In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC2002)*

Numerical Optimization with Neuroevolution

Brian Greer
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78705 USA
bgreer@cs.utexas.edu

Henri Hakonen
Systems Analysis Laboratory
Helsinki University of Technology
02015 Espoo, Finland
henri.hakonen@hut.fi

Risto Lahdelma
Department of Computer Science
University of Turku
20520 Turku, Finland
risto.lahdelma@cs.utu.fi

Risto Miikkulainen
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78705 USA
risto@cs.utexas.edu

Abstract - Neuroevolution techniques have been successful in many sequential decision tasks such as robot control and game playing. This paper aims at establishing whether they can be useful in numerical optimization more generally, by comparing neuroevolution to linear programming in a manufacturing optimization domain. It turns out that neuroevolution can learn to compensate for uncertainty in the data and outperform linear programming when the number of variables in the problem is small and the required precision is low, but the current techniques do not (yet) provide an advantage in problems where many variables must be optimized with high precision.

I. INTRODUCTION

Evolving neural networks with genetic algorithms has proven to be a powerful approach to many sequential decision making tasks such as robot control and game playing (Cliff et al. 1993; Fogel 2001; Gomez and Miikkulainen 1999, 2001; Moriarty et al. 1999; Nolfi et al. 1994; Whitley et al. 1993; Yao 1999). In such domains, a policy needs to be learned where optimal actions are identified for each state. Usually the state and the actions are discrete, or require only a limited amount of numerical precision. An open question at this point is whether neuroevolution methods could also solve large-scale numerical optimization problems such as those occurring in resource management, manufacturing, and process control.

Numerical optimization is usually performed using the well-known technique of Linear Programming (LP; see e.g. Taha 1987). LP can solve the optimization problem exactly, given that the constraints and the objective function are linear and deterministic. However, in real world applications there is often uncertainty in the data. Uncertainty makes it necessary to maintain safety margins, which in turn leads to suboptimal solutions.

One way to solve this problem in principle would be to analyze the distribution of the uncertainty in the data and use the observed regularities to improve the optimization process. It is difficult to make use of such distributions in LP because the problem effectively becomes nonlinear. An alternative is to replace LP with a nonlinear neural network, trained to make optimal decisions under uncertainty. Because the optimal decisions are not known, standard training methods such as backpropagation cannot be applied. However, a successful network can be evolved using genetic algorithms to maximize optimization performance.

In this paper, this idea is put to test in a challenging real-world application: optimizing a metal recycling process. The system has to represent a large number of mass, concentration, and cost variables accurately, and optimize these variables under uncertainty. Neural networks were evolved using a method called Enforced Sub-Populations (ESP), because this method has been shown more powerful than standard reinforcement learning techniques in many control tasks (Gomez and Miikkulainen 1997, 1999, 2001).

The main result was that neuroevolution found better solutions than Linear Programming when the number of variables was low. Neuroevolution was also able to improve its solution as the amount of precision required was reduced. This is a promising result, which also presents a significant challenge for further neuroevolution research.

II. THE ESP NEUROEVOLUTION METHOD

ESP is an extension of Symbiotic Adaptive Neuroevolution (SANE; Moriarty and Miikkulainen 1996, 1997). Unlike standard neuroevolution, where the population consists of full networks, SANE evolves a population of neurons. Each neuron has its own input and output weights, so that a 3-layer feed-forward network can be formed by placing a number of these neurons together into the hidden layer. The neurons are evaluated based on how well the different networks in which they

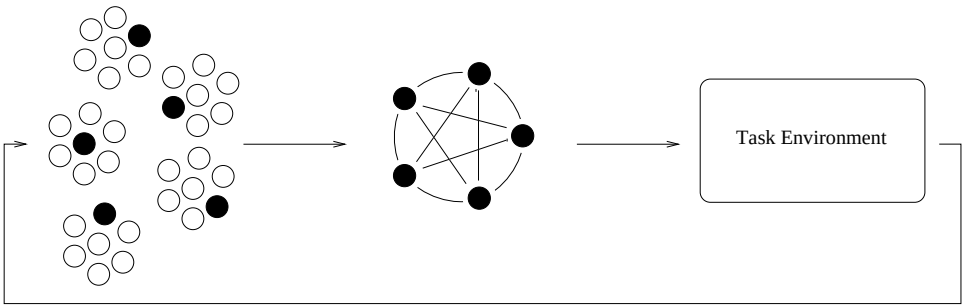

Fig. 1. **The Enforced Sub-Populations Method (ESP).** The population of neurons is segregated into subpopulations shown here as clusters of circles. The network is formed by randomly selecting one neuron from each subpopulation. Each network is tested in the task and a fitness score is assigned to each of the neurons depending on how well the networks in which they participate perform on average. This way evolution searches for compatible subtasks and optimizes each subtask separately, which is faster and more powerful than a direct search of full networks.

participate perform in the task. Since good networks require different kinds of neurons, diversity is automatically maintained in the population, which allows evolution to proceed more effectively than is possible in network-level evolution.

ESP takes this idea one step further by evolving each hidden neuron in its own subpopulation Gomez and Miikkulainen 1997, 1999, 2001. Crossover only occurs between neurons of a particular subpopulation, allowing each neuron to specialize in its role in the network (figure 1). In effect, evolution not only maintains diversity automatically, it also defines compatible subtasks, and finds the solution by optimizing each subtask. Such subtasking makes the search for solution more efficient.

ESP has been shown to perform well in many benchmark tasks such as prey capture and double pole balancing (Gomez and Miikkulainen 1997, 1999, 2001). It can solve easier tasks faster than standard reinforcement learning methods and other neuroevolution methods, and it can solve harder versions of such problems. However, ESP and other neuroevolution methods have not been tested before in numerical optimization problems requiring high precision and high dimensionality.

III. EXPERIMENTS

A. *The Optimization Domain*

In the experiments, ESP was applied to the domain of optimizing a metal recycling process (Lahdelma et al. 1999). In the most general form, the problem is as follows: A recycling plant receives raw materials with stochastic amounts and intervals. Using these raw materials, the task is to produce alloys with specific concentrations of elements (such as Cu, Si, Fe...) through a melting process to fill outstanding orders, which also arrive stochastically. The goal is to maximize long-term profits, taking into account raw material costs and product values, quality tolerances, and storage costs.

Uncertainty is introduced in the melting process in several places. The element concentrations in the raw materials are rough estimates; raw materials may interact in the furnace; some elements may burn off more quickly than others, changing the concentrations of the final product unexpectedly. These random factors make the optimization problem difficult to solve reliably and accurately.

Rather than optimizing the entire process, the experiments in this paper focus on robust optimization under uncertainty. To reduce confounding factors, the problem was limited to optimizing the value of one order the same way it is commonly done in practice: the goal is to produce as much of the product as possible in one charge to the furnace. More specifically, the task is to find the masses x_j of raw materials j such that

$$\text{minimize} \quad x_{\max} - \textstyle\sum_j x_j, \tag{1}$$

$$\text{subject to}$$

$$\textstyle\sum_j x_j \le x_{\max}, \tag{2}$$

$$0 \le x_j \le x_{j,\max}, \tag{3}$$

$$\textstyle\sum_j r_i p_{ij} a_j x_j \ge z t_{i,\min}, \tag{4}$$

$$\textstyle\sum_j r_i p_{ij} a_j x_j \le z t_{i,\max}, \tag{5}$$

where $x_{\max}$ is the maximum mass that may be placed in the furnace, $x_{j,\max}$ is the total amount of material j that is available, $r_i \in [0..1]$ is the yield of element i, p_{ij} is the concentration of element i in material j, $a_j \in [0..1]$ is the yield of material j, $z = \sum_j a_j x_j$ is the mass of the product, and $t_{i,\min}$ and $t_{i,\max}$ are the minimum and maximum allowed concentrations of element i in the product.

A database of 1000 actual orders of the Kuusakoski aluminum recycling plant (Lahdelma et al. 1999) was used in training and testing the system. Each order specified the product mass and boundaries for the element concentrations, as well as the available raw material amounts and concentrations. There were 13 raw materials in the system and concentrations were specified for 10 different elements for each raw material (as shown in figure 2).

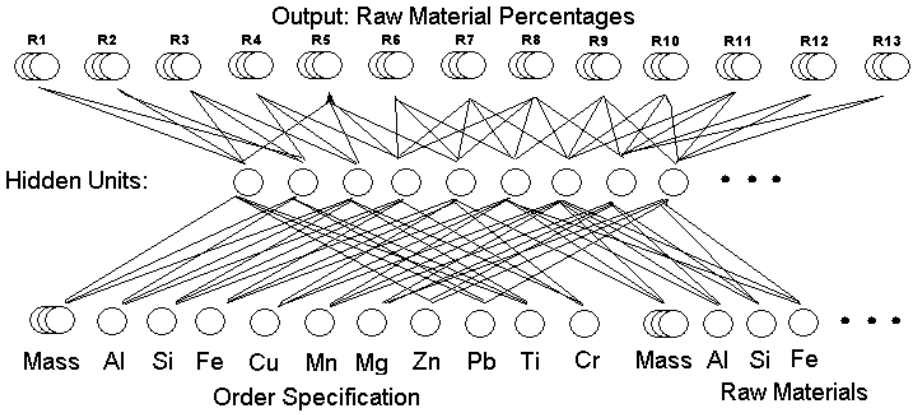

Fig. 2. **An optimization network.** Each network had 20 hidden units, fully connected to the input and output (only a subset of the connections are shown). The inputs consist of the desired product mass and element concentrations, and the mass and element concentrations of each raw material. The outputs consist of the percentages of each raw material to be used in the process. The masses and the percentages are mapped to intervals represented by (max) 20 units (shown as clusters of units); the concentrations were scaled between 0 and 1 and represented by activation values of single units.

The input to the optimizer (a network or the LP program) consisted of the masses and element concentrations of the available raw materials and the target product mass and concentration boundaries. The outputs indicated the percentages of each raw material to be used for the charge. In the domain simulation, the product mass and concentrations were first calculated exactly using the output values, and a significant amount of noise (25%) was then added to the calculated values. Such stochasticity models the uncertainty in the melting process and in the raw material concentrations. Although there are no hidden regularities in the distributions in this simple model, the evolution process should be able to adjust to the level of uncertainty to improve its performance.

B. *Optimization under Uncertainty*

The optimization problem specified above makes sense if the values are exact. However, the task needs to be modified slightly to allow optimization under uncertainty.

Because of the noise, LP optimization of equations 1–5 would often generate products that did not meet the constraints 2-5. Therefore, safety margins were introduced to problems given to LP. The element concentration tolerance, i.e. the difference between $t_{i,\min}$ and $t_{i,\max}$, was reduced 50%. This way LP was forced to find solutions away from the boundaries, leaving a safety margin against noise. The 50% reduction was the maximum safety margin possible: tighter bounds would have made the optimization problems infeasible.

The objective function also needed to be modified to serve as a fitness function for ESP. Due to the explorative nature of the evolutionary search process, a large number of the ESP solutions are infeasible during learning. If the charge was too large, violating constraint (2), a penalty equal to undershooting by the equal amount was introduced to fitness equation (1). Because the raw material amounts represent percentages, constraint (3) was never violated. If the element concentration bounds (4) or (5) were exceeded, exponential penalties s_i were introduced:

$$s_i = \lambda \begin{cases} e^{t_{i,\text{avg}} - c_i} & \text{if } t_{i,\min} > c_i, \\ e^{c_i - t_{i,\text{avg}}} & \text{if } c_i > t_{i,\max}, \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

where c_i is the concentration of element i in the product and $t_{i,\text{avg}}$ is the average of $t_{i,\min}$ and $t_{i,\max}$, i.e. the minimum and maximum concentrations of element i. The constant λ can be adjusted experimentally to encourage feasibility without overwhelming the other constraints. The value $\lambda = 100$ was found to work well in these experiments.

Because of the uncertainty in the domain, both LP and ESP still sometimes generate infeasible solutions. The objective function therefore cannot be directly used to compare their performance across a large set of orders. Instead, a profit function was used which takes into account the cost of raw materials, the cost of additives to fix a product if possible, and the value of the product:

$$P = \alpha m - \sum_i c_i A_i - \sum_j d_j x_j, \tag{7}$$

where α is the value per unit of mass for the product in the order, m is the total mass of the product produced, c_i is the cost of the additive i per unit of mass and A_i its mass, and d_j is the cost of raw material j per unit mass. This way, if some concentrations were low and it was possible to bring them within tolerances by adding pure elements, it was done and the additive costs reduced the profit. If the concentration

constraints could not be met, the mass m was set equal to 0, so that the order resulted in negative profit.

C. *Neuroevolution Implementation*

A crucial issue in implementing numerical tasks with neural networks is the representation of numbers at the input and output. Possible approaches include: (1) scaling numerical values between 0 and 1 and representing them as activation values of single units; (2) representing each possible value as a separate unit, with only one unit on at any one time (i.e. the value value unit approach); and (3) activating such value units according to a Gaussian pattern whose mean identifies the actual value (Ballard 1987). With neuroevolution, the number of units is also a confounding issue because it is generally more difficult to optimize large networks than small ones.

After considerable experimentation, a combination of the above three techniques was found to perform the best (figure 2). In the input, each mass variable was mapped to 5 value units, representing equal intervals between maximum and minimum values. These units were activated with a Gaussian pattern whose mean identified the actual value. The concentrations turned out to require less precision, and were represented by single unit activations between 0 and 1, representing the minimum and maximum values. At the output, each raw material quantity was represented by (max) 20 value units: the most highly activated unit identified the percentage of that raw material to be used in the charge.

Each ESP network had 20 hidden units, fully connected to the input and output. [1] There were therefore 20 subpopulations, each consisting of 100 neuron chromosomes. Each chromosome was formed by concatenating the floating point numbers representing the input-to-hidden and the hidden-to-output weight values of the neuron. One-point crossover was used to generate two offspring, and each offspring chromosome was mutated with a probability of 0.2, replacing a randomly chosen weight in the encoding with a random value within [0.0, 1.0]. At each generation, the top 25% of the population was randomly mated to replace the bottom 50% of the population. The offspring was evaluated by randomly forming 200 networks from the current subpopulations.

IV. RESULTS

Figure 3 depicts the progress of a typical evolution simulation, in this case with full dimensionality and precision. ESP's performance first improves significantly during the first 200 generations of evolution, but then levels off, and very little further progress is made even in prolonged evolution. Each of the simulations were run until such stagnation was observed,

[1] In preliminary experiments, computing the output percentage in terms of activation-weighted average was also tried but the results were not significantly different. Also, hidden layers with 5, 20, 50, and 100 units were tried, but they did not significantly differ in performance.

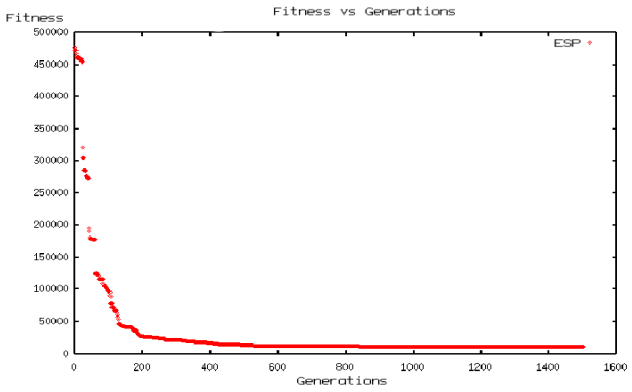

Fig. 3. **Progress of ESP Performance over Time.** Fitness of the best network is plotted over time (i.e. generations of evolution) for the case of 13 raw materials and 20 output units for each percentage. The initial progress is quite rapid but eventually the performance levels off and does not improve even in prolonged evolution.

which usually took about 1000–2000 generations, or 3–5 days on a 1-GHz PIII machine.

In order to determine how ESP compares to LP with different degrees of input dimensionality, both methods were tested using a varying number of input raw materials: 2, 3, 6, 9, 11, and 13. The orders were chosen so that it was always possible to fill them with the available raw materials. In each case a 5-fold cross-validation experiment was performed on the 1000 order database. In each of the 5 runs, a set of 200 orders was chosen randomly for each evaluation out of an 800 order training set and used in measuring fitness during evolution. A set of 200 orders were used to measure the final network performance after evolution had been observed to stagnate. The same test set was used to measure the performance of LP.

Figure 4 summarizes the performance of each methods. When only two raw materials were used, ESP achieved a fitness of 994,661 which was significantly better than that of LP. The difference was insignificant for 4, 6, and 9 raw materials, but for the full 13, LP performed significantly better, at 867,178. In other words, when the input dimensionality is low, ESP significantly outperforms LP, apparently by compensating for the uncertainty in the data as we hypothesized. As the number of dimensions grows, however, ESP begins to have more trouble and eventually cannot compete with LP.

In previous work, neuroevolution has been shown effective even with very high-dimensional input when the required output precision is low (as e.g. in selecting the best move in game playing or prey capture Gomez and Miikkulainen 1997; Moriarty et al. 1999). To test the effect of precision on performance, ESP was also tested on optimizing the full set of 13 raw materials within a limited degree of output precision, i.e. 3, 4, 5, 10, and 20 possible values for each output percentage. A 5-fold cross-validation experiment was performed on

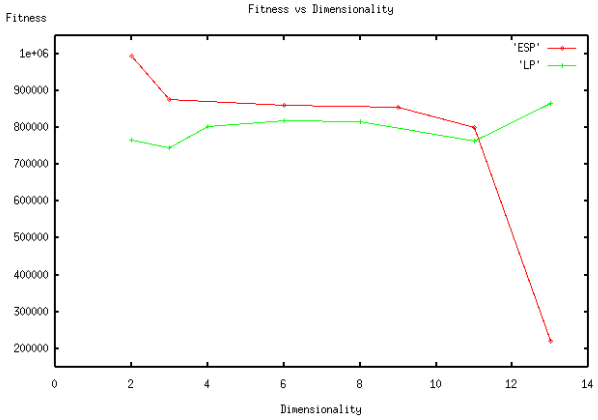

Fig. 4. **Neuroevolution Performance with Varying Dimensionality.** With a low number of input dimensions, ESP performs significantly better than LP, but cannot compete with full 13 raw materials as inputs. In the mid-range, the differences are not significant. The plots are averages of five runs.

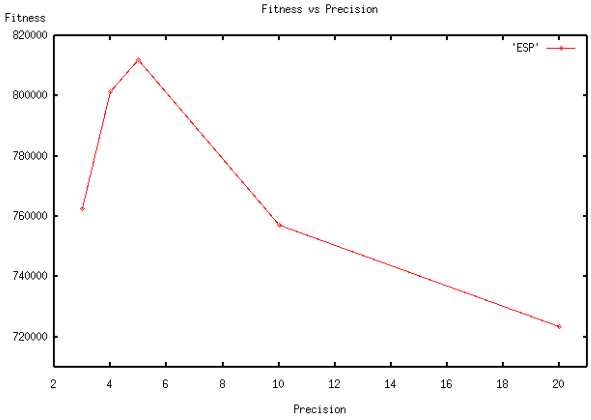

Fig. 5. **Neuroevolution Performance with Varying Precision.** With only 5 units to represent the different percentage values, the performance of ESP is comparable to that of LP, but decreases when more precision is required. The plot is an average of five runs.

the same data in each case.

The results are seen in figure 5. Interestingly, the best performance is achieved with very low levels precision. With only 5 output percentage levels to choose from, the performance is 811,870, i.e. comparable to that of LP. As the number of output units increases, the performance decreases; apparently ESP has trouble optimizing a large number of output weights. With fewer than 5 output units for each percentage, the performance also declines because there are not enough options to construct a feasible order within the constraints.

Together these results suggest that neuroevolution is able to compensate for the uncertainty, but currently does not scale up very well when the required dimensionality and precision is increased.

V. DISCUSSION AND FUTURE WORK

The results present both a promise and a challenge for neuroevolution research. They are promising in that it is possible to learn to compensate for uncertainty in the data, and therefore probably also to utilize hidden regularities in the data. Such a capability should be valuable in many optimization problems in the real world. For example, it may be possible to train another neural network (with backpropagation) to predict the outcome of the melting process, thereby capturing the otherwise unknown regularities in the domain. Neuroevolution can then be used to develop a decision making network that performs optimally wrt. the neural network model of the domain, Such a decision making network should then perform very well in the actual real world task as well.

The challenge is to improve the methods so that they can deal with more variables accurately. Fine tuning neural networks with evolution in general is difficult. Unlike with gradient descent methods, where the gradient can be followed by making arbitrarily small changes to all weights, in neuroevolution the changes tend to be more drastic: swapping parts of the network, or mutating an individual weight significantly. Such a process is good for discovery, but may need to be augmented by e.g. stochastic hillclimbing, value function iteration, or particle swarming (Conradie et al. 2002) to achieve the desired precision in many dimensions at once. It may also be possible to achieve higher accuracy with indirect, compressed representation of the network weights (Whitley et al. 1995). Methods that optimize the structure of the network as part of the evolution (Stanley and Miikkulainen 2002; Whitley et al. 1995) may also help balance the demands for high dimensionality and high precision.

It is worth noting that numerical optimization with neuroevolution has one clear advantage over standard methods such as LP: optimization requires only one propagation through the network, which is extremely fast. With full dimensionality and precision, our evolved network was 600 times faster than LP. With fixed-size hidden layer, the optimization time grows linearly with dimensionality. Such extremely fast approximate optimization may turn out highly useful in e.g. real-time domains.

VI. CONCLUSION

Neuroevolution has been shown to perform well in many control tasks including pole balancing and robot arm control. These domains require either low precision with many parameters (e.g. pole balancing) or high precision with few parameters (e.g. robot arm). In such domains, neuroevolution is able to learn and utilize the underlying regularities in the domain, and can perform significantly better than direct mathematical

optimization methods such as Linear Programming. However, the current neuroevolution methods do not provide an advantage if the problem requires a high degree of precision with many parameters simultaneously, posing an interesting challenge for future neuroevolution research.

Acknowledgments

This research was supported in part by the National Science Foundation under grant IIS-0083776, by the Texas Higher Education Coordinating Board under grant ARP-003658-476-2001, and the AMRO (Adaptive Metallurgical Raw-material Optimization) project of the National Technology Agency of Finland.

References

Ballard, D. H. (1987). Interpolation coding: A representation for numbers in neural nets. *Biological Cybernetics*, 57:389–402.

Cliff, D., Husbands, P., and Harvey, I. (1993). Evolving recurrent dynamical networks for robot control. In *Proceedings of ANNGA93, International Conference on Artificial Neural Networks and Genetic Algorithms*. Innsbruck: Springer-Verlag.

Conradie, A., Aldrich, C., and Miikkulainen, R. (2002). Intelligent process control utilizing symbiotic memetic neuroevolution. In *Proceedings of the 2002 Congress on Evolutionary Computation*.

Fogel, D. B. (2001). *Blondie24: Playing at the Edge of AI*. San Francisco, CA: Morgan Kaufmann.

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

Gomez, F., and Miikkulainen, R. (1999). Solving non-Markovian control tasks with neuroevolution. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*. Denver, CO: Morgan Kaufmann.

Gomez, F., and Miikkulainen, R. (2001). Learning robust nonlinear control with neuroevolution. Technical Report AI01-292, Department of Computer Sciences, The University of Texas at Austin.

Lahdelma, R., Hakonen, H., and Ikäheimo, J. (1999). AMRO: Adaptive metallurgical raw material optimization. In *Proceedings of the 1999 Conference of the International Federation of Operational Research Societies*.

Moriarty, D. E., and Miikkulainen, R. (1996). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32.

Moriarty, D. E., and Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive co-evolution. *Evolutionary Computation*, 5:373–399.

Moriarty, D. E., Schultz, A. C., and Grefenstette, J. J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:199–229.

Nolfi, S., Elman, J. L., and Parisi, D. (1994). Learning and evolution in neural networks. *Adaptive Behavior*, 2:5–28.

Stanley, K., and Miikkulainen, R. (2002). Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation*.

Taha, H. (1987). *Operations Research: An Introduction*. Macmillan Publishing Company.

Whitley, D., Dominic, S., Das, R., and Anderson, C. W. (1993). Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, 13:259–284.

Whitley, D., Gruau, F., and Pyeatt, L. (1995). Cellular encoding applied to neurocontrol. In Eshelman, L. J., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, 460–469. San Francisco, CA: Morgan Kaufmann.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.