An Empirical Comparison of Genetically Evolved Programs and Evolved Neural Networks for Multi-agent Systems Operating under Dynamic Environments

Jaime Dávila
Hampshire College, School of Cognitive Science
893 West Street
Amherst, MA, USA
jdavila@hampshire.edu

ABSTRACT

This paper expands on the research presented in [2] by comparing the performance of genetically evolved programs operating under dynamic game environments with that of neural networks with evolved weights. On the genetic programming side, the maximum allowed tree depth was varied in order to study its effect on the evolutionary process. For evolution of neural networks, encoding included direct encoding of weights and three different L-Systems. Empirical results show that genetic evolution of neural networks weights provided better performance under dynamic environments when evolved to choose which of several high-level actions to perform, such as "defend" or "attack". On the other hand, genetic programming evolved better solutions for low-level actions, such as "move left," "move right," or "accelerate." Solutions are analyzed in order to explain these differences.

Categories and Subject Descriptors

•Computing methodologies → Multi-agent planning;
•Computer systems organization → Neural networks;
•Software and its engineering → Software evolution;

Keywords

Genetic algorithms; Genetic programming; Multi-agent systems; Neural networks; Dynamical optimization

1. INTRODUCTION

Multi-agent systems in competitive environments need to be able to adapt to varying competition, while at the same time maintaining some basic approaches in the face of changing opponents. A good machine learning system for a competitive environment should be able to learn general strategies that apply to all opponents while being able to quickly modify its behavior when needed. During game playing, a good system should be able to modify details of how to best attack or defend an opponent based on that opponent's particular style of play.

Oliveto and Sarges [4] have looked at the effect of looking at different types of diversity, such as genotype and fitness value, for this type of problem. For static environments, while Kitano found L-Systems to outperform direct encoding of weights for evolution of neural networks [3], Siddiqi and Lucas have found conflicting results [6]. Also for static environments, Bornhofen and Lattaud have looked at the effect of different mappings from genotypes to phenotypes in evolved L-systems, and their ability to store useful information in dormant production rules [1].

Davila has empirically compared the efficiency of evolving neural networks (NN) through direct encoding of weights versus using L-Systems for multi-agent systems under dynamic environments [2] . Results showed that different coding schema provided different levels of gene expression, and that these levels affected the ability of the evolutionary process to adapt to new conditions while making high-level game decisions. In the research discussed here, I have expanded on this research to include performance analysis for genetic programming (GP), and to expand the game situations to include low-level multi-agent decisions.

2. GENERAL EXPERIMENTAL PARAMETERS

The results presented here were obtained for Multi-agent systems playing a virtual version of a Capture the Flag game. Solutions were evolved to play against a particular opponent for 80 generations, then changed to compete and evolve against a different opponent for 40 generations, finally returning to the original opponent for 40 generations more. Systems evolved included neural networks with direct weight encoding, three different types of Lindenmayer systems, and genetic programming. The full description of the Capture the Flag environment and the neural networks used can be found in [2]. The description of the genetic programming system is included in the next section.

Evolved systems played under two types of competition. Following the terminology used in [7], high-level decisions determined which of four pre-determined roles an agent would take: defending, floating, flexing, or attacking. Extending on [2], I also evolved systems to perform two different types of low level decisions. In one of them the agents needed to learn to approach an object in the playing field. In the sec-

GECCO '15 Companion, July 11-15, 2015, Madrid, Spain

ACM ISBN 978-1-4503-3488-4/15/07.
DOI: http://dx.doi.org/10.1145/2739482.2764717

ond low level task, agents needed to take into consideration the position and behavior of other agents (i.e. attacking while being aware of defenders in the opposing team). During these two low-level decision-making, the evolved systems determined the direction an agent would move in.

3. GP PARAMETERS

Evolved programs could make use of the following primitive operations: add, subtract, multiply, protected divide, square root, absolute value, sine, cosine, tangent, arcosine, arcosine, cotangent, maximum, and minimum. Evolutionary runs were repeated with maximum tree depths of 4, 10, and 20. During low level decision making, the output of the GP was interpreted as the angle in which the agent should move. For high level tasks, the angle was interpreted as which direction in the field the agent should take a role in: defending if pointing towards the far end of its own side of the field; sweeping if pointing towards the center portion of its side of the field; flexing if pointing towards the center portion of the opposing team's side of the field; and attacking if it was pointing towards the far end of the opposing team's side of the field.

4. RESULTS

4.1 Low Level Decisions

GP systems managed to evolve solutions for both types of low level decision-making problems mentioned above. For problems where agents needed to approach an object in the field, GP systems found simple ways to evaluate a path: compute the cosine to the object given vectors in the x and y coordinates, and compute the arcosine of that value. GP systems with maximum tree depth of 4, 10, and 20 levels managed to find this logic.

For low level tasks that required taking into consideration the position and behavior of other game objects, evolved solutions checked to see if the distance to the enemy flag was bigger than one unit. If it was, it approached the flag; otherwise it moved towards its side of the field, since it had the flag in its possession. While the tree for this solution had a depth of 5, only GP systems that allowed for trees with depth of 20 managed to evolve this type of code.

NN systems failed to evolve solutions to either of these low level problems. This failure to evolve NN solutions is not to say that neural networks are not able to solve this problem, but only that evolving network weights with the methods discussed here did not find an appropriate set of weights. In fact, standard backpropagation [5] is able to find a solution to the first low level problem discussed here in less than 500 training epochs.

4.2 High Level Decisions

Evolved neural networks outperformed evolved programs in the high level task described above. L-systems where gene expression was high, as further detailed in [2], outperformed both GP and direct-weight-encoding NN. Particularly interesting is the fact that GP systems perform very similarly to direct encoding NN, and these are both systems where all genetic information expresses into their corresponding phenotypes. Both of these systems adjusted to new opponents fairly quickly, but without seeming to take much advantage of previous evolution when returning to a prior opponent, which is one of the important behaviors of some of the L-systems used.

5. ACKNOWLEDGMENTS

Thanks to Josiah Erickson, from Hampshire College's IT Department, for the configuration and maintenance of the computing cluster where the experiments were run. The Breve simulation environment in which the CTF simulation runs was developed and made publicly available by Jonathan Klein. All neural network computations where performed with the freely available Emergent software package, developed at the University of Colorado. Genetic programming experiments were carried out with the pyEvolve library developed by Christian Perone.

6. CONCLUSIONS

The experiments presented here demonstrate that different types of evolutionary processes are optimal for different types of problems, even within the same domain. GP proved optimal for problems that required systems to pay attention to a small and specific set of input variables. GP systems needed depths twice as big as the optimal solution trees in order to successfully evolve solutions. Evolved NN proved to be better in problem that required observing a high number of input variables. Evolution of NN failed to find solutions to low level tasks, even though training with backpropagation was able to find good solutions for the same problems. This points towards the need for more elaborate types of NN evolution for those tasks, such as evolution of topologies as well, or instead, of evolution of weights.

7. REFERENCES

[1] S. Bornhofen and C. Lattaud. On hopeful monsters, neutral networks and junk code in evolving l-systems. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, pages 193–200, New York, NY, USA, 2008. ACM.

[2] J. J. Dávila. Genotype coding, diversity, and dynamic evolutionary neural network environments: A study on a multi-agent system. In *2014 World Congress on Computational Intelligence*, pages 2306–2313. IEEE, July 2014.

[3] H. Kitano. Designing neural networks using genetic algorithms with graph generation systems. *Complex Syst. J.*, 4:461–476, 1990.

[4] P. S. Oliveto and C. Zarges. Analysis of diversity mechanisms for optimisation in dynamic environments with low frequencies of change. In *GECCO*, pages 837–844. ACM, 2013.

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back- propagating errors. *Nature*, 323:533–536, 1986.

[6] A. Siddiqi and S. M. Lucas. A comparison of matrix rewriting versus direct encoding for evolving neural networks. In *1998 IEEE International Conference on Evolutionary Computation Proceedings*, pages 392–397. IEEE, 1998.

[7] P. Stone and M. Veloso. A layered approach to learning client behaviors in the robocup soccer server. *APPLIED ARTIFICIAL INTELLIGENCE*, 12, 1998.