# Evolution of Cartesian Genetic Programs Capable of Learning

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

Julian F. Miller
Intelligent System Design Group
Electronics Department
University of York
jfm7@ohm.york.ac.uk

## ABSTRACT

We propose a new form of Cartesian Genetic Programming (CGP) that develops into a computational network capable of learning. The developed network architecture is inspired by the brain. When the genetically encoded programs are run, a networks develops consisting of neurons, dendrites, axons, and synapses which can grow, change or die. We have tested this approach on the task of learning how to play checkers. The novelty of the research lies mainly in two aspects: Firstly, chromosomes are evolved that encode programs rather than the network directly and when these programs are executed they build networks which appear to be capable of learning and improving their performance over time solely through interaction with the environment. Secondly, we show that we can obtain learning programs much quicker through co-evolution in comparison to the evolution of agents against a minimax based checkers program. Also, co-evolved agents show significantly increased learning capabilities compared to those that were evolved to play against a minimax-based opponent.

## Categories and Subject Descriptors

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

## General Terms

Algorithms, Design, Performance

## Keywords

Cartesian Genetic Programming, Computational Development, Co-evolution, Artificial Neural Networks, Checkers

## 1. INTRODUCTION

In our view the process of biological development underpins learning. Since in biology all learning occurs during development, and DNA does not in itself encode learned information. This raises the question: How is a *capability* for learning encoded at a genetic level? We are also interested in finding out how important for learning, is the interaction between two systems developing in response to each other? In this paper, we evolve genotypes that encode programs that *when executed* gives rise to a neural network that plays checkers. In particular, we demonstrate how important it is to co-evolve and co-develop two agents, instead of evolving and developing a single agent for learning.

Following Khan et al. the genotype we evolve is a set of computational functions that are inspired by various aspects of biological neurons [10]. Each agent (player) has a genotype that grows a computational neural structure (phenotype). The initial genotype that gives rise to the dynamic neural structure is obtained through evolution. As the number of evolutionary generations increases the genotypes develop structure that allow the players to play checkers increasingly well.

Our method employs very few, if any, of the traditional notions that are used in the field of Artificial Neural Networks. Unlike traditional ANNs we do not evolve or directly adjust set of numbers that defines a network. We run evolved programs that can adjust the network indefinitely. This allows our network to learn while it develops during its lifetime. The network begins as small randomly defined networks of neurons with dendrites and axosynapses. The job of evolution is to come up with genotypes that encode *programs* that when *executed* develop into mature neural structures that learn through environmental interaction and continued development.

ANNs can only solve a specific problem as they model learning through synaptic weights. Whereas memory and learning in brains is caused by many other mechanisms. Synaptic weights are only responsible for extremely short term memory. Also if very complex tasks are required to be solved with say, billions of weights, current traditional approaches won't scale. In principle ours will as the network complexity is not related to the complexity of the evolved programs. So in a nutshell we choose to model at this particular level of abstraction because we feel it has the plasticity we need and will scale better. What we do is *inspired* by biology. We are not trying to model biology. We expect the additional model complexity to pay off when we allow it to develop in interaction with the environment over long time scales and on different problems simultaneously.

There are a number of techniques in which an agent can


*GECCO'09,* July 8–12, 2009, Montréal Québec, Canada.

the highest potential makes the jumping move. In addition, there are also five output dendrite branches distributed at random locations in the CGPCN grid. The average value of these branch potentials determine the direction of movement for the piece. Whenever a piece is removed its dendrite branch is removed from the CGPCN grid.

## 7.2 CGP Computational Network (CGPCN) Setup

The CGPCN is arranged in the following manner for this experiment. Each player CGPCN has neurons and branches located in a 4x4 grid. Initial number of neurons is 5. Maximum number of dendrites is 5. Maximum number of dendrite and axon branches is 200. Maximum branch *statefactor* is 7. Maximum soma *statefactor* is 3. Mutation rate is 5%. Maximum number of nodes per chromosome is 200. Maximum number of moves is 20 for each player.

## 7.3 Fitness Calculation

The fitness of each agent is calculated at the end of the game using the following equation:

$Fitness = A + 200(K_P - K_O) + 100(M_P - M_O) + N_M$,

Where $K_P$ represents the number of kings, and $M_P$ represents number of men (normal pieces) of the player. $K_O$ and $M_O$ represent the number of kings and men of the opposing player. $N_M$ represents the total number of moves played. A is 1000 for a win, and zero for a draw. To avoid spending much computational time assessing the abilities of poor game playing agents we have chosen a maximum number of moves. If this number of moves is reached before either of the agents win the game, then A =0, and the number of pieces and type of pieces decide the fitness value of the agent.

# 8. RESULTS AND ANALYSIS

In two independent evolutionary runs we evolved agents against MCP (evolution) and co-evolved agents for one thousand (1000) generations. Then we took the best players from generations 50 to 1000 (in 50 generation intervals) from the co-evolutionary runs and let them play against the players evolved against the MCP at the same generation. In this way we could assess whether co-evolved players play better at the same generation than the agents that played only against the professional checker software (whose level of play does not change during the course of game). We evaluate their performance over the five game series by calculating their average fitness using the fitness function that was used in evolution. It is important to note that over the five game series *there is no evolution*. We just begin with a small random network and run the programs that were evolved at the generation in question over the sequence of five games.

In Figure 5 we have plotted the average fitness of both co-evolved and evolved player when playing each other in a five game series for different generations. The co-evolved player in almost every case beats the evolved player by a large margin. We also repeated these experiments under exactly the same conditions but where the players played a ten game sequence of games. In Figure 6 we have plotted the average fitness in the same way as before. Comparing the two figures, reveals that the players that were obtained through co-evolution perform even better than the five game players against the same players evolved against the MCP. This indicates that on average the players who play a ten

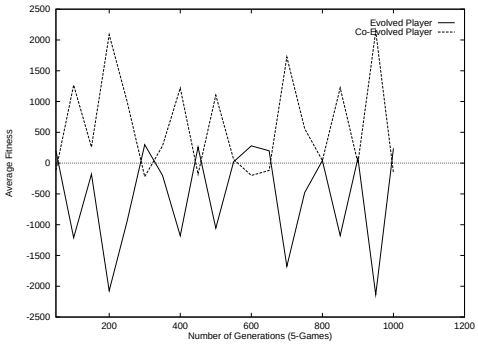


**Figure 5: Average fitness of Co-evolved player against an evolved player for five games**

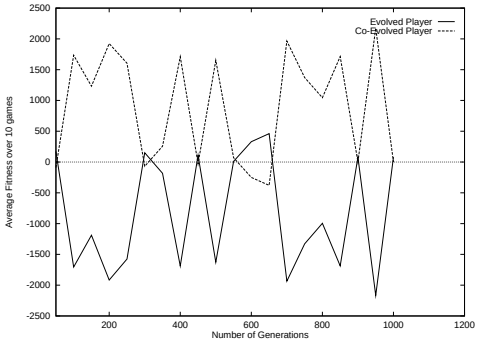


**Figure 6: Average fitness of Co-evolved player against an evolved player for ten games**

game series play checkers at a higher level than the players who play the five game series. It is important to note that the evolved programs for both cases are the same, the only difference is that in one case the programs play a series of ten games and the other they played only five games. This is strong evidence that the programs are actually learning how to play checkers better through experience alone.

To asess how large these margins of victory were, we plotted the cumulative fitness (where each plotted fitness is added to the previous) for both the agents playing a five game series and those playing a ten game series. This is shown in Figure 7 and we have plotted what the cumulative fitness would be if the co-evolved agents won every game against the evolved agents with one piece advantage, two pieces advantage (or one King), three pieces (a king and a piece), eight pieces (4 kings) or ten pieces (5 kings) advantage. From these graphs, it is evident that the co-evolved agent continues to perform better and wins every game by a margin greater than nine pieces on average. In fact, the ten game co-evolved players almost always beat the MCP evolved players by more than eight pieces (4 kings), whereas the five game players win by more than five pieces, but less than six. The figure also shows that the players with ten games experience are much superior to the same starting players but who have only five game experience.

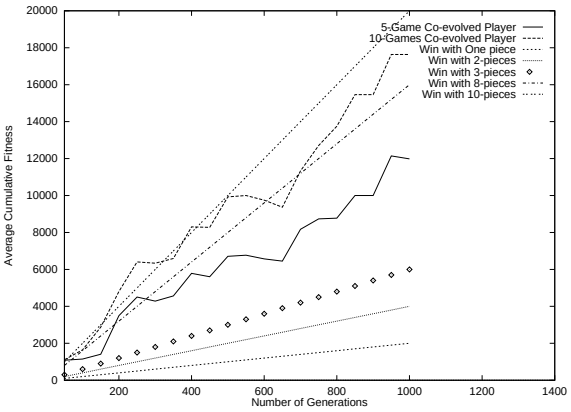


**Figure 7: Average cumulative fitness of Co-evolved player against an evolved player for five and ten games**

# 9. CONCLUSION

We have investigated the evolution and co-evolution of checkers playing agents that are controlled by developmental programs. The agents evolve intelligent behaviour much quicker through co-evolution rather than evolution against a minimax based program. We also have shown that the co-evolved agents improve with experience, and it appears that we have successfully evolved CGP programs that encode *an ability* to learn 'how to play' checkers. In future, we are planning to coevolve agents for longer, and allow more developmental experience through longer sequence of games, after evolution is finished.

# 10. REFERENCES


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

[2] F. Dalaert and R. Beer. 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, 1994.

[3] R. Dawkins and J. R. Krebs. Arms races between and within species. In *Proceedings of the Royal Society of London Series B*, volume 205, page 489Ú511, 1979.

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

[5] D. Fogel. *Blondie24: Playing at the Edge of AI*. Academic Press,London, UK, 2002.

[6] F. Gruau. Automatic definition of modular neural networks. *Adaptive Behaviour*, 3:151–183, 1994.

[7] W. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Artificial life 2*, pages 313–324, 1991.

[8] N. Jacobi. *Harnessing Morphogenesis, Cognitive Science Research Paper 423, COGS*. University of Sussex, 1995.

[9] G. Kendall and G. Whitwell. An evolutionary approach for the tuning of a chess evaluation function using population dynamics. In *IEEE. CEC. 2001*, pages 995–1002, 2001.

[10] G. Khan, J. Miller, and D. Halliday. Coevolution of intelligent agents using cartesian genetic programming. In *Proc. GECCO*, pages 269 – 276, 2007.

[11] A. Lubberts and R. Miikkulainen. Co-evolving a go-playing neural network. *in Coevolution: Turning Adaptive Algorithms upon Themselves, Belew R. and Juille H (eds.)*, pages 14–19, 2001.

[12] J. Miller, D. Job, and V. Vassilev. Principles in the evolutionary design of digital circuits – part i. *Journal of Genetic Programming and Evolvable Machines*, 1(2):259–288, 2000.

[13] J. F. Miller and P. Thomson. Cartesian genetic programming. In *Proc. EuroGP*, volume 1802 of *LNCS*, pages 121–132, 2000.

[14] D. Moriarty and R. Miikulainen. Discovering complex othello strategies through evolutionary neural networks. *Connection Science*, 7(3-4):195–209, 1995.

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

[16] S. Nolfi, O. Miglino, and D. Parisi. 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, 1994.

[17] J. Paredis. Coevolutionary constraint satisfaction. In *Proceedings of the third international conference on parallel problem solving from nature, Springer- Verlag*, volume 866, pages 46–55, 1994.

[18] J. Paredis. Coevolutionary computation. *Artificial Life*, 2(4):355–375, 1995.

[19] J. Pollack, A. Blair, and M. Land. Coevolution of a backgammon player. In *In: Langton, C. (ed),Proceedings artificial life 5*. MIT Press.

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

[21] C. D. Rosin. *Coevolutionary search among adversaries*. Ph.D. thesis, University of California, San Diego., 1997.

[22] A. Rust, R. Adams, and B. H. 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, 2000.

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

[24] J. Schaeffer. *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer, Berlin, 1996.

[25] G. Shepherd. *The synaptic organization of the brain*. Oxford Press, 1990.

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