of evolution by an agent and can therefore be transferred from generation to generation through the genetic code. The CGPCN setup (number of neurons, neurites and threshold parameters) is the same as for first case both for the agent and the Wumpus.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

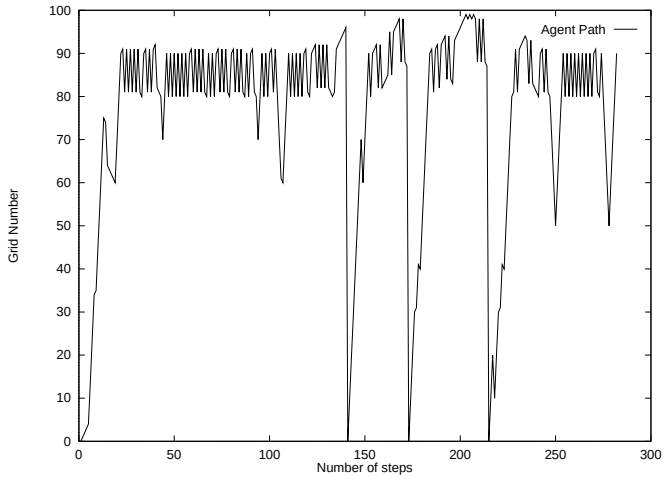

Figure 19: Path followed by the agent in graphical form for the four journeys illustrated in figure 17, with grid points enumerated along rows from top left to bottom right (0 - 99). The oscillatory behaviour of the agent is highlighted in this graph. The agent exhibits more oscillations in the first journey, while trying to stabilize its network, and less oscillations in the second and third journeys when it took less time to find the gold.

5.3.2 Results and Analysis

Figure 21 shows how the fitness of the agent and Wumpus change, in one particular evolutionary run, over 1250 generations. It is evident that there are increases and decreases in fitness at different stages for each agent and Wumpus with either of them having the "upper hand".

The y-ordinate divided by 1000 gives the number of times an agent or Wumpus achieves its goals. There are different kinds of dynamic behaviors that the agent and Wumpus adopt due to the interactions between neurons, resulting from their internal genetic codes. At the start, both the agent and Wumpus do not know anything about the gold, pits, their interaction with each other, and the signals that indicate the presence of these objects nearby. As they evolve they develop their own memory of life experiences and their ability to do this is encoded genetically. Each agent starts with a fixed randomly assigned neural structure that develops during the agent's lifetime making it capable to achieve the goal. The fitness graphs in figure 21 show that initially, the fitness of the agent and Wumpus varies a great deal. However, as the number of generations increases, both the agent and Wumpus become more skillful and the frequency of variations in the fitness reduces. This suggests that the two CGPCN networks behave so that one of them benefits at the expense of the other. There are also some points along the fitness graph when the fitness of both the agent and Wumpus goes down, this occurs when they are both unable to achieve their goals. Often in the very next generation, however, they find a way to achieve their goals. It is interesting to observe that around generation 680, both agent and Wumpus become reasonably skillful. The agent obtains the gold twice (and on one occasion 3 times), while at the same time the Wumpus catches the agent twice (and later three times). This is followed by agent and Wumpus having fluctuating fortunes. Following that, at around genera-

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

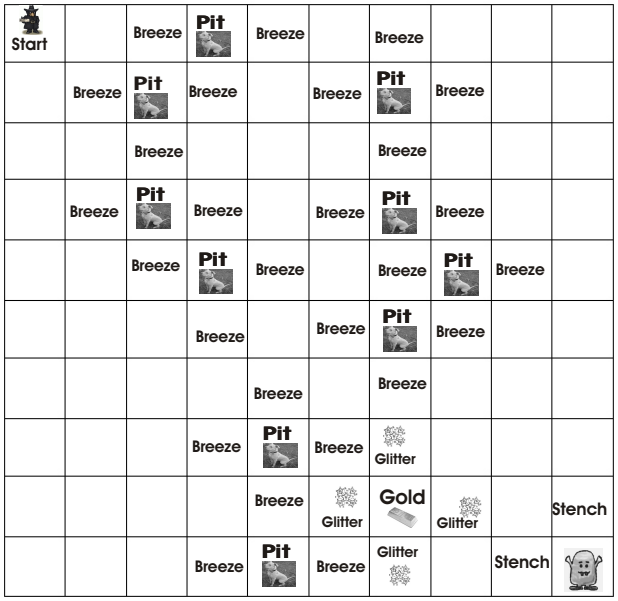

Figure 20: A two dimensional grid 10x10, showing Wumpus World environment, having one Wumpus starting from bottom right corner, one agent on the top left corner square, ten pits at various place and gold.

tion 1100 both the agent and Wumpus achieving are more successful in achieving their goals, with the Wumpus being the more successful.

5.3.3 Further Analysis

In further experimental analysis we looked at the behaviour of the agent and Wumpus in detail. Initial energy levels of the agent and Wumpus were increased to 300 (from 100 previously used during evolution). This allowed the agent to find the gold five times (see figure 22) while the Wumpus caught the agent once. Fig. 22 illustrates the movements of the agent (black arrows) and Wumpus (grey arrows) over the 6 journeys, the Wumpus died during its fourth outing (panel D), thus is not shown in panels E or F. Squares are numbered from 0 - 99 along rows from top left to bottom right, thus the gold is on square 86. In Figure 22A both the agent and Wumpus begin with the same randomly assigned initial networks which are converted into mature networks by running the seven CGP programs. The agent takes a fairly direct route toward the gold, encountering two pits along the way. The Wumpus spends a great deal of time on squares 98 and 88 and moves toward the gold just after the agent has been replaced on its home square after obtaining the gold. Figure 22B illustrates the next journey of the agent, which takes a different route to the gold (in fact following the minimum path)

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

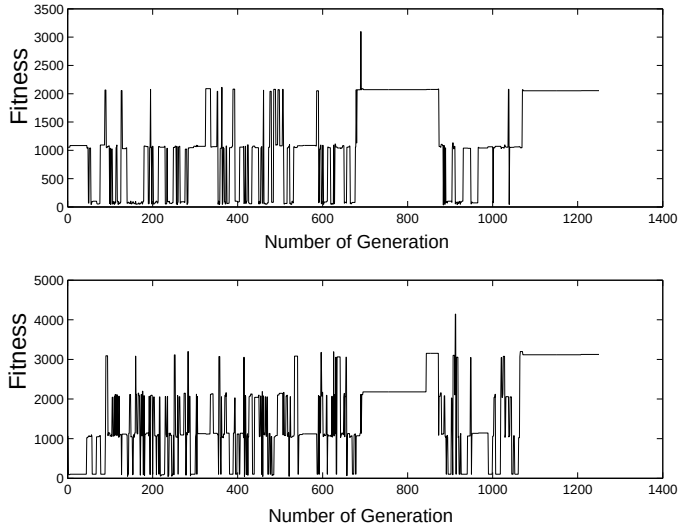

Figure 21: Variation in fitness of the best agent (top) and Wumpus (lower) with evolutionary generations. The traces show oscillatory behaviour in fitness variation of both at the start, with either being fitter from generation to generation. After 680 generations both the agent and Wumpus reach a stable behaviour with the agent getting the gold twice, and the Wumpus catching the agent two or three times. At generation 692 the agent retrieves the gold three times.

and avoiding all pits. Meanwhile, the Wumpus just lurks near the gold, spending all its time on squares 96 and 86. We generally found that the Wumpus exhibited this kind of behaviour. This is a good strategy for it, as the agent (if any good) will eventually come close to the gold and this gives the Wumpus a chance of catching it. On this occasion, the Wumpus is unlucky and jumps away from the gold at the same time that the agent obtains it. We did not allow the agents to remain stationary, they always have to make a move and go to a different square. Figure 22C illustrates their third journey. The agent follows an identical path to the gold as it did at the start. This is 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 in its network. However, when it arrives near the gold (square 96) it is attacked by the Wumpus and is relocated to its home square. Its subsequent behaviour (in Figure 22D) is interesting. It now follows a very different, and meandering, path to the gold. It spends some time alternating between squares 8 and 9, before turning back and arriving home again, only to set off

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

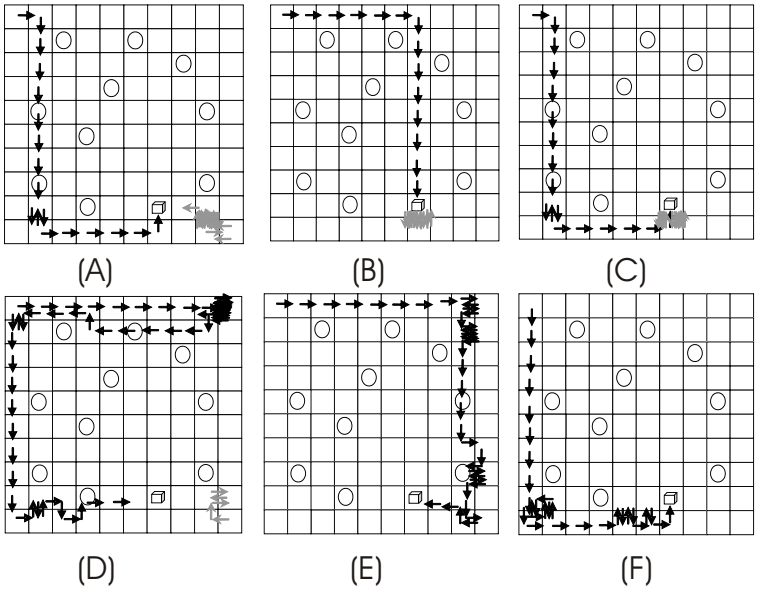

Figure 22: (A-F) Different paths followed by the agent and Wumpus in successive journeys of the coevolutionary task. The agent starts toward the gold from its home square in the upper left corner, the Wumpus starts toward the agent from its home square in the lower right corner. Movement of the agent is shown by black arrows, movement of the Wumpus by grey arrows. The pits are indicated by circles, the gold by a box. The Wumpus CGPCN dies during journey D, therefore it is not available in E-F.

down the left hand side, in the direction of the gold. The behaviour of the Wumpus is odd. Having been replaced to its home square (99) after attacking the agent, it moves around briefly in the bottom right four squares before its CGP computational network dies. This illustrates an interesting, but puzzling, phenomenon that we observed with other evolved agents and Wumpuses. Often, their CGPCN dies when their energy level is a small number and becomes more active (with many branches and synapses) when they have a high value of energy level. This is puzzling, since the energy level is not supplied as an input to the CGPCN. Beneficial encounters often result in more neurons and branching, whereas deleterious encounters often result in removal of neurons and branches from the network. In figures 22E and 22F we see the subsequent behaviour of the agent, where it successfully obtains the gold again (panel E), and dies before it reaches the gold (panel F).

These results suggest that the agent produces a memory map of the environment early in its evolutionary history. It is interesting to note that after the agent is attacked by the Wumpus its CGPCN 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 Wumpus (with some minor variations), eventually finding the gold. In the subsequent trials it appears to be looking for the shortest path to the gold. Even after the events shown in figure 22E we found that when the agent is reallocated to its starting position, it again followed a short path to get to the gold, but unfortunately as it reached the gold its network died.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

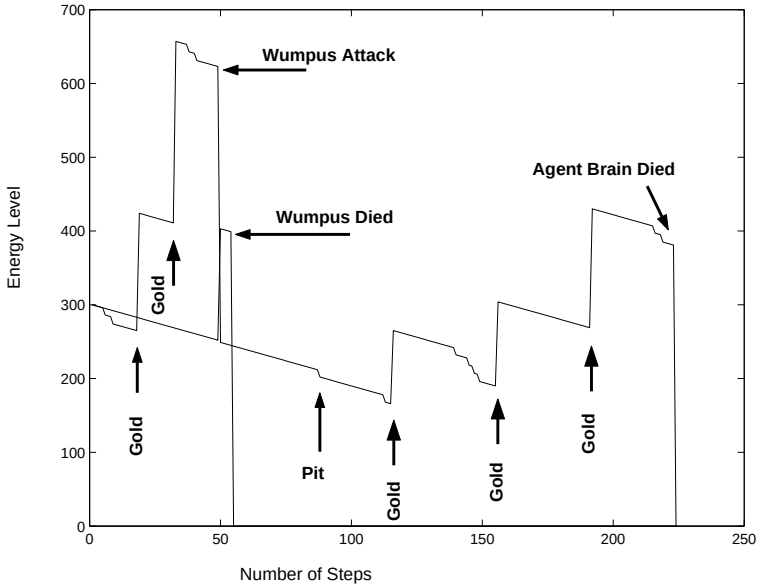

Figure 23: Variation in energy level of both agent and Wumpus during the consecutive journeys illustrated in A-F, figure 22. The energy shows a continual decrement of 1 per step, with larger changes when they achieve their respective goals. When the agent finds the gold, its energy is increased by 60%. When the agent encounters a pit the energy level is decreased by 10, and when caught by the moving Wumpus its energy level is decreased by 60%. The Wumpus has its energy level increased by 60% when it catches the agent. In the scenario illustrated, the agent finds the gold 5 times. The Wumpus catches the agent once, around step 50, shortly after this the Wumpus CGPCN dies, indicated by the drop to zero in energy.

We also examined the behaviour of this agent in a number of other situations. We removed all pits and found that the agent moved around the environment, apparently 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 Wumpus to square 56 (and disabled its ability to move from this square). This lies directly on the path of the first agent in Fig. 22B. We found that the behaviour of the agent was identical to its previous behaviour except that it was caught by the Wumpus (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 Wumpus. But this encounter affects its network and subsequently causes it to follow a totally different path to avoid the Wumpus. The agent and Wumpus used in these experiments came from generation 220, so they are not highly evolved. At this stage in evolution the agent has not fully learned what to do in any situation, so the agent does not fully respond to the presence of the Wumpus and the degree to which it influences its energy level. We ran the experiment for longer (more than 2000

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

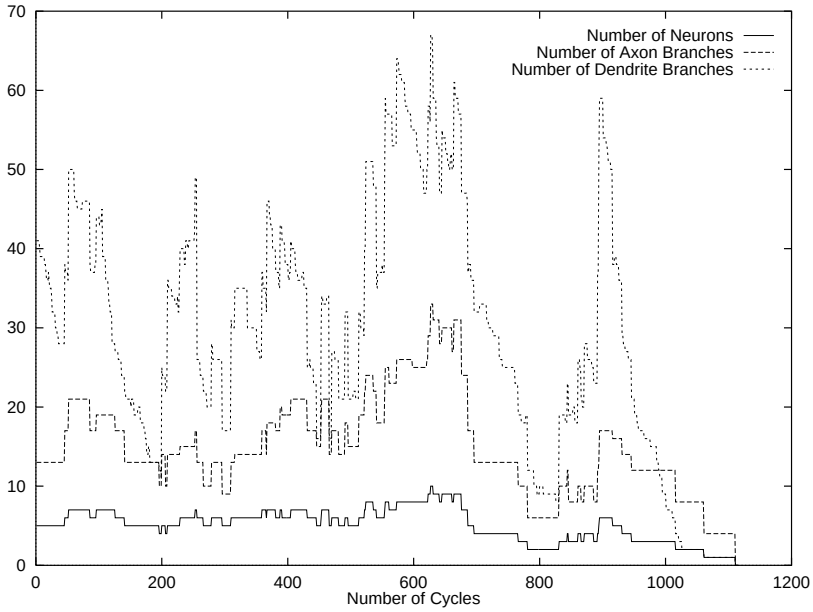

Figure 24: Variation in the numbers of neurons, dendrite branches and axon branches (plotted against number of cycles) of the agent's CGPCN during the co-evolutionary task illustrated in figure 22. Changes in the network structure appear to coincide with environmental events illustrated in figure 23. Note that each step in figure 23 requires 5 cycles to complete.

generations), but as the agent got better, so did the Wumpus, thus ending up with a stable point where the agent got the gold a few times and the Wumpus caught it a few times as evident from figure 21.

In earlier generations both the agents wander around in the Wumpus World environment. They do not achieve any goals (in fact they do not know what their goals are) thus causing their fitness value to be minimum, so they are not selected during the course of evolution. Once during evolution they happen to achieve a goal their fitness increases and they are selected for the next generation. From this point onwards these agents have a sense of their goal. Evolution make one agent a Wumpus and the other an agent that looks for gold. The results above (in Figure 22 and 23) are for well evolved agents when they have a sense of their goals.

During an agent's and Wumpus's lifetime the structural development and activity of the CGPCN changes considerably. We illustrate this by looking at the variation in energy level and the changes in network morphology during the lives of the agent and the Wumpus whose behaviour was shown in Figure 22. Figure 23 shows the variations in energy level of the agent and Wumpus. Changes in energy level reflect their experiences. For the agent, increases show the number of times it found the gold. Decreases are of two kinds: smaller decreases after encountering a pit, larger decreases when attacked by the Wumpus. Superimposed is a steady decrement in energy of 1 unit per step. Energy levels for agent and Wumpus only increase when they achieve their respective goals. When the Wumpus catches the agent its energy level is increased by

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

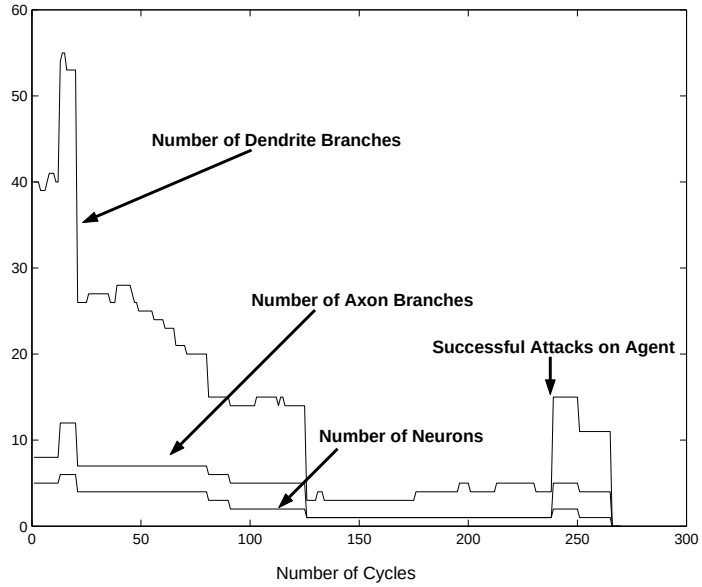

Figure 25: Variation in the numbers of neurons, dendrite branches and axon branches (plotted against number of cycles) of the Wumpus CGPCN during the co-evolutionary task illustrated in figure 22. There is a continuous drop in numbers over time, resulting in the death of the Wumpus at cycle 270 (step 54 in figure 23).

60%. It can be seen how the energy level of the agent drops rapidly (by 60%) immediately after it is attacked by the Wumpus. Shortly after this, rather strangely, all the neurons in the Wumpus CGPCN network die.

Figure 24 shows the variation in numbers of neurons, axon branches and dendrite branches during the agents life. Figure 25 shows the numbers for the Wumpus.

We have also observed that, in most cases once the agent is caught by the Wumpus it is never able to get the gold again during its lifetime because the interaction with Wumpus affected its network leading to neuron death. In order to assess the validity of this argument we tested the system by increasing its initial energy level as demonstrated in figure 22. It was only through doing this that we found an agent was able to get the gold after being caught by the Wumpus. However, in this case the Wumpus died soon after encountering the agent. It is evident that the network of the Wumpus was about to die (see figure 22) before it caught the agent, but the encounter appears

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

to have increased its life span by triggering a brief increase in the numbers of neurons and branches.

We also evolved agent and Wumpus with various values of initial energy. If the energy is increased (up to 300) it diminishes the influence of deleterious and beneficial environmental encounters. As it is decreased (to 50) the deleterious encounters (pits and Wumpus) outweigh the beneficial effects (obtaining gold). We found that an initial life of 100 proved to be a good balance.

The CGPCN builds the network and any learned behavioural memory in the environment, so the system 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.

6 Conclusion

We have presented a method for evolving programs that construct a dynamic computational network inspired by neuroscience. We used a technique called Cartesian Genetic Programming (CGP) to encode and evolve seven computational functions inspired by the functions of biological neurons. We evaluated this approach on a classic AI problem called Wumpus World and also a coevolutionary version of this environment. Results are 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 stability of the network and the ability to avoid pits and the Wumpus. It is not possible, at present, to compare directly the effectiveness 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 CGP computational network (CGPCN) 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 CGPCNs 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' (McCloskey and Cohen (1989), Ratcliff (1990))).

We have also described a co-evolutionary competitive learning environment in which the CGPCNs 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 an internal map of their environment.

The model presented in this paper is admittedly very complex and future work will look to see how it can be simplified. The compartmental nature of the genotype is useful as various chromosomes could be disabled or replaced with fixed rules. Careful experiments could then establish the importance of the various mechanisms. The model also requires extensive experiments using a variety of parameters using more and longer evolutionary runs. Unfortunately, running that many programs in a developed network is very computationally expensive and either the system needs to be evaluated on a large cluster of machines or the model needs to be simplified. We hope, however, that the model will encourage further investigation into more biologically-inspired models of developmental neural networks. Establishing a minimally complex but computationally powerful model will require the effort of many researchers over a substantial period of time.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

References

Alberts, B., Johnson, A., Walter, P., Lewis, J., Raff, M., and Roberts, K. (2002). *Molecular Biology of the Cell*. Garland Science, 3rd edition edition.

Arel, I., R. D. and Karnowski, T. (2009). A deep learning architecture comprising homogeneous cortical circuits for scalable spatiotemporal pattern inference. In *Proc. NIPS 2009 Workshop on Deep Learning for Speech Recognition and Related Applications*.

Becerra, J., Duro, R., and Santos, J. (2002). Self pruning gaussian synapse networks for behaviour based robots. In *Proceedings of CEC 2005 IEEE Congress on Evolutionary Computation, J.R. Dorronsoro (Ed.): ICANN 2002, LNCS 2415, Springer-Verlag Berlin Heidelberg*, page 837843.

Blynel, J. and Floreano, D. (2002). Levels of dynamics and adaptive behavior in evolutionary neural controllers. In *In*, pages 272–281. MIT Press.

Boers, E. J. W. and Kuiper, H. (1992). *Biological metaphors and the design of modular neural networks*. Masters thesis, Department of Computer Science and Department of Experimental and Theoretical Psychology, Leiden University.

Cangelosi, A., Nolfi, S., and Parisi, D. (1994). Cell division and migration in a 'genotype' for neural networks. *Network-Computation in Neural Systems*, 5:497–515.

Chalup, S. K. (2001). Issues of neurodevelopment in biological and artificial neural networks. *Proceedings of the Fifth Biannual Conference on Artificial Neural Networks and Expert Systems (ANNES'2001)*, pages 40–45.

Chen, X. and Hurst, S. (1982). A comparison of universal-logic-module realizations and their application in the synthesis of combinatorial and sequential logic networks. *IEEE Transactions on Computers*, 31:140–147.

Cunningham, P., Carney, J., and Jacob, S. (2000). Stability problems with artificial neural networks and the ensemble solution. *Artificial Intelligence in Medicine*, 20(3):217–225.

Dalaert, F. and Beer, R. (1994). Towards an evolvable model of development for autonomous agent synthesis. In *Brooks, R. and Maes, P. eds. Proceedings of the Fourth Conference on Artificial Life*. MIT Press.

Dawkins, R. and Krebs, J. R. (1979). Arms races between and within species. In *Proceedings of the Royal Society of London Series B*, volume 205, pages 489–511.

Debanne, D., Daoudal, G., Sourdet, V., and Russier, M. (2003). Brain plasticity and ion channels. *Journal of Physiology-Paris*, 97(4-6):403–414.

Downing, K. L. (2007). Supplementing evolutionary developmental systems with abstract models of neurogenesis. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 990–996, New York, NY, USA. ACM.

Federici, D. (2005). Evolving developing spiking neural networks. In *Proceedings of CEC 2005 IEEE Congress on Evolutionary Computation*, pages 543–550.

Floreano, D. and Urzelai, J. (2000). Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, 13:431–443.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

Frey, U. and Morris, R. (1997). Synaptic tagging and long-term potentiation. *Nature*, 6;385(6616):533–6.

Gaiarsa, J., Caillard, O., and Ben-Ari, Y. (2002). Long-term plasticity at gabaergic and glycinergic synapses: mechanisms and functional significance. *Trends in Neurosciences*, 25(11):564–570.

George, D. and Hawkins, J. (2005). A hierarchical bayesian model of invariant pattern recognition in the visual cortex. In *In Proceedings of the International Joint Conference on Neural Networks. IEEE*, pages 1812–1817.

Graham, B. (2002). Multiple forms of activity-dependent plasticity enhance information transfer at a dynamic synapse. *,J.R. Dorronsoro (Ed.): ICANN 2002, Springer-Verlag Berlin Heidelberg,ICANN*, LNCS 2415:45–50.

Gruau, F. (1994). Automatic definition of modular neural networks. *Adaptive Behaviour*, 3:151–183.

Harding, S., Miller, J. F., and Benzhaf, W. (2010). Developments in cartesian genetic programming:selfmodifying cgp. *GPEM*, 11(2):397–439.

Hebb, D. (1949). *The organization of behavior*. Wiley, New York.

Hillis, W. (1990). Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42:228–234.

Hinton, G. E. and Osindero, S. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18:2006.

Hornby, G., Lipson, H., and Pollack, J. B. (2003). Generative representations for the automated design of modular physical robots. *IEEE Transactions on Robotics and Automation*, 19:703–719.

Jakobi, N. (1995). *Harnessing Morphogenesis, Cognitive Science Research Paper 423, COGS*. University of Sussex.

Kandel, E. R., Schwartz, J. H., and Jessell, T. (2000). *Principles of Neural Science, 4rth Edition*. McGraw-Hill.

Kleim, J., Napper, R., Swain, R., Armstrong, K., Jones, T., and Greenough, W. (1998). Selective synaptic plasticity in the cerebellar cortex of the rat following complex motor learning. *Neurobiol. Learn. Mem.*, 69:274–289.

Koch, C. and Segev, I. (2000). The role of single neurons in information processing. *Nature Neuroscience Supplement*, 3:1171–1177.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural selection*. MIT Press.

Kumar, S. (Editor), B. J. (2003). *On Growth, Form and Computers*. Academic Press.

Lindenmeyer, A. (1968). Mathematical models for cellular interaction in development, parts i and ii. *Journal of Theoretical Biology*, 18:280–315.

London, M. and Husser, M. (2005). Dendritic computation. *Annual Review of Neuroscience*, 28:503–532.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

Malenka, R. (1994). Synaptic plasticity in the hippocampus: Ltp and ltd. *Cell*, 78:535–538.

Marcus, G. (2004). *The Birth of the Mind*. Basic Books.

Marcus, G. F. (2001). Plasticity and nativism: towards a resolution of an apparent paradox. pages 368–382.

McCloskey, M. and Cohen, N. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 24:109–165.

Miller, J. and Smith, S. (2006). Redundancy and computation efficiency in cartesian genetic programming. *IEEE Trans. Evol. Comp.*, 10:167–174.

Miller, J. F. (2004). Evolving a self-repairing, self-regulating, french flag organism. In *Proc of GECCO, Deb. K. et al (Eds.)*, volume 3102, pages 129–139.

Miller, J. F. and Thomson, P. (2000). Cartesian genetic programming. In *Proc. of the 3rd European Conf. on Genetic Programming*, volume 1802, pages 121–132.

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.

Miller, J. F., Vassilev, V. K., and Job, D. (2000). Principles in the evolutionary design of digital circuits-part i. *genetic programming*. volume 1:1/2, pages 7–35.

Nguy en, Q.T., L. J. (1996). Mechanism of synapse disassembly at the developing neuromuscular junction. *Curr Opin Neurobiol*, 6:104–112.

Nolfi, S. and Floreano, D. (1998). Co-evolving predator and prey robots: Do 'arm races' arise in artificial evolution? *Artificial Life*, 4:311–335.

Nolfi, S., Miglino, O., and Parisi, D. (1994). Phenotypic plasticity in evolving neural networks. in gaussier, d.p, and nicoud, j.d., eds. In *Proceedings of the International Conference from perception to action*. IEEE Press.

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, Springer-Verlag Berlin Heidelberg,ICANN*, LNCS 2415:896–901.

Paredis, J. (1995). Coevolutionary computation. *Artificial Life*, 2:355–375.

Parisi, D. (1997). Artificial life and higher level cognition. *Brain and Cognition*, 34:160–184.

Parisi, D. and Nolfi, S. (2001). *Development in Neural Networks. In Patel, M., Honovar, V and Balakrishnan, K.eds. Advances in the Evolutionary Synthesis of Intelligent Agents*. MIT Press.

Penn, A. A. and Shatz, C. J. (1999). Brain waves and brain wiring: The role of endogenous and sensory-driven neural activity in development. *Pediatric Research*, 45:447–458.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

Quartz, S. and Sejnowski, T. (1997). The neural basis of cognitive development: A constructivist manifesto. *Behav. Brain. Sci*, 20:537–556.

Ratcliff, R. (1990). Connectionist models of recognition and memory:constraints imposed by learning and forgetting functions. *Psychological Review*, 97:205–308.

Roberts, P. and Bell, C. (2002). Spike-timing dependent synaptic plasticity in biological systems. *Biological Cybernetics*, 87:392–403.

Roggen, D., Federici, D., and Floreano, D. (2007). Evolutionary morphogenesis for multi-cellular systems. *Journal of Genetic Programming and Evolvable Machines*, 8:61–96.

Rose, S. (2003). *The Making of Memory: From Molecules to Mind*. Vintage.

Rosin, C. D. and Belew, R. K. (1997). New methods for competitive evolution. *Evolutionary Computation*, 5.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence, A Modern Approach*. Prentice Hall.

Rust, A., Adams, R., and H., B. (2000). Evolutionary neural topiary: Growing and sculpting artificial neurons to order. In *Proc. of the 7th Int. Conf. on the Simulation and synthesis of Living Systems (ALife VII)*, pages 146–150. MIT Press.

Rust, A. G. and Adams, R. (1999). Developmental evolution of dendritic morphology in a multi-compartmental neuron model. In *Proc. of the 9th Int. Conf. on Artificial Neural Networks (ICANN'99)*, volume 1, pages 383–388. IEEE.

Rust, A. G., Adams, R., George, S., and Bolouri, H. (1997). Activity-based pruning in developmental artificial neural networks. In *Proc. of the European Conf. on Artificial Life (ECAL'97)*, pages 224–233. MIT Press.

Sebastian Risi, C. E. H. and Stanley, K. O. (2010). Evolving plastic neural networks with novelty search. *Adaptive Behavior*, 18(6):470–491.

Shepherd, G. (1990). *The synaptic organization of the brain*. Oxford Press.

Sims, K. (1994). Evolving 3d morphology and behavior by competition. In *Artificial life 4 proceedings*, pages 28–39. MIT Press.

Smythies, J. (2002). *The Dynamic Neuron*. BradFord.

Soltoggio, A., Bullinaria, J., Mattiussi, C., Dürr, P., and Floreano, D. (2008). Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In Bullock, S., Noble, J., Watson, R., and Bedau, M. A., editors, *Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, pages 569–576. MIT Press, Cambridge, MA.

Song, S., Miller, K., and Abbott, L. (2000). Competitive hebbian learning through spike-time -dependent synaptic plasticity.

Spector, L. (1996). Simultaneous evolution of programs and their control structures. In Angeline, P. J. and K. E. Kinnear, J., editors, *Advances in Genetic Programming 2*, pages 137–154, Cambridge, MA, USA. MIT Press.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

Spector, L. and Luke, S. (1996). Cultural transmission of information in genetic programming. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 209–214, Stanford University, CA, USA. MIT Press.

Stanley, K. and Miikkulainen, R. (2004). Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 3:21:63–100.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural network through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

Stuart, G., Spruston, N., and Hausser, M. e. (2001). *Iterative Broadening: Dendrites*. Oxford University Press.

Sutskever, I. and Hinton, G. (2006). Learning multilevel distributed representations for high-dimensional sequences. Technical report.

Taylor, G. W., Hinton, G. E., and Roweis, S. (2006). Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems*, page 2007. MIT Press.

Terje, L. (2003). The discovery of long-term potentiation. *Philos Trans R Soc Lond B Biol Sci*, 358(1432):617–20.

Traub, R. (1977). Motoneurons of different geometry and the size principal. *Biological Cybernetics*, 25:163–176.

Urzelai, J. and Floreano, D. (2001). Evolution of adaptive synapses: Robots with fast adaptive behavior in new environments. *Evolutionary Computation*, pages 495–524.

Van Ooyen, A. and Pelt, J. (1994). Activity-dependent outgrowth of neurons and overshoot phenomena in developing neural networks. *Journal of Theoretical Biology*, 167:27–43.

Van Rossum, M. C. W., Bi, G. Q., and Turrigiano, G. G. (2000). Stable hebbian learning from spike timing-dependent plasticity. *Journal of Neuroscience*, 20:8812–8821.

Van Valin, L. (1973). A new evolutionary law. *Evolution Theory*, 1:130.

Vassilev, V. K. and Miller, J. F. (2000). The advantages of landscape neutrality in digital circuit evolution. In *Proc. of the 3rd ICES, Springer-Verlag*, volume 1801, pages 252–263.

Yamauchi, B. and Beer, R. (1994). Integrating reactive, sequential, and learning behavior using dynamical neural networks. In *Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3: from animals to animats 3*, pages 382–391, Cambridge, MA, USA. MIT Press.

Yob, G. (1975). Hunt the wumpus. *Creative Computing*, pages 51–54.

Yu, T. and Miller, J. (2001). Neutrality and the evolvability of boolean function landscape. In *Proc. of the 4th EuroGP, Springer-Verlag*, pages 204–217.

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

A Operation of the Network

Information processing in the network starts by selecting the list of active neurons in the network (those with $statefactor = 0$) and processing them in a random sequence. This sequence is called a cycle. The processing of neural components is carried out in time-slices so as to emulate parallel processing. Each neuron takes the signals from the dendrites by running the dendrite electrical processing chromosomal program to update the potentials in the dendrites. These potentials from all dendrites are averaged and applied to the soma CGP program together with the soma potential. This program is run to obtain the final value of soma potential, which is used to decide whether to fire or not. If the soma fires, the action potential signal is transferred to other neurons through axosynaptic branches. The same process is repeated in all neurons.

After each cycle in the neural network, the potential and *statefactor* of the soma and the branches are reduced. The *statefactor* is decremented by 1 (unless it is already zero). The potential is reduced by 5%. The reduction in potential occurs to emulate the ubiquitous leakage effects that occur in excitable membranes in the brain. The reduction in *statefactor* moves inactive branches and neurons incrementally toward an active state ($statefactor = 0$). After a user defined number of cycles of the network (5 cycles in this case), the *health* and weights of all neurons and branches are reduced by 5%. This introduced a tendency for neurons and branches to fade away, therefore evolution is responsible for producing program that maintain stability against this process of background decay.

A.1 CGPCN Operation

1. **Initialise network and parameters**.

1.1. *Produce a random neural network* (the default structure) by specifying the following network properties. Default values for the present study are indicated in brackets.

1.1.1. The initial number of neurons (5).
1.1.2. Maximum number of Dendrites per neuron (5).
1.1.3. Maximum number of branches per dendrite (5).
1.1.4. Maximum number of branches per axon (5).
1.1.5. Maximum soma *statefactor* (3).
1.1.6. Maximum branch *statefactor* (7).
1.1.7. Mutation rate (5%).
1.1.8. Dimension of 2D space for neurons, rows and columns in CGPCN grid (3×4).
1.1.9. Threshold value for soma *health*, dendrite branch *health* and axon branch *health* that triggers soma or branch death (10% of Maximum value (255, 8-Bits)).
1.1.10. Threshold value for soma *health*, dendrite branch *health* and axon branch *health* that triggers soma or branch offspring (90% of Maximum value).
1.1.11. Fixed reduction after each step for soma *health*, dendrite branch *health* and axon branch *health* (5%).
1.1.12. Reduction factor after each step for soma *weight*, dendrite branch *weight* and axon branch *weight* (5%).
1.1.13. Fixed reduction after each cycle for soma potential, dendrite branch potential and axon branch potential (5%).
1.1.14. *Initial Structure*: A number of neurons are generated with a random number of dendrites, each dendrite has a random number of branches. Each

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

neuron has one axon with a random number of axon branches. The neurons and branches (dendrite & axon) are placed at different random locations in the 2-D CGPCN grid. An initial value of *health* is assigned to soma and branches. A random *weight* value is assigned to each soma, each dendrite branch and each axon branch. Branches are also assigned a random value of *resistance*. Thresholds for all operations are specified.

1.2. *Create initial population of genotypes*, each consisting of seven chromosomes. Values used indicated in brackets.

1.2.1. Specify the number of offspring (λ) in the $(1 + \lambda)$ evolutionary strategy (4).

1.2.2. Number of nodes per chromosome (100).

1.2.3. The set of node functions (4, specified in section 3) and the number of connections per node (3).

1.3. Specify the number of inputs and output of neural network, and distribute them at random locations in the network (5 inputs; 5 outputs).

2. **Processing in the CGPCN**

3. When applying the external input to the network obey the following rules. Note the network is run for five cycles (5 cycles = 1 step) before taking the output (see 3.11.).

3.1. Every time the soma fires, the soma threshold is set to a higher value(255), and the soma life cycle is run. After firing the soma is kept inactive for a number of cycles by assigning its state factor to the maximum value (3).

3.2. If the soma potential is below a threshold (25% of maximum 255), its state factor remains high so that the soma is kept inactive.

3.3. If the soma potential is in an intermediate range (above the inactivity threshold and below the firing threshold), the soma is kept inactive for the next cycle.

3.4. The activity of the branches (via their *statefactor*) are also affected by the transformation (the change in potential during axo-synapse and dendrite electrical processing) of potential across it, if the modification in signal is more it is kept active otherwise inactive, also its life cycle is run only if it is active.

3.5. The electrical processing in the soma is affected only by the active dendrite branches. Once the neuron fires, the signal is transferred only through active axon branches, if none of the axon branches is active the signal is not transferred to other neurons.

3.6. The action potential signal is only transferred to one of the active dendrite branches in the same CGPCN grid square as the axon branch (making synapse with dendrite branch in vicinity, as done in biology.

3.7. The potential of each soma, each dendrite branch, and each axon is reduced by a fixed amount after every cycle (5% in this case).

3.8. After 1 step, the *health* and *weight* of each soma, each dendrite branch and each axon branch are reduced by a fixed amount (5% in this case).

3.9. The soma threshold potential reduction after each cycle is twice the reduction applied to the soma potential.

3.10. The *statefactor* of all soma and branches where $statefactor > 0$ are reduced by one unit after every cycle, allowing them to move toward activity.

3.11. **Run the network for one step**.

3.11.1. repeat for all inputs to the network.

3.11.1.1. Find the location of each input to the CGPCN grid. Select the active dendrite branches in that grid square. The input is applied to the

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

network using vector CGP processing, where 10 copies of the input are applied as 'virtual' inputs to the axo-synapse electrical chromosome (CGP Program). The axo-synapse program updates the values of the dendrite branch potentials.

3.11.1.2. At the start of every cycle, select the list of all the active neurons. Start processing each **neuron** one by one in random order.

3.11.1.3. For each neuron, start processing each **dendrite** for any signal received from other neurons.

3.11.1.3.1. For each **dendrite** select all the active **dendrite branches** attached to it, and apply their potential values along with a biased soma potential (biased by creating 10 copies in the input vector) to the CGP dendrite electrical processing chromosome. This produces their updated values as output.

3.11.1.3.2. Each branch potential, P, is further updated with contributions from *resistance*, *weight* and *health* using the following equation:

$$P = \left(\acute{P} + 0.02H + 0.05W - 0.05R\right) \ \& \ \text{mask}$$

Where P is the final value of potential, and $\acute{P}$ is the potential updated by the CGP program. H, W and R are the *health*, *weight* and *resistance* of the dendrite branch respectively. The processed potential is masked to avoid overflow, based on the number of bits used for CGP processing, the $\&$ operation is a logical `AND`.

3.11.1.3.3. The **life cycle program of any active dendrite branch is run** by applying its *resistance* and *health* as input to the CGP dendrite branch life cycle chromosome. This updates these values, and depending on the updated value of *resistance* a decision on whether to move the branch from the current location is taken. The *health* decides whether the branch should produce offspring, die or remain the same. Producing offspring results in creation of another branch at the same location with an initial *health*, and random *weight* and random *resistance* values. If the branch dies it is removed from the dendrite.

3.11.1.3.4. The same process is repeated for all the dendrites and their branches. After processing all dendrites, the average potentials of all the dendrites is taken, which in turn is the average of all the active dendrite branches attached to them. This grand average potential and the soma potential are applied as inputs to the CGP soma electrical processing chromosome. This produces the updated value of the soma potential ($\acute{P}$) as output.

3.11.1.3.5. This updated soma potential is further processed to create a final potential using the *health*, H, and *weight*, W of the soma using the following equation:

$$P = \left(\acute{P} + 0.02H + 0.05W\right) \ \& \ \text{mask}$$

As for the dendrite potential, the processed potential is masked to avoid overflow, based on the number of bits used.

3.11.1.3.6. After processing, the **soma potential is compared with the soma threshold potential**, if it is higher then the soma fires. This means that the soma potential is set to the maximum value of

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

255 (8-bits). The soma threshold potential and soma *statefactor* are set to their maximum values, thus keeping the soma inactive for a number of cycle (creating a refractory period).

3.11.1.3.7. **If the soma fires its life cycle is run**, and the potential is transferred to other neurons through axo-synaptic branches by running the CGP axo-synaptic electrical processing chromosome.

3.11.1.3.8. **If the soma does not fire**, the value of processed potential is noted, and the following actions taken:
- If it is below one third of the maximum value, its *statefactor* is set to maximum value(7), therefore it is kept inactive for maximum number of cycles.
- If its value is above one third and below half then it is kept inactive for two cycle.
- If it is higher then half the maximum value the soma is kept active for the next cycle, and its life cycle is run.

3.11.1.3.9. **If the soma life cycle is run**, it takes the *health* and *weight* of the soma as input and produces updated values as output. If the soma *health* is below the soma life threshold (one tenth of the maximum in this case) it will die, which removes the neuron and all its branches from the network. If its value is above soma offspring threshold (90% of maximum(255) in this case), then it will produce another neuron in the network at the same location with a random number of dendrites and branches.

3.11.1.3.10. **If the soma fires** the signal is transferred to other neurons. This is done as follows: **the axo-synaptic CGP electrical processing chromosome is run for any active axon branches**. This program selects the potential of all active dendrite branches in its vicinity, and applies these with a biased soma potential (10 copies of soma potential in the input vector) as inputs to the CGP electrical processing chromosome. The chromosome updates potentials of the dendrite branches, along with the axo-synaptic potential. Next the axo-synaptic potential is processed using the same equation as for the dendrite branches, except that P is the synaptic potential.
The weight processing CGP chromosome is run, this takes the *weight* of any active dendrite branches in the vicinity of the axon branch and its own axo-synaptic *weight* as input and generates an updated weights as output. The axo-synaptic potential is assigned (replacing the existing value) to the dendrite branch whose *weight* is the maximum after *weight* processing.

3.11.1.3.11. **After axo-synaptic electrical processing**, if the potential of the axo-synaptic branch is raised above a threshold level(20%), it is kept active and its life cycle is run.

3.11.1.3.12. The life cycle of axo-synapse takes the *health* and *resistance* of the axon as input and updates their values. The change in *resistance* of the branch is compared with a threshold value to decide whether the branch should move or stay at the same location. If the *health* of the branch is above the offspring threshold, a new branch is generated at the same location, with an

Evolutionary Computation Just Accepted MS.
doi:10.1162/EVCO_a_00043

initial *health* and random *weight* and *resistance*. If the health falls below the life threshold the axon branch dies and is removed from the axon.

3.11.1.3.13. The same process is repeated in all the axon branches.

3.11.2. **Taking output from the network**: The output branches are affected by the network processes, through their updated potential values and after every five cycles the output is taken by reading the potential values of the virtual dendrite branches (output branches), and averaging these to generate an output potential.

3.12. After completing all steps. Assess fitness:

3.12.1. The one with highest fitness is selected.

3.12.2. Using mutation new offspring chromosomes are produced and we repeat the network process again.