

# Numerical Optimization with Neuroevolution

Brian Greer

Department of Computer Sciences

University of Texas at Austin

Austin, TX 78705 USA

*bgreer@cs.utexas.edu*

Senior Honors Thesis. Advisor: Risto Miikkulainen

December 8, 2001

## 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 accuracy is low, but the current techniques do not (yet) provide an advantage in problems where many variables must be optimized with high accuracy.

## 1 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 (Gomez and Miikkulainen 1999, 2001; Moriarty et al. 1999; 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 accuracy. 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 or manufacturing.

This paper presents an application of neuroevolution to such a task: 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 other reinforcement learning techniques in many control tasks (Gomez and Miikkulainen 1997, 1999, 2001).

The main result was that neuroevolution found more accurate solutions than Linear Programming when fewer variables were present. Neuroevolution was also able to improve its performance with a constant number of variables as the amount of accuracy required diminished. This is a promising result, which also presents a significant challenge for further neuroevolution research.

## 2 Operations Research

Operations research attempts to optimize the use of time, money, raw materials, production facilities, and shipping capabilities. Increasing overall profits is always the bottom line, which means that cost minimization is central to the equation. Optimization of the cost function is commonly formulated as a multi-objective constrained optimization problem, which attempts to minimize the cost function within the constraints of the process domain. A generic formulation of the problem is of the form:

$$\begin{array}{ll} \text{minimize} & F(x), \\ \text{subject to} & b_1 \leq Ax \leq b_2 \end{array}$$

where  $F(x)$  is the cost function,  $x$  is a vector of variables that are involved in the manufacturing process,  $A$  is the matrix of known coefficients,  $b_1$  and  $b_2$  are the lower and upper bounds, respectively, of the manufacturing process.

Inventory scheduling is the area of operations research that deals with the efficient allotment of raw materials to create finished goods. Costs, amounts, and specifications for each of the raw material must be considered in order to efficiently optimize the overall cost of producing a finished product. Optimization in the metal recycling plant is an instance of such an inventory scheduling problem.

## 2.1 Linear Programming

Numerical optimization has been effectively tackled using the well-known technique of Linear Programming (LP; see e.g. Dantzig 1967; Taha 1987), which can solve the problem exactly, given that the constraints and the objective function are linear and deterministic.

The simplex method is the most common LP method used in operations research because it guarantees that an optimal solution will be found within a finite amount of time. It explores the boundary between the feasible and infeasible regions of the the state space by fixing some of the variables in the optimization equation to one of their boundary conditions in order to create a “basic” solution. The algorithm then moves along that boundary looking for a minimal feasible solution by assigning the remaining variables unique values (Dantzig 1967).

Uncertainty in manufacturing data often makes the optimization problem nonlinear, which causes LP to deliver only approximate 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. For example, Genetic Algorithms can be used to detect these kinds of distributions, and they can be encoded in neural networks, which then utilize them during decision making. The ability of neural networks to utilize these observed regularities is the motivation for their use.

## 3 The Neuroevolution Method

### 3.1 Genetic Algorithms

Genetic Algorithms (GAs) are inspired by Darwin’s theory of biological evolution (Goldberg 1989; Holland 1975). GAs are distinguished from other global search techniques by the fact that they maintain a population of prospective solutions instead of only optimizing a single solution. The result of the process, however, is still a single individual that represents an optimal solution. Complex problems are approached from the standpoint that a population of parallel search agents is able to converge on a global optimum much more efficiently than a single agent.

Biological organisms encode their traits and characteristics (genomes) in a chromosome. The chromosome is a fundamental aspect in evolution since it contains the hereditary information of the organism that will be passed on to its offspring. Genetic computing must reproduce the chromosome in such a way as to facilitate the use of evolutionary operators (crossover and mutation) on members of the population to create offspring. Neuroevolution methodologies often diverge with respect to the definition of the chromosome in an attempt to find an optimal encoding structure. Often the entire network may be encoded in the chromosome so that the search space contains all networks of a certain structure, and all network weights are evolved simultaneously. Other encoding schemas include encoding individual neurons (Moriarty and Miikkulainen 1996a) or encoding the topology of the network (Stanley and Miikkulainen 2001).

As in biological evolution, the genetic operators of selection, crossover, and mutation are applied to the members of the simulated populations to create successive generations. Selection determines which members of the population are allowed to reproduce and with what frequency. Crossover represents biological reproduction, and mutation simulates random mutations that may occur during reproduction. During each generation, members of the population are

evaluated on a given task and assigned a fitness value. Those members of the population with higher fitness values are more likely to be selected for reproduction thereby allowing whatever dominant traits they possess to help solve the problem to carryover to the next generation. This model is constructed in a similar fashion to a “survival of the fittest” approach in Darwinian evolution where members of the population with desirable traits are more likely to be selected by other members of the population for reproduction.

GAs have proven to be effective in many real world optimization tasks such as electromagnetic system design (Michielssen and Weile 1995) and aerodynamic design (Periaux et al. 1995).

### **3.2 Neuroevolution**

GAs have been used in conjunction with neural networks to evolve weights in fixed architecture networks, evolve network topologies, and to select training data and interpret output behavior of neural networks. When neural networks are evolved using GAs the method is referred to as neuroevolution.

The training is done in neuroevolution using a reinforcement system whereby individual networks are tested on a given task and assigned a fitness, which then determines their relative position in a population of networks. Superior networks (networks that perform well in the given task) are allowed to reproduce more often (have a higher selection probability). A population of networks is evolved in parallel in this manner until some threshold fitness is met or the performance of the population stagnates.

The reinforcement aspect of neuroevolution stems from the fact that networks do not require immediate feedback during the training process, rather a general measure of fitness is calculated after the entire task has been completed. Previous knowledge about the domain is also not required with this method since pertinent knowledge is acquired through the evolution process. This allows the networks to ascertain what knowledge is required in order to solve the problem sufficiently.

Stagnation of the population is handled differently for different methodologies. One possible solution to stagnation is to replace some percentage of the unfit members of the population by random mutations of the best network. This is how Delta-Coding in ESP works as discussed below.

### **3.3 SANE**

David Moriarty altered the classic approach to neuroevolution by evolving populations of neurons instead of populations of entire networks. This method is known as Symbiotic Adaptive Neuroevolution (SANE; Moriarty and Mikkulainen 1996a, 1997). Each neuron is encoded with its input and output weights in a string (chromosome). Three-layer feedforward networks are built using a randomly chosen neuron from each population to form the hidden layer. The network is then evaluated and given a fitness value. This fitness value is passed on to each neuron that participated in the network. Neurons are then selected for based on how well they performed in each of the networks in which they participated.

Since good networks require different kinds of neurons, diversity is automatically maintained in the population. The network architecture itself, however, is fixed during evolution so that only the weight space of the individual neurons is explored.

### **3.4 ESP**

SANE was extended by Faustino Gomez by grouping hidden neurons into subpopulations in a method known as Enforced Sub-populations (ESP; Gomez and Mikkulainen 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 was also used with Delta-Coding turned on in order to handle stagnation. Whenever ESP detects stagnation in the current population, a percentage of the population with the lower fitness values is replaced by perturbations of the best neuron.

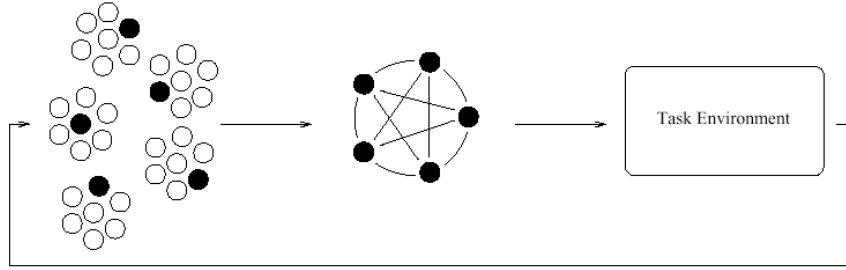


Figure 1: **The Enforced Sub-Populations Method (ESP)**. The population of neurons is segregated into subpopulations shown here as clusters of circles. The network is formed by randomly selecting one neuron from each subpopulation. Each network is tested in the task and a fitness score is assigned to 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.

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. ESP and other neuroevolution methods, however, have not been tested in numerical optimization problems of high accuracy and dimensionality.

## 4 Experiments

### 4.1 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, and the properties of the different materials are known only approximately. Using these raw materials, the task is to produce alloys with specific concentrations of elements (such as Cu, Si, Fe, etc) 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, 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.

Optimization in this process was originally done by experts who knew the manufacturing process well and were able to utilize their experience to make “well-informed” decisions on how to compensate for randomness in the process. Linear Programming was implemented to automate the decision process for which raw materials to use, and what amounts, for each order. Decisions are made with a strong focus not only on creating feasible orders, but, more importantly, on maximizing the profitability of each order.

### 4.2 Problem Formulation

The experiments in this paper focus on robust optimization under uncertainty. To reduce confounding factors, the problem was limited to optimizing the value of only one order. That is, the goal was to find the amounts of raw materials that result in as large an amount of the product as possible in one charge to the furnace. More specifically, the task was to find the masses  $x_j$  of raw material  $j$  such that

$$\text{minimize} \quad z - \sum_j a_j x_j, \quad (1)$$

subject to

$$\sum_j a_j x_j \leq z, \quad (2)$$

$$0 \leq x_j \leq x_{j,\max}, \quad (3)$$

$$\sum_j r_i p_{ij} a_j x_j \geq z t_{i,\min}, \quad (4)$$

$$\sum_j r_i p_{ij} a_j x_j \leq z t_{i,\max}, \quad (5)$$

where  $z$  is the maximum mass that may be placed in the furnace,  $a_j \in [0..1]$  is the yield of material  $j$ ,  $x_{j,\max}$  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  $j$ , 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. The maximum order size had 13 raw materials that could be used to satisfy each order. Each raw material had 10 elemental concentrations specified.

### 4.3 Network Architecture

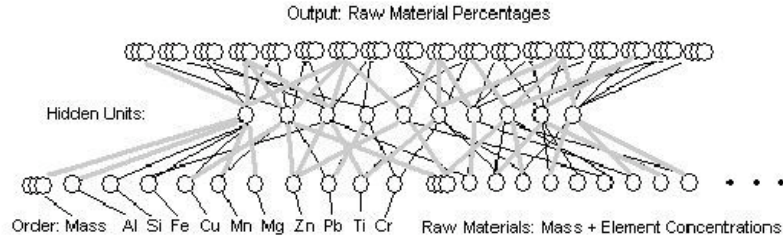


Figure 2: **An optimization network.** Each network had 20 hidden units, fully connected to the input and output. The inputs consist of the desired product mass and element concentrations, and the amount and element concentrations of each raw material. The outputs consist of the percentages of each raw material to be used in the process. The masses and the percentages are mapped to intervals represented by (max) 20 units, 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 percentage of each raw material to use. In the domain simulation, the product mass and concentrations were first calculated exactly using these percentages. A significant amount of noise (25%) was then added to the mass and element concentrations. Such stochasticity models the uncertainty in the melting process and in the raw material concentrations.

Numerical values in the input and output of the neural network were represented partly by range coding and partly by activation coding (figure 2). In the input, each mass variable was mapped to 10 units, each representing an interval between maximum and minimum value. The interval assigned to each node was made in order to account for the distribution of the data. These units were activated with a gaussian pattern whose mean identified the actual value. The concentrations turned out to require less accuracy, and were represented by single unit activations between 0 and 1, interpreted to represent minimum and maximum values. At the output, each percentage value was represented by (max) 20 units: the most highly activated unit identified the percentage to be used for that raw material. Each ESP network had 20 hidden units, fully connected to the input and output. With full dimensionality (13 raw materials) the input size was 290 units. The largest number of bins in the output (20) yielded 260 output units.

Initially networks were tested using continuous values in the input to represent raw material charges, and continuous values were used in the output layer to represent the percentage values. Results from these experiments have been omitted here because the value unit approach resulted in higher fitness networks that corresponded more closely with LP’s fitness. Discretizing the outputs provided the networks with a finite choice of percentage values allowing for an experimental review of the accuracy that the networks could handle in such a large-scale, continuous optimization problem. For a reference on value unit encoding see Ballard 1987.

## 4.4 Algorithms

The above optimization problem can be solved directly with Linear Programming; however, such LP optimization does not take into account uncertainty in the inputs and in the melting process. To make the optimization more robust against noise, the element concentration tolerance, i.e. the difference between  $t_{i,\min}$  and  $t_{i,\max}$ , was reduced 50% whenever possible (i.e. if LP was still able to find feasible solutions). This way LP would find solutions away from the boundaries, leaving a safety margin against noise.

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. 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:

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

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  $\lambda = 100$  is a weighting constant determined experimentally.

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:

$$P = \alpha m - \sum_i c_i A_i - \sum_j d_j x_j, \quad (7)$$

where  $\alpha$  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.

In order to determine how ESP scales with dimensionality and with accuracy, it was tested using varying number of input raw materials (2, 3, 6, 9, 11, and 13) and with varying amounts of output accuracy (3, 4, 5, 10, and 20 possible values for each output percentage). 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 be stagnate.

## 5 Results

In the case of full dimensionality and full accuracy (13 input raw materials with 20 possible percentage values in the output). ESP performance first improved significantly during the first 200 generations of evolution, but then stagnated at a level inferior to that of LP. This result is evident in figure 3 where there is a rapid descent in  $f$  followed by

essentially a flat fitness (average over the 5 runs). ESP had a fitness of 220,763 at the time of stagnation, while LP had 867,178.

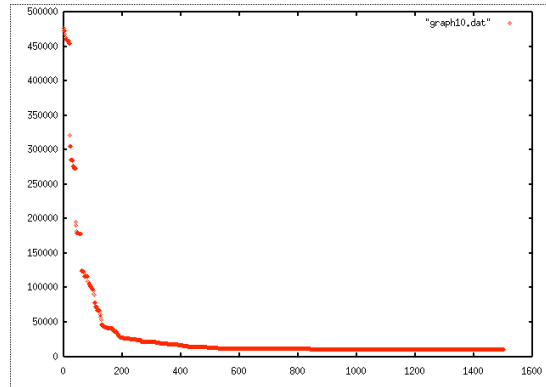


Figure 3: **Progress of Neuroevolution with High Accuracy and High Dimensionality.** Fitness of the best network is plotted over time (i.e. generations of evolution). ESP makes a reasonable initial progress but levels off at a fitness of 220,763, which is considerably weaker than that of LP at 867,178.

To find out what was causing ESP to stagnate, the effects of input dimensionality and output accuracy were tested. In each test, different sizes of the hidden layer were tried as well (5, 20, 50, and 100). A smaller hidden layer yielded weaker performance, but expanding the hidden layer did not significantly affect the performance.

In the first test, the number of raw materials in the input was varied (between 2, 3, 6, 9, 11, and 13), with orders chosen so that it was always possible to fill them with the available raw materials. The number of output bins was fixed at 5 since experiments with accuracy had shown that 5 bins delivered close to optimal performance. Figure 4 summarizes the results. When only two raw materials were used, ESP achieved a fitness of 994,661 which was significantly better than the 867,178 of LP. The difference was insignificant for three, four, six, nine and eleven raw materials, and as we already saw, for the full 13, LP performed significantly better. In other words, when the input dimensionality is low, ESP significantly outperforms LP, apparently by compensating for the uncertainty in the data. As the number of dimensions grows, however, ESP begins to have more trouble and eventually cannot compete with LP.

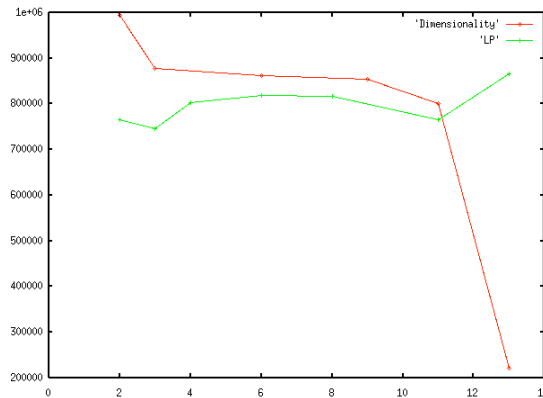


Figure 4: **Neuroevolution Performance with Varying Dimensionality.** With a low number of input dimensions, ESP performs significantly better than LP, but cannot compete with full 13 raw materials as inputs. In the mid-range, the differences are not significant.

In previous work, neuroevolution has been shown effective even with very high-dimensional input and output when the required accuracy 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 accuracy on performance, ESP was also tested on optimizing the full set of 13 raw materials within a limited degree of output accuracy (3, 4, 5, 10, and 20 output units).

In each of these tests the dimensionality was fixed at 13 input raw materials.

The results are seen in figure 5. The best performance is achieved with the lower output accuracy until 4 output bins are used and the performance declines again. With only 5 output percentage levels to choose from, the performance (fitness 811,870) is 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 begins to decline again because there are not enough options to construct a feasible order within the constraints.

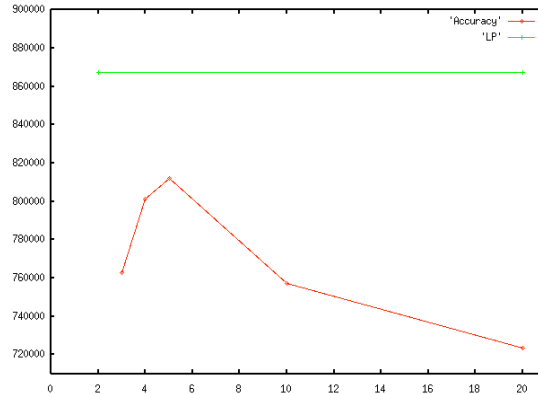


Figure 5: **Neuroevolution Performance with Varying Accuracy.** With only 5 units to represent the different percentage values, the performance of ESP is compatible to that of LP, but decreases when more accuracy is required, and when fewer output bins are available.

## 6 Discussion and Future Work

The results present both a promise and a challenge for neuroevolution research. It is possible to learn to compensate for uncertainty in the data, and perhaps even to utilize hidden regularities in the data, which property will be valuable in many optimization problems in the real world. 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 or value function iteration to achieve the desired accuracy in many dimensions at once.

In cases where neuroevolution was able to handle the problem with decreased accuracy fine tuning of the solution may be accomplished using alternate optimization methods such as swarm optimization. Another network may also be evolved to fine tune the solution after an initial network has come up with an approximate solution. This setup would be similar to evolving obstacle avoidance in Moriarty and Miikkulainen 1996b.

Incremental evolution of network topologies (Stanley and Miikkulainen 2001) may also yield better networks assuming that structure is a limiting factor in the current setup.

The current magnitude of the problem (in terms of dimensionality as well as accuracy) may mean that a hierarchical setup of networks may be able to efficiently reduce the size of the problem by utilizing multiple networks to handle the inputs and the outputs.

## 7 Conclusion

Neuroevolution has previously been shown to perform well in many control tasks including pole balancing and robot arm control. These domains require either low accuracy with many parameters (e.g. pole balancing) or high accuracy

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 accuracy 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, and the AMRO (Adaptive Metallurgical Raw-material Optimization) project of the National Technology Agency of Finland.

Special thanks to my advisor, Risto Miikkulainen, for his contributions and guidance throughout this project. I would also like to thank Henri Hankonen for his help in understanding the metal recycling process and his work on the LP optimizer and furnace simulator; Rich Robison for creating tools that interface with Henri's LP optimizer that have helped me immensely; and Risto Lahdelma for his advice and contributions about the LP simulation.

### **References**

- Ballard, D. H. (1987). Interpolation coding: A representation for numbers in neural nets. *Biological Cybernetics*, 57:389–402.
- Dantzig, G. B. (1967). *Linear Programming and Extensions*. Princeton University Press.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley.
- 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.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Ann Arbor, MI: University of Michigan Press.
- 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*.
- Michielssen, E., and Weile, D. (1995). Electromagnetic system design using genetic algorithms. In Winter, G., Periaux, J., Galan, M., and Cuesta, P., editors, *Genetic Algorithms in Engineering and Computer Science*. John Wiley & Sons Ltd.
- Moriarty, D. E., and Miikkulainen, R. (1996a). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32.
- Moriarty, D. E., and Miikkulainen, R. (1996b). Evolving obstacle avoidance behavior in a robot arm. Technical Report AI96-243, Department of Computer Sciences, The University of Texas at Austin.
- 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.

- Periaux, J., Sefrioui, M., Stoufflet, B., Mantel, B., and Laporte, E. (1995). Robust genetic algorithms for optimization problems in aerodynamic design. In Winter, G., Periaux, J., Galan, M., and Cuesta, P., editors, *Genetic Algorithms in Engineering and Computer Science*. John Wiley & Sons Ltd.
- Stanley, K., and Miikkulainen, R. (2001). Evolving neural networks through augmenting topologies. Technical Report AI2001-290, Department of Computer Sciences, The University of Texas at Austin.
- 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.
- Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.