2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Genotype Coding, Diversity, and Dynamic Environments: A Study on an Evolutionary Neural Network Multi-agent System

Jaime J. Dávila

***Abstract*—This paper reports the effects that different coding schemes at the genetic level have on the evolution of neural network multi-agent systems that operate under dynamic (changing) environments. Types of NN encoding include direct encoding of weights and three different L-Systems. Empirical results show that even variations within the same type of coding scheme can have considerable effects on evolution. Several different analysis of both genotypes and phenotypes are used in order to explain the differences caused by the coding schemes.**

I. 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. For example, under the RoboCup competition, which is commonly used as a testbed for multi-agent systems ([1]), virtual players controlled by an intelligent computer program should know how to perform basic tasks like kicking/passing the ball to a teammate or to a goal, or intercepting a pass or a shot, regardless of the particular strategy being used by their opponent. At the same time, 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. Stone and Veloso have discussed ways to learn basic, low-level skills like the ones mentioned above, while leaving open the question of how to learn high level strategies ([2])

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. In cases where learning is performed via evolutionary computation, learning should balance exploration of new alternatives with exploiting solutions that have proven successful in the past. When looking at a complete evolutionary computation system, partial solutions to different variations of a common problem could be found in different segments of population elements, in the information coded in specific genes, or in the diversity of elements in the current population. Previous research in evolutionary computation has looked at the effect of maintaining population diversity when operating under dynamically changing environments ([3]). This is based on the premise that having diversity in the population will help in finding good solutions when variations in the problem space take place, avoiding the well known problem of early convergence to sub-optimal solutions. 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 ([5]), 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 ([7]).

In the research discussed in this paper, I have evolved neural networks (NN) multi-agent systems (MAS) to play the team-based game of capture the flag (CTF), testing the effect that genetic coding scheme and genotype to phenotype translation have on the ability to deal with changes in their opponents. Empirical results indicate that different coding schemata provide the systems with differing ability to both adapt to change and to store solutions to past environmental conditions for later use.

The rest of this paper is organized as follows: Section II describes the game of CTF as implemented for these experiments; Section III describes the evolutionary process being used to evolve neural network agents, as well as the parameters being controlled in the experiments; Section IV describes and analyzes the the results obtained; Section V mentions future research suggested by the results presented here; Section VI summarizes results and conclusions; section VII acknowledges important people that have proven essential in the development and maintenance of the computing platform used to run these experiments.

II. The capture the flag (CTF) challenge

In this version of CTF, two teams of five players each start the game in opposite sides of a two-dimensional square playing area. Each team has a flag that they must defend, which is originally placed on their side of the field. In order to win a game, players must grab and carry the flag belonging to the opposing team and bring it into their own half, while at the same time keeping the opposing team from doing the same. If a player is "tagged" by an opposing player while in the opposer's side of the field, they are sent to "jail," where they must remain until a teammate frees them by touching the jail they are in. Figure 1 shows a segment of a sample game situation, where the blue team has grabbed the red flag and there is a blue player in jail.

In these experiments, neural networks will need to play against one of two different opposing teams. One of them,

* J. Dávila is with the School of Cognitive Science, Hampshire College, Amherst, MA 01002 USA (phone: 413-559-5687; fax: 413-559-5438; e-mail: jdavila@hampshire.edu).

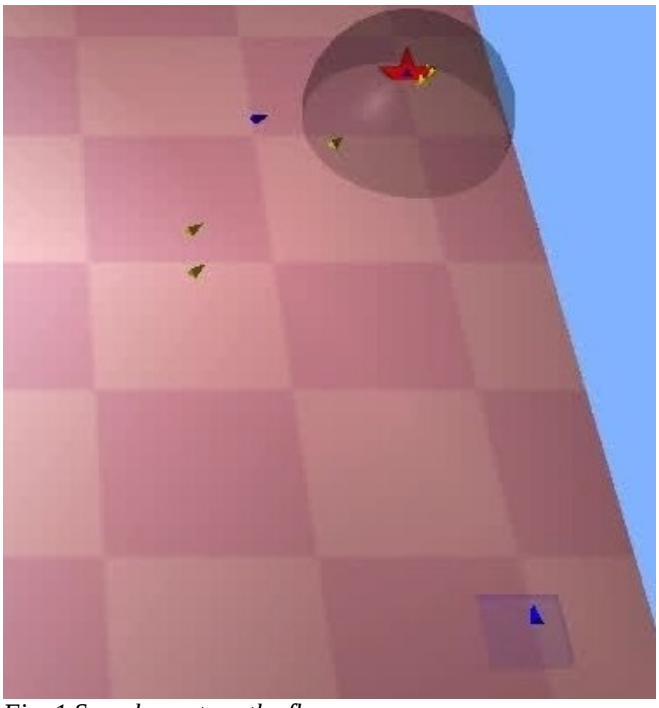

Fig. 1.Sample capture the flag scene

called *Champion*, has two defenders that stay close to their flag and try to tag enemies within a radius of ten distance units from them, and three attackers that either try to grab the enemy flag, or go towards the jail if there is a teammate in the jail and they are the closest attacker to the jail. The other team, called *runner-up*, plays by distributing players, at the beginning of the game, in the following way: two *guards* to protect their own flag; one *attacker* to try to grab the opposing flag; one *sweeper* to stay close to the middle of the field tagging opponents close to it, and one *flexer* to free any teammates that might be in jail, if any, otherwise behaving like an attacker. Based on these descriptions, we can see that these two teams are not completely different, but they nevertheless use different tactics. As documented later in Section IV, these differences in the teams' playing tactics are enough to make a difference during evolution. Games last for five minutes, or until a flag is captured and taken across the field's center line. If a flag is captured this way, the capturing team receives 200 points. In addition, a team receives one point for each second of the game that they had the enemy flag in their possession, minus the number of seconds the opposing team had their flag.

III. The Evolutionary System

A. Neural Networks Being Evolved

A team of five agents controlled by homogeneous neural networks is evolved to play against the two types to teams described above. These neural networks have 27 inputs and four outputs. The 27 inputs are distributed as follows: four two-dimensional vectors to each of its teammates; five two-dimensional vectors to each of the opposing players; a two-dimensional vector to its own flag; a two-dimensional vector to the opposing team's flag; a two-dimensional vector to the center of the field; the number of teammates that have been captured and are in jail; the number of opposing players that have been captured and are in jail; and a number between one and five that identifies which member of the team this one is. All vectors are normalized based on the size of the playing field. The number of players in each jail is normalized based on the total number of players in teams.

The input layer of these networks is fully connected to a hidden layer with twenty nodes. This first hidden layer is then fully connected to a second hidden layer of twenty nodes. Finally, the second hidden layer is fully connected to an output layer with four nodes. These four NN outputs indicate a preference towards performing one of four roles: defending, sweeping, attacking, or flexing. These four tasks are defined the same way as for the runner-up team described in section II. After generating these four outputs, the agent performs the role indicated by the output node with the highest activation value. This topology provides for enough connections as to allow the evolutionary systems described below to show different behaviors under changing conditions. Fig. 2 displays a network with this particular topology, with node colors representing activation values after processing a sample game situation. Arrows between layers represent full connectivity in the direction shown.

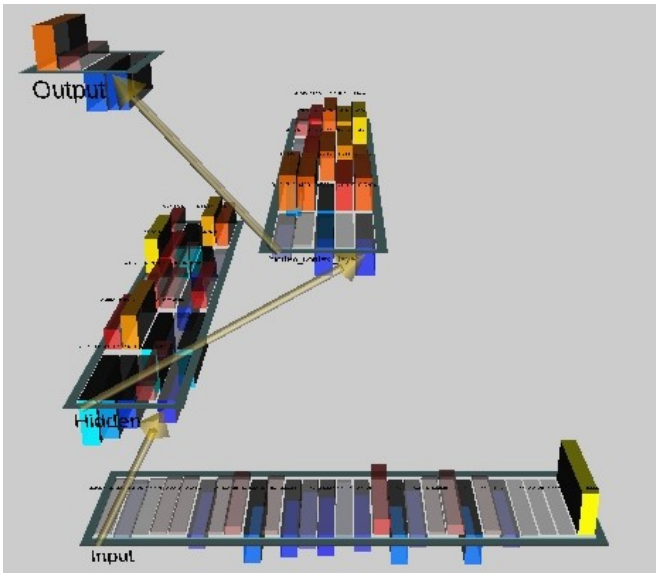

Fig. 2. NN topology

B. Coding of NN Weights in Chromosomes

The weights used by these neural networks are evolved by a genetic algorithm system using one of several possible coding schemata. In one system, which I will refer to as *DC*, direct encoding of weights is used. The genomes used in this system have 1400 genes, one for each connection in the networks described above. There are also three different

Lindenmayer systems, or L-systems ([8]). Following the work of Kitano, a genetic algorithm is used to evolve a set of rules defining different L-Systems ([5]). These L-Systems are then used to generate a set of weights for the NN connections following the algorithm outlined below.

C. L-Systems

An L-System is a collection of production rules of the form A → B, read as "A goes to B," where A is a non-terminal symbol, and B is a collection of one or more terminals and non-terminals. In the experiments reported here, a terminal is a symbol that represents a specific decimal number between -1 and 1. To generate a set of connection weights with an L-System, we begin by expanding the s*tarting symbol* S, into whatever the right side of that production rule indicates, substituting any terminals with the appropriate number based on those terminals' rule. If there are any non-terminals after expanding the S symbol, we continue expanding each of these non-terminals based on their own rules, until we have generated the desired number of weight values, or until we detect that no new weights will be generated regardless of continuing to expand non-terminals. At that point we substitute any remaining non-terminal with zeros, and use the generated numbers as connection weights in a neural network. Some of the characteristics of L-Systems that make them attractive for evolutionary neural networks are that they can represent a large number of weights with a smaller genome ([5]), and that they can facilitate the creation of modularity in the evolved networks ([9]).

In the experiments reported here, I have used three different types of L-Systems. One of them has sixteen non-terminals and five terminals, another has 21 non-terminals and five terminals, and a third one has 38 non-terminals and ten terminals. In each of these L-systems, non-terminals can expand to 4, 36, and 36 elements at the right side of a production rule, respectively. In the rest of this paper, these three L-Systems are referred to as LS-16-5-4, LS-21-5-36, and LS-38-10-36, respectively.

TABLE I
GENETIC CODING SCHEMA

Name	Non-terminals	Terminals	Expansion rate	# of genes
DC	N/A	N/A	N/A	1400
LS-16-5-4	16	5	4	69
LS-21-5-36	21	5	36	761
LS-38-5-36	38	5	36	1373

Experimenting with these three L-Systems and with direct encoding allows for an examination of the effect of different genome sizes and genotype-to-phenotype translation on the evolutionary process.

At the genetic level, each rule in these L-System is represented by a collection of N genes, where N is the expansion rate for the system. The first group of N genes stores what the S symbol expands to, the next group of N genes stores what the second non-terminal expands to, and so forth for each non-terminal. Finally, the last T genes represent what specific number between -1 and 1 each of the T terminals represents. Therefore, an L-System requires N*(expansion-rate) + T genes. Table I shows the number of terminals, non-terminals, expansion rate and total number of genes per L-System for the three cases used in the experiments reported in this paper.

D. Evolutionary Parameters

Regardless of coding scheme, populations had a constant size of one hundred elements. At each generation, 90% of offspring were generated by uniform crossover, 5% by copying, and 5% by mutation. Elements for each of these operations were selected with a tournament selection size of 50, where 50 elements were grabbed at random from the population and the best one of those 50 being selected for the operation. Outside of the 5% copying mechanism, no element, regardless of fitness value, was copied into future generations. This made processing through genetic schema the only way by which to propagate specific genes from one generation to the next.

TABLE II
EVOLUTIONARY PARAMETERS

Parameter	Value
Population size	100
Crossover rate	90%
Copying rate	5%
Tournament size	50
Mutation rate	5%

Systems were always evolved for eighty generations against one opponent, and then switched to a different one. That is, systems were either evolved against the runner-up for 80 generations before switching to evolve against the champion, or evolved against the champion for 80 generations before changing to evolving against the runner-up. Results reported in this paper are the averages for eight runs under the same experimental conditions.

IV. RESULTS FROM EVOLUTIONARY PROCESS

A. Champion and Runner-up as Different Environmental Challenges

Since the central topic of this paper is the evolution of neural networks under dynamic environments, it is fair to ask if the two challenges being presented to the system (i.e. playing against Champion and against runner-up) are in fact different from each other.

Fig. 3 shows the fitness results of all runs evolved first against the runner-up, with an average fitness at generation

80 of -783.88, with an average immediate drop to -395.5 when switching to evolving against the champion. Fig. 4 shows fitness results of all runs evolved first against the champion, with an average fitness at generation 80 of -301.45, with an average increase to 331.8 in four generations. Since the champion presents a harder challenge to the evolutionary system, the rest of this paper concentrates on two versions of dynamic changes during evolution: how the coding schema react to evolving to play against the runner-up for 80 generations and then evolving against the champion for 40 generations; and evolving against the champion for 80 generations, then switching to the runner-up for 40 generations, and finally evolving against the champion once more for 40 additional generations.

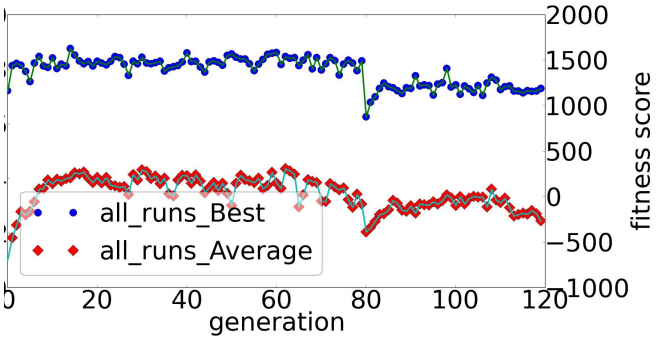

Fig. 3: fitness values, across all runs evolving against runner-up first.

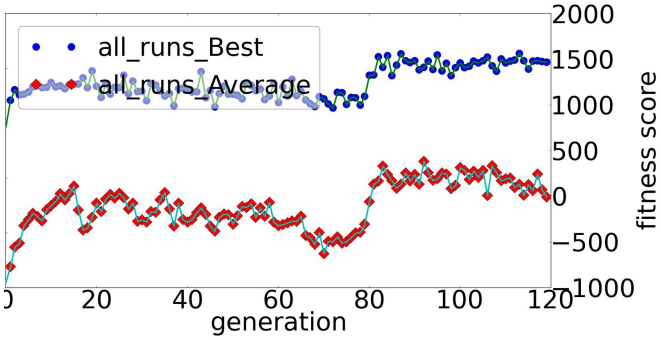

Fig. 4: fitness values, across all runs evolving against the champion first

B. On the Ability of Direct Encoding and L-Systems to Find Good Solutions

Both direct encoding and L-Systems are able to find good solutions to the problem presented. Table III shows fitness values for all runs, for direct encoding and each L-System. While different coding schema end up being superior at different points in the evolutionary process, direct encoding and LS-38-5-36 produce better values the most often. As seen in Fig. 5, direct encoding suffers a smaller drop in fitness when switching from evolving against the runner-up to evolving against the champion, and also improves faster once they start evolving against the champion, compared to L-Systems. Closer examination of L-Systems, presented later in this paper, show that the L-Systems used in these experiments behave differently from each other, and that LS-38-5-36 produces better values the most often. As seen in Fig. 5, direct encoding suffers a smaller drop in fitness when switching from evolving against the runner-up to evolving against the champion, and also improves faster once they start evolving against the champion, compared to L-Systems.

TABLE III
RESULTS OF EVOLUTIONARY PROCESS

Coding schema	After 80 generations against the runner-up	After 40 additional generations against the champion
All runs	1493.4	1187.66
Direct Encoding	1526.65	1257.62
All L-Systems	1478.56	1160.95
LS-16-5-4	1460.95	968.87
LS-21-5-36	1282.39	1046.9
LS-38-5-36	2132.26	1221.14

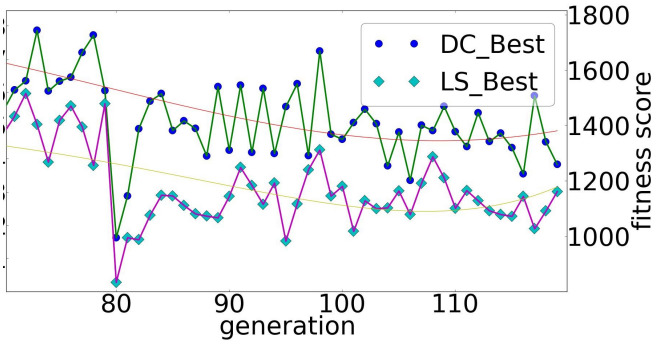

Fig. 5: direct encoding vs L-Systems, when switching to harder task

C. On what Different Types of Diversity Tell Us about the Evolutionary Process

Observing diversity values during evolutionary runs allows us to understand why some coding schemes are performing better than others in these tasks, and in particular when switching from one opponent to another. First, let us define two types of diversity to look at: genetic diversity and phenotypic diversity. I define genetic diversity as the average difference between genes occupying the same

position in different genomes, across all elements of a population. For direct encoding schemes, the difference between two genes can be obtained by subtracting one gene value from another, obtaining the formula in equation [1].

$$\frac{\sum_{i=1}^{|P|}\sum_{j=i+1}^{|P|}\sum_{k=1}^{|G|}|g(i,k)-g(j,k)|/|G|}{|P|} \quad (1)$$

where P is the current population, G is the genome in the coding scheme, and g(i,k) is the k^{th} gene of element i. For L-Systems, where the genome uses symbols instead of floating point numbers, I define the difference between genes as zero when the symbols are the same and one when the symbols are different, producing the algorithm in Fig. 6, with term definitions as above.

```
Difference = 0
For i = 1 to |P|:
  For j = 1 to |P|:
    For k = 1 to |G|:
      If g(i,k) <> g(j,k):
        Difference = Difference + 1
    Difference = Difference/|G|
Return (Difference/|P|)
```

Fig. 6: algorithm for computing genetic diversity in L-Systems

Phenotypic diversity, on the other hand, is defined by looking at the weights of the neural networks that are deterministically produced from the genomes at the moment of evaluating their fitness. In the experiments reported in this paper, where all the networks have the same topology, only the 1400 weights between nodes are changing, giving us the formula in equation (2)

$$\frac{\sum_{i=1}^{|P|}\sum_{j=i+1}^{|P|}\sum_{k=1}^{1400}|w_k(i)-w_k(j)|}{1400} \quad (2)$$

where P is again the current population, and $W_x(y)$ is the x^{th} weight of the network generated by genome y.

In direct coding schemata, genetic and phenotypic diversity are of course the same. Plotting genetic and phenotypic diversity for L-Systems, on the other hand, shows different behavior for different codings. While genetic and phenotypic diversity are never identical, figs. 7-9 show how the relationship between genetic and phenotypic diversity is strongest for LS-38-5-36, weaker for LS-21-5-36, and even weaker for LS-16-5-4. This exactly maps with the fitness these systems obtain after evolving first against the runner-up and then against the champion. Notice how for LS-16-5-4 systems the diversity values cross each other multiple times during runs, in LS-21-5-36 systems phenotypic diversity is always below genetic diversity, and for LS-38-5-36 systems phenotypic diversity is always below genetic diversity and their curves have the same general shapes. The high discrepancy between diversity values in LS-16-5-4 systems is caused by changes in the rules being evolved that do not lead to changes in the resulting neural networks. That is, the rules that are changing are not expressing into phenotypic changes. This situation diminishes for LS-21-5-36 systems, and still exists but is even smaller for LS-38-5-36 systems.

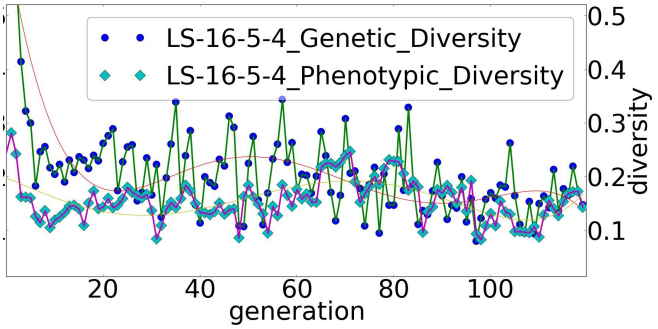

Fig. 7: diversities for LS-16-5-4 system

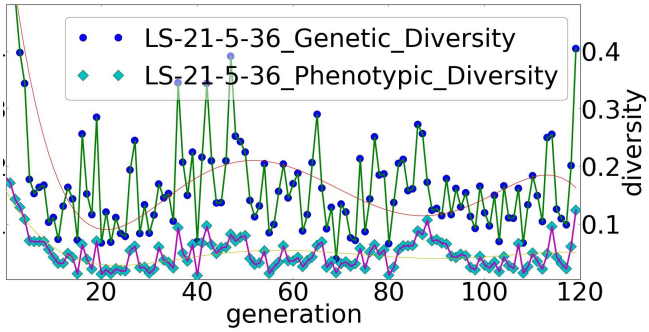

Fig. 8: diversities for LS-21-5-36 systems

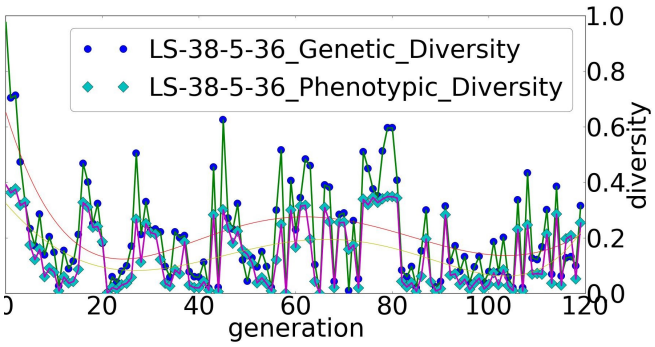

Fig. 9: diversities for LS-38-5-36 systems

These differences in how coding schemes manage to express or fail to express their genes can explain the performance difference between them. As direct encoding manages to express 100% of its genes, it can react to changes in the environment more quickly. Among L-Systems, those coding schemes that tend to express more of their genes also present better performance.

Next I take a look at two additional criteria that have been used to evaluate L-systems and direct encoding: their ability to create modules that become useful in different part of the phenomes, and their ability to store information in their genotype that becomes useful when facing environmental changes.

D. On the ability of different coding schemata to evolve modularity in neural networks

Given that LS-38-5-36 systems are managing to respond to environmental changes faster than direct encoding, we could ask if they are doing this while losing the ability to create modular neural networks, specially since its behavior is closer to that of direct encoding than either of the other L-Systems. While the literature shows disagreement on how to define modularity ([5], [6], [10]), we can set its highest limit as a count of how often weights are being repeated throughout network topologies in the elements of a population. Fig. 10 shows this information for the three L-Systems used.

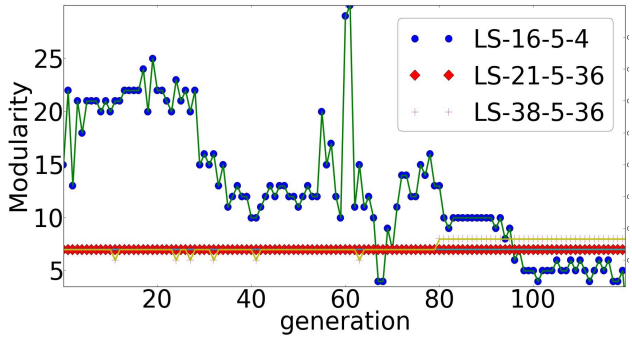

Fig. 10: module generation in L-systems

While LS-16-5-4 can produce higher modularity, this modularity varies more and on occasions drops to values lower that under other schemata. LS-21-5-36 and LS-38-5-36 show almost identical constant modularity.

E. On the ability of different coding schemata to retain learning under differing environmental conditions

Some of the literature on genetic schemata for evolution of neural networks centers around their ability keep useful learned traits in their genotype, under changing conditions, which can then form part of good solutions when those conditions arise again. To analyze the coding schemata discussed in this paper based on this property, I ran the following evolutionary experiments: solutions were evolved for eighty generations to play against the champion, then they evolve for forty additional generations to play against the runner-up, finally evolving for forty additional generation playing against the champion again. While direct encoding continues to show the ability to quickly adapt to environmental changes (Fig. 11), LS-38-5-36 outperforms it, suffering little performance decrease when switching back to evolving against the champion., compared with how it was doing forty generations before when last playing against this opponent (Fig. 12).

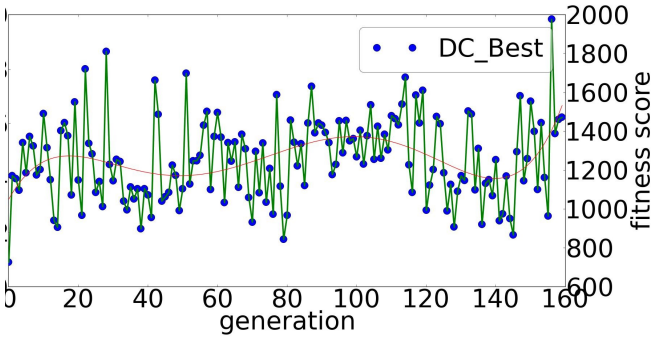

Fig. 11: Direct encoding fitness, when returning back to champion

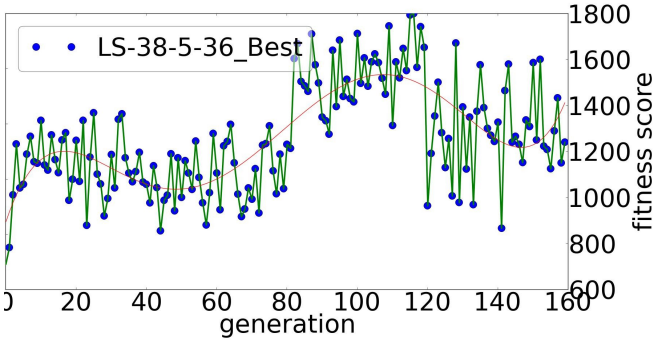

Fig. 12: LS-38-5-36 fitness, when returning back to champion

Notice how LS-38-5-36 behaves differently when originally evolving against the champion from generation zero. This new behavior is not caused by the forty additional

generations evolving against the champion, as seen from Fig 13, where LS-38-5-36 evolves for 160 consecutive generations against the champion.

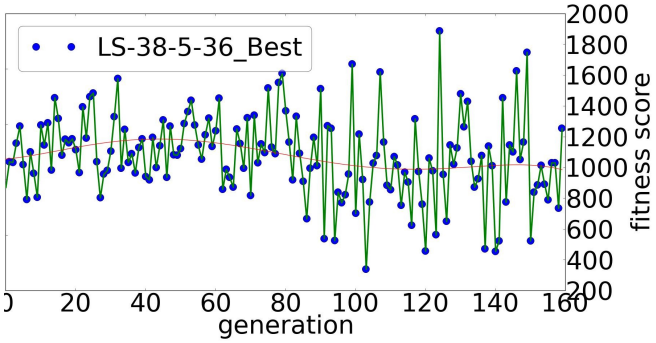

Fig. 13: LS-38-5-36 fitness, 160 generations against the champion

These patterns are not seen with LS-16-5-4 or LS-21-5-36, and in fact LS-16-5-4 fails to re-learn how to play against the champion, never achieving its original performance against it, as seen in Fig. 14.

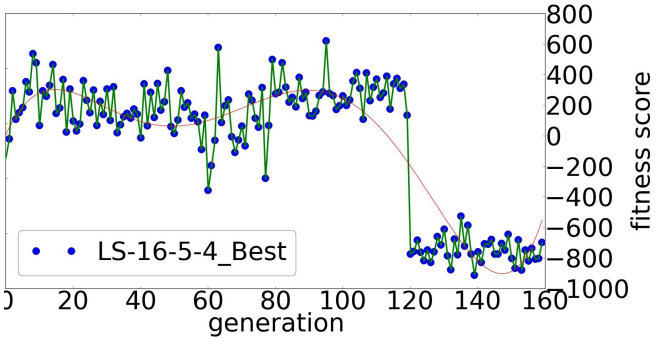

Fig. 14: LS-16-5-4 fitness, when returning back to champion

To understand why LS-16-5-4 and LS-38-5-36 behave so differently, I have taken a look at the neural network connection weights present in the population after initially evolving against the champion, and seeing how many are repeated after evolving against the runner-up.

With the algorithm shown in figure 15, the number of repetitions are shown in Table IV. LS-38-5-36 manages to keep 21.73% more connections present after switching opponents.

TABLE IV
WEIGHT REPETITIONS

Coding schema	Connection weights from generation 80 repeated in generation 120
LS-16-5-4	558462
LS-38-5-36	713514

```
maintained = 0
For i = 1 to |P80|:
  For j = 1 to |P120|:
    For k = 1 to |G|:
      If g(i,k) == g(j,k):
        maintained = maintained + 1
Return (maintained)
```

Fig. 15: algorithm for computing capacity to maintain phenotypic characteristics through environmental change

LS-38-5-36, then, is seen to have both the capacity to maintain characteristics across environmental changes and the capacity to create some modules. On the downside, this schema uses 1373 genes, close to the same number of connection weights that need to be generated.

V. FUTURE RESEARCH

While results of the experiments reported in this paper clearly demonstrate that coding schemata are making a difference in performance, and that those differences are based on the generated phenomes, it is not clear why these phenomes behave differently for this task. Experiments with neural networks of different topologies and L-System with different parameters (such as numbers of terminals, non-terminals, expansion rates, and total number of genes) will help clarify the relationship between these variables and system performance. Aside from genetic and phenotypic diversity, I have started to look at behavioral diversity, which is based on the specifics of how the agent systems behave against different opponents. Other future research includes generating neural networks via genetic programming (GP) instead of genetic algorithms, as GP have also been found to provide for scalability and modularity ([11], [12]. Finally, I am interested in how the methods presented in this paper perform on other types of tasks. I am currently configuring poker-playing environments where GA/NN systems learn to detect high-level playing pattern in their opponents, and configuring CTF games where neural networks determine low level decisions, such as in which direction to move, instead of the high level decisions they perform in the experiments reported here.

VI. CONCLUSIONS

This paper has presented results obtained by evolving neural networks for agent-based high-level decision making in dynamic environments. Both direct encoding of connection weights and L-Systems are able to find quality high level strategies for a team-based game with multi-agent systems. The schema used at the genetic level to encode

neural network connectivity is found to make substantial differences in performance. Simultaneously looking at genetic and phenotypic diversity proves useful in understanding the performance of genetic coding methods. Direct encoding, where all genes are expressing as weights of a network, adjusts to environmental changes the fastest. From among L-Systems, where differing genes sometimes but not always express into different connection weights, those methods that have higher gene expression outperform others. We also see that systems that manage to carry phenotypic characteristics through differing environments have better performance when returning to past environmental conditions. While the L-System that reacts the quickest to environmental changes has close to the same number of genes as direct encoding (losing the commonly mentioned advantage of scalability), this L-System also has a higher ability to maintain genes across environmental changes that more compact L-Systems.

VII. ACKNOWLEDGEMENTS

Thanks to Josiah Erickson (Institutional Technology, Hampshire College) for maintaining the computing cluster where these experiments were run. The Breve simulation environment was developed and made publicly available at http://www.spiderland.org/ by Hampshire graduate Jonathan Klein. All neural network computations done in the Emergent software package, freely available at http://grey.colorado.edu/emergent/index.php/Main_Page . The Capture the Flag game/simulation was developed by a course taught by Lee Spector, Professor of Computer Science at Hampshire College. As part of this same course, the champion CTF player was developed by Paul Swartz and the runner-up was developed by Mikel Waxler. Graphs presented in this paper were produced with Python scripts using the PyLab set of libraries.

REFERENCES

[1] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawai, and H. Matsubara, “Robocup: A challenge problem for ai and robotics,” in *RoboCup-97: Robot Soccer World Cup I*, Springer, 1998, pp. 1–19.

[2] P. S. and M. Veloso, “A Layered Approach to Learning Client Behaviors in the RoboCup Soccer Server.”

[3] F. L. Minku and X. Yao, “Using diversity to handle concept drift in on-line learning,” in *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, 2009, pp. 2125–2132.

[4] P. S. Oliveto and C. Zarges, “Analysis of diversity mechanisms for optimisation in dynamic environments with low frequencies of change,” in *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, 2013, pp. 837–844.

[5] H. Kitano, “Designing Neural Networks Using Genetic Algorithms with Graph Generation System,” *Complex Syst. J.*, vol. 4, pp. 461–476, 1990.

[6] A. A. Siddiqi and S. M. Lucas, “A comparison of matrix rewriting versus direct encoding for evolving neural networks,” in , *The 1998 IEEE International Conference on Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence*, 1998, pp. 392–397.

[7] 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*, 2008, pp. 193–200.

[8] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[9] J. Fekiac, I. Zelinka, and J. Burguillo, “Proceedings 25th European Conference on Modelling and Simulation: ECMS 2011 : june 7th-10th, 2011, Krakow, Poland,” 2006.

[10] H. Lipson, J. B. Pollack, and N. P. Suh, “Promoting modularity in evolutionary design,” in *Proceedings of DETC*, 2001, vol. 1, pp. 9–12.

[11] Woodward, “Modularity in Genetic Programming,” in *Proceedings of EuroGP 2003*, 2003.

[12] N. NourAshrafoddin, A. R. Vahdat, and M. M. Ebadzadeh, “Automatic design of modular neural networks using genetic programming,” in *Artificial Neural Networks–ICANN 2007*, Springer, 2007, pp. 788–798.