A HYBRID GA/PSO TO EVOLVE ARTIFICIAL RECURRENT NEURAL NETWORKS[1]

MATTHEW L. SETTLES
University of Idaho
Department of Computer Science
Moscow, Idaho 83844

TERENCE SOULE
University of Idaho
Department of Computer Science
Moscow, Idaho 83844

ABSTRACT

In this paper we propose a novel hybrid algorithm (GA/PSO) combining the strengths of particle swarm optimization with genetic algorithms to evolve the weights of recurrent neural networks. Particle swarm optimization and genetic algorithms are two optimization techniques that have proven to be successful in solving difficult problems, in particular both can successfully evolve recurrent neural networks. The hybrid algorithm combines the standard velocity and update rules of PSOs with the ideas of selection and crossover from GAs. We compare the hybrid algorithm to both the standard GA and PSO models, when evolving weights to two recurrent neural network problems. This paper presents results using the new hybrid algorithm and describes the hybrids benefits.

INTRODUCTION

Genetic algorithms (GA) and particles swarm optimization (PSO) are both population based algorithms that have proven to be successful in solving very difficult problems, including recurrent artificial neural networks (RANN). However, both models have strengths and weaknesses. Comparisons between GA and PSOs have been performed by both Eberhart (1998) and Angeline (1998) and both conclude that a hybrid of the standard GA and PSO models could lead to further advances. In this paper we present a novel hybrid algorithm, combining the strengths of GAs with those of PSO. The hybrid algorithm is compared to the standard GA and PSO models in evolving the weights of two RANN problems.

Traditional neural network (NN) training algorithms, such as back propagation, are based on gradient descent. They systematically and incrementally reduce the output error. Although gradient descent approaches are very effective for a wide range of problems, they suffer from two significant drawbacks. First, they are generally restricted to finding local minimum. Second, they may get stuck in flat regions of a search space. In such cases the typical recourse is to restart the algorithm from a new random location. In contrast, population based searches are not restricted to a local search and can easily move across flat regions. However, they tend to be slower than gradient

[1] This work is supported by NSF EPSCoR EPS-0132626. The experiments were performed on a Beowulf cluster built with funds from NSF grant EPS-80935 and a generous hardware donation form Micron Technologies.

descent methods and so are only appropriate for particularly difficult problems, such as finding the weights for a RANN.

RANNs are NNs in which nodes in the 'later' layers of the network may feed back to nodes in 'earlier' layers. RANNs have several significant features missing from feedforward NNs. Feedforward NNs are limited to input vectors of fixed dimension; data received in a more general format may need extensive preprocessing (Hammer, 2000). A typical example of this difference occurs in time series data. A feedforward NN is limited to looking at a fixed size window of data, determined by the number of input nodes. Associations in the data that extend beyond this window cannot be found by the NN. In contrast, in a RANN the recurrent connections allow the network to 'store' information received earlier by feeding it back to an earlier layer of the network. Thus, the RANN's calculations are not limited by its input window.

Unfortunately, RANNs are very difficult to train using gradient descent based approaches. First, the additional recurrent connections greatly expand the overall search space, making local minima and flat regions much more likely. Second, gradient descent training requires 'unfolding' the network. To train a network in which a particular input influences the network N time steps into the future, requires training the network as if it were a feedforward network with N times as many layers (Hammer, 2000). This greatly increases the time and difficulty of training the RANN.

GAs and PSO do not suffer these drawbacks. Because they are performance (fitness) based no unfolding is required. Furthermore because both methods are population based they are much less likely to become stuck at a local optima or in a flat region of the search space. Thus, they are much better suited to the complex search spaces created by RANNs. There have been several successful examples of using a GA to evolving RANN architectures and weights (Angeline et al, 1994, Saunders et al, 1994, Mandischer, 1995). Salerno (1997) used a PSO to evolve the weights of a RANN to solve the XOR problem and parsing natural language phases.

Both GA and PSO, however, have their own set of strengths and weaknesses. The PSO algorithm is conceptually simple and can be implemented in a few lines of code. PSOs also have memory, whereas in a GA if an individual is not selected the information contained by that individual is lost. However, without a selection operator PSOs may waste resources on a poor individual that is stuck in a poor region of the search space.

A PSO's group interactions enhances the search for an optimal solution, whereas GAs have trouble finding an exact solution and are best at reaching a global region (Eberhart, 1998). Hybrid GA/Gradient Searches are often used to overcome this problem.

Here we propose a hybrid algorithm (GA/PSO) combining the strengths of GAs with PSO to evolve recurrent neural networks. The hybrid algorithm combines the standard velocity and position update rules of PSOs with the ideas of selection and crossover from GAs. The algorithm is designed so that the GA performs a global search and the PSO performs a local search. Other hybrid GA/PSO algorithms have been proposed, and have been tested on function minimization problems, but not NN or RANN (Lvbjerg et al, 2001, Robinson et al., 2002).

In previous work, a GA and PSO were tested and compared when evolving the weights for a fixed topology RANN (Settles and Soule, 2003). Both the GA and PSO were successful in evolving RANNs to produce a periodic output in response to a fixed input signal. However, the algorithms produced different results for different network sizes. Thus, demonstrating that they implement different search strategies. We believe that the correct combination of both models has the potential to achieve better results faster. This paper is an extension of that research.

GENETIC ALGORITHM

Genetic Algorithms were first introduces by John Holland in the 1960s. In this problem, the GA uses a chromosome consisting of real values. Each real value corresponds to the weight between one pair of nodes (or between a node and itself, which is allowed in our strongly connect RANNs).

The genetic algorithm is generation. The five best individuals are copied into the next generation (elitism). Tournament selection is used, with a tournament of size 3. The mutation rate is 1% and can change a weight by up to $\pm 25\%$ of the weight's original value.

When crossover is applied to two individuals the same random (non-input) node is chosen in both individuals. The weights of all connections leading *into* the chosen node are exchanged between the two individuals. The crossover rate used is 0.8.

PARTICLE SWARM OPTIMIZER

The PSO algorithm was first proposed by Kennedy and Eberhart (2001). The PSO algorithm is an adaptive algorithm based on a social-psychological metaphor; a population of individuals (referred to as particles) adapts by returning stochastically toward previously successful regions of the search space.

The PSO consists of a population of particles, where each particle keeps track of its current position, velocity, best position thus far, best fitness thus far and current fitness. The position and velocity vectors refer to the particle's position and velocity within the search space. In this problem, a particles position and velocity are real valued vectors, with one value for each network weight. The best fitness value achieved thus far is retained, as well as the particles position at that fitness value. The current fitness value is obtained by evaluating an error function at the particle's current position. Thus the movement of each particle through the search space is governed by Equations (1) and (2).

$$\vec{v}_i(t) = I * \vec{v}_i(t-1) + \rho 1(\vec{p}_g - \vec{x}_i(t-1)) + \rho 2(\vec{p}_i - \vec{x}_i(t-1)) \quad (1)$$

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \quad (2)$$

Where $\vec{x}_i(t)$ is the position of the particle at time t, $\vec{v}_i(t)$ is the velocity of the particle at time t, I is the *inertia* weight, reduced linearly from 0.9 to 0.2 by epoch in this problem, $\vec{p}_g$ is the position of the global best particle, $\vec{p}_i$ is the particles previous best position and ρ_1 and ρ_2 are random values in the range [0.0,1.0].

To prevent the particle from venturing outside the search space a limit, called V_{max}, is applied to the velocity equation. If the velocity term exceed V_{max}, V_{max} is used as the new velocity for the particle in that dimension, keeping the particle in the search space.

THE HYBRID (GA/PSO) MODEL

Both Angeline (1998) and Eberhart (1998) have suggested that a hybrid combination of the GA and PSO models could produce a very effective search strategy. Our goal is to introduce an adjustable hybrid model that makes it possible to optimize the GA/PSO hybrid. Our results show that with the correct combination of GA and PSO the hybrid does outperform both the standard PSO and GA models, when evolving the weights of RANNs.

The hybrid algorithm combines the standard velocity and position update rules of PSOs with the ideas of selection and crossover from GAs. Two additional parameters are added in order to give preference to either the GA or PSO. The PSO's velocity vector is multiplied by an *influence* term, when this term is set to 0 the PSO has no effect on the population, when set to 1 the PSO runs as a standard PSO described above. For intermediate values the PSO functions normally, but the size of the steps taken by the particles is reduced.

The GAs selection operator has a *replacement* term added which determines how many individuals in the population get replaced and crossed over in the current generation.. When the *replacement* term is 0 no individuals/particles are selected or crossed and the GA has no effect on the population. When the *replacement* term is set to 1 the entire population is replaced in the generation. The GA mutation operator was not used in the hybrid algorithm.

First, the hybrid algorithm performs the standard velocity and position update rules, with the *influence* term. Selection and Crossover then occur on the appropriate number of individuals determined by the *replacement* term, the top (*population size – replacement term*population size)* individuals, based on fitness, are copied into the new population. Tournament selection, with a tournament size of 3, is used to select parents to fill the remainder of the population. Two crossover operators for the hybrid were used in these experiments, node crossover, as described above, and a new variation of crossover: node weighted average crossover. Node weighted average crossover is a hybrid crossover that incorporates the PSO velocity vector. The goal is to create two new child particles whose position is between the parents position, but accelerated away from the current direction to increase diversity. Node weighted average crossover chooses a random (non-input) node and takes the average of the two weights multiplied by the negative velocity of one of the parent weights, this applies to all of the connections leading into the chosen node. Equations (3) and (4) show how the new child weights are determined.

$$child1(xi) = -parent1(vi) * (parent1(xi) + parent2(xi)) / 2.0 \quad (3)$$

$$child2(xi) = -parent2(vi) * (parent1(xi) + parent2(xi)) / 2.0 \quad (4)$$

The children's velocity vectors are exchanged at the same node and the previous best vector is set to the new position vector, effectively restarting the children's memory. Fitness is evaluated on the new population and the process is repeated.

Four tests of the hybrid algorithm, using different *influence* and *replacement* parameters and crossover operators, were tried (see Table 1).

All tests are performed with a population size of 500 and each test is run 50 times.

Table 1: Test algorithm parameters and crossover type. $0.0 \rightarrow 1.0$ means that the parameter begins at 0.0 in the first generation and is linearly increased, by generation, to 1.0 in the last generation.

Model	Influence	Replacement	Crossover Type
GA	$0.0 \rightarrow 0.0$	$0.99 \rightarrow 0.99$	Node crossover
PSO	$1.0 \rightarrow 1.0$	$0.0 \rightarrow 0.0$	Node crossover
Hybrid-1	$0.0 \rightarrow 1.0$	$0.99 \rightarrow 0.0$	Node crossover
Hybrid-2	$1.0 \rightarrow 0.0$	$0.0 \rightarrow 0.99$	Node crossover
Hybrid-3	$1.0 \rightarrow 1.0$	$0.5 \rightarrow 0.5$	Node crossover
Hybrid-4	$1.0 \rightarrow 1.0$	$0.5 \rightarrow 0.5$	Node weighted average crossover

The rows in Table 1 are as follows:

- GA - used an *influence* term of 0, so the PSO has no effect on the population. Replacement was set to 0.99 throughout, which is effectively a normal generational GA with elitism of 5. A mutation operator was added in this test.
- PSO - used an *influence* term of 1 throughout the entire test. Replacement was set to 0, so the GA has no effect on the population. This implementation of the PSO is a standard implementation.
- Hybrid-1 - increases the *influence* term linearly from 0.0 to 1.0, based on the current generation. The *replacement* term is reduced linearly from 0.99 to 0.0, based on the current generation. This test was designed with the expectation that the best results would be with the GA performing an initial global search with the PSO finishing as a local search.
- Hybrid-2 - reduces the *influence* term from 1.0 to 0.0 and increases the *replacement* term from 0.0 to 0.99. This test was designed with the expectation that the PSO performing an initial global search with the GA finishing as a local search would produce poor results.

The results (described below) ran counter to our expectations, so two additional test are included.

- Hybrid-3 - sets the *influence* term to 1.0 and the *replacement* term to 0.5, throughout the entire run. Node crossover is used. The goal

of this test was to allow the PSO to naturally converge (local search) on a solution, while allowing half the population to "spread out" through crossover.

- Hybrid-4 - Is an extension to Hybrid-3. The *influence* and *replacement* terms remain the same, but node weighted average crossover was used. Here the PSO is again allowed to naturally converge on a solution, while half the population, due to the averaged weights and negative velocity, actively "disperses", increasing diversity.

RECURRENT ARTIFICIAL NEURAL NETWORK MODEL

The neural network used in this experiment is a strongly connected recurrent neural network. It has a single input node and a single output node. Every internal node is connected to every other node and to itself. A connection between the input node and the output node also exits. The number of hidden layers and nodes per hidden layer is fixed for each experiment. The number of weights in the network is

$$(L * N)^2 + 2(L * N) + 1 \tag{5}$$

where L is the number of layers and N is the number of nodes per layer.

At each time step the input node's activation level (0 or 1) is set. Then for each hidden layer the activation level of the nodes in that layer are calculated. The activation levels for a layer are updated to the new values only after all of the activation levels for that layer have been calculated. Then the process repeats for the next hidden layer.

Effectively the activation levels of the neurons in the same hidden layer are updated simultaneously. The activation levels of the neurons in different layer are updated sequentially.

FINITE STATE MACHINE

To compare the ability of the hybrid model, we chose a finite state machine. The problem is to learn the minimal six state finite state machine shown in Figure 1.

A complete set of 254 strings consisting of all strings of length one through seven were used to train and test the evolved network. The set of strings was randomly divided into a training set and a testing set, each consisting of 127

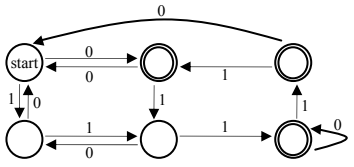

Figure 1: A six state finite state machine.

strings. The training set consisted of 58 positive cases and 69 negative cases. The testing set consisted of 55 positive cases and 72 negative cases. Both sets were ordered lexographically.

A network with a size of 2 layers and 3 nodes per layer consisting of 49 weights was used in this problem. Training was allowed to continue for 1000 generations. The initial weights were randomly chosen in the range [-2.0,2.0].

All nodes in the network used the symmetric sigmoid activation function, with an output in the range [-1.0,1.0], except the output node which used the standard sigmoid function, with output in the range [0.0,1.0]. An output equal to or greater than 0.5 was considered a positive classification of the current input string, outputs less than 0.5 were considered a negative classification.

Table 2 summarizes the results for each experiment. Figures 2 and 3 show the best and average results for the GA, PSO, Hybrid-1 and Hybrid-4, respectively.

Hybrid-4, the standard PSO algorithm and Hybrid-3 perform the best on this problem, with Hybrid-4 marginally outperforming Hybrid-3 and the PSO. Additionally, Hybrid-4 finds a solution more quickly than any other algorithm. Hybrid-4 also has the best overall run with a training error of 10% and testing error of 24%. Interestingly Hybrid-4 has the poorest average results (Figure 3). This occurs because node weighted average crossover actively disperses the

Table 2: Percentage classification error on the FSM experiment for each training model.

Model	Training error		Testing error	
	Average Best	Std-dev	Average Best	Std-dev
GA	16.0	8.2	36.2	7.5
PSO	13.9	4.5	34.4	5.9
Hybrid-1	24.3	4.6	37.7	7.3
Hybrid-2	14.4	3.9	33.5	5.6
Hybrid-3	13.9	4.7	33.6	4.7
Hybrid-4	13.8	3.6	32.5	5.0

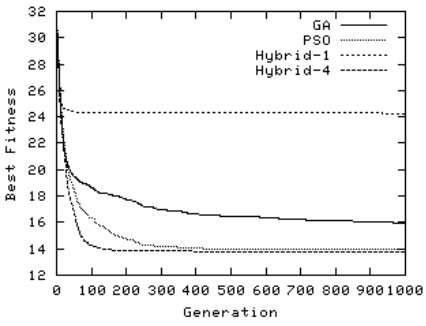

Figure 2: Best Fitness values of each algorithm over 50 runs

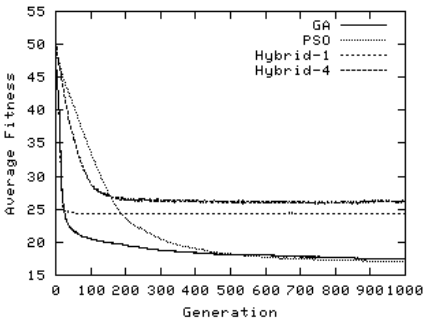

Figure 3: Average fitness values of each algorithm over 50 runs

particles, leading to a higher average fitness.

Hybrid-2 and the standard GA algorithm both perform reasonable well. Hybrid-1 did not perform well on this problem. We believe that the combination of GA selection removing the poor individuals and the PSO moving all individuals closer together creates to much pressure for convergence, leading the population to converge on a suboptimal solution after only a few generations.

SIMULATING A BIOLOGICAL NEURON

The second problem was to evolve a network that produces a simple pulsed output when an activation 'voltage' is applied to the networks input. The test lasts for 100 time steps. The activation signal is applied to the input node at 10 time steps and is turned off at 90 time steps (I.e. the input node is set to one from time step 10 to step 90 and is set to zero at all other times).

While the input is high the goal is to produce a pulsed output with a period of 20 time steps. The error is the sum of the absolute value of the difference between the desired output and the actual output at each time step plus a penalty of 0.5 if the slope of the desired output differs in the direction from the slope of the actual output.

As with the previous problem, a network with 2 layers and 3 nodes per layer consisting of 49 weights was used. All nodes in the network use a symmetric sigmoid activation function, with an output in the range [-1.0,1.0].

Training was allowed to continue for 250 generations. The initial weights were randomly chosen in the range [-2.0,2.0]. A successful run was defined as a periodic output of the desired frequency.

Table 3 summarizes the results for each experiment. Figures 4 and 5 show the best and average results for the GA, PSO, Hybrid-1 and Hybrid-4, respectively.

Again Hybrid-4 and the standard PSO algorithm outperform all other models on this problem, with Hybrid-4 marginally outperforming the PSO algorithm. (From previous experiments we know that the GA can outperform the PSO on this problem with slightly larger networks (Settles and Soule, 2003).) Again Hybrid-4 is able to find a better solution faster than any other algorithm and Hybrid-4 has the best overall run with an output more closely matching that of the desired output. As before Hybrid-4 has relatively poor average results due to the modified crossover operation.

Hybrid-3 and the standard GA algorithm perform the next best with 24 and 21 successes, respectively. Hybrid-1 and Hybrid-2 performs poorly on this problem, Hybrid-1 performs extremely poorly, with the population converging on a suboptimal solution after 30 generations.

CONCLUSIONS

An adjustable, hybrid model incorporating GA and PSO searches was proposed. This hybrid allows for varying levels of influence for the GA and PSO portions of the hybrid. The performance of the hybrid is compared to that of the standard GA and PSO models. Two test problems were used in this research. In both tests a properly balanced hybrid algorithm (Hybrid-4)

Table 3: Experimental results from simulating a biological neuron.

Model	# of Successes	Best Fitness Average	Best Fitness Std-dev	Average Fitness Average	Average Fitness Std-dev
GA	21	22.7	9.2	27.3	8.3
PSO	28	22.0	9.6	30.1	10.4
Hybrid-1	8	31.1	4.7	31.1	4.7
Hybrid-2	14	27.7	8.6	27.7	8.7
Hybrid-3	24	23.7	8.9	24.2	9.8
Hybrid-4	30	21.5	9.1	34.5	7.3

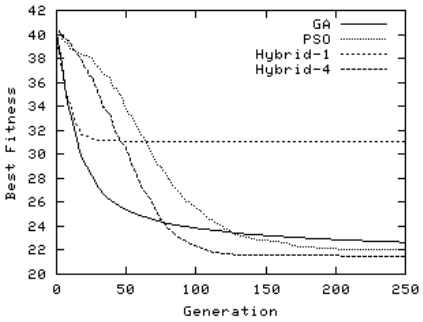

Figure 4: Best Fitness values of each algorithm over 50 runs

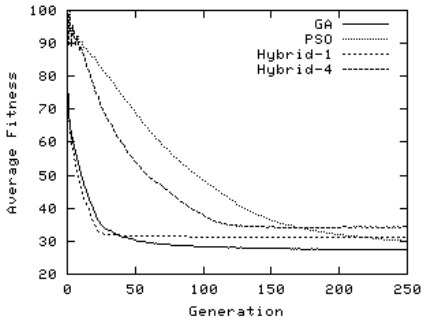

Figure 5: Average fitness values of each algorithm over 50 runs

outperformed all other models.

Our initial hypothesis was that the best hybrid would use the GA to perform an initial global search and the PSO would perform a local search at the end of the run. This hypothesis led to Hybrid-1. In fact, Hybrid-1 performs very poorly. We believe that Hybrid-1 performs poorly because the combination of GA selection and PSO convergences produces premature convergence. In the PSO algorithm the global best particle actively pulls the population towards itself. While in the GA the selection operator removes the particles farthest from the global best and creates child particles closer to the global best. We believe that this causes the population to converge on a sub-optimal solution much to quickly, often in 20 generations or less.

Two additional hybrids (Hybrid-2 and Hybrid-3) that varied the relative influence of the GA and the PSO were then introduced. They perform marginally better than the initial hybrid, but it appeared that convergence was still a problem.

Hybrid-4 was an extension to Hybrid-3 using a new crossover operator. The choice of values for the *influence* and *replacement* parameters allow half the population, due to the PSO portion of the algorithm, to naturally converge on a solution. The other half the population is subject to selection and crossover. Our choice for the crossover operator, weighted averaged node crossover, actively "disperses" the population, resulting in increasing diversity throughout the run

(see Figures 3 and 5). The increased diversity in the population did improve the overall results.

Hybrid-4 was able to successfully solve both test problems better than the other algorithms, and, based on these limited results, shows real promise in other problem domains, such as function minimization. It is important to note that these results suggest that any hybrid of PSO and GA will require some additional diversity maintenance. Future work will include testing this model in other problem domain and changing parameters.

REFERENCES

Angeline, P. J., Saunders, G. M., and Pollack, J. P., 1994. "An Evolutionary Algorithm that Constructs Recurrent Neural Networks," *IEEE Transactions on Neural Networks*, Vol.. 5, Num. 1, pp. 54-65.

Angeline, P.J., 1998. "Evolutionary Optimization Versus Particle Swarm Optimization: Philosophy and Performance Differences," *Evolutionary Programming VII, Lecture Notes in Computer Science* 1447, pp. 601-601.

Eberhart, R.C., and Shi, Y., 1998. "Comparison between Genetic Algorithms and Particle Swarm Optimization," *Evolutionary Programming VII, Lecture Notes in Computer Science* 1447, pp. 611-616.

Hammer, B., 2000. *Learning with Recurrent Neural Networks*. Vol. 254. Springer.

Horne, B. G., and Giles, C. L., 1995. "An Experimental Comparison of Recurrent Neural Networks," *In Advances in Neural Information Processing Systems*, G. Tesauro, D. Touretzky, and T. Leen, Eds., Vol. 7, The MIT Press, pp. 697-704.

Kennedy, J., and Eberhart, R., 2001. *Swarm Intelligence*. Morgan Kaufmann Publishers, Inc., San Francisco, CA.

Lvbjerg, M., Rasmussen, T., and Krink, T., 2001. "Hybrid Particle Swarm Optimizer with Breeding and Subpopulations," *In Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO-2001).

Mandescher, M., 1995. "Evolving Recurrent Neural Networks with Non-binary Encoding," *In Proc. Second IEEE Int'l Conf. Evolutionary Computation*, Vol. 2, pp. 584-589, Perth, Australia. IEEE Press, Priscataway, NJ.

Robinson, J., Sinton, S., and Rahmat-Samii, Y.., 2002."Particle Swarm, Genetic Algorithm, and Their Hybrids: Optimization of a Profiled Corrugated Horn Antenna," 2002 IEEE Antennas and Propagation Society International Symposium and URSI National Radio Science Meeting, San Antonio, TX.

Salerno, J. 1997. "Using the Particle Swarm Optimization Technique to Train a Recurrent Neural Model," 9th *International Conference on Tools With Artificial Intelligence* (ICTAI '97), IEEE Press.

Saunders, G.M., Angeline, P.J., and Pollack, J.B., 1994. "Structural and Behavioral Evolution of Recurrent Networks," *In Advances in Neural Information Processing Systems*, J.D. Cowan, G. Tesauro, and J. Alspector, Eds., Vol. 6, Morgan Kaufmann Publishers, Inc., pp. 88-95.

Settles, M., Soule, T., 2003. "Comparison of Genetic Algorithm and Particle Swarm Optimizer When Evolving a Recurrent Neural Network," *In Proceeding of the Genetic and Evolutionary Computation Conference* (GECCO-2003), In Press.