Evolution of Optimal ANNs for Non-Linear Control Problems using Cartesian Genetic Programming

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

***Abstract*—A method for evolving artificial neural networks using Cartesian Genetic Programming (CGPANN) is proposed. The CGPANN technique encodes the neural network attributes namely weights, topology and functions and then evolves them. The performance of the algorithm is evaluated on the well known benchmark problem of double pole balancing, a non-linear control problem. The phenotype of CGP is transformed into ANN and tested under various conditions in the task environment. Results demonstrate that CGPANN has the ability to generalize neural architecture and parameters in substantially fewer number of evaluations in comparison to earlier neuro-evolutionary techniques. We have also tested the CGPANN for generalization with different initial states (not encountered during evolution) over a range of evolved genotypes and obtained good results.**

***Index Terms*—Genetic Programming, Artificial Neural Networks, Non-Linear Control Problems**

I. Introduction

Artificial neural networks (ANNs) are computational systems made up of interconnected neurons. These neurons have properties inspired by biological neurons, and can exhibit complex global behaviour depending on the connections between neurons, their internal parameters and neuron functions. Just like biological neurons, artificial neurons are bound together by connections that determine the flow of information between neurons. Signals are transmitted from one neuron to another using these connections. ANNs are used in many real life applications including function approximation, time series prediction, classification, sequence recognition, data processing, filtering, clustering, blind signal separation, compression, system identification and control, pattern recognition, medical diagnosis, financial applications, data mining, visualisation and e-mail spam filtering [5], [4], [16], [22], [13], [15].

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

Various ANN training algorithms have been published that finds an appropriate topology with optimum weights which can accurately represent and solve the task under consideration. Neuroevolution is an approach to generate optimal neural networks using genetic algorithms. Numerous applications such as face recognition, manufacturing, aircraft control, robot navigation, game playing have been addressed using the Neuroevolutionary approach [7]. For decades the performance of the Neuroevolutionary algorithms have been tested on the non-linear control problem of pole balancing [10], [1], [9], [17], [18], [7], [8], [3], [12].

In this paper, we demonstrate a novel Neuroevolutionary method based on Cartesian Genetic Programming known as CGPANN. The results we obtained demonstrate that the proposed technique out performs all the previous methods explored to date.

The paper is organized as follows. Section 2 describes the background information on Cartesian Genetic Programming. Section 3 is an overview on NeuroEvolution and the different algorithms developed so far. Section 4 describes the Neuroevolutionary algorithm based on CGP in detail. Section 5,6,7 and 8 describes the algorithm applied on the standard bench mark problem i.e. the double pole balancing task along with simulation results. Finally, section 9 concludes with findings and future work.

II. Cartesian Genetic Programming (CGP)

Cartesian Genetic Programming (CGP) is an Evolutionary Programming technique developed by Miller and Thomson [11]. In CGP genotype are represented as finite length integers. The genotype consists of genes representing 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 logical or arithmetic function. 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.

We give a simple example of CGP in Figure 1 where (a) shows a 2x2 (rows x columns) genotype for an application

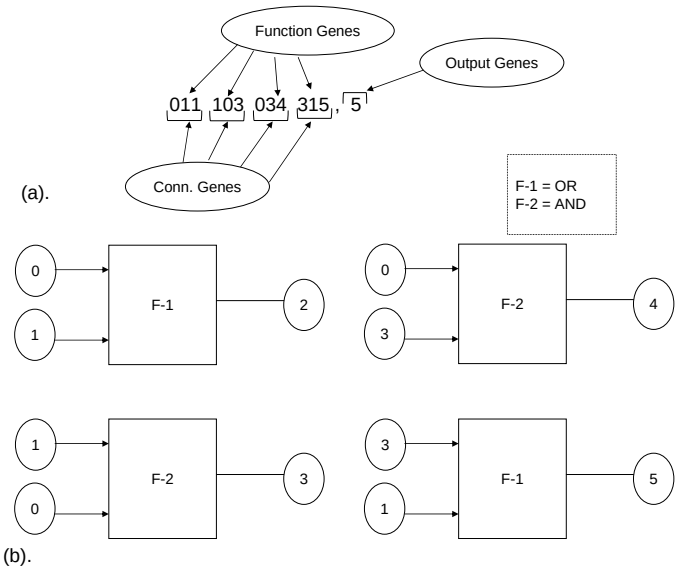

Fig. 1. (a)CGP based Genotype of an application with 2-bit inputs (b) Graphical Representation of the Genotype in (a)

with 2-bit input and 1-bit output. The function used are logical 'OR' and logical 'AND'. The input to each node is set to 2. Figure 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 observed that in CGP there are many non-coding (junk) genes (in the example nodes with outputs 2 and 4 are both non-coding).

III. Neuro-Evolution

This section describes artificial evolution on ANNs known as Neuroevolution (NE). The term NE refers to evolution of different attributes of a neural network. It is achieved through combination of an ANN with a genetic algorithm, with the network working as the phenotype and the genetic algorithm acting on the corresponding genotype. The genotype can include connection weights, connection type, node function or even the topology of the network. The genotype is evolved until the desired phenotypic behaviour is obtained. Since the choice of encoding affects the search space of solutions, it is an important part of the design of any NE system. Some methods evolve only weights of the network, some topology and some evolve both. If weights are evolved in a fixed topology, the network solution space is restricted, also evolution has to work in a more conservative environment and may not be able to find a novel solution to the problem [25].

The choice of selecting the proper topology of the network for a specific problem is always a difficult task. Topology and Weight Evolving Neural Networks (TWEANNs), which evolve both weights and network topologies is a method in which evolution is given the flexibility to select the desired topology and weights for its network. Thus, genotypes in TWEANN encode both the topology and weights of the network. TWEANN has many advantages over a fixed topology network, but they are accompanied by an increase in the number of parameters to be evolved.

A system called ENZO (Evolver and Network optimizer) optimizes both topology and connection weights at the same time [2]. The main features of the ENZO scheme are the introduction of new combinations of the parental properties by merging the parent's genes (crossover with connection specific distance coefficients). This speeds up the learning process through the inheritance of knowledge from the parents (weight transmission). Pujol and Poli used genetic programming to evolve weights, topology and activation functions of ANNs. They tested the system for development of a neural controller for the pole balancing problem with promising results [14].

Yao reviewed different combinations of ANNs and evolutionary algorithms (EAs) that evolved ANN connection weights, architectures, learning rules, and input features [25]. He identified different search operators which have been used in various EAs and pointed out possible future research directions for neuroevolution. His analysis showed that the evolution of connection weights provides a global

approach to connection weight training, especially when gradient information of the error function is difficult or costly to obtain. Due to the simplicity and generality of the evolution and the fact that gradient-based training algorithms often have to be run multiple times in order to avoid being trapped in a poor local optimum, the evolutionary approach is quite competitive. His analysis of evolving neural architecture showed that evolution can find a near-optimal ANN architecture automatically. He used both direct and indirect encoding scheme and pointed out that the direct encoding scheme of ANN architectures is very good at fine tuning and generating a compact architecture, whereas the indirect encoding scheme is suitable for finding a particular type of ANN architecture quickly. In further analysis he identified that separating the evolution of architectures and connection weights can cause fitness evaluation to mislead evolution, whereas simultaneous evolution of ANN architectures and connection weights produces better results.

Stanley presented a new TWEANN, known as NeuroEvolution of Augmenting Topologies (NEAT) [20], [19], [21]. He identified 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 double pole balancing problem [17], [7], [18].

Wieland proposed a conventional neuroevolutionary technique. 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 is better than cooperative coevolution (ESP) as it evolves genotypes at the network level rather than neuron level, thus isolating the performance [23].

Moriarty presented Symbiotic, Adaptive Neural Evolution (SANE) in which 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 blue-prints that results in good neuron combination are promoted to the next generation and recombined to produce even better combinations [12]. Symbiotic, Adaptive Neural Evolution (SANE) has successfully been applied to double pole balancing task obtaining the desired behaviour within relatively few evaluations.

Enforced Sub-Population (ESP) is an extension to SANE where instead of one population of neurons, a sub-population of hidden layer neurons is evolved [6]. During genotype reproduction the neurons are only produced from their own subpopulation and the produced offspring remain in their parent's subpopulation. It produced better results than SANE [18], as shown in table IV.

Cooperative Synapse Neuroevolution (CoSyNE) evolves neural network at the level of synaptic weights only [8]. 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 out-performed all the previous approaches on double pole balancing problem [8].

IV. Cartesian Genetic Programming Artificial Neural Network (CGPANN)

In this section NeuroEvolution based on Cartesian Genetic Programming known as CGPANN is proposed. The idea of CGPANN method is to encode neural network parameters i.e. topology, weight and function as CGP genotype.

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 Figure 2(a). Inputs can be program inputs or outputs of the previous nodes. An input is said to be connected if the connection is 1 and unconnected if the connection is 0. Weights are randomly generated between -1 to +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 an output at each node. This output can either be the input to the next node or output of the system. The CGPANN genotype is then evolved from one generation to next (through the process of mutation) until the desired behaviour is achieved.

Figure 2(a) is a block representation of a 2-input CGPANN node with inputs (I_1, I_2), weights(W_{13}, W_{23}) and connection

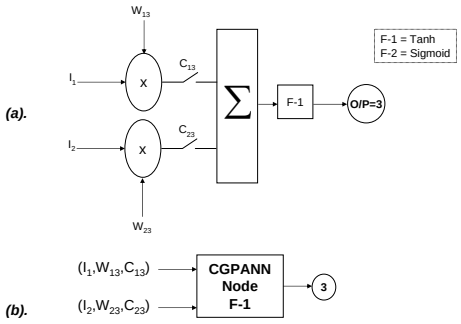

Fig. 2. (a)Inside Process of CGPANN for node (b)CGPANN node with two inputs in(a)

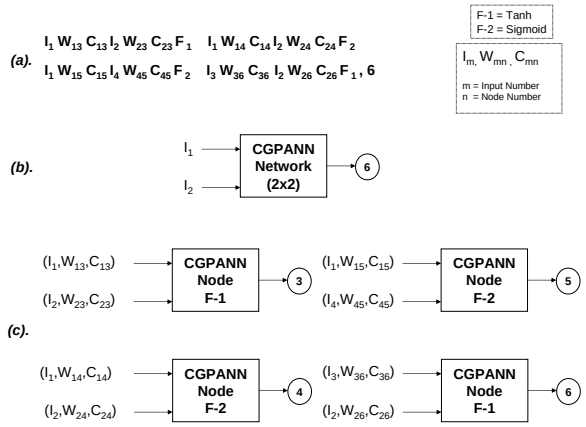

Fig. 3. (a) Genotype of a 2x2 CGPANN network, (b) Block Representation of the Genotype in (a), (c) Graphical Representation of the Genotype in (a)

switches (C_{13}, C_{23}) respectively. The output of the nodes are referenced in the same way as in CGP so that the first node's reference is the number of inputs. Fig. 2(a) represents the parameters of node '3' (For this example we we assume two inputs). W_{13} corresponds to a weight assigned to I_1 and W_{23} represents weight of I_2 for node '3' respectively.

Figure 2(b) displays the inside view of CGPANN node. The two node inputs (I_1, I_2) are multiplied with the corresponding weights (W_{13}, W_{23}). In this case both connection switches are zero so that the input to the node function is zero. In general, the result after summation of the connected inputs is given to the node function which generates an output value for the node '3'. In this paper node functions are either hyperbolic tangent or sigmoidal function.

Figure 3(a) shows a CGPANN genotype of a 2x2 network assuming 2 program inputs(I_1, I_2), 2 functions(F-1 and F-2) and 1 output. Figure3(b) represents the block diagram of the genotype in Figure3(a). The CGPANN network has an output node '6'. Figure 3(c) shows the inside view of the network in Figure 3(b) . The node '6' includes input I_3 and I_2 where I_3 is the output of node 3 and I_2 is one of the inputs to the system. Thus the network first computes the output of node '3' and then passes the result to node '6'. In this example the remaining nodes 4 and 5 are not used (non-coding).

Figure 4 represents the neural architecture of the genotype of Figure 3(a). The network derived shows that neurons are not fully connected because they may be non-coding and may or may not use all the inputs supplied to them (because of binary connection switch genes). This differs from standard ANN architecture. An evolutionary algorithm operating on this representation has the possibility to select topologies and weights of the networks simultaneously using this indirect representation.

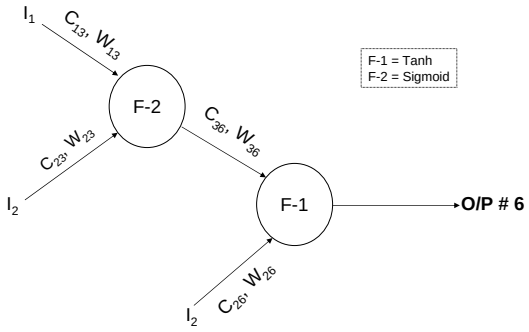

Fig. 4. Phenotype of the Genotype in Figure 3(a)

V. CONTROL PROBLEM: DOUBLE POLE BALANCING

Pole balancing (also known as pole-cart, broom balancer, stick balancer, inverted pendulum) is a standard benchmark problem in the field of control theory and artificial neural networks for designing controllers for unstable and non-linear systems [7]. Various NeuroEvolutionary Algorithms are evaluated on single and double pole balancing tasks.

Double Pole Balancing task requires balancing two poles hinged on a wheeled cart with a finite length track defined in the interval (-2.4$<x<$+2.4) and exerting a constant force, F, in left and right direction. The pole is said to be unbalanced if either the angle of poles exceeds the threshold (-36° $< \theta <$ +36°) or the cart leaves the track.

Figure 5 represents a double pole balancing system with a cart and two poles having lengths l_1 and l_2. Different

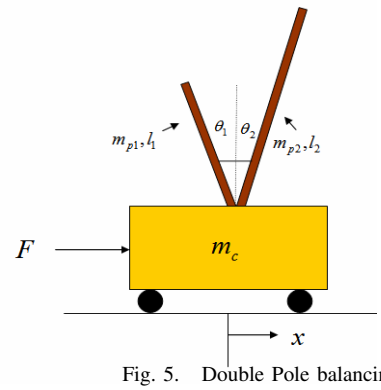

Fig. 5. Double Pole balancing Scenario

parameters of the poles are calculated from the force on the cart using equations below [24]. Where τ is the simulation time interval, g the acceleration due to gravity.

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

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

$$\ddot{x} = \frac{F - \mu_c sgn(\dot{x}) + \sum_{i=0}^{N} \widehat{F_i}}{M + \sum_{i=0}^{N} \widehat{m_i}} \tag{3}$$

$$\widehat{F_i} = m_i l_i \theta_i^2 sin\theta_i + \frac{3}{4} m_i cos\theta_i \left(\frac{\mu_{pi}\theta}{m_i l_i} + gsin\theta_i\right) \tag{4}$$

$$x[t+1] = x[t] + \tau\dot{x}[t] \tag{5}$$

$$\dot{x}[t+1] = \dot{x}[t] + \tau\ddot{x}[t] \tag{6}$$

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

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

Equations (1) - (4) are used to compute the effective mass of poles, angular acceleration of poles, acceleration of cart, and effective force on each pole. The remaining for equations (5) - (8) calculates the next state of the angle of the poles $(\theta_{1,2})$, velocity of poles $(\dot{\theta}_{1,2})$, position of the cart (x) and velocity of the cart $(\dot{x})$.

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

VI. EXPERIMENTAL SETUP

A random population of five CGPANN networks is generated at the start of each evolutionary run. The inputs to the CGPANN networks are pole-angles (θ_1, θ_2) , velocity of the poles $(\dot{\theta_1}, \dot{\theta_2})$, position of the cart (x) and velocity of the cart $(\dot{x})$. The activation functions used are sigmoid and hyperbolic tangent. The number of inputs to each node is 6. Results are

TABLE I
PARAMETERS FOR DOUBLE POLE BALANCING TASK

Parameters	Value
Mass of cart (m_c)	1 Kg
Mass of Poles (m_1,m_2)	0.1,0.01 Kg
Half Length of Poles (l_1,l_2)	0.5,0.05 m
Friction of Pole Hinges (μ_{p_i})	0.000002
Friction of the cart (μ_c)	0.0005
Width of the Track (h)	4.8m

simulated for random initial values in the following ranges: $(-0.6rad < \theta_{1,2} < 0.6rad)$ and $(-2.4 <$x$< +2.4)$ in one case, and zero initial values in second case. Mutation rates of 10% and 20% are used for the two experimental scenarios respectively. In all cases the results are average of 50 independent evolutionary runs. The constant force on the cart will be +10N if output of the genotype (force) is greater than or equal to zero and -10N if less than zero. Each discrete time step corresponds to 0.02 sec (τ). The objective is to balance the poles for approximately 30 minutes which is equivalent to 100,000 simulation steps. Table 2 and 3 represents the results for inputs $(x, \dot{x}, \theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$ starting with zero and random initial states. The performance of the algorithm is based on the average number of balancing attempts (genotype evaluations) to achieve the desired performance.

VII. RESULTS AND DISCUSSION

We carried out 50 independant evolutionary runs for each of the six different cases of CGPANN with varying network sizes and mutation rates for single and multiple outputs. Multiple outputs are averaged to produce the final output of the system. Results are tabulated in Table II and III.

From Table II, it is observed that with increasing network size and constant mutation rate the number of evaluations to find the optimal solution reduces. Using a mutation rate of 0.1 with a network size of 15x15 we found we could solve the double-pole balancing problem using an average of 77 evaluations . This number of evaluations is considerably fewer than all previously published figures for other neuroevolutionary algorithms. It is also observed from Table II that when the number of nodes is increased the problem of finding a pole-balancing solution becomes easier to evolve.

Often during the course of evolution, it was observed that the evolved genotype had produced neural network structures that did not use all the inputs (i.e. the physical parameters of the pole and cart). This indicates that CGPANN also functions as a feature extractor and exploits those parameters that help in the generalization of the problem (i.e. it ignores extraneous inputs values).

Figure 6(a) and 7(a) displays neural network structures of the evolved genotype with 2 and 3 inputs respectively. Figure 6(b) and 7(b) represents the corresponding pole an-

TABLE II
PERFORMANCE OF CGPANN BASED ON ZERO INITIAL VALUES

Network Size	Mutation Rate	Average Evaluations with single output (rounded)	Average Evaluations with 4-Outputs (rounded)
5x5	0.1	205	241
10x10	0.1	93	157
15x15	0.1	77	77
5x5	0.2	141	117
10x10	0.2	97	133
15x15	0.2	93	93

TABLE III
PERFORMANCE OF CGPANN BASED ON RANDOM INITIAL VALUES

Network Size	Mutation Rate	Average Evaluations with single output (rounded)	Average Evaluations with 4-Outputs (rounded)
5x5	0.1	1183	333
10x10	0.1	437	201
15x15	0.1	181	93
5x5	0.2	297	121
10x10	0.2	205	133
15x15	0.2	162	93

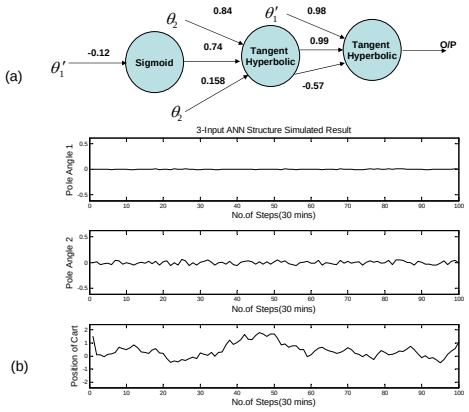

Fig. 6. (a) Phenotype of the Evolved Genotype (3-input ANN) (b) Pole Angle and Position of Cart simulated for the ANN in (a) for 100,000 steps

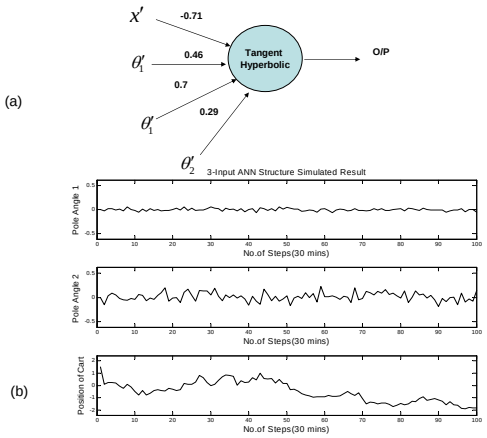

Fig. 7. (a) Phenotype of the Evolved Genotype (3-input ANN) (b) Pole Angle and Position of Cart simulated for the ANN in (a) for 100,000 steps

gles and position of cart simulated for 30 minutes respectively(100,000 steps are down-sampled to produce 100 steps for demonstration purpose only).

Similar results were obtained for inputs with random initial states as shown in Table III. With increasing network size the average number of evaluation decreased. For a 0.1 mutation rate the 15x15 network has produced the minimum evaluation of 181. Similarly for a 0.2 mutation rate the 15x15 has produced the minimum evaluation of 162. Also the evolved genotype had produced network structures with minimum 3 inputs.

It is evident from Table III that using 4-Outputs and taking its average as the network output further reduces the number of evaluations needed to obtain the desired result. It is expected that the greater number of outputs should produce faster convergence as the probability of mutating an *inactive* (junk) node increases.

Figure 8(a) and 9(a) displays neural network structures of the evolved genotype with 4 inputs respectively. Figure 8(b) and 9(b) displays the corresponding pole angles and position of the cart simulated for 30 minutes respectively.

Table IV presents the comparison of different neuro-evolutionary algorithms based on average number of evaluations over fifty independent runs for the double pole balancing task. It is clear that the CGPANN aproach outperforms other techniques by a huge margin.

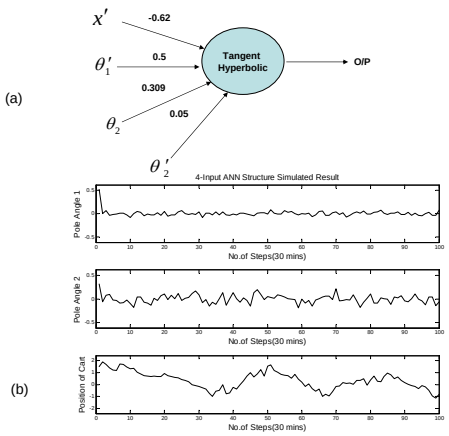

Fig. 8. (a) Phenotype of the Evolved Genotype (4-input ANN) (b) Pole Angle and Position of Cart simulated for the ANN in (a) for 100,000 steps

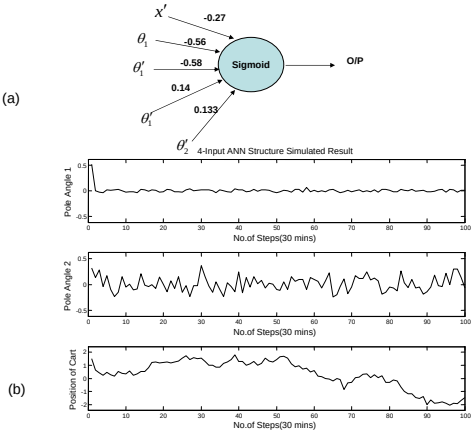

Fig. 9. (a) Phenotype of the Evolved Genotype (4-input ANN) (b) Pole Angle and Position of Cart simulated for the ANN in (a) for 100,000 steps

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

Method	Evaluations
CNE	22100
SANE	12600
ESP	3800
NEAT	3600
CoSyNE	954
CGPANN	***77***

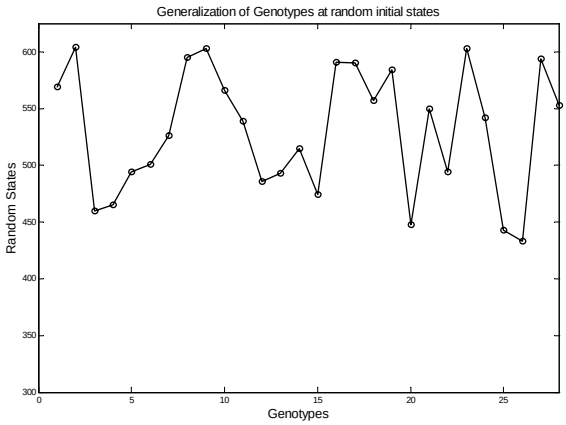

Fig. 10. Generalization of the Genotypes for 625 random initial states

VIII. GENERALIZATION

In addition to speed of evolution, we have also tested the robustness of CGPANN. We have tested a number of evolved genotypes (28) with different (625) random initial states and tested if they are able to control the system for 1000 steps. CGPANN was able to achieve the desired behaviour on an average of 532 out of 625 for twenty eight (28) different evolved genotypes. Figure 10 shows the performance of various evolved genotypes tested with 625 random initial states. This shows that the solutions (evolved genotypes) can cope with a large number of different initial conditions and that many solutions exhibit general behaviour.

It is important to mention that the performance of the CGPANN is based on a number of parameters namely mutation rate, network size, number of inputs to each node, averaging the number of outputs and the functions used. Thus proper selection of such parameter would be likely to locate even faster or more optimal evolutionary runs.

IX. CONCLUSION & FUTURE WORK

In this paper, the CGPANN method is proposed to generate ANN solutions to problems amenable to reinforcement learning. CGPANN algorithm is used to simultaneously generate good topologies, connections, weights and functions. The CGPANN approach was tested on the double pole balancing task. It was observed that CGPANN has generated solutions in much fewer evaluations than other published techniques. CGPANN not only generated optimum solution but also operated as a feature extractor in that many successful ANNs were found that did not use the six physical inputs. CGPANN proved to be a very flexible representation since it could handle weights, topology and neuron functions. CGPANN was

also tested on varying the number of outputs and computing its average, this further reduced the number of evaluations. Clearly the CGPANN stands competitive with established Neuro-Evolutionary techniques like NEAT, SANE, ESP and CoSyNE.

Future work in CGPANN involves modifying the CGPANN algorithm to incorporate levels back (a connectivity parameter in CGP), arity, adaptive mutation rate and adaptive network size. It is also planned to evaluate CGPANN genotypes for a larger class of ANN problems.

REFERENCES

[1] C. Anderson. Strategy learning with multilayer connectionist representations. In *In Proceedings of the Fourth International Workshop on Machine Learning*, pages 103–114. Morgan Kaufmann, 1987.

[2] H. Braun and J. Weisbrod. Evolving feedforward neural networks. In *Proceedings of ICANNGA93, International Conference on Artificial Neural Networks and Genetic Algorithms. Innsbruck: Springer-Verlag*, 1993.

[3] 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.

[4] G. Dorffner. Neural networks for time series processing. *Neural Network World*, 6(4):447–468, 1996.

[5] G. Dorffner and G. Porenta. On using feedforward neural networks for clinical diagnostic tasks. *Artificial Intelligence in Medicine*, 6(5):417–435, 1994.

[6] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[7] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Efficient non-linear control through neuroevolution. In *Proceedings of the European Conference on Machine Learning (ECML),Lecture Notes in Computer Science*, volume 4212. Springer, 2006.

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

[9] F. J. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuro-evolution. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1356–1361, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[10] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. *Congress on Evolutionary Computation*, 4(2):2588 – 2595, June 2003.

[11] 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.

[12] D. Moriarty. *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks*. PhD thesis, University of Texas at Austin, Tech Rep. UT-AI97-257, 1997.

[13] S. Murray. *Neural Networks for Statistical Modeling*. Van Nostrand Reinhold, 1993.

[14] J. C. F. Pujol and R. Poli. Evolution of the topology and the weights of neural networks using genetic programming with a dual representation. *Technical Report CSRP-97-7, School of Computer Science, The University of Birmingham, Birmingham B15 2TT, UK*, 1997.

[15] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[16] J. Sjoberg, H. Hjalmarsson, and L. Ljung. Neural networks in system identification, 1994.

[17] K. Stanley and R. Miikkulainen. Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*. IEEE, 2002.

[18] K. O. Stanley. *Efficient Evolution of Neural Networks through Complexification*. PhD Thesis, University of Texas at Austin, Report AI-TR-04-314, 2004.

[19] K. O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), San Francisco: Kaufmann.*, 2002.

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

[21] K. O. Stanley and R. Miikkulainen. Evolving a roving eye for go. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Berlin: Springer Verlag.*, 2004.

[22] M. Timothy. *Signal and Image Processing with Neural Networks*. John Wiley & Sons, Inc, 1994.

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

[24] A. P. Wieland. Evolving controls for unstable. systems. *Proc. Connectionist Models Summer School*, pages 91–102, 1991.

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