Design Issues In Hill Climbing For Neural Network Training

Stephan Chalup **Frederic Maire**
Machine Learning Research Centre, Neurocomputing Laboratory
School of Computing Science, FIT, Queensland University of Technology
2 George Street, Brisbane Qld 4001, Australia
phone: +61-7-3864-2575, fax: +61-7-3864-1801
{chalup,maire}@fit.qut.edu.au

Abstract

Hill climbing algorithms can train neural control systems for adaptive agents. They are an alternative to gradient descent algorithms especially if neural networks with non-layered topology or non-differentiable activation function are used, or if the task is not suitable for backpropagation training. This paper describes three variants of generic hill climbing algorithms which together can train nearly any type of neural network. Variations of the algorithms are analysed and tested on the 5-bit parity data and practical design rules are inferred. A hill climbing algorithm which uses inline search is proposed. Experimental results show that it can accelerate learning. It can be recommended for population based training.

1 Introduction

Since artificial neural networks are able to learn, they can serve as control systems for adaptive agents, see [Sharkey and Heemskerk, 1997]. In most applications these are small standard feed-forward neural networks trained by a gradient descent algorithm. In [Nolfi and Parisi, 1997] a genetic algorithm is used to evolve a population of simple feed-forward neural networks, each controlling the behaviour of a small mobile robot. An earlier example in this direction is [Braitenberg, 1984]. As soon as the agent's control system requires a more sophisticated neural network [Scutt and Damper, 1997] a training with gradient descent algorithms might not always be appropriate or applicable [Bengio *et al.*, 1994] and hill climbing or simulated annealing are worthwhile alternatives. E.g. [Bengio *et al.*, 1994] shows that learning long-term dependencies with gradient descent is difficult and simulated annealing is a better alternative. In [Blair and Sklar, 1998] neural controllers of artificial hockey players are evolved by means of hill climbing in a situation where backpropagation could not be applied straightforwardly. Hill climbing algorithms have the big advantage to be easy to implement for a large variety of applications.

Historically hill climbing first was used for maximisation tasks, but in spite of the name it applies to minimisation tasks as well. Hill climbing and simulated annealing [Kirkpatrick *et al.*, 1983] can be regarded as evolutionary algorithms for populations of size two and have been investigated extensively in the areas of evolutionary computation, see e.g. [Ackley, 1987], [Chen and Flann, 1994] or [Yagiura and Ibaraki, 1996]. A comparison of genetic algorithms for neural network training and backpropagation is reported in [Kitano, 1990] and [Heistermann, 1994]. Despite their simple form hill climbing algorithms can play an important role in current research, see [Pollack and Blair, 1998].

The intention of this study is to give better insight into the behaviour and applicability of hill climbing algorithms for training neural networks. After reviewing some key factors that have influence on the training performance we describe three closely related algorithms, hill climbing, inline hill climbing and simulated annealing. We analyse the behaviour of a population of 40 networks when trained with these algorithms on the 5-bit parity classification task. Several variations of the algorithms are tested and practical rules for designing these algorithms are inferred.

2 Background

The training performance of the neural networks depends on several factors:

- The type of data.
- The neural network architecture.
- The initial distribution of the neural network's weights.
- Other random variables (e.g. the components of the step matrix dNW or the *stepsize*, see section 3) which are part of the training algorithm.
- The type of random number generator or the computer/compiler which is used.

2.1 Random numbers

Hill climbing uses random numbers at least at three steps (see pseudo code in table 1):

1. Generation of the initial weight matrix W.
2. Selection of a random number C which determines the variable *stepsize*.
3. Calculation of the step matrix dNW whose components all are new random numbers C.

The simulated annealing algorithm requires a fourth random variable U to decide if a new step is accepted or not, see table 3.

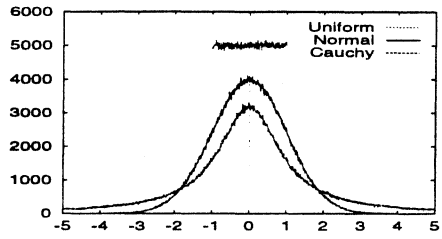

Figure 1: Histograms of 1,000,000 pseudo random numbers for each distribution (Uniform, Normal and Cauchy, from the top to the bottom) accumulated in 2000 bars.

We have used the algorithms 'ran1' and 'gasdev' from [Press *et al.*, 1995] to be independent of a compiler or machine specific random number generator. They calculate pseudo random numbers which are uniformly or normally distributed, respectively. If $X, Y \in N(0,1)$ are two normally distributed random variables with 0 mean and standard deviation 1, a Cauchy distributed random variable is obtained by taking the quotient X/Y with $Y \neq 0$. The Cauchy distribution has fatter tails than the normal distribution, see figure 1. Uniform random variables are taken from the interval (-1,1) and are denoted by $U \in U(-1,1)$. Normal distributed random variables have 0 mean and standard deviation 1 and are denoted by $N \in N(0,1)$. Derived Cauchy distributed random variables are denoted by $C \in C(0,1)$. It took at least double the time to generate N compared to U and again more than double the time to calculate C.

2.2 The networks and the data

A fully connected feed-forward architecture with two hidden units was used which had all possible shortcut and forward connections, see figure 2. This architecture has 18 connections, i.e. 6 more than the standard feed-forward network. All non-input units had a bias and a sigmoidal activation function.

The training data consisted of the 32 patterns of the 5-bit parity task which is a standard benchmarking set [Blake *et al.*, 1998]. It is a medium difficult task

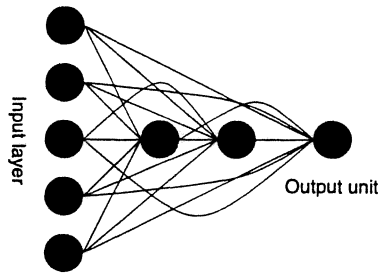

Figure 2: Feed-forward network with two hidden units and all additional shortcut connections.

for neural network training [Stork and Allen, 1992] and it was just the right challenge for the hill climbing algorithm to show its capabilities.

3 The training algorithms

Three algorithms are investigated which all have the structure of a *hill climbing* (HC) algorithm, see table 1. $W = (w_{ij})$ is the matrix of the neural network's weights. Each weight w_{ij} connects unit j to unit i. NW is a new weight matrix which is obtained by adding a step matrix dNW to W. The step matrix consists by default of Cauchy distributed random numbers C whose standard deviation is the parameter *stepsize*. The neural networks are trained in batch mode where the performance is measured by the root mean squared error (*rmse*) over the training set. The aim of the learning process is to minimise the *rmse* within a maximum number of epochs which is bounded by the factor nEpochs.

A delicate issue in the design of the algorithms is the way the step size and the steps (dNW) are determined. Following experience from previous studies [Chalup and Maire, 1999] and to decrease the number of parameters we decided on a formula for the step size which is adapted by the *rmse*:

$$stepsize := C * (sf * MIN(rmse, 1) + sl)$$

C is by default a Cauchy distributed random variable as explained in section 2.1. For comparison a normal and a uniform probability distribution have been used in section 4.4 instead of the Cauchy distribution. The standard deviation of the distribution is determined by the *rmse* and by the two parameters *sl* and *sf*. The step size factor *sf* has been varied, see section 4.4 while the lower step size limit was left constant ($sl = 0.001$) in all experiments.

If the part of the pseudo code which is emphasised in italic in table 1 is substituted by the code in table 2 a variant of HC which we call *inline hill climbing* (IHC) is obtained. IHC continues to use the same step matrix to proceed inline as long as it decreases the *rmse*

```
/* hill climbing algorithm (HC) */
  evaluate network with the initial weights W
  ⟶ rmse;
  WHILE((limit ≤ rmse) and (counter ≤ nEpochs))
     stepsize := C * (sf * MIN(rmse,1) + sl)
     FOR all weights of the neural net
        dNW := C * stepsize          /* step matrix */
        NW := W + dNW
     end FOR
     evaluate network with NW
     ⟶ rmse_new
     Δ := rmse_new - rmse
     IF(Δ < 0)
        W := NW          /* weight update */
        rmse := rmse_new
     end IF
     counter := counter + 1
  end WHILE
  return rmse_new
```

Table 1: Hill Climbing (HC).

```
WHILE(Δ < 0 and il_steps ≤ il_max)
    W := NW
    rmse := rmse_new
    NW := W + dNW
    evaluate network with NW
    ⟶ rmse_new
    il_steps := il_steps + 1
    counter := counter + 1
end WHILE
```

Table 2: Inline Hill Climbing (IHC). Substitute in table 1.

but at most for il_max steps. IHC with $il_max = 1$ is equivalent to HC.

If the code of table 3 is substituted in table 1 the algorithm turns into a *simulated annealing* (SA) algorithm [Kirkpatrick *et al.*, 1983], [Dracopoulos, 1997], [Ingber, 1993]. We regard SA as a hill climbing algorithm which sometimes allows backward steps. The temperature annealing schedule is determined by an adaptive formula which depends on the $rmse$ or the epoch counter and on two parameters $sa1$ and $sa2$. SA with $sa_temp = 0$ is equivalent to HC.

4 Experiments

We report results from a comparative study of HC, IHC and SA and from experiments with HC in which one type of parameter was varied while the other parameters were left unchanged. The results which are displayed in table 4 and figure 3 are part of a larger investigation [Chalup and Maire, 1999]. They reflect general tendencies and typical behaviour of the algorithms when trained on the 5-bit parity task. A different task might need a different parameter tuning.

```
sa_temp = MIN(rmse*sa1, (1-counter/nEpochs))*sa2))
IF(Δ < 0 or U < 1/(1+exp(Δ/sa_temp)))
   W := NW
   rmse := rmse_new
end IF
```

Table 3: Simulated Annealing (SA). Substitute in table 1.

4.1 Fixed parameters

- The network architecture with two hidden units was kept constant, see figure 2.
- After some initial tests with population sizes of 10, 20, ... ,100 networks we decided to use a constant population size of 40 networks. A comparison of several populations of size 40 and larger behaved essentially the same while results from populations with sizes smaller than 20 had a high variance.
- By default the weights were initialised with random numbers from a normal distribution.
- Each network was trained for 250,000 epochs unless the stopping condition $rmse < 0.001$ was met earlier.
- The average of the 40 learning curves ($rmse$) of the population of networks has error bars, see figure 3. They indicate the confidence intervals for the mean of a normal distribution. The confidence level is set to 95%.

4.2 Variable parameters

- Different values for sf have been tested while $sl = 0.001$ was left constant.
- The Cauchy distributed random variables C in tables 1 to 3 can be replaced by uniformly distributed random variables U or normally distributed random variables N.
- The weights were initialised using normally distributed random numbers with 0 mean. Different standard deviations have been tested.

4.3 Comparison of HC, IHC and SA

In a comparative experiment IHC performed best of all three algorithms, see the first line in figure 3. IHC produced the fastest network and the average learning curve was significantly below the learning curve of the standard HC. Always $il_max = 10$ was used. Tests with other parameter settings [Chalup and Maire, 1999] supported

Algo-rithm	Distribution used for initial weights	step-size	step-matrix	sf	Nets with rmse $\leq$ 0.001 Epoch of earliest	Median of them	Numb. of	After 250,000 epochs Nets with 100% acc.	Average accuracy of the population
HC	N(0,0.5)	C	C	0.001	28040	40013	5	5	93% (0.072 std)
IHC	N(0,0.5)	C	C	0.001	25751	38043	14	14	95% (0.047 std)
SA	N(0,0.5)	C	C	0.001	160275	160275	1	1	82% (0.161 std)
HC	N(0,1.0)	C	C	0.001	29527	38455	3	3	92% (0.074 std)
HC	N(0,1.0)	N	N	0.001	194106	195016	2	8	93% (0.079 std)
HC	N(0,1.0)	U	U	0.001	-	-	0	1	92% (0.067 std)
HC	N(0,2.0)	C	C	0.001	26091	35137	7	7	90% (0.117 std)
IHC	N(0,2.0)	C	C	0.001	25229	31336	7	7	93% (0.060 std)
HC	N(0,1.0)	C	C	0.004	28117	31371	6	6	93% (0.073 std)

Table 4: Overview of the experimental results (Corresponding graphs are in figure 3).

IHC's advantage but the differences were not always significant.

4.4 Variations of the distributions for stepsize and step matrix and the stepsize factor sf

Three different distributions, a Cauchy (C), a normal (N) and a uniform distribution (U), see figure 1, were used for generating both *stepsize* and step matrix *dNW* in HC.

Experimental results with the same distribution for both *stepzise* and step matrix *dNW* are displayed in lines 4-6 of table 4 and in the middle lines in figure 3. The Cauchy distribution is a clear winner. But this example emphasises the differences. If *sf* is increased the picture can change and combinations of different distributions for *stepsize* and the step matrix are candidates for good results, too [Chalup and Maire, 1999].

The stepsize factor *sf* changes the standard deviation of the distributions. On average the networks converged faster when the stepsize factor *sf* was increased from 0.001 to 0.004, compare lines 4 and 9 of table 4.

4.5 The initial weights

HC is sensitive to the choice of the range of the initial weights. The first column in figure 3 shows results from training with weights initialised from a normal distribution with 0 mean and standard deviations 0.5, 1.0 and 2.0. Faster converging networks were obtained when the standard deviation of the initial weight distribution was increased.

5 Design issues

We observed that hill climbing algorithms can converge fast but they also can get stuck easily without converging. The simulated annealing algorithm could not in our experiments compete with the hill climbing algorithm. HC and IHC were faster than SA, but it can be shown [Aarts and Korst, 1989] that SA asymptotically always converges to a global minimum. An appropriate setting of the parameters is crucial. SA will be an advantage on other training data or networks where small local minima are an issue. For each task a suitable hill climbing algorithm can be adapted.

During hill climbing training often a smaller step size can increase the probability to sample a successful direction but at the same time it slows down the process of convergence. The IHC algorithm resolves this dilemma. Small steps are tested to find a good direction. Then the same small step is made quickly several times in sequence which has the effect of a big step.

There are several strategies for using the parameter *il_max* in IHC. One of them is to set *il_max* to a relatively low value to stop most of the inline walks before they reach an edge position where no further progress inline is possible. The earlier positions are assumed to be better starting points for a new search than the edge points.

Another strategy is to set *il_max* medium high to limit inline exploitation on long ridges in favour of earlier exploration of better directions.

On other tasks hill climbing strategies can be completely different. One example is described in [Pollack and Blair, 1998] where only a fraction of a successful step is used to update the weights.

Since hillclimbing is sensitive to the range of the initial weights it is worth to test initial weight distributions with higher standard deviations instead of initialising the weights always very close to zero.

Hill climbing uses random numbers at several steps. Hence the choice of the random number distributions is a design issue. Results of this study show advantages of the Cauchy distribution.

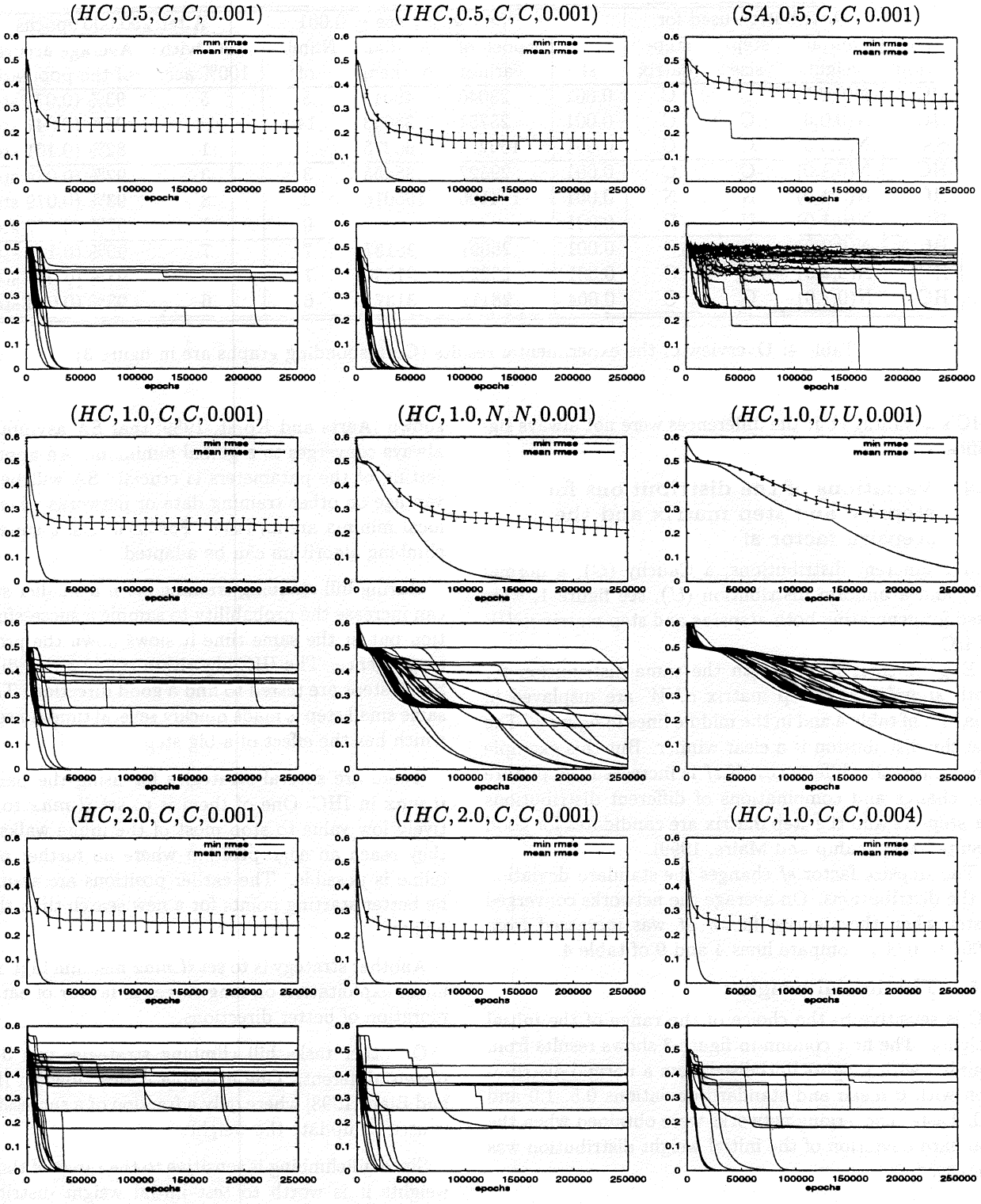

Figure 3: Results of nine experiments, each of them displayed in two graphs. The upper graph shows the learning curve of the "fastest network" (*min rmse*) and the average learning curve of the population (*mean rmse*). The x-coordinate is the number of epochs and the y-coordinate is the *rmse*. The error bars indicate the confidence intervals for the mean of a normal distribution for a confidence level of 95%. The lower graph is the superposition of all 40 learning curves. Each experiment is labelled by a vector which specifies the following parameters *(type of algorithm, standard deviation of the initial weight distribution, type of distribution used for the stepsize, type of distribution used for the stepmatrix, sf)*.

6 Conclusion

The algorithms which we have described in this paper are easy to implement and they are able to train almost any neural network structure. We have proposed the *inline hill climbing algorithm* for training neural networks. It confirmed our geometric intuition and performed best in experiments on the 5-bit parity task. We recommend inline hill climbing for population based training where one fast solution is favoured over a high average fitness of the population. The 5-bit parity data is a medium difficult standard task and offered good possibilities to observe the algorithms' behaviour. Hill climbing is flexible and often an alternative to gradient descent algorithms.

Acknowledgements

Thanks to Alan Blair, Lawrence Buckingham and Ross Hayward for helpful comments and to all colleagues at the MLRC and at the FoD Machine Intelligence Workshop at Mullumbimbi which participated in supporting discussions. The first author is supported by the Machine Learning Research Centre.

References

[Aarts and Korst, 1989] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley, Chichester, 1989.

[Ackley, 1987] D. H. Ackley. *A Connectionist Machine for Genetic Hillclimbing.* Kluver Academic Publishers, Boston, 1987.

[Bengio *et al.*, 1994] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[Blair and Sklar, 1998] A. Blair and E. Sklar. The evolution of subtle manoeuvres in simulated hockey. In *Fifth Conference on Simulation of Adaptive Behaviour, Zurich*, pages 280–285, 1998.

[Blake *et al.*, 1998] C. Blake, E. Keogh, and C.J. Merz. UCI repository of machine learning databases, 1998.

[Braitenberg, 1984] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology.* The MIT Press, Cambridge, MA, 1984.

[Browne, 1997] A. Browne. *Neural Network Perspectives On Cognition And Adaptive Robotics.* Institute of Physics Publishing, 1997.

[Chalup and Maire, 1999] S. Chalup and F. Maire. A study on hillcliming algorithms for neural network training. Technical report, Machine Learning Research Centre, FIT, Queensland University of Technology, Brisbane, Australia, 1999. in preparation.

[Chen and Flann, 1994] I. Chen and N. S. Flann. Parallel simulated annealing and genetic algorithms: a space of hybrid methods. Number 866 in Lecture Notes in Computer Science, pages 428–438. Springer-Verlag, 1994.

[Dracopoulos, 1997] D. C. Dracopoulos. *Evolutionary Learning Algorithms for Neural Adaptive Control.* Springer-Verlag, 1997.

[Heistermann, 1994] J. Heistermann. Different learning algorithms for neural networks - a comparative study. Number 866 in Lecture Notes in Computer Science, pages 386–396. Springer-Verlag, 1994.

[Ingber, 1993] L. Ingber. Simulated annealing: Practice versus theory. *Mathl. Comput. Modelling*, 18(11):29–57, 1993.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 222:671–680, 1983.

[Kitano, 1990] H. Kitano. Empirical studies on the speed of convergence of neural network training using genetic algorithms. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pages 789–795. MIT Press, Cambridge, 1990.

[Nolfi and Parisi, 1997] S. Nolfi and D. Parisi. Learning to adapt to changing environments in evolving neural networks. *Adaptive Behaviour*, 5(1):75–98, 1997.

[Pollack and Blair, 1998] J. B. Pollack and A. Blair. Coevolution in the successful learning of backgammon strategy. *Machine Learning*, 32:225–240, 1998.

[Press *et al.*, 1995] W. H. Press, S. A. Teukolsky, Vetterling W. T., and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, second edition, 1995.

[Scutt and Damper, 1997] T. W. Scutt and R. I. Damper. *Designing a nervous system for an adaptive mobile robot*, chapter 11. In Browne [1997], 1997.

[Sharkey and Heemskerk, 1997] N. E. Sharkey and J. N. H. Heemskerk. *The Neural Mind and the Robot*, chapter 9, pages 169–194. In Browne [1997], 1997.

[Stork and Allen, 1992] D. G. Stork and J. D. Allen. How to solve the n-bit parity problem with two hidden units. *Neural Networks*, 5:923–926, 1992.

[Yagiura and Ibaraki, 1996] M. Yagiura and T. Ibaraki. Metaheuristics as robust and simple optimization tools. In *Proceedings 1996 IEEE Conference on Evolutionary Computation*, pages 541–546, 1996.