Evolution of Neural Networks using Cartesian Genetic Programming

Maryam Mahsal Khan, Gul Muhammad Khan, Julian F. Miller

***Abstract*—A novel Neuroevolutionary technique based on Cartesian Genetic Programming is proposed (CGPANN). ANNs are encoded and evolved using a representation adapted from the CGP. We have tested the new approach on the single pole balancing problem. Results show that CGPANN evolves solutions faster and of higher quality than the most powerful algorithms of Neuroevolution in the literature.**

I. Introduction

In the past decade, a number of algorithms have been produced to generate artificial neural networks based on the evolution of weights and topology - also known as TWEANN (Topology and weight evolving artificial neural network). TWEANN algorithms can be divided into direct and indirect encoding. Direct encoding schemes use the genotype to represent all connection and nodes that appears in the phenotype. Direct encoding are generally thought of as better at fine tuning and generating compact architecture [1], [12], [25], [3], [19], [26], [14]. Indirect encoding schemes specify rules for constructing the phenotype. The genotype does not represent the connection and nodes that appear in the phenotype. The indirect encoding method includes artificial embryogenesis and produces phenotypes from evolved genotypes using a small phenotypical structure. Indirect encodings are thought to be better at finding a particular types of ANN architecture (i.e. topologies) [1], [8], [10], [13], [9].

In Conventional Neuroevolution (CNE) a genotype represents the whole neural network. The neural network weights are encoded with real numbers. It uses rank selection and burst mutation. The conventional algorithm has advantages over cooperative coevolution (ESP) as it evolves genotypes at the entire network level rather than neuron level, thus allowing potential global solutions to be found for a predefined network topology and size [5].

In Symbiotic, Adaptive Neural Evolution (SANE) the neuron population along with the network topologies representing the blue-prints are simultaneously evolved. Neurons that are combined to represent good networks scores and high fitness are promoted to the next generation. These are recombined to produce one single population. Also blueprints that results in good neuron combination are promoted to the next generation and recombined to produce even better combinations [17]. Symbiotic, Adaptive Neural Evolution (SANE) has successfully been applied to pole balancing task obtaining the desired behaviour within relatively few evaluations [6].

Enforced Sub-Population (ESP) is an extension to SANE where instead of one population of neurons, a subpopulation of hidden layer neurons is evolved. During genotype reproduction the neurons are only produced from their own subpopulation and their offspring remain in their parent's subpopulation. It produced better results than SANE [23].

Cooperative Synapse Neuroevolution (CoSyNE) evolves neural network at the level of synaptic weights only. For each network connection, there is a different subpopulation consisting of real-valued synaptic weight. Coevolution of the synaptic weight is performed by permutating the subpopulation that ultimately present a different network in the next generation. Multi-point crossover and probabilistic mutation is applied based on Cauchy distributed noise to produce offspring for the next generation. The CoSyNE approach has been shown to out-perform all the previous approaches on pole balancing problem [2].

Indirect encoding of network parameters in the form of Fourier coefficients is also exploited for generating efficient neural structures. The weight matrix is generated by taking the inverse Fourier transform of the coefficients matrix. As frequency domain representation decorrelates signal, the search of dimensionality space can be reduced in a principled manner by discarding high frequency. Thus the search starts from low frequency weights and then successively progresses toward high frequency weights [11].

Recently Cooperative Co-evolution has been introduced in the area of evolutionary computation focused on the evolution of co-adapted subcomponents. In Cooperative Co-evolutionary model instead of evolving complete networks only subnetworks are evolved. The cooperation among the individuals is encouraged by rewarding the individuals based on how well they cooperate to solve a problem. These subnetworks together with the best combinations of subnetworks are then evolved together. Different neural structures are formed by blending different subnetworks together. COVNET uses this technique and has been shown to produce better generalization and smaller networks [4].

Stanley presented a new TWEANN, known as NeuroEvolution of Augmenting Topologies (NEAT). He identified

Maryam Mahsal Khan is with the Department of Computer System Engineering, NWFP University of Engineering and Technology, Peshawar,Pakistan.
E-mail: maryam@nwfpuet.edu.pk
Gul Muhammad Khan is with the Department of Electrical Engineering, NWFP University of Engineering and Technology, Peshawar,Pakistan.
E-mail: gk502@nwfpuet.edu.pk
Julian Francis Miller is with Intelligent System Group, Electronics Department, University of York, UK.
E-mail: jfm7@ohm.york.ac.uk

the three major challenges for TWEANNs and presented solutions to each of them. The first one is tracking genes with historical markings to allow easy crossover between different topologies, the second is protecting innovation via speciation, and the third is starting from a minimal structure and "complexifying" as the generations pass. NEAT was shown to perform faster than many other neuro-evolutionary techniques. Unlike a conventional neural network whose topology is defined by the user, NEAT allows the user to evolve the network topology. Thus the complexity of the network changes, and is not constant as in fixed topology networks. The NEAT approach begins with a simple structure, with no hidden neurons. It consists of a simplistic feed-forward network of input and output neurons, representing the input and output signals. As evolution progresses, the topology of the network is augmented by adding a neuron along an existing connection, or by adding a new connection between previously unconnected neurons. The algorithm is notable in that it evolves both network weights and structure, and efficiently balances between the fitness and diversity of evolved solutions. NEAT has also produced good results on pole balancing problem [21], [20].

Continuous Time Recurrent Neural Network (CTRNN) are derived from dynamical neuron models known as leaky-integrators and their behaviour is mathematically modeled from system of differential equations. It is an intermediate step between sigmoidal and spiking neuron. CTRNN utilizes important characteristics such as spiking neuron-like input integration over time and variable internal state. The latter can also dynamically change state in the absence of external inputs. CTRNN are evolved using NEAT and applied on the pole balancing problem and have produced better results than the standard NEAT algorithm [15].

Neuroevolutionary algorithms are also used with adaptive mutation rates. One such strategy was used in NeVA where adaptive mutation of connection and weights of a network is performed. A direct encoding strategy is used where weights are encoded with 17 bits with a precision of 0.001. The network is evolved using mutation and crossover. Based on individual peculiar phenotype addition or deletion of new elements is performed without the use of global limiting variables or deterministic rules as the adaptive selection of the type of mutation do not require that. Adaptive mutation rate was found to improve the performance of NeVA for the pole balancing problem [22].

MBEANN, is a TWEANN algorithm where evolution is based on mutation only. A new encoding technique is introduced where a string is defined as a set of substrings called operons. Mutation of connections is applied to each operon at a constant probability. Weight mutation is based on adding a Gaussian random number with a standard deviation of 0.05 and mean zero. Add-Node mutation is applied at each operon at a constant probability. This mutation removes one of the synaptic connections, which is randomly selected and then adds a new hidden node and two associated synaptic connections. The algorithm was applied on the pole balancing problem and has proved to be robust [18].

It is observed that neuroevolution based on topology and weight produces better results than topology or weight alone [24]. In this paper, we show that evolving topology, weight and function can increase performance. We present a new TWEANN i.e. CGPANN. Our results show that this technique outperforms all the previous NE techniques. Also Cartesian genetic programming provides a specific topology that speed up the evolution process. The paper is organized as follows. Section II describes the background information on Cartesian genetic Programming. Section III describes the Neuroevolutionary algorithm based on CGP in detail. Section IV - VI describes the algorithm CGP and CGPANN applied on the standard bench mark problem i.e. the pole balancing task along with simulation results. Section VII concludes with discussions and future work.

II. CARTESIAN GENETIC PROGRAMMING

Cartesian Genetic Programming (CGP) is an Evolutionary Programming technique developed by Miller and Thomson [16]. CGP has proven to be a powerful and effective system in a wide array of experimental domain. CGP is a feedforward network. In CGP genotype are represented as finite length integers. The genotype consists of nodes, where each node is comprised of inputs and function. The inputs can be program inputs or inputs from the previous nodes. The function can be any mathematical function like logical OR, logical AND, Sigmoid, Tangent-Hyperbolic, Step, Linear etc. The output(s) of the genotype can be the output(s) of any node or from the program input(s).

The 1 + λ (where λ = 4) evolutionary strategy is used to evolve the genotypes from one generation to the next. In this strategy the parent genotype is unchanged and 4 offspring are produced by mutating the parent genotype. A simple example of CGP is shown in Fig.1. Fig.1(a) shows a 2x2 (rows x columns) genotype for an application with 2-bit input and 1-bit output. The function used are logical 'OR' and logical 'AND'. The number of inputs to each node is set to 2. Fig.1(b) shows the graphical representation of the genotype in Fig. 1(a). The output gene 5 is selected as the output of the network. The genotype represents the following mathematical equation, y =$x_0.x_1 + x_1$ where '.' represent the logical AND operation and '+' the logical OR. It should be noted that in CGP there are many non-coding (junk) genes (in the example shown, nodes with outputs 2 and 4 are both non-coding having no contribution in the output).

III. TWEANN BASED ON CARTESIAN GENETIC PROGRAMMING

In this section NeuroEvolution based on Cartesian Genetic Programming known as CGPANN is explained. This algorithm adopts the direct encoding scheme. CGPANN encodes topology, weight and functions in one genotype and then evolves it for augmented topology, best possible weights and functions. TWEANN algorithms for finding optimal topological structure are either constructive or destructive [24]. CGPANN is both constructive and destructive algorithm. It

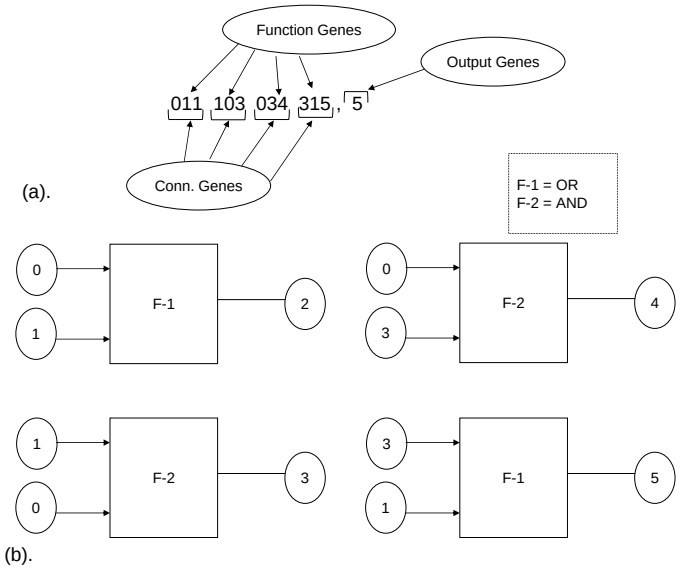

Fig. 1. (a)CGP based Genotype of an application with 2-bit inputs (b) Graphical Representation of the Genotype in (a)

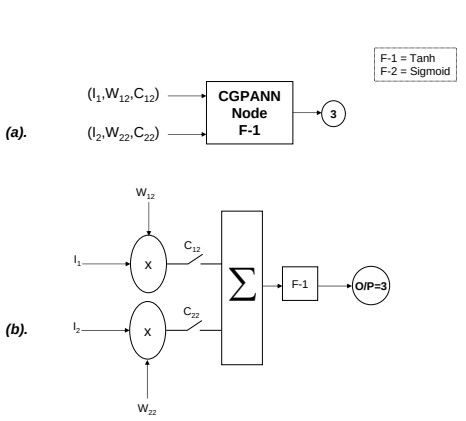

Fig. 2. (a)CGPANN node with two inputs (b) Inside Process of CGPANN for node in(a)

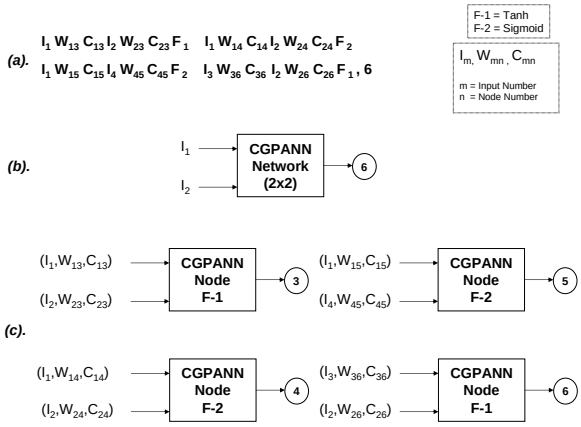

Fig. 3. (a) Genotype of a 2x2 CGPANN network, (b) Block Representation of the Genotype in (a), (c) Graphical Representation of the Genotype in (a)

begins with random topological features and incrementally removes and adds new features. It does so by mutating functions, inputs, weights, connection types and outputs. The genotype consists of nodes, where each node corresponds to a neuron of ANN. The node includes inputs, weights, connection and function as shown in Fig.2(a). Inputs can be program inputs or inputs from the previous nodes. An input is said to be connected if the Connection is 1 otherwise the Connection is 0. Weights are randomly generated in the range of -1 and +1. The output(s) of the genotype can be output(s) from any node or program input(s). Input and Weight are multiplied for all the connected inputs and is summed up. It is then forwarded to a non-linear function such as sigmoid or tangent hyperbolic to produce the output of each node. This output can either be the input to the next node or output of the system or not used (junk node). The CGPANN genotype is evolved from one generation to next (through the process of mutation) until the desired behaviour is achieved. When a connection is disabled through mutation it is not fully removed. It has a probability to become re-enabled in the later generations. Using different functions helps not only in quickly finding solution but also helps in generating diverse solutions to the same problem. Fig.2(a) is a block representation of a 2-input CGPANN node with inputs (I_1, I_2), weights (W_{13}, W_{23}) and connections (C_{13}, C_{23}). The output of the node is referenced as a finite number greater than the number of inputs to the system. Thus Fig.2(a) represents the parameters of node '3'. W_{13} corresponds to a weight assigned to I_1 and W_{23} represents

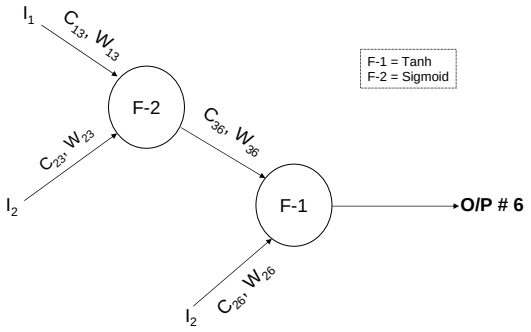

Fig. 4. Phenotype of the Genotype in Figure 3(a)

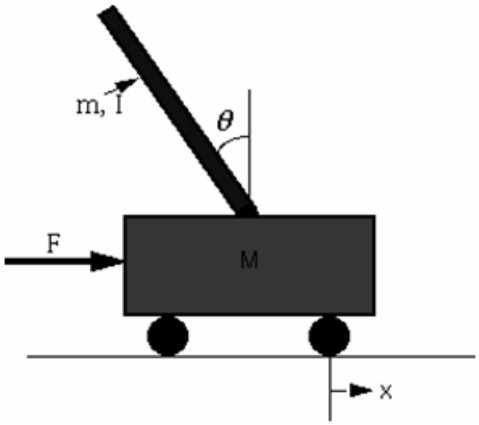

Fig. 5. Single Pole Balancing Scenario

weight of I_2 for node '3' respectively. Fig.2(b) displays the internal view of CGPANN node. The two inputs (I_1, I_2) are multiplied with the corresponding weights (W_{13}, W_{23}) (both the inputs are unconnected in this case). The result after summation is then forwarded to a tangent hyperbolic function that generates an output value for the node '3'.Fig.3(a) is a CGPANN genotype of a 2x2 network with 2 inputs (I_1, I_2), 2 functions (F-1 and F-2) and 1 output. Fig.3(b) represents the block diagram of the genotype in Fig.3(a). The CGPANN network has an output node '6'.

Fig.3(c) shows the inside view of the network in Fig.3(b). The node '6' takes input I_3 and I_2 where as I_3 is the output of node 3 and I_2 is one of the input to the system. Thus the network first computes the output of node '3' and then processes the result further.The resultant genotype is transformed to the neural architecture shown in Fig. 4. The network derived shows that neurons are not fully connected and that the program input(s) are not supplied to every neuron in the input layer which is different than the traditional ANN architecture. Thus such an evolutionary algorithm has the possibility to produce topologies that are efficient in terms of hardware implementation and time.

IV. SINGLE POLE BALANCING

Pole balancing is a standard benchmark problem in the field of control theory and artificial neural networks used for designing controllers for unstable and non-linear systems [7]. In this section, we will demonstrate the performance of CGP & CGPANN on the single pole balancing problem. Single pole balancing problem consists of a pole attached by a hinged to a wheel cart, where the track of the cart is limited to $-2.4 < x < 2.4$. The objective is to apply force F to the cart where the angle of the pole doesnt exceed (-12°$< \theta <$ +12°) and the cart doesnt leave the track. The controller has to balance the pole for approximately 30 minutes which corresponds to 100,000 time steps. Thus the neural network must apply force to the cart to keep the pole balanced for as long as possible. If the output of the CGPANN/CGP is greater than or equal to zero then a force F equal to +10N will be applied on the cart and if it is less than zero then a force 'F' equal to -10N is applied. The system inputs are the pole-angle (θ_p), angular velocity of pole (θ'_p), position of cart (x) and velocity of cart (x'). Fig.5 shows a single pole balancing scenario.

TABLE I
PARAMETERS FOR SINGLE POLE BALANCING TASK

Parameters	Value
Mass of cart (m_c)	1 Kg
Mass of Pole (m_p)	0.1 Kg
Length of Pole (l_1)	0.5 m
Width of the Track (h)	4.8m

Table 1 shows the standard numerical values used for simulating the pole-balancing problem.

$\widehat{m_i}=m_i\left(1-\frac{3}{4}cos^2\theta_i\right)$ **.....Eq.1**

$\ddot{\theta}_i=-\frac{3}{4l_i}\left(\ddot{x}cos\theta_i+gsin\theta_i+\frac{\mu_{pi}\dot{\theta}_i}{m_i l_i}\right)$ **.....Eq.2**

$\ddot{x}=\frac{F-\mu sgn(\dot{x})+\widehat{F_i}}{M+\widehat{m_i}}$ **.....Eq.3**

$\widehat{F_i}=m_i l_i\theta_i^2 sin\theta_i+\frac{3}{4}m_i g cos\theta_i sin\theta_i$ **.....Eq.4**

$x[t+1]=x[t]+\tau\dot{x}[t]$ **.....Eq.5**

$\dot{x}[t+1]=\dot{x}[t]+\tau\ddot{x}[t]$ **.....Eq.6**

$\theta_i[t+1]=\theta[t]+\tau\dot{\theta}_i[t]$ **.....Eq.7**

$\dot{\theta}_i[t+1]=\dot{\theta}_i[t]+\tau\ddot{\theta}_i[t]$ **.....Eq.8**

Equations (1) - (4) represents the general case for multiple poles balancing scenario. These equations are used to compute the effective mass of the poles (m_p), acceleration of the poles, acceleration of cart, and effective force (F_e) on each pole. (5) - (8) calculates the next state of angle and

angular velocities of poles, and position and velocity of cart. The pole and the cart motion are considered frictionless.

V. EXPERIMENTAL SETUP

A. CGP Network Settings:

In the single pole balancing task CGP networks of different topological architectures are generated. The inputs to the CGP network are the pole-angle (θ_p), angular velocity of pole (θ'_p), position of the cart (x) and velocity of the cart (x'). The functions used are addition, subtraction, multiplication and division. The maximum number of inputs to each node is fixed to 4. The mutation rate of 10% and 20% is used for all the networks generated. The performance of the CGP network is tested for 1 output and an average of 4 outputs. Various genotypes are produced by randomly generating inputs and functions for both the scenarios.

B. CGPANN Network Settings

Similarly CGPANN of different network sizes are generated. The inputs to the CGPANN network are the pole-angle (θ_p), angular velocity of pole (θ'_p), position of the cart (x) and velocity of the cart (x'). The activation functions used are sigmoid and hyperbolic tangent. Weights are randomly generated between -1 to +1. The number of inputs to each node is 4. The mutation rate of 10% and 20% is used for all models. Similarly the performance of the CGPANN network is tested for 1 output and average of 4 outputs. Various genotypes are produced by randomly generating connections, weights, outputs and inputs.

For both the CGP and CGPANN networks, results are simulated for inputs with random and zero initial values. The objective is to balance the pole for approximately 30 minutes which is equivalent to 100,000 simulation steps where each discrete time step corresponds to 0.02 sec. The output of the genotype is computed in each step. If the output is greater than or equal to zero then +10N Force is applied on the pole otherwise -10N. The performance of the algorithm is based on the average number of balancing attempts (evaluations). Table 2 represents the results for inputs (θ_p, θ'_p, x, x') starting from initial states of zero and random with one and average of 4 output nodes for both the CGP and CGPANN evolved genotypes. The results are the average of 50 independent evolutionary runs. Table 3 represents the average minimum and maximum number of active nodes (coding genes) with changing network sizes, mutation rate and outputs. The results are also average of 50 independent evolutionary runs.

VI. DISCUSSION AND ANALYSIS OF RESULTS

Table 2 displays a random trend of evaluations with changing network sizes for a CGP based network. While generally in the CGPANN case the number of evaluation seems to reduce with increasing network sizes. With increasing network size the search space increases hence the possibility of locating the best possible genotype increases, thus the desired behaviour is produced in minimum number of evaluations. Also by taking the average of 4 outputs the number of evaluations to obtain the desired results decreases further as the possibility of mutating a junk node increases. For a 0.2 mutation rate the 10x10 network has produced the minimum evaluation of 19. While for the CGP with 0.2 mutation rate the 5x5 has produced the minimum evaluation of 153.

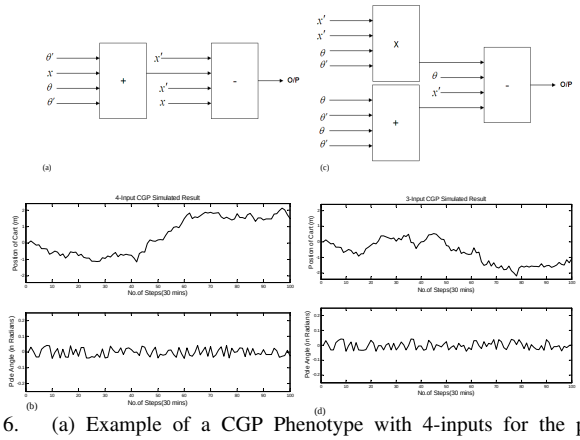

Fig. 6. (a) Example of a CGP Phenotype with 4-inputs for the pole balancing task (b) Simulation of the genotype in (a) for 30 minutes (100,000 time steps) (c). Phenotype of a CGP evolved genotype with only 3-inputs for the pole balancing task (b) Simulation of the genotype in (c) for 30 minutes (100,000 time steps)

From Table 3 it is observed that as we increase the number of output nodes from 1 to 4, the number of active nodes increases, while in most cases the number of evaluation decreases. This is due to the probability of enabling an inactive node which is directly or indirectly connected to the output. But ultimately this results in an increase in computational time in the evaluation of a genotype in a generation. Such a scenario is useful in simulating complex problems with huge network sizes. It was also observed that the evolved genotype in both the CGP and CGPANN cases had produced network structures with inputs less than 4. This indicates that CGP and CGPANN selects distinctive attributes and takes advantage of only those that help in the generalization of the problem.

Fig.6(a,c) displays the phenotype of the CGP evolved genotype with 2 and 3 inputs respectively. Fig.6(b,d) represents the pole angle and position of cart simulated for 100,000 steps (30 minutes) for the phenotype in Fig.6(a,c). Fig.7(a,c) displays neural network structures of the CGPANN evolved genotype with 2 and 3 inputs respectively. Fig.7(b,d) represents the pole angle and position of cart simulated for 100,000 steps (30 minutes) for the network in Fig.7(a,c). In Fig.6 & Fig.7 100,000 steps are down-sampled to produce 100 samples for demonstration purpose only.

Table 4 represents the comparison of different evolutionary algorithms for the single pole balancing phenomena, based on average number of evaluations for 50 independent evolutionary runs. CGP has been found to produce an average of 153 while CGPANN generate solution in only 19 evaluations. Thus, the proposed algorithm produce solutions in fewer evaluations than all other Neuroevolutionary techniques proposed to date.

TABLE II
PERFORMANCE OF CGP AND CGPANN FOR SINGLE/MULTIPLE OUTPUTS

Network Representation			Evaluations (CGP)		Evaluations (CGPANN)	
Initial State	Network Arch.	Mutation Rate	1 O/P	4 O/P	1 O/P	4 O/P
Zero	5x5	0.1	1881	549	477	1065
	10x10	0.1	613	577	157	317
	15x15	0.1	1477	1873	113	1637
	5x5	0.2	2409	373	35.48	25
	10x10	0.2	2697	933	31.4	**19**
	15x15	0.2	4241	605	25.32	21.8
Random	5x5	0.1	233	325	65	57
	10x10	0.1	461	269	41	65
	15x15	0.1	441	537	21	33
	5x5	0.2	177	**153**	172.2	65
	10x10	0.2	189	185	97	45
	15x15	0.2	217	209	95.56	45

TABLE III
NUMBER OF ACTIVE NODES OF CGP AND CGPANN FOR SINGLE/MULTIPLE OUTPUTS

Network Representation			Active Nodes (CGP)				Active Nodes(CGPANN)			
Initial	Network	Mutation	1 O/P		4 O/P		1 O/P		4 O/P	
State	Arch.	Rate	Min	Max	Min	Max	Min	Max	Min	Max
Zero	5x5	0.1	1	9	4	15	1	12	3	15
	10x10	0.1	1	14	5	21	1	23	5	37
	15x15	0.1	1	25	6	22	1	54	4	63
	5x5	0.2	1	9	7	14	1	12	2	15
	10x10	0.2	1	18	8	23	1	24	6	29
	15x15	0.2	1	43	9	22	1	54	4	63
Random	5x5	0.1	2	9	10	15	1	8	2	15
	10x10	0.1	1	27	11	28	1	26	7	34
	15x15	0.1	1	42	12	52	2	38	4	63
	5x5	0.2	2	12	13	15	1	9	1	17
	10x10	0.2	2	26	14	31	1	24	6	33
	15x15	0.2	2	52	15	38	1	35	10	58

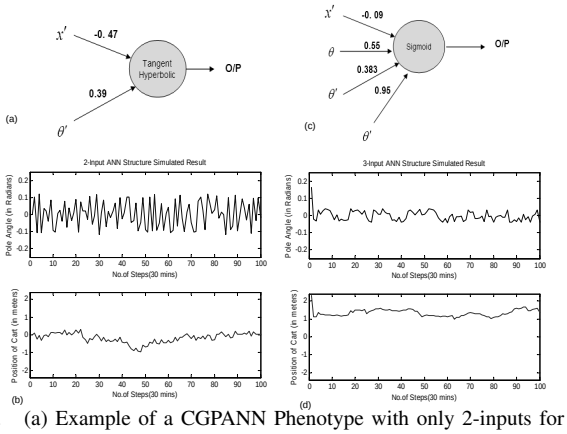

Fig. 7. (a) Example of a CGPANN Phenotype with only 2-inputs for the pole balancing task (b) Simulation of the genotype in (a) for 30 minutes (100,000 time steps) (c). Phenotype of a CGPANN evolved genotype with only 3-inputs for the pole balancing task (b) Simulation of the genotype in (c) for 30 minutes (100,000 time steps)

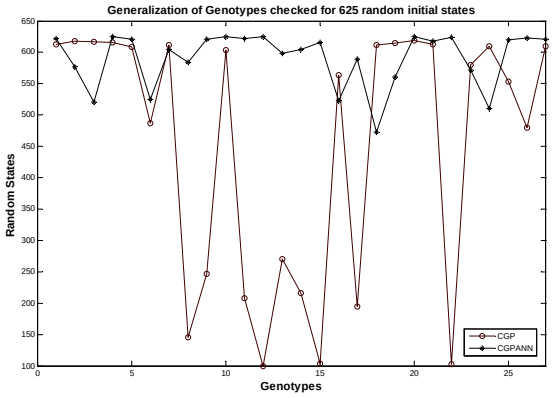

Fig. 8. Genotypes tested for generalization on 625 different random initial states

The robustness of both the algorithms was tested for generalization. Generalization refers to the successful balancing attempts starting from 625 random initial values for 1000 steps only. CGP was found to achieve an average of 450/625, whereas CGPANN was able to attain an average of 590/625 for 50 different genotypes. Fig.8 displays the performance of the 27 evolved genotypes tested with 625 random initial states for both the CGP and CGPANN based structures. This indicates that many evolved genotypes demonstrate a generalized behaviour.

VII. CONCLUSION AND FUTURE WORK

In this paper, Cartesian Genetic Programming and NeuroEvolution based on Cartesian Genetic Programming (CGPANN) was applied to a standard benchmark control problem i.e. single pole balancing. The result presented in this paper shows that CGPANN extends the powerful representation of

TABLE IV
COMPARISON OF CGPANN WITH OTHER NEURO-EVOLUTIONARY ALGORITHMS APPLIED ON THE SINGLE POLE BALANCING TASK

Method	Evaluations
NEAT	743
CNE	352
NeVA	349
SANE	302
ESP	289
CGP	**153**
CoSyNE	98
CGPANN	***19***

CGP to evolution of neural networks.It was observed that not only CGP but CGPANN had generated solutions with fewer numbers of evaluations. Also with the increase number of output nodes the number of evaluations decreased with an increase in coding genes. Both CGP and CGPANN were found to work as feature extractor by generating phenotypes with inputs less than 4. CGPANN produced the desired results in fewer numbers of evaluations as compared to the Neuroevolutionary algorithm formulated till date. Simulation results also shows that the CGPANN produces robust controllers with better generalization ability. The devised controllers are stable under a wide range of initial states. This shows that the CGPANN stands among competitive TWEANN algorithms such as NEAT, ESP, SANE, CoSyNE etc.

For the single pole balancing task with incomplete state recurrent networks are required. CGP and CGPANN algorithm will be modified to generate recurrent networks and tested accordingly. The algorithm will also be tested on more complex problem i.e. double pole balancing without velocity information.

REFERENCES

[1] J. C. Bongard and R. Pfeifer. Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In *Spector et al. 2001*, pages 829–836, 2001.

[2] C. Conforth and Y. Meng. Toward evolving neural networks using bio-inspired algorithms. In *Proceedings of the International Conference on Artificial Intelligence*, pages 413–419, 2008.

[3] D. Dasgupta and D. McGregor. Designing application-specific neural networks using the structured genetic algorithm. In *Proceedings of the International Conference on Combinations of Genetic Algorithms and Neural Networks*, pages 87–96, 1992.

[4] N. García-Pedrajas, C. Hervás-Martínez, and J. Mu noz-Pérez. Covnet: A cooperative coevolutionary model for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 14(3), 2003.

[5] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *J. Mach. Learn. Res.*, 9:937–965, 2008.

[6] F. J. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *IJCAI'99: Proceedings of the 16th international joint conference on Artificial intelligence*, pages 1356–1361, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[7] F. J. Gomez, J. Schmidhuber, and R. Miikkulainen. Efficient non-linear control through neuroevolution. In *ECML:17th European Conference on Machine Learning, Berlin, Germany*, pages 654–662, 2006.

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

[9] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks: In koza, j. r., goldberg, d. e., fogel, d. b., and riolo, r. l., editors, genetic programming 1996. In *Proceedings of the First Annual Conference*, pages 81–89. MIT Press., 1996.

[10] G. S. Hornby and J. B. Pollack. Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8, 2002.

[11] J. Koutnik, F. Gomez, and J. Schmidhuber. Searching for minimal neural networks in fourier space. In *Proceedings of the Third Conference on Artificial General Intelligence*, page to appear, 2010.

[12] C.-H. Lee and J.-H. Kim. Evolutionary ordered neural network with a linked-list encoding scheme. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 665–669, 1996.

[13] M. Mandischer. Representation and evolution of neural networks. in albrecht, r. f., reeves, c. r., and steele, u. c., editors. *Artificial Neural Nets and Genetic Algorithms*, pages 643–649, 1993.

[14] V. Maniezzo. Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions on Neural Networks*, 5(1):39–53, 1994.

[15] C. G. Miguel, C. F. d. Silva, and M. L. Netto. Structural and parametric evolution of continuous-time recurrent neural networks. In *SBRN '08: Proceedings of the 2008 10th Brazilian Symposium on Neural Networks*, pages 177–182. IEEE Computer Society, 2008.

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

[17] D. E. Moriarty, R. Miikkulainen, B. Kuipers, R. Mooney, B. Porter, and J. Shavlik. Symbiotic evolution of neural networks in sequential decision tasks. In *Evolutionary Synthesis of Neural Systems. MIT-Press*, 1997.

[18] K. Ohkura, T. Yasuda, Y. Kawamatsu, Y. Matsumura, and K. Ueda. Mbeann: Mutation-based evolving artificial neural networks. In *ECAL*, pages 936–945, 2007.

[19] D. W. Opitz and J. W. Shavlik. Connectionist theory refinement: Genetically searching the space of network topologies. *Journal of Artificial Intelligence Research*, 6:177–209, 1997.

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

[21] K. O. Stanley and R. Miikkulainen. Efficient evolution of neural network topologies. In *CEC '02: Proceedings of the Evolutionary Computation on 2002.*, pages 1757–1762, Washington, DC, USA, 2002. IEEE Computer Society.

[22] Y. Tsoy and V. Spitsyn. Using genetic algorithm with adaptive mutation mechanism for neural networks design and training. In *proceeding of third international conference on Genetic Algorithms*, page 709 714, 2005.

[23] Wieland. Evolving neural network controllers for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks (Seattle, WA)*, pages 667–673. NJ: IEEE., 1991.

[24] X. Yao. Evolving artificial neural networks. In *Proceedings of the IEEE*, volume 87(9), pages 1423–1447, 1999.

[25] X. Yao and Y. Liu. Towards designing artificial neural networks by evolution. *Applied Mathematics and Computation*, 91(1):83–90, 1996.

[26] B. Zhang and H. Muhlenbein. Evolving optimal neural networks using genetic algorithms with occams razor. *Complex Systems*, 7:199–220, 1993.