

Toward Evolving Neural Networks using Bio-Inspired Algorithms

Matthew Conforth and Yan Meng

Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ, USA.

Abstract - *The SWarm Intelligence-based Reinforcement Learning (SWIRL) method is proposed in this paper to efficiently generate Artificial Neural Network (ANN) based solutions to various problems. Basically, two swarm intelligence based algorithms are combined together in SWIRL to train the ANN models. Ant Colony Optimization (ACO) is applied to optimize ANN topology, while Particle Swarm Optimization (PSO) is applied to adjust ANN connection weights. To evaluate the performance of the ANN models trained by SWIRL, the XOR and double pole balance problem are utilized as case studies. Extensive simulation results successfully demonstrate that SWIRL offers performance that is competitive with modern neuroevolutionary techniques, as well as its viability for real-world problems.*

Keywords: Artificial Neural Network, Swarm Intelligence, Particle Swarm Optimization, Ant Colony Optimization

1 Introduction

Artificial Neural Networks (ANNs) exhibit remarkable properties, such as adaptability, capability of learning by examples, and ability to generalize. One of the most used ANN models is the well-known Multi-Layer Perceptron (MLP) [1]. Training neural networks is a complex task for reinforcement learning methods. The training process for MLPs for pattern classification problems consists of two tasks. The first one is the selection of an appropriate architecture for the problem, and the second is the adjustment of the connection weights of the network. Extensive research work has been conducted to attack this issue. Most of available training methods for ANNs only focus on the adjustment of connection weights with fixed topology. Few works have investigated training methods for ANNs on both topology and connection weights simultaneously.

Recently, swarm intelligence (SI) has attracted extensive attention in various research areas. SI is an innovative computational and behavioral metaphor for solving distributed problems by taking its inspiration from the behavior of social insects swarming and flocking, herding, and shoaling phenomena in vertebrates. The social insect colonies are able to build sophisticated structures and regulate the activities of millions of individuals by endowing each individual with simple rules based on local perception.

Among many successful bio-inspired swarm intelligence based computational paradigms, two well-known approaches are known as Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO). The ACO algorithm, proposed by Dorigo et al. [2], is essentially a system that simulates the natural behavior of ants, including mechanisms of cooperation and adaptation. The involved agents are steered toward local and global optimization through a mechanism of feedback of simulated pheromones and pheromone intensity processing. Particle Swarm Optimization (PSO) was proposed by Kennedy and Eberhart [3]. PSO is a biologically-inspired algorithm motivated by a social analogy, such as flocking, herding, and schooling behavior in animal populations. Both algorithms have been applied to solve problems in various areas, such as clustering, data mining, dynamic task allocation, and optimization [4].

In this paper, the SWarm Intelligence based Reinforcement Learning (SWIRL) method is proposed to train ANNs so that an optimal model for the problem of interest can be achieved. The basic idea of the SWIRL method is that ACO is used to optimize the topology structure of the ANNs, while PSO is used to adjust the ANN connection weights within a given topology structure. This is designed to split the problem such that ACO and PSO can both operate in the environment they are most suited for. ACO is ideally applied to finding paths through graphs. One can treat the ANN's neurons as vertices and its connections as directed edges, thereby transforming the topology design into a graph problem. PSO is best used to find the global maximum or minimum in a real-valued search space. Considering each connection weight plus one associated fitness score as orthogonal dimensions in a hyperspace, each possible weight configuration is merely a point in that hyperspace. Finding the optimal weights is thus reduced to finding the global maximum of the fitness function.

This paper is organized as follows. An overview of related work is discussed in Section 2. Relevant background information on artificial neural networks, ant colony optimization, and particle swarm optimization is provided in Section 3. The SWIRL method is proposed in Section 4. Section 5 describes the simulation setup and experimental results for two case studies: exclusive OR and the double pole balance problem. The conclusion and future work are presented in Section 6.

2 Related Work

The ANN has been applied to solve various real world problems due to its adaptability, capability of learning by examples, and ability to generalize. The theoretical capabilities of ANN solutions have been comprehensively investigated in [5] and [6]. To efficiently apply the ANN model to various problems, the optimization of ANNs for each specific problem is critical for problem solving. There are plenty of efficient searching/optimization algorithms available for the ANN model optimization, such as evolutionary algorithms (EA) [7], simulated annealing (SA) [8], tabu search (TS) [9], ant colony optimization (ACO) [2], particle swarm optimization (PSO) [3], genetic algorithm (GA) [10], etc.. Among these searching/optimization techniques, some of them have been applied for connection weights adjustment or architecture optimization of ANNs. For example, a genetic algorithm was hybridized with local search gradient methods for the process of ANN training in weight adjustment with fixed topology in [11]. Ant colony optimization was proposed to optimize the neural network with fixed topology in [12]. In [13], tabu search was used for the ANN training with fixed topology. Simulated annealing and genetic algorithms were compared for the training of ANNs with fixed topology in [14], where the GA-based method performs better than the simulated annealing method. Simulated annealing and the backpropagation variant Rprop [15] were combined for MLP training with weight decay in [16].

Recently, PSO has attracted considerable attention in various areas. Particle swarm optimization and some of its many variants were applied to MLP training in [17] without generalization control with fixed topology. Carvalho and Ludermir [18] proposed a hybrid training method for ANNs using PSO, where they analyze the use of the PSO algorithm and two variants with a local search operator for ANN training and investigate the influence of the GL5 stop criteria in generalization control for swarm optimizers.

However, sometimes it is difficult to predefine a fixed topology which is optimal for the problem of interest even with the adjustment of connection weights. Therefore, some methods were proposed to train the topology and the connection weights of the ANNs simultaneously to achieve optimal ANN models. In [19], simulated annealing and tabu search are hybridized to simultaneously optimize the weights and the number of active connections of MLPs, with the aim of producing classifiers with good classification and generalization performance. The NeuroEvolution of Augmenting Topologies (NEAT) [20] method evolves efficient ANN solutions quickly by complexifying and optimizing simultaneously. It achieves performance that is superior to comparable fixed-topology methods. An evolutionary programming based method was proposed in [21] to generate valid solutions to the double-pole and jointed-pole balance problems. Genetic algorithms proposed in [22] have solved many variations of the pole balance problem. The neuroevolutionary method Enforced Sub-

populations (ESP) was proposed in [23] and shown to be effective at solving both the standard and non-Markovian forms of the double pole balance problem.

3 Background

3.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is a connectionist computational model inspired by the nervous systems of biological entities. ANNs consist of processing elements called neurons and the weighted connections between them, as shown in Fig. 1. A neuron operates by passing the sum of its inputs, which may include a bias, through a transfer function to produce its output value. This is described by the following equation:

$$A_j = \Theta_j \left(b_j + \sum_i w_{ij} A_i \right) \quad (1)$$

where A_j is the activation level of neuron j , Θ_j is the transfer function, b_j is the bias, and w_{ij} is the weight of the connection from neuron i to j . Typically either sigmoid or linear transfer functions are used. An individual neuron cannot accomplish much, but the cumulative effect of many neurons connected together is effectively unlimited in complexity.

Cybenko [5] proved that a feedforward neural network with one hidden layer is capable of approximating any continuous, multivariate function to arbitrary precision. This was extended by Siegelmann and Sontag [6] who proved that a recurrent architecture with rational valued weights has the full power of a Universal Turing Machine, while support for irrational values as weights can produce an ANN with trans-Turing computational power. Almost all the work associated with ANNs is therefore not searching for improvements to ANNs themselves, but rather for effective methods to design, train, generate, or otherwise produce ANNs that solve particular problems/tasks.

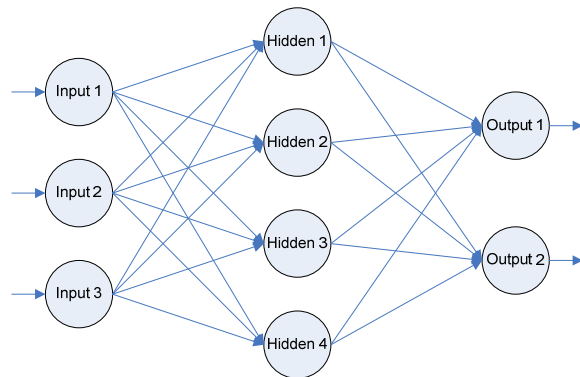


Fig. 1. Diagram of an artificial neural network.

3.2 Ant Colony Optimization

Dorigo et al. proposed the Ant Colony Optimization (ACO) algorithm in [2]. ACO is essentially a system that simulates the natural behavior of ants, including mechanisms of cooperation and adaptation. The involved agents are

steered toward local and global optimization through a feedback mechanism of simulated pheromones and pheromone intensity processing. It is based on the following ideas. First, each path followed by an ant is associated with a candidate solution for a given problem. Second, when an ant follows a path, the amount of pheromone deposited on that path is proportional to the quality of the corresponding candidate solution for the target problem. Third, when an ant must choose between two or more paths, the path(s) with higher pheromone intensity are more attractive to the ant. The probability of a given ant moving from vertex i to vertex j is a function of the pheromone intensity on edge ij and the desirability of edge ij . The desirability is generally an a priori value such as the inverse of the distance. After a sufficient number of iterations, the ants will converge to a short path, which is expected to be the optimum or a near-optimum solution for the target problem.

The pseudo-code for the ACO algorithm is as follows:

```

Procedure ACO_MetaHeuristic
  while (not_termination)
    Generate solutions;
    Update pheromone;
    Select actions;
  end while
end procedure

```

ACO can be represented in the terms of sequential decision processes and Monte Carlo sampling-based learning. More precisely, ACO can be treated as a policy search strategy aimed at learning the distributed parameters, called pheromone variables, of the stochastic decision policy which is used by ant agents to generate solutions. Each ant represents one independent sequential decision process aimed at constructing a possibly feasible solution for the optimization problem at hand by using only local information. Ants are repeatedly and concurrently generated in order to sample the solution set according to the current policy. The outcomes of the generated solutions are used to partially evaluate the current policy, spot the most promising search areas, and update the policy parameters in order to focus the search in those promising areas while keeping a satisfactory level of overall exploration.

3.3 Particle Swarm Optimization

The PSO algorithm [3] is a population-based optimization method, where a set of potential solutions evolves to approach an optimal solution (or set of solutions) for a problem. The social metaphor that led to this algorithm can be summarized as follows: the individuals that are part of a society hold an opinion that is part of a “belief space” (the search space) shared by every possible individual. Individuals may modify this “opinion state” based on three factors: (1) the search for new knowledge of the environment (explorative factor); (2) the individual’s previous history of states (cognitive factor); (3) the previous history of states of the individual’s neighborhood (social factor).

Basically, the PSO algorithm can be represented as a swarm of particles in n -dimensional Euclidean space, where n is the

number of attributes that fully define a particle’s state. The explorative factor is emulated by giving the particles inertia and randomly weighing each influence every iteration. The cognitive factor is modeled as an attractive force towards the individual’s best previous state. Similarly, the social factor is simulated as an attractive force towards the best previous state out of all the particles in the neighborhood.

Therefore, the basic idea is to propel the swarm towards a probabilistic median, where the explorative factor, cognitive factor (individual particles’ respective views), and social factor (global swarm wide view) are considered simultaneously and try to merge these three factors into consistent behaviors for each particle.

4 The SWIRL Method

In this section, the SWIRL (Swarm Intelligence based Reinforcement Learning) method is proposed to train the topology and connection weights of ANNs so that an optimal ANN model can be achieved for the problem of interest. Basically in the SWIRL method, the ACO algorithm is utilized to select the topology of the ANNs, while the PSO algorithm is employed to optimize the weights of the ANNs.

4.1 The SWIRL System

The objective of the SWIRL system is to achieve an optimal model for the problem of interest through reinforcement learning, which is analogous to a school system. Therefore, the SWIRL system can be modeled as a school system, where the ACO module, the PSO modules, and the untrained ANNs taking on the roles of administrator, teachers, and students, respectively. In a school system, students learn from teachers, teachers train students, and administrators allocate resources to teachers. In the SWIRL system, the ACO algorithm (the administrator) allocates training iterations to the PSO algorithms (teachers). The PSO algorithms (teachers) then run for their allotted iterations to train their ANNs (students). The global best score for all the ANNs trained by a particular PSO instance is then used by the ACO algorithm to allocate the next set of training iterations. Fig. 2 gives a high-level overview of this SWIRL system.

This composition of the ACO and PSO algorithms is designed to best leverage their inherent strengths. The ACO algorithm excels at finding the optimum path through a graph. It has been used to produce near-optimal solutions to traveling salesman-type problems. In problems where the graph may change dynamically, ACO’s ability to rapidly adapt and de-converge from a solution that is no longer optimal provides a major advantage of the ACO algorithm over other methods.

The optimization of an ANN topology can be treated as a graph problem where the neurons are vertices and the connections are directed edges. The value of a path is the best possible performance of the ANN whose topology is defined by that path. At first, this value is completely unknown, but as training progresses it can be estimated with gradually increasing accuracy. This makes the graph extremely dynamic, with edge values changing every iteration. Thus,

the ACO algorithm can be used to optimize the ANN topology.

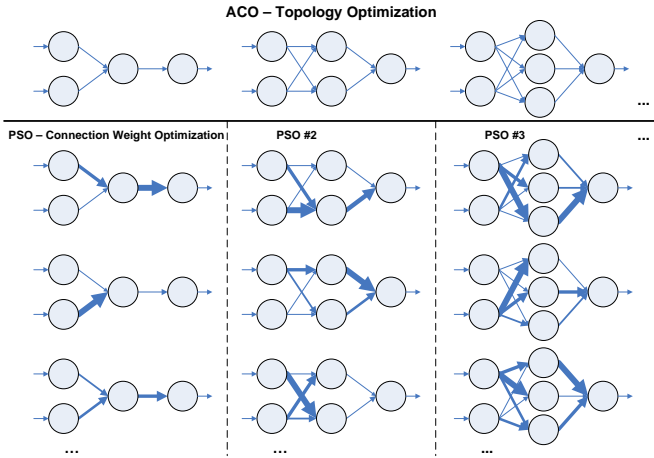


Fig. 2. SWIRL overview. The ACO algorithm is used to optimize the ANN topology. This is shown as a horizontal list of topologies under consideration at the top of the diagram. A separate instance of the PSO algorithm is used to optimize the connection weights for a particular topology. This is depicted as a vertical list of successive configurations, where the arrow thickness is used to indicate the connection weight.

The PSO algorithm is essentially a stochastic search for the maximum or minimum value of a function in a real-valued hyperspace. In a sense, this is similar to the optimization of connection weights for a given ANN topology. The goal is to find the global maximum of the reinforcement (i.e. fitness) function, where each connection corresponds to an orthogonal dimension in the search space. Thus, the $(n+1)$ -tuple of n connection weights and 1 associated fitness score defines a point in the search space. Therefore, the PSO algorithm can be used to optimize the ANN connection weights.

4.2 ACO-based Topology Optimization

As previously stated, the ACO algorithm can be used to allocate training iterations among a set of candidate network topologies. The corresponding desirability in the ACO algorithm can be defined as:

$$d(i) = \frac{1}{h+1} \quad (2)$$

where h is the number of hidden nodes in ANN i . ANN i represents i th topology of a set of ANNs with different numbers of hidden nodes.

The pheromone intensity $\tau_i(t)$ is initialized to 0.1, so that the ants' initial actions are based primarily on desirability. Thereafter, $\tau_i(t)$ values are updated according to the following equation:

$$\tau_i(t+1) = \rho \cdot \tau_i(t) + n_i \cdot \frac{g_i}{\sum_{j=1}^m g_j} \quad (3)$$

where ρ is the pheromone persistence. n_i is the number of ants returning from ANN i . g_i is the global best for ANN i , and

$\sum_{j=1}^m g_j$ is the sum of all the current global bests. Each ant

represents one training iteration for a PSO teacher. During each major iteration (i.e. one ACO step), the ants go out into the topology space. The probability of a given ant picking ANN i is provided by the following equation:

$$p(i) = \frac{[\tau(i,t)]^\alpha [d(i)]^\beta}{\sum_{j=1}^i \{[\tau(j,t)]^\alpha \cdot [d(j)]^\beta\}} \quad (4)$$

Where $p(i)$ represents the probability of an ant picking topology i from a set of ANNs with different topologies. α and β are constant factors that control the relative influence of pheromones and desirability, respectively.

4.3 PSO-based Weight Adjustment

ANNs are then trained via the PSO algorithm for a number of iterations determined by the number of ants at that node. Each PSO teacher starts with a group of networks whose connection weights are randomly initialized. The student ANNs are tested on the chosen problem and each receives a score. The PSO teacher keeps track of the global best configuration and each student's individual best configuration. A configuration for an ANN with n connections can be considered as a point in an $n+1$ dimensional space, where the extra dimension is for the reinforcement score. After every round of testing, the teacher updates the connection weights of the student ANNs according to the following equations:

$$\mathbf{v}_{t+1} = c_{inr} \mathbf{r}_1 \bullet \mathbf{v}_t + c_{cgn} \mathbf{r}_2 \bullet (\mathbf{x}_{pb} - \mathbf{x}_t) + c_{scl} \mathbf{r}_3 \bullet (\mathbf{x}_{gb} - \mathbf{x}_t) \quad (5)$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_{t+1} \quad (6)$$

where the position and velocity vectors are denoted by \mathbf{x} and \mathbf{v} , respectively. The big dot symbol is for Hadamard multiplication. The \mathbf{r}_i represents vectors where each element is a new sample from the unit-interval uniform random variable. Personal best, \mathbf{x}_{pb} , is the point in the solution-space where that particular student received its highest score so far. Global best, \mathbf{x}_{gb} , is the point with the highest score achieved by any student of this PSO teacher. The three constants, c_{inr} , c_{cgn} , and c_{scl} , allow the adjustment of the relative weighting

for the inertial, cognitive, and social components of the velocity, respectively.

After exhausting its allotted training iterations, the PSO teacher reports the global best to the ACO administrator. If/when it is allocated additional training iterations, the PSO teacher resumes training from exactly where it left off.

4.4 SWIRL Algorithm Summary

The pseudo-code for the SWIRL algorithm can be summarized as follows:

```
function SWIRL
  initialize ACO_Administrator;
  for(candidate topology i=1,...,N)
    create PSO_Teacher(i)
    for(ANN_Student j=1,...,M)
      initialize ANN_Student
    end for
  end for
  while(TRUE)
    compute ant movement probability CDF;
    ants allocate training iterations;
    for(PSO_Teacher i=1,...,N)
      while(iterations < allocation)
        for(ANN_Student j=1,...,M)
          test ANN_Student(j);
          if(ANN_Student(j) is a solution)
            return ANN_Student(j)
          end if
        end for
        for(ANN_Student j=1,...,M)
          update weights ANN_Student(j);
        end for
      end while
      return global best
    end for
    update pheromone concentrations
  end while
end
```

5 Simulation

5.1 Simulation Platform and Parameters Setup

The SWIRL system is implemented in Java for simulation testing. There is a 5:1 ratio of ants to candidate ANN topologies. It is assumed that the candidates have only one hidden layer with 1 through 5 hidden nodes. In the following simulations, the pheromone influence factor, α , the desirability influence factor, β , and the pheromone persistence, ρ , are set to 2, 1, and 0.5, respectively. The initial pheromone level is set to 0.1 for all topologies. Each PSO teacher has 100 ANN students. The PSO particle velocity is capped at 5. The velocity factors are set to 0.8 for the inertial constant, 2 for the cognitive constant, and 2 for the social constant. The ANNs are fully connected, with initial connection weights randomized uniformly in the range (-5, 5). Hyperbolic tangent is used as the transfer function. The ANNs operate entirely with double-precision floating point values.

Two case studies are conducted in this paper to evaluate the performance of the proposed SWIRL algorithm for ANNs. The first case study is exclusive or (XOR). It is one of the classic ANN problems. XOR is not expected to be much of a challenge, but it is sufficient to prove that the SWIRL algorithm can solve problems that are not linearly separable. Also, XOR requires at least 2 hidden nodes because our test implementation does not support connections bypassing the hidden layer. The second case study is the double pole balance problem, also known as the double inverted pendulum problem. This is a fairly difficult test that verifies SWIRL's ability to solve control tasks. This test was also selected because of its popularity, so that SWIRL's performance can be compared with other ANN reinforcement learning methods.

5.2 Exclusive OR

The exclusive or (XOR) logical operation is a simple example of a function that is not linearly separable. The difficulty of the XOR problem can be straightforwardly scaled to anywhere in the range from "extremely easy" to "quite easy" depending on how the test setup handles the numerical precision of values. At the easiest end of the spectrum, connection weights are integer valued and the output is thresholded to 0 or 1. The test used for these results is instead made to be as difficult as possible. As previously stated, double-precision floating point values are exclusively used in our simulations for the SWIRL algorithm. The ANNs are denied the assistance of any rounding or thresholding. A valid solution is defined in this test as an ANN that produces output in double-precision floating point of exactly 0.0 or 1.0 according to the truth table for XOR. SWIRL's XOR solution has 2.79 hidden nodes on average, with a standard deviation of 0.95 nodes. Table 1 gives a more detailed breakdown of the solution topology selections.

TABLE I
DETAILED SWIRL RESULTS FOR EXCLUSIVE OR

Hidden Nodes	Frequency
1	0 %
2	51 %
3	27 %
4	15 %
5	7 %

XOR provides a simple but relevant demonstration that the SWIRL algorithm can solve problems which are not linearly separable. The main source of difficulty in this test for stochastic methods such as the SWIRL algorithm is that there are only 4 training points, and the very simple OR function will classify 3 of them correctly. One advantage of the SWIRL algorithm in this regard is that the PSO teachers

operate independently, so that even if a couple of the teachers get stuck on local maxima, the SWIRL system as a whole can still generate a valid solution. The results show that the SWIRL algorithm favors the simplest viable topology, but will readily utilize others in the event of a “lucky start” or a swarm getting stuck on a local maximum. This test also shows that the initial topology desirability values do not dominate the ACO administrator.

5.3 Double Pole Balance Problem

The double pole balance problem is setup as described in [20]. A cart is placed on a 4.8 meter track, and two poles of length of 1 meter and 0.1 meter respectively are attached to the top of the cart via hinges. Both poles have a linear density of 0.1 kilogram per meter. The mass of the cart is 1 kilogram. The state of the system is defined by the position and velocity of the cart, and the angles and angular velocities of the poles. Control input applies lateral thrust in the range of (-10, 10) newtons to the cart. The ANN must keep both poles within 36° of vertical positions without letting the cart go off either end of the track. The system dynamics are modeled using the Runge-Kutta fourth-order method and 0.01 second time steps. Fig. 3 shows a visualization of the double pole balance simulation.



Fig. 3. Double pole balance simulator GUI display. The pink block represents the cart. The navy and blue poles indicate the deflection from vertical of the short and long pole, respectively. The line beneath the cart is the track. Letting a pole fall more than 36° from vertical or running off of the track constitutes failure. To prove itself to be a valid solution, an ANN must pass 10 trials, each lasting 100,000 simulation steps.

For these trials, the problem is considered solved when the ANN can keep both poles balanced in 10 trials of 100,000 simulation steps each. Table 2 shows the simulation results. Evolutionary programming results were obtained in [21]. Conventional neuroevolution data was reported in [22]. SANE and ESP results were reported in [23]. The results of the SWIRL algorithm are the mean of 1000 tests. NEAT [20] results are averaged over 120 experiments. All other results are the mean of 50 runs. The standard deviation of the SWIRL results is 1033 evaluations. The NEAT trials had a standard deviation of 2,704 evaluations. Standard deviations for other methods were not reported. From the simulation results in Table 2, it can be seen that the SWIRL method is competitive with advanced neuroevolutionary methods. Total evaluations are the important performance measurement data since they provide the best approximation of real execution

time on conventional computer systems. The value for the SWIRL algorithm is the summation over all the topologies being tested, not just the topology that reached a solution. Evolutionary generations and swarm iterations are not directly comparable, but these values are included to address performance on systems where arbitrarily-many ANNs can be tested simultaneously.

The average number of hidden nodes in the solution network was 1.22 with a standard deviation of 0.55. Over the course of the 1000 tests, each candidate topology was used to produce the solution at least once.

The results demonstrate that the SWIRL algorithm performs comparably to advanced neuroevolutionary methods such as NEAT and ESP for the Markovian double pole balance problem. The improvement over conventional neuroevolution, evolutionary programming, and SANE is quite drastic, so we can say with confidence ($p < 0.001$) that the SWIRL algorithm is superior for this task. SWIRL, NEAT, and ESP perform quite similarly, and given the variances involved it would take a prohibitive number of tests to establish their ranking to a high degree of certainty.

TABLE II
PERFORMANCE COMPARISON OF DIFFERENT ALGORITHMS SOLVING THE
DOUBLE POLE BALANCE PROBLEM

Method	Total Evaluations	Generations/Iterations
Evolutionary programming	307200	150
Conventional Neuroevolution	80000	800
SANE	12600	63
ESP	3800	19
NEAT	3600	24
SWIRL	3516	15

It is also important to keep in mind that the performance of the SWIRL algorithm is contingent on many parameters, such as the number of ants, the number of particles, the initial particle distribution, the social, cognitive, and inertial constants, the particle velocity cap, etc. For the purpose of fairness, these values were not tuned to this particular test at all; they are nice round numbers that reflect common sample values from theoretical discussions of PSO and ACO. Better values could improve the performance, but that would obviously reduce the general applicability of the results.

6 Conclusions

In this paper, the SWIRL method is proposed to generate ANN solutions to tasks/problems amenable to reinforcement learning. Basically, the ACO algorithm is applied to optimize the topology selection, while the PSO algorithm is utilized to adjust the connection weights of the selected topology. Two

case studies have been conducted to evaluate the performance of the proposed SWIRL algorithm. The XOR case study verifies that SWIRL can solve problems which are not linearly separable and that it can abandon topologies too simple to solve the problem at hand. The double pole balance case study demonstrates that SWIRL is competitive with advanced neuroevolutionary techniques such as NEAT and ESP. By utilizing the ACO algorithm and the PSO algorithm in contexts for which they are well-suited, the SWIRL algorithm provides an efficient method for ANN solution generation which complements the innate strength of ANNs to adapt and generalize. Future work would give the ACO system full control over the ANN topology, including support for recurrent links, bypass links, and fractured nets, etc.

7 References

- [1] S. Haykin, *Neural Networks: A comprehensive Foundation*. 2nd Edition, Prentice Hall, 1998.
- [2] M. Dorigo, V. Maniezzo and A. Coloni. "Ant System: optimization by a colony of cooperating agents", *IEEE Transactions on Systems, Man and Cybernetics - Part B*, vol. 26, no. 1, pp. 29-41, 1996.
- [3] J. Kennedy and R. Eberhart, "Particle Swarm Optimization", in: *Proc. IEEE Intl. Conf. on Neural Networks (Perth, Australia)*, IEEE Service Center, Piscataway, NJ, IV:1942-1948, 1995.
- [4] L. Lhotská, M. Macaš, and M. Burša, *PSO and ACO in Optimization Problems*, E. Corchado et al. (Eds.). 2006 *Intelligent Data Engineering and Automated Learning (IDEAL 2006)*, LNCS 4224, pp. 1390 – 1398, 2006.
- [5] G.V. Cybenko. "Approximation by Superpositions of a Sigmoidal Function," *Mathematics of Control, Signals and Systems*, vol. 2 pp. 303-314. 1989.
- [6] H.T. Siegelmann; E.D. Sontag. "Analog Computation Via Neural Networks," *Theoretical Computer Science*, v. 131, no. 2, pp. 331-360. 1994.
- [7] E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Natural Computing Series. MIT Press. Springer. Berlin. (2003).
- [8] S. Kirkpatrick, C.D. Gellat Jr. and M.P. Vecchi, "Optimization by simulated annealing", *Science*, 220: 671-680, 1983.
- [9] F. Glover, "Future paths for integer programming and links to artificial intelligence", *Computers and Operation Research*, Vol. 13, pp. 533-549, 1986.
- [10] J.H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, MI, 1975.
- [11] E. Alba and J.F. Chicano, "Training Neural Networks with GA Hybrid Algorithms", K. Deb(ed.), *Proceedings of GECCO'04*, Seattle, Washington, LNCS 3102, pp. 852-863, 2004.
- [12] C. Blum and K. Socha, "Training feed-forward neural networks with ant colony optimization: An application to pattern classification", *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, pp. 233-238, 2005.
- [13] R.S. Sexton, B. Alidaee, R.E. Dorsey and J.D. Johnson, "Global optimization for artificial neural networks: a tabu search application", *European Journal of Operational Research*, (106)2-3, pp.570-584,1998.
- [14] R.S. Sexton, R.E. Dorsey and J.D. Johnson, "Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing", *European Journal of Operational Research*, (114), pp.589-601,1999.
- [15] M. Riedmiller. "Rprop - description and implementations details", Technical report, University of Karlsruhe, 1994.
- [16] N.K. Treadgold and T.D. Gedeon, "Simulated annealing and weight decay in adaptive learning: the SARPROP algorithm", *IEEE Transactions on Neural Networks*, 9:662-668,1998.
- [17] F. van den Bergh, *An Analysis of Particle Swarm Optimizers*, PhD dissertation, Faculty of Natural and Agricultural Sciences, Univ. Pretoria, Pretoria, South Africa, 2002.
- [18] M. Carvalho and T.B. Ludermir, *Hybrid Training of Feed-Forward Neural Networks with Particle Swarm Optimization*, I. King et al. (Eds.): *ICONIP 2006, Part II*, LNCS 4233, pp. 1061–1070, 2006.
- [19] T.B. Ludermir ; Yamazaki, A.; Zanchetin, Cleber . "An Optimization Methodology for Neural Network Weights and Architectures" (. *IEEE Transactions on Neural Networks*, v. 17, n. 5, 2006.
- [20] K. O. Stanley and R. Miikkulainen, *Evolving Neural Networks through Augmenting Topologies*, *Evolutionary Computation*, 10(2): 99-127. 2002.
- [21] Saravanan, N. and Fogel, D.B. (1995). "Evolving Neural Control Systems." *IEEE Expert*, 10(3):23-27.
- [22] Wieland, A. (1991). "Evolving Neural Network Controllers for Unstable Systems." In *Proceedings of the International Joint Conference on Neural Networks*, pp. 667–673, IEEE Press, Piscataway, New Jersey.
- [23] Gomez, F. and Miikkulainen, R. (1999). "Solving Non-Markovian Control Tasks with Neuroevolution." In Dean, T., editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pp. 1356–1361, Morgan Kaufmann, San Francisco, California.