Efficient representation of Recurrent Neural Networks for Markovian/Non-Markovian Non-linear Control Problems

Maryam Mahsal Khan
Department of Computer Systems Engineering
University of Engineering & Technology
Peshawar, Pakistan
Email: maryam@nwfpuet.edu.pk

Gul Muhammad Khan
Department of Electrical Engineering
University of Engineering & Technology
Peshawar, Pakistan
gk502@nwfpuet.edu.pk

Julian F. Miller
Department of Electronics
University of York
UK
jfm7@ohm.york.ac.uk

***Abstract*—A novel representation of Recurrent Artificial neural network is proposed for non-linear markovian and non-markovian control problems. The network architecture is inspired by Cartesian Genetic Programming. The neural network attributes namely weights, topology and functions are encoded using Cartesian Genetic Programming. The proposed algorithm is applied on the standard benchmark control problem: double pole balancing for both markovian and non-markovian cases. Results demonstrate that the network has the ability to generate neural architecture and parameters that can solve these problems in substantially fewer number of evaluations in comparison to earlier neuroevolutionary techniques. The power of Recurrent Cartesian Genetic Programming Artificial Neural Network (RCGPANN) is its representation which leads to a thorough evolutionary search producing generalized networks.**

***Keywords*-Artificial Neural Network, Pole Balancing, Non-Linear Control Problems, NeuroEvolution, Recurrent Networks.**

I. Introduction

Recurrent Neural Network (RNN) represents a class of artificial neural architecture that exploits the dynamic behaviour of a system. RNN are used in systems which are represented by complex sets of constraints and where a simple feed forward structure fails to converge. RNN are applied in a wide variety of applications ranging from control to chemical processes and manufacturing, speech detection, lung sound detection, time series prediction etc [9].

Numerous evolutionary algorithms producing RNN architecture have been applied on various tasks like developing linear and non-linear controller of a standard benchmark problem - inverted pendulum, optimum localization of mobile robots, auto-pilot helicopter or aircraft controller, automobile crash warning system, rocket control, routing over a data network, coordinating multi-rover systems and octopus arm task [14], [16], [7], [8], [3], [12], [2]. All the developed systems have been found efficient and powerful as compared to the systems produced by conventional methods.

In this paper, we present a novel neuroevolutionary algorithm that is based on the representation of Cartesian Genetic Programming producing a recurrent architecture known as RCGPANN. CGPANN generating feedforward architecture has already been invesitigated [8]. RCGPANN is applied on a standard benchmark problem - pole balancing which ranges from simpler setup to extremely difficult versions. The results obtained demonstrates that the proposed technique out performs all the previous methods explored to date. The paper is organized as follows. Section II is an overview on NeuroEvolution and the different algorithms developed so far. Section III describes the background information on Cartesian Genetic Programming. Section IV describes the Neuroevolutionary algorithm based on Cartesian Genetic Programming in detail. Section V - VIII describes the algorithm applied on the standard benchmark problem i.e. double pole balancing task along with simulation analysis and results. Section IX concludes with discussions and future work.

II. NeuroEvolution

Neuroevolution (NE) refers to the evolution of different attributes of a neural network i.e. connection weights, connection type, node (neuron) function or even the topology of the network. The genotype represent these parameters and 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 one parameter of the network, some evolve two or three in combination. If only weights are evolved in a fixed topology, the network solution space is restricted, and evolution may not be able to find a novel solution to the problem [17].

In Symbiotic, Adaptive Neural Evolution (SANE) the neuron population along with the network topologies representing the blue-prints are simultaneously evolved [12]. It has successfully been applied to 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 subpopulation of hidden layer neurons is evolved. It produced better results than SANE [4].

978-1-4244-8136-1/10/$26.00

In Conventional Neuroevolution (CNE) a genotype represents the whole neural network. It uses rank selection and burst mutation. The conventional algorithm has advantages over cooperative coevolution 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 [3].

Stanley presented an algorithm called NeuroEvolution of Augmenting Topologies (NEAT). He identified and solved the three major challenges: tracking genes with historical markings to allow easy crossover between different topologies, protecting innovation via speciation, and starting from a minimal structure and "complexifying" as the generations pass. NEAT was shown to perform faster than many other neuro-evolutionary techniques. NEAT has also produced good results on pole balancing problem [15].

Cooperative Co-evolution has been introduced in the area of evolutionary computation focused on the evolution of co-adapted subcomponents. In Cooperative Coevolutionary 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. COVNET has shown to produced better generalized and smaller networks [2].

Cooperative Synapse Neuroevolution (CoSyNE) evolves neural network at the level of synaptic weights only. Coevolution of the synaptic weight and multi-point crossover and probabilistic mutation is applied based on Cauchy distributed noise. The CoSyNE approach has been shown to out-perform all the previous approaches on pole balancing problem [1].

Continuous Time Recurrent Neural Network (CTRNN) is 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 [10].

III. Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) is an Evolutionary Programming technique developed by Miller and Thomson for digital circuits optimization [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).

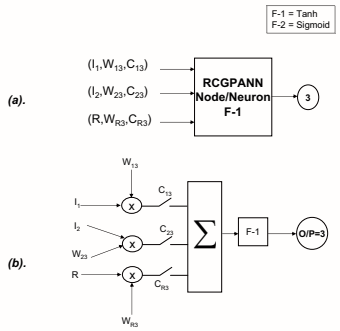

Figure 1. (a)RCGPANN node with two inputs (b) Inside View of RCGPANN for node in(a)

The 1+ λ (where $\lambda = 4$) 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.

IV. Recurrent Cartesian Genetic Programming based Artificial Neural Network

In order to capture a broader domain of systems that are dynamic and non-linear, the need for recurrent networks becomes essential. This section describes a NeuroEvolutionary algorithm that exploits the powerful representation of CGP in generating recurrent artificial neural architecture.

Recurrent Architecture are NeuroEvolved based on Cartesian Genetic Programming hereby known as RCGPANN. It follows the direct encoding strategy where it encodes topology, weight and functions in one genotype and then evolves it for augmented topology, best possible weights and functions. The 1+4 evolutionary strategy is adopted to generate offsprings.

In general TWEANN- (Topology and Weight Evolution) algorithms are either constructive or destructive [17]. RCGPANN is both constructive and destructive algorithm. It begins with random topological features and incrementally removes and adds new features. It does so by mutating functions, inputs, weights, connection types and outputs. When a connection is disabled through mutation it is not fully removed. It has a probability to become re-enabled in the later generation.

The network derived constitutes neurons that 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. The recurrent network produced is a representation of 'Jordan Network' [13].

The RCGPANN genotype consists of nodes that represents neurons of ANN. The node includes inputs, weights, connection and function as shown in Fig.1(a). Inputs can

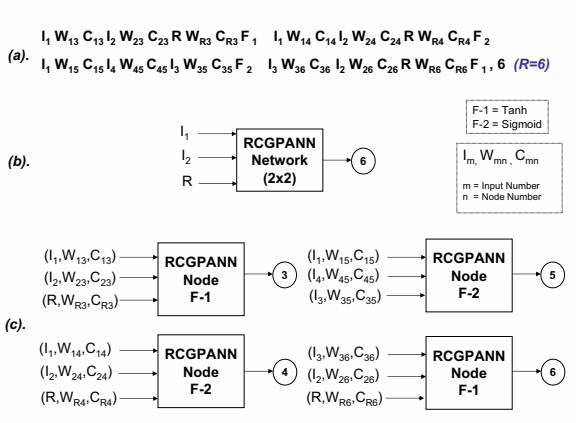

Figure 2. (a) Genotype of a 2x2 RCGPANN network, (b) Block Representation of the Genotype in (a), (c) Graphical Representation of the Genotype in (a)

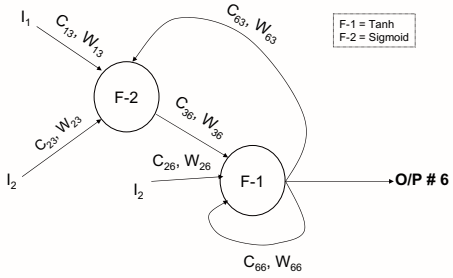

Figure 3. Phenotype of the Genotype in Figure 2(a)

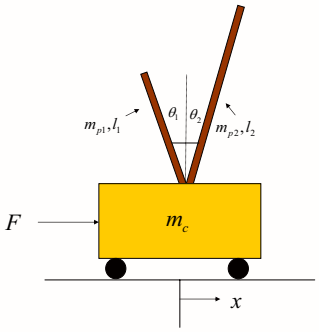

Figure 4. Double Pole Balancing

be program inputs or inputs from the previous nodes or a feedback. The first layer of the RCGPANN genotype consists of inputs that are recurrent. The layers following that might have recurrent connections based on whether the feedback input is randomly selected as an input to the node or not. An input is said to be connected if the Connection is 1 otherwise the Connection is 0. Weights are randomly generated between -1 to +1 but for the feedback input the weight is always assigned a value 1. Input and Weight are multiplied for all the connected inputs and is summed up. It is then forwarded to a linear or a non-linear function such as sigmoid, tangent hyperbolic, step or linear to produce an output at each node. This output can either be the input to the next node or output of the system. The output(s) of the genotype can be output(s) from any node or program input(s). The output of the genotype is fed back to the nodes if the recurrent input is enabled or connected.

The RCGPANN genotype is then evolved from one generation to next (through the process of mutation) until the desired fitness is achieved. No mutation is applied on the connection or weight of the state units (feedback units). The resultant genotype is then transformed to an artificial neural architecture.

Fig.1(a) is a block representation of a 3-input RCGPANN node with inputs (I_1, I_2, R), weights (W_{13}, W_{23}, W_{R3}) and connections $(C_{13}, C_{23}, C_{R3} = 1)$. Fig.1(a) represents the parameters of node '3'. W_{13} corresponds to a weight assigned to I_1 , W_{23} represents weight of I_2 and R represents a recurrent input which is the output of the system fedback as an input having a weight of W_{R3}=1 for node '3' respectively. The initial value of the recurrent input R is taken as 0. Fig.1(b) displays the internal view of RCGPANN node. The three inputs (I_1, I_2, R) are multiplied with the corresponding weights (W_{13}, W_{23}, W_{R3}) (all 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.2(a) is a RCGPANN genotype of a 2x2 network with 3 inputs (I_1, I_2, R), 2 functions (F-1 and F-2) and one output node '6'. Fig.2(b) represents the block diagram of the genotype in Fig.2(a). Fig.2(c) shows the inside view of the network in Fig.2(b). The node '6' takes input I_3 , I_2 and its feedback value which for initial step is 0. I_3 is the output of node 3 having I_1 , I_2 and a feedback of the system (in this case R represents output of node '6') as inputs with function F_1. 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. 3.

V. Case Study : Pole Balancing

Pole balancing is a standard benchmark problem from the field of control theory and artificial neural networks for designing controllers for unstable and non-linear systems [5]. In this section, we will demonstrate RCGPANN on the double pole balancing experiments.

Double pole balancing requires balancing two poles, 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 doesn't exceed

Table I
PARAMETERS FOR DOUBLE POLE BALANCING TASK

Parameters	Value
Mass of cart (m_c)	1 Kg
Mass of Poles (m_{p1},m_{p2})	0.01,0.1 Kg
Length of Poles (l_1,l_2)	0.05,0.5 m
Friction of the Poles (μ_p)	0.000002
Friction of the cart (μ_c)	0.0005
Width of the Track (h)	4.8m

(-36° $<\theta_{1,2}<$+36°) and the cart doesn't leave the track. The controller has to balance the pole(s) 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(s) balanced for as long as possible where a force 'F' greater than or equal to zero corresponds to +10N and 'F' less than zero corresponds to -10N. The system inputs are the pole-angle(s) θ_1 and θ_2 ,velocity of pole(s) $\dot{\theta}_1$ and $\dot{\theta}_2$, position of cart x and velocity of cart $\dot{x}$. Fig.4 shows the double pole balancing scenario.

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

Equations that are used to compute the effective mass of poles, acceleration of poles, acceleration of cart, effective force on each pole, the next state of angle of poles, velocity of poles, position of cart and velocity of cart can be found in [8].

The proposed NeuroEvolutionary algorithm shall be used on the double pole balancing tasks on the following sets of conditions:

A. With Velocity - Zero and Random Initial states

In Case(A) the network or controller has access to six (x,$\dot{x}$,θ_1,$\dot{\theta}_1$,θ_2,$\dot{\theta}_2$) input state variables.

B. Without Velocity - Zero and Random Initial states

Case(B) is a non-markovian problem where the input state variables are reduced to three (x,θ_1,θ_2) variables.

In both the cases, the networks shall be generated with initial input values of zero or with random values of $(-0.6rad < \theta_{1,2} < 0.6rad)$ and $(-2.4 <$x$< +2.4)$.

VI. EXPERIMENTAL SETUP

For the two cases, RCGPANN genotypes of different network sizes with randomly generated connections, weights, outputs and inputs are created. Weights are randomly produced between -1 to +1. The activation functions used are sigmoid and tangent-hyperbolic. The mutation rate of 10% and 20% are used for all models. Results are simulated for zero and random initial states. The performance of the algorithm is based on the average number of balancing attempts (evaluations).

Table II
PERFORMANCE OF RCGPANN ON DOUBLE POLE - WITH VELOCITIES

Network Representation			Evaluations
Initial State	Network Arch.	Mutation Rate	RCGPANN
Zero	5x5	0.1	229
	10x10	0.1	213
	15x15	0.1	**197**
	5x5	0.2	309
	10x10	0.2	289
	15x15	0.2	224
Random	5x5	0.1	419
	10x10	0.1	335
	15x15	0.1	241
	5x5	0.2	314
	10x10	0.2	941
	15x15	0.2	569

Table III
PERFORMANCE OF RCGPANN ON DOUBLE POLE - WITHOUT VELOCITIES

Network Representation			Standard Fitness	Damping Fitness
Initial State	Network Arch.	Mutation Rate	RCGPANN	RCGPANN
Zero	5x5	0.1	294	2308
	10x10	0.1	**216**	1074
	15x15	0.1	231	429
	5x5	0.2	469	917
	10x10	0.2	317	**387**
	15x15	0.2	409	450
Random	5x5	0.1	1033	2349
	10x10	0.1	1215	2184
	15x15	0.1	**881**	1581
	5x5	0.2	1789	41491
	10x10	0.2	1725	**1007**
	15x15	0.2	1302	1409

A. Double Pole: With Velocity

As the double pole balancing involves balancing two poles. This adds a non-linear interaction between the poles. The input parameters to the network are: position of cart x, angle of both the poles $\theta_{1,2}$, velocity of cart $\dot{x}$ and the velocity of both the poles $\dot{\theta}_{1,2}$. The number of inputs per node (neuron) is set to 7.

Table II represents the comparison of the average number of balancing attempts for inputs starting from initial states of zero and random values of network with varying network sizes and mutation rates. The results are the average of 50 independent evolutionary runs.

B. Double Pole: Without Velocity

For the Without Velocity case the controller does not have any information of the velocity of cart or poles, thus making the problem non-markovian. The input to the network are position of cart x and the angle of poles $\theta_{1,2}$. The input to the RCGPANN node is set to 4 where the additional input represents a feedback. With such limited information the networks thus have to compute the hidden velocity information from the internal connections. As the velocity information is not provided to the controller it

requires recurrent networks to compute the hidden velocity information of the pole(s) and the cart [4], [12], [15].

The RCGPANN networks are trained on the standard fitness and the damping fitness function. The damping fitness function devised by Gruau [6] represents two fitness measurements simulated at 1000 time steps expressed by Eq.(1) and Eq.(2).

$$f_1 = \frac{t}{1000} \tag{1}$$

$$f_2 = \frac{0.75}{\sum_{i=t-100}^{t}(x^i + \dot{x}^i + \theta_2^i + \dot{\theta}_2^i)} \tag{2}$$

The intent of introducing the damping fitness function is to force the network to generate internal velocity information and also to penalize oscillations. The network is simulated for 1000 time steps and the best network from each generation is tested on the standard fitness function. If the networks fails to pass the standard fitness then the evolutionary algorithm moves on to the next generation.

Table III represents the comparison of the average number of balancing attempts for inputs starting from initial states of zero and random values of network with varying network sizes and mutation rates. The results are the average of 50 independent evolutionary runs.

VII. Results And Discussions

Table II demonstrates the performance of RCGPANN on the double pole balancing task with velocity information. Random behaviour of evaluation was observed with increasing network size. For RCGPANN with 0.1 mutation rate the 15x15 has produced the minimum evaluation of 197.

Table III exhibits the performance of RCGPANN on the double pole balancing task without velocity information. As the velocity information is removed from the network it makes the problem non-linear and non-markovian [3]. The network controller has to compute velocity information internally from the network connections. The number of evaluations was found to randomly change with increasing network size and constant mutation rates. Using a mutation rate of 0.1 with a network size of 10x10 the RCGPANN network was able to found the solution at an average of 216 evaluations for the standard fitness function while at 0.2 mutation rate the 10x10 has produced the minimum evaluation of 387 for the damping fitness function.

Fig.5(a,c) displays the neural network structures of the RCGPANN evolved genotypes for with and without velocity conditions. Fig.5(b,d) represents the corresponding pole angles and position of cart simulated for 30 minutes respectively (100,000 steps are down-sampled to produce 100 steps for demonstration purpose only).

Table IV represents the comparison of RCGPANN with other neuroevolutionary techniques for both markovian and non-markovian cases of the double pole balancing task.

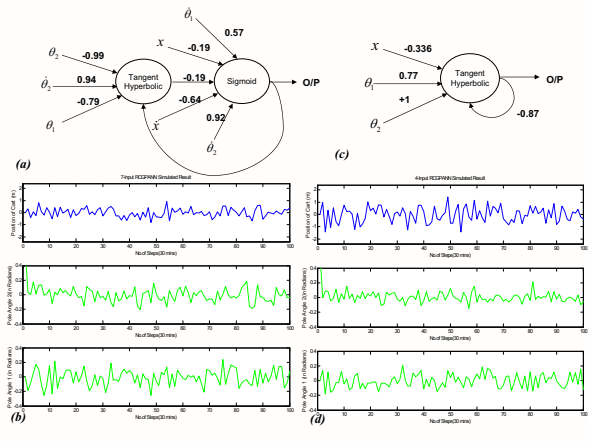

Figure 5. Double Pole Balancing Task (a) Phenotype of the Evolved Genotype With Velocity (b) Pole Angle and Position of Cart simulated for the ANN in (a) for 100,000 steps (c)Phenotype of the Evolved Genotype Without Velocity (d) Pole Angle and Position of Cart simulated for the ANN in (c) for 100,000 steps

Table IV
Comparison of CGPANN with other NeuroEvolutionary Algorithms Applied on Double Pole Balancing Task

Method	With Velocity	Without Velocity	
	Standard Fitness	Standard Fitness	Damping Fitness
CNE	22100	76906	87623
SANE	12600	262700	451612
ESP	3800	7374	26342
NEAT	3600	—	6929
CoSyNE	954	1294	3416
RCGPANN	**197**	**216**	**387**

With velocity information RCGPANN produced a minimum evaluation of 197. While for the without velocity, the RCGPANN network solved the problem in a minimum of 216 evaluations for the standard fitness and 387 for the damping fitness function. The number of evaluations attained is considerably fewer than all previously published figures of other neuroevolutionary algorithms. This exemplifies the potential of RCGPANN network in generating optimum and quicker solutions.

VIII. Generalization

'Generalization' refers to the successful balancing attempts starting from 625 random initial values for 1000 steps only. Networks with higher generalization score generalizes better.

For 50 different genotypes at without velocity conditions, the RCGPANN scored 335.84/625 for the damping fitness function. This indicates that many evolved genotypes demonstrate a generalized behaviour. Table V exhibits the generalization ability of various NeuroEvolutionary algorithms for the double pole balancing scenerio using the

Table V
COMPARISON OF GENERALIZATION WITH NEUROEVOLUTIONARY ALGORITHMS - DOUBLE POLE (DAMPING FITNESS)

Method	Value
CE	300
ESP	289
NEAT	286
RCGPANN	**335.84**

damping fitness function. It is observed that the RCGPANN exhibits greater generalization ability as compared to other techniques.

It is important to mention that the performance of the RCGPANN 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 likely enhance the learning speed.

IX. CONCLUSION

In this paper NeuroEvolution based on Cartesian Genetic Programming with a recurrent architecture (RCGPANN) was exploited on a standard benchmark control problem i.e. double pole balancing task with and without velocity information. The results presented in this paper shows that RCGPANN extends the powerful and flexible representation of CGP through evolution of neural network parameters and that it has the potential to generate ANN solutions to problems amenable to reinforcement learning.

It was observed that RCGPANN has generated solutions with fewer evaluations for the double pole balancing tasks as compared to other published techniques at markovian and non-markovian states. Simulation results also shows that the RCGPANN produces robust controllers with better generalization ability. The devised controllers are stable under a wide range of initial states. Clearly RCGPANN stands competitive with established Neuro-Evolutionary techniques like NEAT, SANE, ESP and CoSyNE.

Future work in CGPANN involves modifying the RCGPANN algorithm to incorporate levels back (a connectivity parameter in CGP), arity, adaptive mutation rate and adaptive network size. It is also planned to evaluate RCGPANN genotypes for a larger class of ANN problems. This opens new avenues of applying the proposed technique to any non-linear and dynamic controller.

REFERENCES

[1] C. Conforth and Y. Meng. Toward evolving neural networks using bio-inspired algorithms. In *Proc. Int. Conf. on Artificial Intelligence*, pages 413–419, 2008.

[2] 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.

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

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

[5] J. Gomez, F. Schmidhuber and R. Miikkulainen. Efficient non-linear control through neuroevolution. In *ECML 2006: Proceedings of the 17th European Conference on Machine Learning*. Springer, 2006.

[6] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the First Annual Conference*, pages 81–89. MIT Press., 1996.

[7] J. Koutnik, F. Gomez, and J. Schmidhuber. Evolving neural networks in compressed weight space. In *GECCO '10*, pages 619–626. ACM, 2010.

[8] M. M. Khan, G. M. Khan, and J. F. Miller. Evolution of Optimal ANNs for Non-linear Control problems using Cartesian Genetic Programming. In *Proceedings of IEEE International Conference in Artificial Intelligence*, 2010.

[9] P. A. Mastorocostas and I. B. Theocharis. A stable learning algorithm for block-diagonal recurrent neural networks: Application to the analysis of lung sounds. *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, 36(2):242–254, 2006.

[10] 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.

[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. 1997.

[13] P. Smyth, D. Heckerman, and M. Jordan. Probabilistic Independence Networks for Hidden Markov Probability Models. *Neural Computation*, 9:227–269, 1996.

[14] K. Stanley, R. Sherony, N. Kohl, and R. Miikkulainen. Neuroevolution of an Automobile Crash Warning System. In *Proc. GECCO-2005.*, 2005.

[15] 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.

[16] K. Tumer. Coordinating multi-rover systems: Evaluation functions for dynamic and noisy environments. In *GECCO-2005*, pages 591–598. ACM Press, 2005.

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