2008 IEEE Swarm Intelligence Symposium
St. Louis MO USA, September 21-23, 2008

Reinforcement Learning for Neural Networks using Swarm Intelligence

Matthew Conforth and Yan Meng

Abstract— **In this paper, we propose a swarm intelligence based reinforcement learning (SWIRL) method to train artificial neural networks (ANN). Basically, two swarm intelligence based algorithms are combined together to train the ANN models. Ant Colony Optimization (ACO) is applied to select ANN topology, while Particle Swarm Optimization (PSO) is applied to adjust ANN connection weights. To evaluate the performance of the SWIRL model, it is applied to double pole problem and robot localization through reinforcement learning. 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.**

I. 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]. The training process of 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 tackle this issue. Most of the available training methods for ANNs only focus on the adjustment of connection weights within a fixed topology. Only a few works have investigated training methods for ANNs that optimize both topology and connection weights.

Recently, swarm intelligence (SI) has attracted extensive attention in various research areas. SI is an innovative computational and behavioral metaphor for solving distributed problems that takes its inspiration from the behavior of social insects and the swarming, flocking, herding, and shoaling phenomena of vertebrates. Among many successful bio-inspired swarm intelligence computational paradigms, two well-known approaches are Ant Colony Optimization (ACO) [2] and Particle Swarm Optimization (PSO) [3]. ACO can be 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 can be used to find the global maximum or minimum in a real-valued search space. Considering each connection 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 in that hyperspace.

Based on these ideas, in this paper, we propose a SWarm Intelligence-based Reinforcement Learning (SWIRL) method to train ANNs. More specifically, ACO is used to select the topology structure of the ANN models, while the PSO is used to adjust the connection weights of the ANN models based on the selected topology structure from ACO.

The paper is organized as follows. Related work is introduced in Section II. Section III discusses the SWIRL method. Simulation results are presented in Section IV. Finally, conclusion is given in Section V.

II. Related Work

Global search techniques, with the ability to broaden the search space in the attempt to avoid local minima, has been used for connection weights adjustment or architecture optimization of MLPs, such as ant colony optimization (ACO) [2], particle swarm optimization (PSO) [3], evolutionary algorithms (EA) [4], simulated annealing (SA) [5], and tabu search (TS) [6]. A genetic algorithm was hybridized with local search gradient methods for the process of MLP training (weight adjustment) in [7]. In [8], ant colony optimization was used to train the weight of a fixed-topology MLP. In [9], tabu search was used for fixed topology neural networks training. In [10], simulated annealing and genetic algorithms were compared for the training of neural networks with fixed topology, where the GA method outperforms the simulated annealing methods. In [11], simulated annealing and the backpropagation variant Rprop [12] were combined for MLP training with weight decay. In [13], simulated annealing and tabu search were hybridized to simultaneously optimize the weights and the number of active connections of MLP neural networks to achieve good classification and generalization performance. In [14], particle swarm optimization and some variants were applied to MLP training without generalization control.

Zhen et al. [15] proposed a new memetic algorithm called shuffled particle swarm optimization (SPSO), which combines the learning strategy of particle swarm optimization (PSO) and the shuffle strategy of shuffled frog leaping algorithm (SFLA). A co-evolutionary particle swarm optimization was proposed for multiobjective optimization (MO) by Meng et al. [16], where co-evolutionary operator, competition mutation operator, and new selection mechanism were designed for MO problem to

M. Conforth is with the Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ 07030, USA. (e-mail: matthew.conforth@stevens.edu).
Y. Meng is with the Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ 07030, USA. (Phone: 201-216-5496; fax: 201-216-8246; e-mail: yan.meng@stevens.edu)

guide the whole evolutionary process. Carvalho and Ludermir [17] proposed a hybrid training method for neural network using PSO, where they analyzed the use of the PSO algorithm and two variants with a local search operator for neural network training and investigated the influence of the GL5 stop criteria in generalization control for swarm optimizers.

The NeuroEvolution of Augmenting Topologies (NEAT) [18] method evolves efficient ANN solutions quickly by complexifying and optimizing simultaneously; it achieves performance that is superior to comparable fixed-topology methods. In [19], the stochastic methods Adaptive Random Search (ARS) and Simultaneous Perturbation Stochastic Approximation (SPSA) outperformed extended dynamic backpropagation at training a dynamic neural network to control a sugar factory actuator.

III. The Swirl Method

In the SWIRL approach, the ACO algorithm is utilized to select the topology of the neural network, while the PSO algorithm is utilized to optimize the weights of the neural network. The SWIRL approach is modeled as a school, with the ACO, PSO, and neural networks taking on the roles of administrator, teacher, and student, respectively. Students learn, teachers train students, and administrators allocate resources to teachers. In the same fashion, the ACO algorithm allocates training iterations to the PSO algorithms. The PSO algorithms then run for their allotted iterations to train their neural networks. The global best score for all the neural networks trained by a particular PSO instance is then used by the ACO algorithm to reallocate the training iterations. Fig. 1 gives a high-level overview of this SWIRL system.

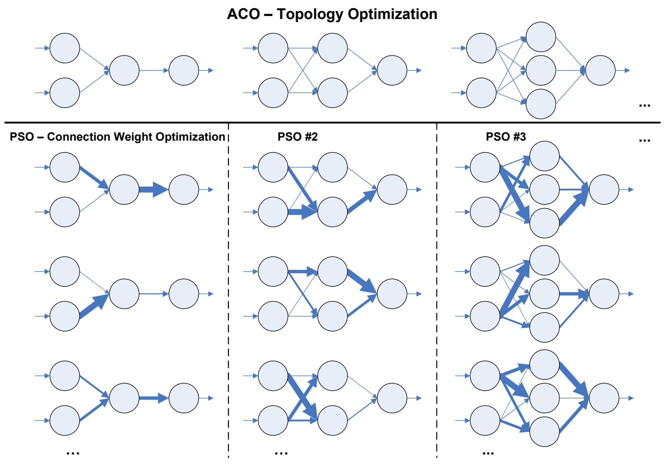
ACO – Topology Optimization
...
PSO – Connection Weight Optimization
PSO #2
PSO #3
...
...
...
...

Fig. 1. SWIRL overview. The ACO algorithm is used to select 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.

A. *ACO based Topology Selection*

Dorigo et al. [2] proposed an Ant Colony Optimization (ACO). 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 mechanism of feedback 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 deposit on that path is proportional to the quality of the corresponding candidate solution for the target problem. Third, when an ant has to choose between two or more paths, the path(s) with a larger amount of pheromone are more attractive to the ant. After some iterations, eventually, the ants will converge to a short path, which is expected to be the optimum or a near-optimum solution for the target problem.

ACO is used to allocate training iterations among a set of candidate network topologies. The desirability in ACO is defined as:

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

where h is the number of hidden nodes in neural network i.

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

$$\tau(i,t+1) = \rho \cdot \tau(i,t) + n_a(i) \cdot \frac{g(i)}{g_{sum}} \tag{2}$$

where τ is the pheromone concentration, ρ is the rate of evaporation, n_a is the number of ants returning from neural network i, $g(i)$ is the global best for i, and g_{sum} 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 can be provided by the following equation:

$$p(i) = \frac{[\tau(i,t)]^{\alpha}[d(i)]^{\beta}}{\sum_{j=1}^{i}\left\{[\tau(j,t)]^{\alpha} \cdot [d(j)]^{\beta}\right\}} \tag{3}$$

where $p(i)$ represents the probability of an ant picking topoplogy 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.

B. *PSO based Weight Adjustment*

The PSO algorithm is a population-based optimization method, where a set of potential solutions evolve to approach a convenient 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 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).

Therefore, the basic idea of the PSO algorithm is to propel towards a probabilistic median, where explorative factor, cognitive factor (local robot respective views), and social factor (global swarm wide views) are considered simultaneously, and try to merge these three factors into consistent behaviors for each robot. The exploration factor can be easily emulated by random movement.

The connection weights of ANNs are then trained via the PSO algorithm as a local search method. Instead of finding a convergent global best solution using PSO with all the necessary iterations, in the SWIRL system, the PSO method only conducts the number of iterations allocated to that topology by the ACO administrator. Each PSO teacher, as a local search method, 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 from the reinforcement (a.k.a. fitness) function for the problem. The PSO teacher keeps track of the global best configuration and each student ANN'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 PSO teacher updates the connection weights of the student ANNs according to the following equations:

$$\begin{aligned}\mathbf{v}_{t+1} = c_{inr}\mathbf{r}_1 \bullet \mathbf{v}_t + c_{cgn}\mathbf{r}_2 \bullet \left(\mathbf{x}_{pb} - \mathbf{x}_t\right) \\ + c_{scl}\mathbf{r}_3 \bullet \left(\mathbf{x}_{gb} - \mathbf{x}_t\right)\end{aligned} \quad (4)$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_{t+1} \quad (5)$$

where the position and velocity vectors are denoted by **x** and **v**, respectively. The big dot symbol is for Hadamard (i.e. element-by-element) 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 ANN received its highest score so far. Global best $\mathbf{x}_{gb}$ is the point with the highest score achieved by any ANN 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 resumes training from exactly where it left off.

C. *SWIRL Algorithm Summary*

The SWIRL algorithm essentially operates as follows. As a global search method, the ACO administrator initializes the training allocation in the topology space, and then creates a PSO teacher as a local search method for each candidate ANN topology. Next, each PSO teacher creates a swarm of randomly initialized student ANNs whose configurations are represented as particles in a hyperspace. The first major iteration begins as the ACO allocates training iterations to different ANN topologies according to (3). Each PSO teacher uses its allocated iterations to train its student ANNs. Training consists of testing each ANN in turn and then updating the weight configurations (i.e. particle positions) according to (4) and (5). If the testing proves that a student ANN is a valid solution as defined by the problem/task, that student ANN is returned and the SWIRL algorithm terminates. Each PSO teacher reports the best score out of all its ANN students back to the TRA administrator. The training iteration distribution in the topology space is updated according to (3), and then the next major iteration begins.

The Pseudo-code for the SWIRL algorithm follows:

```
function SWIRL
  initialize(TRA_Administrator);
  for(candidate_topology i=1,…,N)
    create PSO_Teacher(i);
    for(ANN_Student k=1,…,M)
      initialize(k);
    end for
  end for
  while(TRUE)
    allocate_training_iterations;
    for(PSO_Teacher i=1,…,N)
      while(iterations < allocation)
        for(ANN_Student k=1,…,M)
          test(k);
          if(is_valid_solution(k))
            return k;
            terminate function;
          end if
        end for
        for(ANN_Student k=1,…,M)
          update_weights(k);
        end for
      end while
      report(global_best);
    end for
    update_probability_distribution;
  end while
end
```

IV. Simulation

A. *Simulation Platform*

The SWIRL system is implemented in Java for simulation testing. There is a 5:1 ratio of ants to candidate network topologies. The candidates are 1 through 5 hidden nodes. The pheromone influence factor, desirability influence factor, and rate of pheromone evaporation are set to 2, 1, and 0.5, respectively. The initial pheromone level is 0.1 for all topologies. Each PSO teacher has 100 students. The PSO particle velocity is capped at 5. The velocity factors are 0.8 for the inertial constant, 2 for the cognitive constant, and 2 for the social constant. The neural networks are fully connected, with initial connection weights uniformly random in the range (-5, 5). Hyperbolic tangent is used as the transfer function.

B. *Double Pole Balance Problem*

Double polo balance problem has been used as case studies to evaluate for trained neural networks by many researchers. For example, evolutionary programming generated solutions have been proposed to the double-pole and jointed-pole balance problems in [20]. Genetic algorithms [21] have solved many variations of the pole balance problem. A double CMAC network [22] with one trained for generality and the other trained for accuracy near the target was also applied to the double pole balance problem. The neuroevolutionary method Enforced Sub-populations (ESP) [23] proved effective at solving both the standard and non-Markovian forms of the double pole balance problem. The NeuroEvolution of Augmenting Topologies (NEAT) [18] method was proposed to train topology and connection weights of neural networks simultaneously to solve double-pole problems.

Here, the double pole balance problem is setup as described in [18]. As shown in Fig. 2, 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. The Java code for the physics simulation and GUI display of the double pole balance test is adapted from that included in the ANJI software package.

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 1 shows the simulation results. Evolutionary programming results were obtained in [20]. Conventional neuroevolution data was reported in [21]. SANE and ESP results were reported in [23]. The results of the SWIRL algorithm are the mean of 1000 tests. NEAT [18] 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.

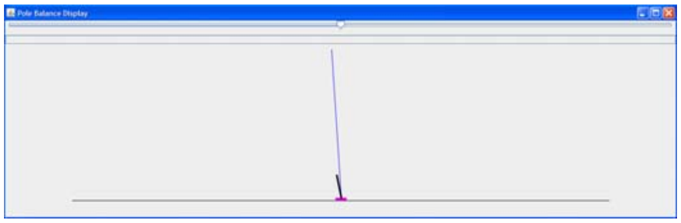

Fig. 2. Double pole balance simulator GUI display. The horizontal block represents the cart. The two vertical lines indicate the deflection the short and long pole, respectively. The line beneath the cart is the track. Letting a pole fall more than 36° from vertical or running off the track constitutes failure. To prove itself to be a valid solution, an ANN must pass 10 trials, each lasting 100,000 simulation steps.

From the simulation results in Table 1, 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.

TABLE I
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

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 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.

It is also important to note 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.

C. *Robot Localization*

The second case study is a mobile robot localization problem using neural networks through reinforcement learning. Localization is a critical problem for an autonomous mobile robot, where a mobile robot has to deduce its own position within a given map using its onboard sensors. Markov localization is a probabilistic approach to estimate the robot location within a given map. Initially, the robot has no idea where it is. By comparing its current sensor readings to the values that it would expect to get at each location on the map, the robot can estimate a probability distribution for where it is likely to be located. As the robot moves, it can update these probabilities by combining its sensor and odometer measurements until it knows its location to a high degree of certainty. It can then use its map to navigate the area. We use a wavefront navigation algorithm in this case study. Although it is still a simulation, this test includes all the issues of noise and mis-calibration that would be encountered in the real world scenarios.

There are some neural network based approaches have been proposed for robot localization. For example, in [24] an ANN was trained to estimate a robot's position relative to a particular local object. Robot localization was achieved by using entropy nets to implement a regression tree as an ANN in [25]. An ANN was trained in [26] to correct the pose estimates from odometry using ultrasonic sensors. In this paper, we apply the SWIRL based neural networks method for robot localization. We developed a robot localization simulator by ourselves written in C++, as shown in Fig. 3.

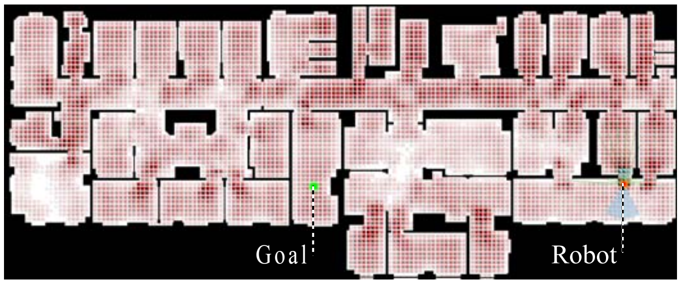

Fig. 3(a). The initialization state of a simulation.

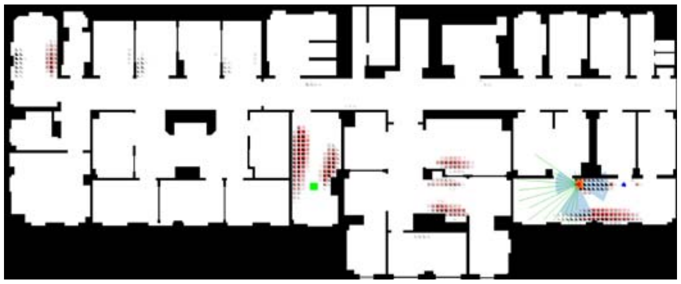
Fig. 3(b). An early state of a simulation.

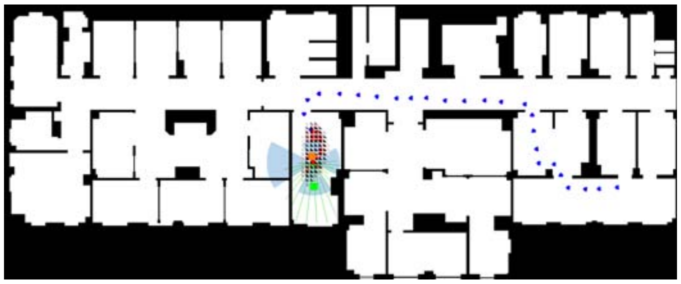
Fig. 3(c). A late state of a simulation.

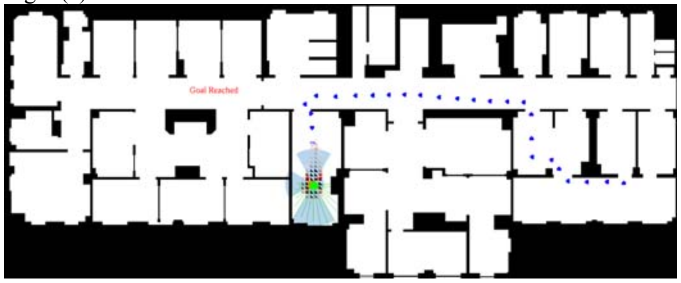

Fig. 3(d). The end of a simulation.

Fig. 3. Start of a simulation. Black = Markov value; Red = ANN value. The robot, shown as a red pentagon, must navigate from its initial position to the goal, the green square. The triangles indicate the likelihood that the robot is in that location according to the beliefs of the localization systems. The green lines show the robot's laser rangefinders and the pale blue wedges show its sonar sensors. The orange square is the robot's most likely position according to whichever localization system is currently in use for navigation. Blue pentagons are periodically stamped on the map to show the path taken by the robot.

The map is divided into a grid of squares, which are each divided into a black and red triangle. The black triangle and red triangle indicate the belief of the Markov algorithm and current best ANN, respectively, that the robot may be located in that particular square. Note that the red triangles do not show up underneath the pale blue wedges; this is merely a color layering issue in the simulator's screen painter. The map is 1086 by 443 pixels in size. The Markov grid squares are 7 pixels on a side, and rotations are multiples of 10°. Note that this quantization applies only to the pose estimation in the localization system. The robot is simulated with a full range of motion using floating point x, y, and θ values. The robot has 4 sonar sensors and 19 laser rangefinders. The sonar sensor range is 75 pixels in the map, whereas the range of the laser rangefinders is 100 pixels.

Fig. 3(a) shows the localization simulator at the beginning of a simulation run. Fig. 3(b) shows an early state of the simulation. The blue pentagons are stamped periodically on the map to indicate the path taken by the robot. At this point, the robot has narrowed down its location to a few regions on the map. In Fig. 3(c), near the end of the simulation, the robot is certain of its general location. Finally, Fig. 3(d) shows the robot when it has successfully reached the goal.

The sensory component of the Markov localization system calculates the probability that the robot is at a particular location by comparing the current sensor readings to the predicted sensor readings for that location which are generated from the map. The following pseudo-code describes this function:

```
function probability(location x)
   prob_match=1.0
   for(i=1,…,Number_Sensors)
      j=normalize(reading(i)-predict(x,i))
      prob_match=prob(j)*prob_match
   end for
   return prob_match
end
```

These match probabilities are then used to update the belief matrix for the robot's location. In order to replace this functionality, the ANN generated by the SWIRL algorithm must take a vector of j's as the input, and provide prob_match as the output. This necessitates that the ANNs have 23 input neurons and 1 output neuron. The obvious source of concern in this scenario is that ANNs inherently deal in weighted sums, whereas joint probability calls for a product. This is solved by taking advantage of the fact that $j > 0$ and $e^{\ln(x)+\ln(y)} = xy$ for x, $y > 0$. Thus, a vector of ln(j)'s is used as the ANN input, and e^{output} is used as prob_match.

In the simulation, a robot must navigate from an unknown starting position to a goal position that is specified on the map. Sensor noise is the main source of difficulty. The sensor noise has 3 components: bias, skew, and incidental. Each sensor has its own bias and skew values that are randomly initialized at the beginning of the simulation, but remain fixed thereafter. The incidental noise is a new Gaussian value generated each time the sensor is read.

The SWIRL system is first trained over one or more training runs. The best ANN produced is then used to replace the sensory component of the Markov localization system for the testing run. For the purpose of training, another Markov localizer with access to noise-free sensors is used to produce the "solution set" for the fitness function. This enables the ANN (ideally) to weigh the sensors according to their relative accuracy. The simulation results are shown in Fig. 4, Fig. 5, and Fig. 6. The results are produced for 1 and 3 training runs.

Unfortunately, the wavefront navigation system used by the Markov localization simulator introduces substantial variability into the robot's performance, as demonstrated by the large standard deviations. Consequently, it would require an exorbitantly large number of tests to establish with a high degree of certainty whether the average SWIRL solution is slightly better or slightly worse than the Markov method. In any case, improving Markov localization is not the goal here. It is clear from the results that SWIRL can indeed produce ANNs that function comparably to the sensory component of the Markov localization system.

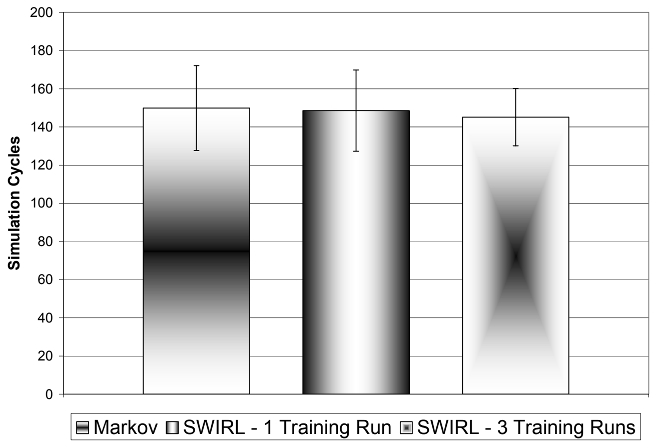

Fig. 4. Simulation cycles required to reach the goal. The bar represents the mean value with standard deviations.

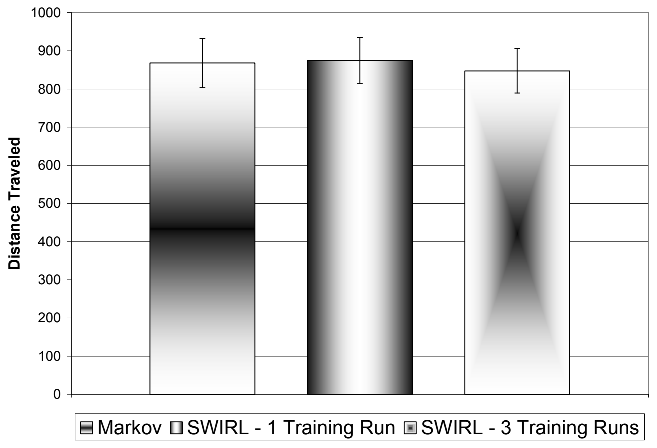

Fig. 5. Distance traveled to reach the goal.

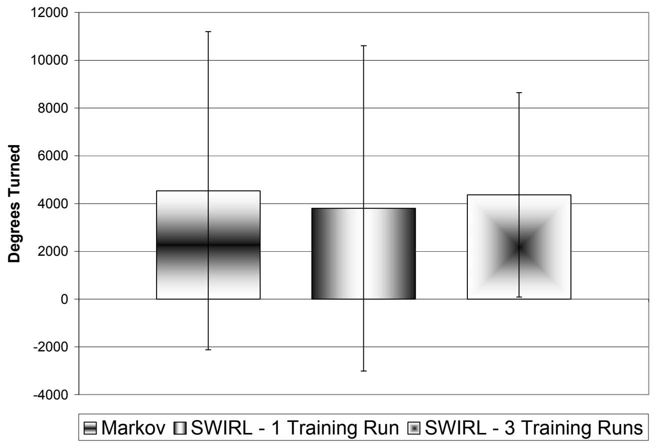

Fig. 6. Degrees turned to reach the goal. The staggering magnitude of the standard deviation values is primarily a consequence of how the robot responds to wall collisions.

SWIRL demonstrates its ability to generate effective solutions in the face of real-world complications such as noise and poor calibration. After only a single test run, the SWIRL solution is already performing comparably to the Markov method. The reason more fine-grained progress is not presented is that the nature of Markov localization is such that early mistakes are compounded each cycle. No matter how good the SWIRL solution gets, its mistakes from earlier in the run will prevent it from making accurate predictions. To make the graphical display useful, every so often during training (only during training) SWIRL's old belief matrix is overwritten with "correct" values so that one can see an estimate of how accurate the best ANN is at that point. However, this obviously gives a rough estimate only, and is not appropriate for a numerical comparison with the Markov predictions.

V. Conclusion

In this paper, a SWIRL algorithm is proposed to generate ANN solutions to tasks/problems amenable to reinforcement learning. Basically, the ACO algorithm is applied to select 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 double pole balance case study demonstrates that SWIRL is competitive with advanced neuroevolutionary techniques such as NEAT and ESP. The localization case study verifies the ability of the SWIRL algorithm to succeed in a noisy, imprecise environment with large-scale number of input signals, as might be encountered in the real world. 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.

As a result of its generality, the SWIRL method is scalable, robust, and can be applied to lots of real world tasks. The implementation in this paper is a proof-of-concept where the ACO algorithm chooses the number of hidden nodes. By limiting the scope to fully-connected feedforward neural networks, this single number fully defines the topology (since the input and output nodes are determined by the test problem). These are limits of this test implementation only; not limits on the SWIRL method.

References

[1] S. Haykin, "Neural Networks: A comprehensive Foundation". 2nd Edition, Prentice Hall, 1998.

[2] M. Dorigo, V. Maniezzo and A. Colorni. "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] E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing.* Natural Computing Series. MIT Press. Springer. Berlin. (2003).

[5] S. Kirkpatrick, C.D. Gellat Jr. and M.P. Vecchi, "Optimization by simulated annealing", *Science*, 220: 671-680, 1983.

[6] F. Glover, "Future paths for integer programming and links to artificial intelligence", *Computers and Operation Research*, Vol. 13, pp. 533-549, 1986.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] M. Riedmiller. "Rprop - description and implementations details", Technical report, University of Karlsruhe, 1994.

[13] T.B. Ludermir, Yamazaki, A. and Zanchetin, Cleber . "An Optimization Methodology for Neural Network Weights and Architectures" (to be published). *IEEE Transactions on Neural Networks*, v. 17, n. 5, 2006.

[14] F. van den Bergh, *An Analysis of Particle Swarm Optimizers*, PhD dissertation, Faculty of Natural and Agricultural Sciences, Univ. Pretoria, Pretoria, South Africa, 2002.

[15] Z. Zhen, Z. Wang, Z. Gu, and Y. Liu, "A Novel Memetic Algorithm for Global Optimization Based on PSO and SFLA", *ISICA 2007*, LNCS 4683, pp. 127–136, 2007.

[16] H. Meng, X. Zhang, and S. Liu, *A Co-evolutionary Particle Swarm Optimization-Based Method for Multiobjective Optimization*, S. Zhang and R. Jarvis (Eds.): AI 2005, LNAI 3809, pp. 349 – 359, 2005.

[17] 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.

[18] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies", *Evolutionary Computation* 10(2): 99-127. 2002.

[19] K. Patan and T. Parisini, "Stochastic learning methods for dynamic neural networks: simulated and real-data comparisons", *Proceedings of American Control Conference*, 2002.

[20] N. Saravanan and D. B. Fogel. "Evolving Neural Control Systems". *IEEE Expert*, 10(3):23-27. 1995.

[21] A. Wieland, Evolving Neural Network Controllers for Unstable Systems. *In Proceedings of the International Joint Conference on Neural Networks*, pages 667–673, IEEE Press, Piscataway, New Jersey. 1991.

[22] Y. Zheng, S. Luo, and Z. Lu, "Control Double Inverted Pendulum by Reinforcement Learning with Double CMAC Network", *The 18th International Conference on Pattern Recognition (ICPR'06).*

[23] F. Gomez and R. Miikkulainen. Solving Non-Markovian Control Tasks with Neuroevolution. In Dean, T., editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1356–1361, Morgan Kaufmann, San Francisco, California. 1999.

[24] J. Racz and A. Dubrawski. "Mobile Robot Localization with an Artificial Neural Network," *International Workshop on Intelligent Robotic Systems IRS '94*, Grenoble, France, July 1994.

[25] I.K. Sethi and G. Yu. "A Neural Network Approach to Robot Localization Using Ultrasonic Sensors," Proceedings of 5th *IEEE International Symposium on Intelligent Control*, 1990. pp. 513-517 vol. 1, 5-7 Sep 1990.

[26] W.S. Choi and S.Y. Oh. "Range Sensor-based Robot Localization Using Neural Network," *International Conference on Control, Automation and Systems, 2007. ICCAS '07.* pp. 230-234, 17-20 Oct. 2007.