Observing, Understanding, and Controlling the Genetic Method

on a Neural Network Application.

870574
Kim Kortermand Hansen
Degree of MSc in Computer Science
(Cand. Scient. i Datalogi)

Department of Computer Science
University of Aarhus

July 1994

Abstract

The purpose of this thesis is to investigate the underlying dynamics of a state of the art genetic algorithm. The population is cluster analyzed, grouping together functionally similar individuals. A snapshot of the sizes of clusters show the diversity of the population at any time step, and in particular detects convergence. The basic idea of the clustering can be used to monitor the population diversity in any genetic algorithm, using any 'closeness' concept. Further investigations are suggested and it is described how cluster analysis can be used to control diversity of the population automatically.

Research on potential interactions between connectionist learning systems and evolutionary search procedures, has attracted a lot of attention recently. Evolutionary artificial neural networks can be considered as the combination of artificial neural networks and evolutionary search procedures.

The focus of attention is the use of evolutionary methods to do supervised learning on feedforward neural networks through adaption of topology *and* weights. When it comes to optimization of topology *or* weights, some work have been done lately, as reported by [Yao 1992] and [Schaffer et al 1992]. However, few have tried to evolve both topology and weights, simultaneously. The genetic algorithm used in this study was constructed using carefully chosen methods reported in the litterature. A genetic algorithm for optimizing the topology and weights in any 3-layer neural network, with respect to a given training set, i.e. a set of input and output patterns is developed and investigated. First, the foundations of genetic algorithm and artificial neural networks, then the current state in genetic optimization of weights and architecture is reviewed.

In the experimental work, each genom (individual) in the genetic soup represents a 3-layer feedforward neural network in terms of all weights associated with each hidden node, i.e. a genom consists of the weights attached to each hidden node, including the bias. Biases to the output nodes is also represented on the genom. To overcome the limitations of the simple, classic stringrepresentation, a recombination operator, due to [Radcliffe 1993] is used. The recombination operator regard an individual as a set of hidden nodes with weights. One advantage of this operator is immunity to permutation of the hidden nodes.

The experiments show that the recombination operator is unlikely to preserve the functionality from the parents to the child. The experiments also show that the dynamic behaviour of the population, i.e. clustering of individuals, has great influence on the success of the search. The investigations confirm suppositions about premature convergence, but also show that premature convergence is escapeable in certain cases. The Cluster Analysis Tool proves useful for monitoring and explaining the evolution process.

Acknowledgements

I would like to thank my supervisor Brian H. Mayoh for helpful guidance, encourage and hospitality, Brian L. Karlsen for careful proofreading of this text, the staff at the Computer Science Department for maintaining the workstations and for hints about the development tools, and especially Bitten for making everything so easy.

Contents

I Introduction **8**

1 The genetic search method **9**
1.1 Foundation of Genetic Algorithms 9
1.2 Genetic algorithm as a search method 10
1.3 Pheno space and Geno space 11
1.4 Domains 12
1.4.1 Fitness functions are domain evaluators 13
1.4.2 Designing a fitness function 14
1.5 Genetic Operators 15
1.5.1 Combination 15
1.5.2 Mutation 15
1.6 The schemata concept 16
1.6.1 The theorem 17
1.7 Lifecycle Administration 18
1.7.1 Generational Replacement 18
1.7.2 Steady State 20
1.7.3 Generational Replacement versus Steady State 20
1.8 Premature Convergence 23

2 Artificial Neural Networks **25**
2.1 Physiology of the artificial neural network 25
2.1.1 Formal definition of neural nets 26
2.2 Feed-Forward Networks 28
2.3 An equivalence relation on the space of networks 30
2.4 Visualization of neural computation 30
2.4.1 Node computation 30
2.5 Internal state of hidden layer 34
2.6 The neural model 35

3 Genetics and Neural Nets **36**
3.1 A word on generalisation capability 41
3.2 The permutation problem 41

4 A new paradigm **44**
4.1 Problems with the genetic method 44
4.2 Modern versus Classic genetic encoding 45
4.3 Formae 47
4.3.1 An illustrative example 49
4.3.2 Criteria for recombination 52
4.4 Use any knowledge of the search space 56

5 NN Recombination **57**
5.1 Formae for networks 57
5.2 Formae for multisets 58
5.3 Example of recombination 61

II The Cluster Analysis Tool **62**

6 Population dynamics analysis **63**
6.1 Introduction to population analysis 63

7 Cluster Analysis Tool **66**
7.1 Formal definition of cluster 68
7.2 Clusters can be defined arbitrarily 69
7.3 Cluster Analysis Bookkeeping algorithm 71
7.4 Several Cluster relations simultaneously 72
7.5 Cluster visualization 72

III The Cluster concept in the future **75**

8 Population Diversity **76**
8.1 Any convergence is premature 79
8.2 Artificial Diversity 80
8.2.1 The algorithm 81
8.2.2 Future extensions to Artificial Diversity 82
8.2.3 Cluster Analysis and Artificial Diversity 82

IV Experiments and Results **83**

9 Visions and limitations **84**
9.1 Setup 85
9.1.1 Choice of genetic algorithm 85
9.1.2 Modern Representation - Phenospace only. 85
9.1.3 Evaluation of fitness 86

9.1.4 Population initialization 86
9.1.5 Recombination explores weighted topologies 87
9.1.6 Mutation explores weights 88

10 Experiments **90**
10.1 8-3-8 multiplexer experiments 90
10.2 Two-Spiral experiment 91

11 Results **94**
11.1 Presentation 96

12 Interpretation **106**

V Discussion and Conclusion **111**

13 Discussion **112**
13.1 Training results 112
13.2 Simultaneous optimisation of topology and weights 113
13.3 Population Dynamics 117
13.3.1 Why is clustering more prominent in the small population ? 117
13.3.2 Escape of Premature convergence 118

14 Conclusion **120**
14.1 Experiments 120
14.1.1 When is search most succesful? 121
14.1.2 When is search most robust? 121
14.2 CATs are widely applicable and useful 122
14.3 Is there a long way yet before the vision comes true? 122

A Using the simulator **123**

Bibliography **130**

List of Figures

1.1 Spinner with 36 arms and weights f1,f2,f3,f4,... 19
1.2 Spinner with 1 arm and weights w1,w2,w3,w4,... 23
1.3 Local minimum and operators on individuals 24

2.1 A fully-connected 3-layer, feed-forward neural network. 28
2.2 The tanh and sigmoid functions. 29
2.3 Interpretation of the output activation. 30
2.4 Surface of dotproduct $5a_1 + 5a_2 - 2$. 31
2.5 Sigmoid activation. Arrows show input (0,0),(0,1),(1,0) and (1,1). 32
2.6 Binary interpretation. 33

3.1 A wireframe representation 39
3.2 Parent networks AB and CD. Possible crossovers AD and CB. . 42

4.1 Space of boats 49
4.2 Three non-seperable formae in S 55

5.1 Above: Parent networks. Below: A possible recombination obeying formae for multisets. 61

7.1 Search space S, with 3 individuals, $N1$, $N2$, and $N3$. 68
7.2 2 examples of S with an cluster equivalence relation with only 4 classes. 4 individuals are within one of the classes. 70
7.3 Example of S with an cluster equivalence relation with many classes. 4 individuals are within four of the classes. 70
7.4 Cluster Analysis Initialisation algorithm 71
7.5 Cluster Analysis Bookkeeping algorithm 71
7.6 Explanatory Cluster graph. 73

8.1 Artificial Cluster Diversity algorithm 81

9.1 Weight initialisation function 87
9.2 The weights mutated by the mutation operators. 88

10.1 The 8-3-8 topology. 90
10.2 The topology used for the Two-Spiral task. 91

10.3 The Two-Spiral input patterns. 92

11.1 The RMSE of the best network during a Run. 95
11.2 %Correct and RMSE of Exp 1. 97
11.3 %Correct and RMSE of Exp 2. 100
11.4 %Correct and RMSE of Exp 3. 102
11.5 %Correct and RMSE of Exp 4. 104

13.1 The Milky Way Galaxy and a 3 level adaptive system. 113
13.2 Cluster sizes and RMSE of Exp1, Run 6. 119
13.3 Cluster sizes and RMSE of Exp1, Run 7. 119

14.1 RMSE and Cluster sizes of Exp 1, Run 4 and 8 121

Part I

Introduction

Chapter 1

The genetic search method

In this chapter the general idea behind the genetic method is outlined. The formal notation used throughout the thesis is introduced. Hollands classic genetic algorithm and its operators are reviewed, followed by later extensions and improvements. The Schemata Theorem is outlined. Appearence of more exotic representations and operators is a new trend in genetic algorithmics which has appeared due to the static limitations in standard bitstring representation. The concepts Modern evolution and Classic evolution are defined to capture the difference between the classic genetic algorithm and the new trend proposed. The concepts give means to describe the new algorihms in the genetic community.

1.1 Foundation of Genetic Algorithms

Consider the task of optimizing a set of solutions to some problem. An exhaustive search for the optimum, can take exponential time in terms of the parameters of the domain. Therefore we must be satisfied with "good" solutions. If some knowledge of the domain is available, it can be used as heuristics guiding the search. If not, or we don't bother implementing heuristics, genetic algorithms offer intriguing possibilities for general purpose adaptive search algoritmhs in artificial intelligence. If good solutions are correlated they can be combined to form other good or better solutions. This is the paradigm of genetic algorithms.

Genetic algorithms, first introduced by [Holland 1975], have been applied to a variety of problems. Search based on evolutionary models have been tried before Holland, though. These models were based on mutation and natural selection, and were not notably succesful. The principal difference of Hollands approach was the incorporation of a 'crossover' operator to mimic the effect of sexual reproduction.

The basic idea of a GA is illustrated.

Initialize:
Create population $\mathcal{B}$ of N members.
Evaluate members $\eta \in \mathcal{B}$.
Lifecycle Administration:
Select parents.
Create child(ren) via genetic operators.
Evaluate child(ren).
Substitute child(ren) in population $\mathcal{B}$.

Several issues concern the construction of a genetic algorithm: Lifecycle adminstration, selection, genetic operators, substitution, parameters and more.

1.2 Genetic algorithm as a search method

The following is an exhaustive overview of search paradigms, from this thesis's point of view. A similar overview can be found in [Goldberg 1989], section "Robustness of traditional optimisation and search methods". The list is ordered in, increasing robustness to size and complexity of search space. The list is meant as background and motivation for the genetic method and the need for domain specific representation. The following list can be considered search paradigms where each paradigm is intended to overcome the weakness(es) of the previous. The term "efficiency" can be considered the probability of finding a point in space within a certain fitness, within a certain maximum of timesteps.

Random search is repeatedly, randomly generated points in space which is tested for goodness. Highly ineffecient.

Guided search A single search point is randomly generated and guided around. Examples are greedy hillclimbing and simulated annealing. The fact that this search method is based on the landscape in immediate neighbourhood of a single search point, in a huge, complex space, makes the search ineffecient. Guided search tends to get stuck in local minima.

Parallel guided search is search with several points in parallel, or equivalently, repeated guided search. Examples are (again) greedy hillclimbing and simulated annealing. Though this does increase probability of succes, it is still based on the immediate neighbourhood of the search points and is thus unlikely to scale with problem size.

Genetic search To overcome the dependence on the immediate neighbourhood Hollands implicit parallism concept helps [Goldberg 1989]. Any search-point(bitstring) is member of several schemata. We say the string is an instance of the schemata. Highly fit, short-defining-length schemata (building blocks) are propagated generation to generation by giving exponentially increasing samples to the, on average, best schemata, under the assumption that most schemata does not contain both high and low fitness individuals. Implicit parallism means parallel search of initially $O(n^3)$ schemata, in a population of size n, which certainly increases efficiency. Each schema constitutes entire hyperplanes in search space. However, when a low-fitness schemata is exterminated from the population, it implies exclusion of several points in space, as the amount of parallism decreases as more and more individuals belong to the same schemata.

Unfortunately the genetic search method still suffers from a few "child diseases", which is 'domain independent combination' and 'premature convergence'. These problems are important, and part of the inspiration of the analysis tool developed in this work. They are treated later.

1.3 Pheno space and Geno space

It has been common to think of two different search spaces, the geno space and the pheno space. In chapter 4 it will be argued that only a single space is necessary, the pheno space, because operators can be designed to manipulate phenotypes directly.

- The structure space or *pheno space* is the search space S, of possible solutions to the problem of concern. The representation of points should be domain inspired. The solutions are often called *individuals*. An individual has certain *features*, which are object to combination.

- The representation space or *geno space* is the coding space C. It should be a "simple" space in which genetic search is possible or more efficient than in structure space. The elements are often called *chromosomes*. A chromosome consist of *alleles*.

It is convenient to introduce a *coding function* from the structure space to the representation space $\rho : S \longrightarrow C$, hopefully observing:

- Every possible individual corresponds to some element in representation space, thus ρ injective. The inverse function we denote $g : C \longrightarrow S$. Decoding a gene into a structure we call 'interpretation' of the gene. In many real applications this may be a noisy function. This can be overcome by reevaluating several times and averaging.

- C is not redundant with respect to ρ, thus ρ bijective. In some cases it can be difficult (impossible) to provide a decoding function with domain C such that $\rho(c) \in S, c \in C$, in which case we call c invalid and restrict to $C' = \{c \in C | \rho(c) \in S\}$. This amounts to an implicit requirement on the representation that all chromosomes represent valid solutions to the problem.

If a one-to-one mapping $\rho : C \longrightarrow S$ can be given, then any operator

$$O_C : C^i \longrightarrow C$$

has its equivalent in S,

$$O_S : S^i \longrightarrow S,$$

where i is whatever the number of arguments the operator takes. As will be apparent, the distinction between S and C is not necessary, and thereby the role of the coding function. Whether both S and C are used or only S, the points sampled to the population are called individuals.

1.4 Domains

This section formalizes the description of the domains which is used in this thesis. Subject of optimization is often: a number of parameters from some space(s), or a set or multiset of parameters, of fixed or variabel size, from the same space. Geno space and pheno space is considered identical here.

- Fixed number of parameters. A chromosome η has l alleles taking values $(\eta_1, ..., \eta_l)$ drawn from sets $(\mathcal{G}_1, ..., \mathcal{G}_l)$ respectively, and η is a member of the search space $S = C = \mathcal{G}_1 \times ... \times \mathcal{G}_l$. This is called a string representation of fixed length.

- Variable sized set. An individual η represents a subset of the set $\mathcal{G}$. The search space is $S = I\!P(\mathcal{G})$ - the power set of $\mathcal{G}$.

- Fixed sized set. An individual η represents a subset of the set $\mathcal{G}$ of size l. The search space S is $\{\eta \subset \mathcal{G} : |\eta| = l\}$.

- Variable and fixed sized multiset. The extension to multisets is straight forward. Let $I\!P_m(\mathcal{G})$ be the *multipower* set of $\mathcal{G}$ and define the *multiplicity* function

  $$m : \mathcal{G} \times I\!P_m(\mathcal{G}) \longrightarrow I\!N$$

  so that $m(x, \eta)$ is the number of copies of x in the multiset η. The search space is $\{\eta \subset I\!P_m(\mathcal{G}) : |\eta| = l\}$ when fixed size. In the case of variabel sized multiset the "l constraint" is relaxed.

1.4.1 Fitness functions are domain evaluators

The object of the adaptive search is to find an individual which performs well in terms of a measure of performance:

$$u : S \longrightarrow I\!R^+ \tag{1.1}$$

In the litterature fitness, performance, evaluation, utility, and goodness are often used interchangeably. *Fitness* will be favored in this paper, with a few exceptions. For simplicity, fitness is assumed to be positive and optimization means minimization here.

Fitnessfunctions can be divided into two conceptually different kinds:

1. Given a real function of some domain, the optimum is sought. The GA is used to optimize that function. This is often the case in mathematics or perhaps statistics. Examples are De Jong's testbed [De Jong 1975].
2. Given a task to be solved, like graphs problems, cybernetics, training/designing neural networks, etc. a good solution is sought. In such a case, the fitness function must be designed carefully such that the global/local optima corresponds to solutions with desired properties.

The last kind is used when searching for neural nets using genetic algorithms. Such fitness functions are inherently difficult to search. The search for Evolutionary Artificial Neural Networks(EANN) is haunted by [Yao 1992]:

- The surface is infinitely large since the number of possible nodes and connections is unbounded.
- The surface is non-differentiable since changes in the number of nodes or connections is discrete (...) .
- The surface is deceptive since EANNs with similar architechtures may have dramatically different information processing abilities and performances.
- The surface is multimodal since EANNs with quite different architectures can have very similar capabilities.

The domain evaluator contains the core problem of concern. This function is the landscape of error, on the search space S. If this is a simple function some conventional algorithmic approach would probably perform better than using a genetic algorithm. Some examples of fitness functions are:

1. Given a function $f : I\!R^n \longrightarrow I\!R$ to be optimized, choose $S = I\!R^n$ and f as the evaluation function.

2. Given a set of pairs $(a_i, b_i) \in A \times B$. In order to find a continuous function $f : A \longrightarrow B$, usually under some syntactic restrictions, such that $f(a_i)$ approximates b_i with respect to some measure of approximation $\phi : (A \longrightarrow B) \longrightarrow I\!R$, choose S as the set of functions allowed. This is the field of genetic programming. [Koza 1991] use genetic programming to build expressions similar to, but far more flexible than, any multilayer neural networks.

3. Given again a set of pairs $(a_i, b_i) \in A \times B$. The task is to find a neural network $\mathcal{N} : A \longrightarrow B$. This is a special case of 2., but neural networks have the advantage of:

   - learning capability.
   - the ability to generalize well, in terms of other sets (a'_i, b'_i).
   - neural networks have a visual interpretation.
   - many documented experiments exists.

   The choice of S could include some representation of topologi and weights. Genetic search of weights can be improved by applying gradient descent algorithms, in which case parameters of the algorithm should be included. Construction of the evaluation function could include the error of the net on training or test set. To enhance ability of generalisation, include terms of network complexity.

1.4.2 Designing a fitness function

When the desired solution must have more than one property, each measured with different terms $f_1, ..., f_n$ these must be combined into one real number expressing the relationship between desired properties:

$$f : I\!R^n \longrightarrow I\!R,$$

such that the desired fitnes is expressed by $f(f_1, ..., f_n)$. For example f can be a linear function so each property is weighted, but this demands some experimentation to determine useable coefficients. If all coefficients are chosen positive, then the landscape of a linear function has the convenient property that optimum is in "downhill direction".

1.5 Genetic Operators

A central idea in any genetic algorithm is the combination of existing solutions to form new solutions inheriting alleles or features, thus exploring new combinations. But this does not introduce new alleles or features, so mutation is also a part of most genetic algorithms.

1.5.1 Combination

The purpose of *combination* is to combine features of individuals to form new individual(s). It is hoped that the features collected from the parents are the better features. It can not be known if the collected features are better than the remaining. Therefore it might be useful to form child number two consisting of the remaining features. If one child has better fitness, the other child is considered genetic garbage, though not necessarily worse than other members.

In the case of a stringrepresentation, wellknown crossovers include 1pt, 2pt, and Uniform crossover. In the case of real alleles, $\mathcal{G}_i \subseteq I\!R$, [Eshelman & Schaffer] has introduced the BLX crossover. A set representation is appropriate for graph domains, like neural nets optimization and the Traveling Salesman Problem(TSP). [Goldberg & Lingle 1985] ([Radcliffe 1993]'s reference) have developed Partially-mapped crossover for the TSP, which is immune to permutation of cities.

It will be convenient to regard a recombination operator as any function which, given some pair of members together with some "control parameter", produces a unique child. This formalism allows the stochastic element in recombination to be introduced simply through the choice of control parameter. It will be assumed that the control parameter is selected from a uniform distribution over the control set, though other distributions could be used.

Definition 1.5.1 *Given any (non-empty) set $\mathcal{K}_X$, and a search space S, any function*

$$X : S \times S \times \mathcal{K}_X \longrightarrow S$$

will be said to be a recombination operator *over S, and the set $\mathcal{K}_X$ will be called the* control set *for X.*

1.5.2 Mutation

The 'genetic algorithm' so far described is incomplete. One of the problems that can be encountered is that every instance of some allele for a particular gene may dissappear from the population. Or it was not instantiated in the initial

population. While combination rarely introduces new alleles, the Lifecycle Administration allow deletion from the population, and it could happen that some allele is lost entirely. Once this happens, some portions of the space might become unavailable to the search, depending on the nature of the recombination operator(s).

Mutation is used as a "background operator" occasionally replacing some alleles with others selected at random. This ensures that no allele is ever permanently lost to the search, even if the measured performances of chromosome which use it are low. The aim is to keep the rate of application of mutation as low as consistent with keeping the gene pool well-stocked, for mutation is intrinsically disruptive, causing alleles from chromosomes which have been performing well to be discarded.

Definition 1.5.2 *Given any (non-empty) set $\mathcal{K}_M$, and a search space S, any function*

$$M : S \times \mathcal{K}_M. \longrightarrow S$$

will be said to be a mutation operator *over S, and the set $\mathcal{K}_M$ will be called the* control set *for M.*

1.6 The schemata concept

This section reviews the concept of schemata and the famous fundamental theorem of genetic algorithms.

[Radcliffe 1990] In order to analyse the behaviour of a genetic algorithm it is useful to introduce the concept of schemata:

$$\Xi := \mathcal{G}_1^* \times ... \times \mathcal{G}_l^*,$$

where

$$\mathcal{G}_i^* := \mathcal{G}_i \bigcup \{\Box\}.$$

It is useful to construct from u, defined in 1.1, a measure

$$\mu : \Xi \longrightarrow I\!R^+$$

which gives the utility of a schema as the average utility of all its members:

$$\mu(\xi) := \frac{1}{|\xi|} \sum_{\eta \in \xi} u(\eta),$$

where $|\xi|$ is the number of chromosomes in C which instantiate schema ξ. Noting that $C \subsetneq \Xi$, it is then immediately apparent that $\mu \restriction_C = u$, so that μ can be used to yield the utility of either a schema or a chromosome.

Holland observed that each evaluation of a chromosome, could be regarded as a statistical sampling event which yields information about the utility of *each* of the 2^n schemata that the chromosome instantiates - a phenomenon known as intrinsic parallelism.

This is true only if most schemata contains individuals with correlated fitness. All search techniques other than enumeration make assumptions about correlations between solution: if correlations do not exist, it is impossible to improve on an enumerative search. In the case of genetic algorithms, the implicit assumption is that there is a correlation between the various instances of a schema! A schema creates an equivalence relation partitioning the representation space into chromosomes belonging to the schema and those not. 1pt, 2pt, and Uniform crossover "obey" any schema in important senses.

- Two members of the same schemata allways combines to a child in the same schemata.
- Two members belonging to each their schemata, with non-empty intersection, has the possibility to combine to a child in the intersection.
- A member combined with itself should yield itself.

These are fundamental properties of combination. The requirement of a correlation between the utilities of the various instances of a schema, can now be reformulated as the requirement that combination of members should yield child(ren) with fitness "similar" to the parents!

1.6.1 The theorem

The way in which instances of above average schemata accumulate in the population can be quantified. Given an operator

$$\Gamma_i : C \longrightarrow C,$$

where any other parameters that the operator may take have been omitted for convenience, let p_i^ξ be the probability that the child of a parent instancing ξ is prevented from instancing the schema itself by the application of Γ_i. Formally,

$$p_i^\xi = P(\Gamma_i(\eta) \notin \xi | \eta \in \xi)$$

Using this idea, [Holland 1975] derived the following remarkable result, though not in this form:

Theorem 1.6.1 (The Fundamental Theorem of Genetic Algorithms) *Let $n_\xi(t)$ denote the number of instances of a schema ξ at time-step t in fixed-size population whose dynamics are governed by a lifecycle administration of type B. Let $\mu_\xi(t)$ denote the sample utility for the schema - the mean utility of every $\eta \in \xi$ in the population at time t - and $\mu(t)$ denote the mean utility of the population at this timestep. Moreover, let there be N_0 genetic operators Γ_i, each applied with probability p_i. Then the expected number of instances of the ξ at timestep $t+1$ is governed by:*

$$< n_\xi(t+1) > \geq \; n_\xi(t) \frac{\mu_\xi(t)}{\mu(t)} (1 - \sum_{i=1}^{N_0} p_i p_i^\xi).$$

It is, in fact, easy to prove this and to fill in bounds for p_i^ξ for the standard operators. It says that schemata with above average utility and small disruption probabilities, will have exponential growing number of instantiations in the population. Otherwise exponential decreasing.

1.7 Lifecycle Administration

This section reviews two opposite kinds of Lifecycle Administration, Generational Replacement and Steady State. A selection strategy is described: fitness scaling. And under Steady State Lifecycle Administration four replacement strategies are mentioned.

1.7.1 Generational Replacement

Generational replacement replaces some of, or the entire population with a new population. This is done by repeatedly selecting(with replacement) a member from the old population, according to fitness, and adding that member to the new population. Since selection is biased by fitness, members having fitness value greater than the population average will on average be represented in the population to a greater extent. Likewise, members with below-average fitness will decrease in extension in the population according to the same ratio.

Generational practitioners typically use Baker's (multi-arm spinner selection method) Stochastic Universial Sampling(SUS), without replacement [Baker 1987].

SUS copies exactly the expected number of each member, i.e. proportional to its fitness. Basically, Baker proprosed using a spinner that has as many equally-spaced arms as there are members in the population. According to its fitness value, each member gets a piece of the spinner pie - a weight. The spinner is spun, and each arm then points to a member that will be reproduced in the new population, see figure 1.1.

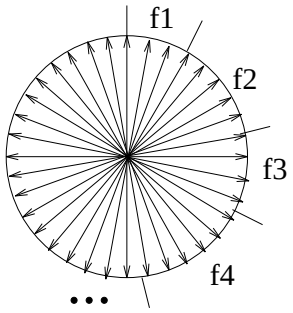


Figure 1.1: Spinner with 36 arms and weights f1,f2,f3,f4,...

This has good statistical properties when the number of arms is approximately the number of members or more. This selection method is universially applicable, for stochastic sampling without replacement.

For the selection of weights [Davis 1991] prefers 'Linear Normalisation': "Order the chromosomes by decreasing evaluation. Create fitnesses that begin with a constant value and decrease linearly." To keep things simple, the weights determinating the probability of selecting an individual are simply whole numbers. The least fit individual has weight 1, incrementing by one, until the fittest gets weight Populationsize.

This is the pseudo algorithm for generational replacement, as first described by [Holland 1975].

Initialize:
Create population $\mathcal{B}(0)$ of k members.
Evaluate members.
Lifecycle Administration:
$\mathcal{B}(t+1)$:= select n members from $\mathcal{B}(t)$.
Apply genetic operators to members in $\mathcal{B}(t+1)$.
Evaluate $\mathcal{B}(t+1)$.

k is a parameter of the algorithm fixed between 2 and size of population.

1.7.2 Steady State

The Steady State(SS) technique replaces only a few members during a generation, typically whatever number is produced by an operator. This is done by selecting an operator from the set of available operators, and selecting the appropriate number of member(s), by fitness.

However, in order to insert the child(ren) into the population, room must be made, demanding a method for selecting a member to be deleted. Most steady-state practitioners do not use random deletion. Instead, they use some other sort of deletion scheme:

Delete least fit Since the least fit member has little chance of being selected for reproduction, it is a good candidate for deletion, says defenders.

Exponential deletion The worst member has some small chance of being deleted, say P_d ($0 < P_d \ll 1$). If not selected, then the next also has P_d chance, and so on. For a population of n, the best member has $(1 - P_d)^{n-1}$ chance of being deleted.

Reverse fitness Each member has a chance of being deleted according to a reverse linear ranking.

Delete parent In the case of 2 or more parents one can be chosen at random or the least fit can be chosen.

This is the pseudo algorithm for Steady State reproduction, as first desribed by [Syswerda 1991].

Initialize:
Create population $\mathcal{B}$ of n members.
Evaluate members.
Lifecycle Administration:
Select an operator from operator set.
Select a number of members from $\mathcal{B}$ acquired by the operator.
Apply genetic operator to the member(s) to create child(ren).
Evaluate child(ren).
Make room for child(ren) in $\mathcal{B}$.
Insert child(ren) into $\mathcal{B}$.

1.7.3 Generational Replacement versus Steady State

According to the schema theorem, the performance of a genetic algorithm is highly dependent on its reproductive behaviour: the number of instantiations of

above average schemata is directly dependent on the reproductive rate of individual population members. In the light of this statement, the first selection/deletion scheme coming to mind is Generational Replacement(GR).

GR replaces whole or part of the entire population. This is done by repeatedly selecting, with replacement, a member from the old population, with probability proportional to fitness, and adding that member to the new population, until size of new population is correct. The genetic operators are applied to the new members. This forms the next generation. Here is a problem. Because each individual is selected with replacement, any individual could be selected any number of times, worst case. This is easily solved by selecting every individual without replacement, and adding a number of copies to the new population proportional to its fitness, scaled such that populationsize remains constant. This copies exactly the expected number of each individual, in accordance with the Schemata Theorem.

Here are two problems. If all individuals have similar fitness, they all gets reproduced exactly once, yielding no progress. Otherwise, if one indiviual have a fitness multiple times the remaining individuals, this elitist will soon dominate the population, restricting further search to a small area of search space. Early in the genetic search, if a few members are considerably better they can easily become domination in the population, and thereby converge on some suboptimum. Further, the selection pressure on a fitness-homogeniuos population will not bias on the better members. This can slow down the search.

Both phenomena can be overcome by ordering the members by their fitness, and give them a fitness according to their number. This is called fitness scaling or rankbased selection: The population is sorted according to fitness and individuals are seleceted according to their rank instead of fitness. The rank is often scaled linearly, exponentially, or other scheme. For simplicity a linear scaling is chosen as ranking method in the experiments of this thesis.

There are several asserted drawbacks with the whole GR concept [Syswerda 1991]:

- even after recombination of the next generation, there might be duplicate individuals, even if the operator ensures each child is not as its parents. Every pair of individuals have to be compared to find duplicates. If found, we could re-recombine them, but then the population must be checked again.

- during recombination of the next generation, individual(s) with very fit properties may very well be created, but these properties are not available to further exploration before the entire population is created.

- GR in its basic form, does not guarantee survival of the fittest.

Therefore [Syswerda 1991] propose the Steady State reproduction scheme, which eliminates the intuitive drawbacks. The Steady State technique replaces only a few members during a generation, usually whatever number is required for an operator. The required members are selected according to their fitness, as with GR. The chosen operator(s) is applied to generate the new member(s). To make room for the new member(s), a deletion method is required. SS allows for convenient control of the population, as will be apparent later. Syswerda has compared the Generational Replacement and Steady State in a simpel, artificial experiment. He used a population of 100 and made experiments compareable by fixing number of created members at 2000.

> "Interestingly, informal testing and comparison of the two approaches indicate that at least for some problems, steadystate GAs do find as good or better solutions in much less time than generational replacement GAs. A careful benchmark of the two approaches are warranted. (...) Users of generational replacement GAs often provide a guarantee that the best member in the current population will be present in the next. This is not necessary with steady-state GAs, since the deletion methods most often used automatically grant elitist status to *all* good members of the population. (...) If a good member is created in a steady-state GA, it is immediately available for use. In generational GAs, a good member is not used until the next generation. (...) Finally steady-state GAs allow us to make another change that significantly improves performance: one can impose the condition that only one copy of any chromosome exist in population at any one time."

In the case of steady-state, he found exponential deletion better than deleting the least fit. Random deletion is worse than either. Syswerda did not try reverse fitness. Notice that multiarm spinner is not necessary, a common variation with just 1 arm suffices here. To each individual corresponds a weight w. Let w be uniformly distributed from 0 to sum of weights. Obviously w "points" at one of the weights and the corresponding individual is chosen, see figure 1.2

Thus if the weight of some individual I is twice the weight of another, then I has twice the probability of being selected. To select another member, if required, the method is repeated, with the restriction not to select same member twice.

Selection of operator is done in the same way. To each operator corresponds a user-specified weight. These weights can remain constant or be automatically adjusted [Davis 1991][Montana & Davis 1989] as evolution proceeds.

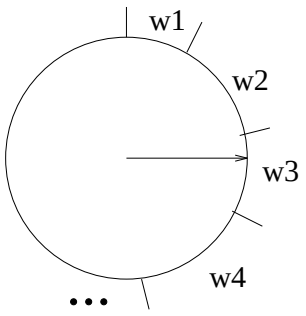


Figure 1.2: Spinner with 1 arm and weights w1,w2,w3,w4,...

1.8 Premature Convergence

GA researchers often experiences varying results on the same experiment. Either the process converges after some time to a satisfiable optimum. Or it converges to some local, unsatisfiable optimum. The latter is known as Premature Convergence, PC. Convergence means (almost) all individuals are similar genotypically or phenotypically. When most of population is concentrated around a local optima, the 'implicit parallelism' is neglected to a small part of the space. Evolution has either stopped or is very difficult. Notice: 'convergence' in the sense that all individuals are similar, not 'convergence' in the sense that some individual(s) is close to a global optimum. This section discusses the phenomena and reviews some attempts to avoid PC.

Recombination operators are intended to combine individuals, creating individuals in the same schemata - exploring new combinations of parents features. Recombination operators tends to produce a solution in some neighbourhood of the parent individuals. Recombination in its basic sense cannot introduce individuals from other schemata than those of the parents. To maintain diversity and explore other schemata something more is needed. Hollands answer was the concept of mutation. Mutation operators are meant to explore parts of the search space which cannot be reached by recombination of present individuals in population.

Mutation operators can be divided in two kinds, those:

- Producing a new individual randomly distributed on some neighbourhood of the parent. It is usually intended to produce an individual randomly distributed from the same schemata as parent.
- Producing a new individual randomly distributed on the entire search space. This is usually done the same way as the initial individuals were created.

To escape convergence by combination is difficult because crossover of two similar individuals often yields a similar child. To escape convergence by mutation is also difficult, because mutation randomly chooses an individual in some neighbourhood of the parent and this is often very similar to the parent or missing important features, see figure 1.3.

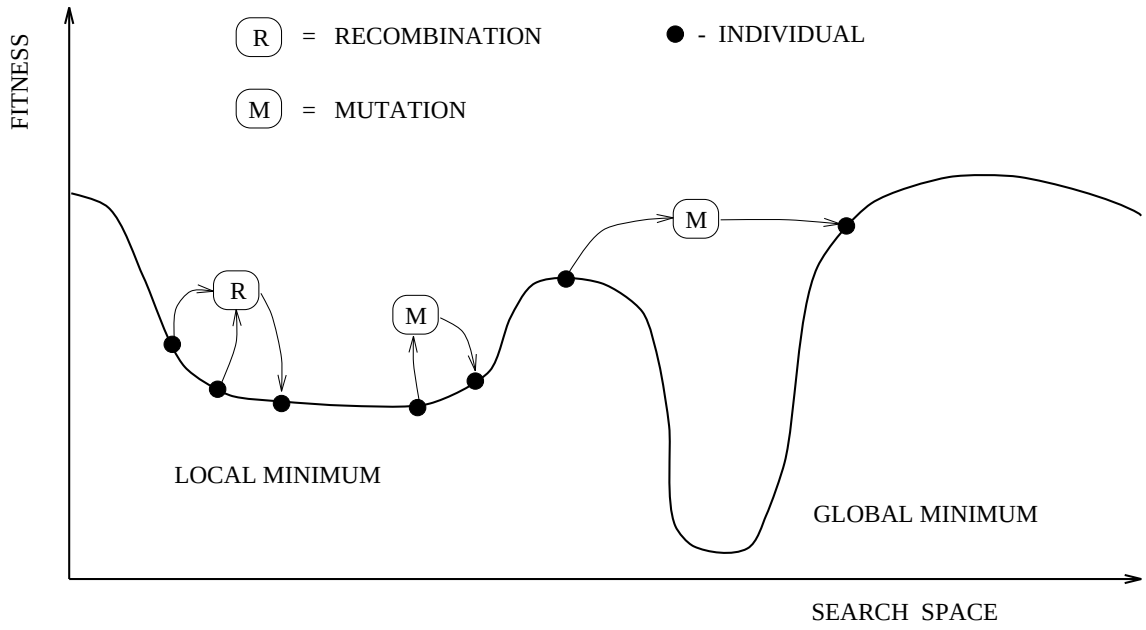


Figure 1.3: Local minimum and operators on individuals

Therefore PC should be avoided in the first place. An algorithm invented by the author is presented in chapter 8.

Chapter 2

Artificial Neural Networks

The field of neural networks is a very broad one, drawing researchers from fields as diverse as physics and neurology, engineering and psychology, biology and computer science. Researchers from these fields often have quite different perceptions of what a neural network is, and radically different reasons for studying them. It will be useful to discuss general features of neural networks, and then to formulate more precisely what will be meant by the term in the remainder of this study.

2.1 Physiology of the artificial neural network

The approach in neural networks(NN) research is essentially to model neurons by *nodes* which implement simple non-linear functions. Each node has a number of directed input and output connections, and associated with each connection is a *weight*. The *potential* of a node is usually taken to be the weighted sum of the outputs from all nodes connected to the node under consideration plus some threshold - the bias. The output from this node is its *activation*, calculated by applying the non-linear function to the potential. A network is determined by the connection between nodes, the connection weights, the biases, the potential function(s) and the activation function(s). If the graph comprised by nodes and connections is cyclic, the update scheme is also important, i.e. the order in which node activations are calculated.

When an input pattern is clamped on the input nodes the activations propagate through the nodes until activations possibly converge to a stable state. The output of the network is this state. In this manner a network can be considered as a function from the space of input patterns to final states. Considering a set of pairs of input patterns and final states as a discrete function, a network can be configured to approximate this function continuously.

The aim of all neural learning paradigms is to automate the configuration of a network given sets of input and target patterns, trying to create a good approximation to all given training sets. Learning in NNs can be divided into supervised and unsupervised. Supervised learning is a weight training process, in which effort is made to find a near optimal set of connection weights for a fixed topology according to some training set of patterns. A pattern consists of an input pattern and a desired target pattern. In the literature 'pattern' and 'example' are often used interchangeably. With a few exceptions, 'pattern' will be favored here.

2.1.1 Formal definition of neural nets

Formally, it will be convenient to take a neural network $\mathcal{N}$ to be a system comprising of:

- A set N of nodes, together with a decomposition function

$$d : N \longrightarrow I\!P(N)^2,$$

  which designates some of the nodes as *input nodes* I and some as *output nodes* O, possibly not disjoint sets. The *hidden nodes* H are defined to be those nodes which are neither input nor output nodes. Then

$$N = I \cup H \cup O$$

  If the node states are real, a network $\mathcal{N}$ effectively acts as a function

$$\mathcal{N} : I\!R^{|I|} \longrightarrow I\!R^{|O|}$$

- A set of thresholds. To each non-input node belongs a threshold $\tau_j \in I\!R$, affecting the activation function(s) at choice. It is equivalent to regard the threshold concept as an independent input node, the bias node, with constant activation value 1, and a connection to all non-input nodes.

- A connectivity. The connectivity of a network can be viewed as a function

$$c : N \times N \longrightarrow \{0, 1\}$$

  which is defined by

$$c_{ij} := \begin{cases} 1, & \text{if node } i \text{ is connected to node } j \\ 0, & \text{otherwise} \end{cases}$$

- A set W of *weights*. There is one real-valued weight associated with each pair of connected nodes, so that

$$\forall i, j \in N : (c_{ij} = 1 \Rightarrow \exists w_{ij} \in I\!R)$$

$$W := \{w_{ij}\}$$

- A set V of $|N|$ activation functions. Given the state of every node and weight in the network the ith activation function is used to update the state of the ith node:

$$V := \{v_i : I\!R^{|N|} \times I\!R^{|W|} \longrightarrow I\!R \quad | \quad i \in N\}$$

The following definitions assume a network $\mathcal{N} \in \Omega$, the set of all neural networks, with nodes $N = I \cup H \cup O$.

A *training set* $\mathcal{T}$, is simply a set of input and target patterns - training patterns. A training pattern $(i, t) \in I\!R^{|I|} \times I\!R^{|O|}$. Given a training pattern we define the degree of approximation by the networks error, ϵ, for that pattern.

$$\epsilon : I\!R^{|O|} \times I\!R^{|O|} \longrightarrow I\!R_0^+$$

is

$$\epsilon(o, t) \ := \ \sum_{j \in O} (o_j - t_j)^2$$

Finally, the global error, $\mathcal{E}$, is defined as the root mean square error(RMSE), over training patterns $\mathcal{T}$:

$$\mathcal{E} \ := \ \sqrt{\frac{1}{|\mathcal{T}|} (\sum_{r=1}^{|\mathcal{T}|} \epsilon(o^{(r)}, t^{(r)})}$$

2.2 Feed-Forward Networks

Layered, feed-forward networks will form the bulk of this study. In these networks nodes are partitioned into disjoint *layers*:
It is useful to introduce a function L assigning a layer to each node in an n-layer network:

$$L : N \longrightarrow \{1, ..., n\}$$

Then

$$c_{ij} = 1 \quad \Rightarrow \quad L_i = L_j - 1$$

Layer 1 is the input layer. Layer n is the output layer.

The *feed* F_j of a node j is defined to be the set of all nodes which can influence it, including bias

$$F_j := \{i \in N | c_{ij} = 1\}$$

the potential

$$\phi_j : N \times I\!R^{|W|} \times I\!R^{|W|} \longrightarrow I\!R$$

is given by

$$\phi_j^{(t)} = \sum_{i \in F_j} a_i^{(t-1)} w_{ij}$$

where t is the time step.

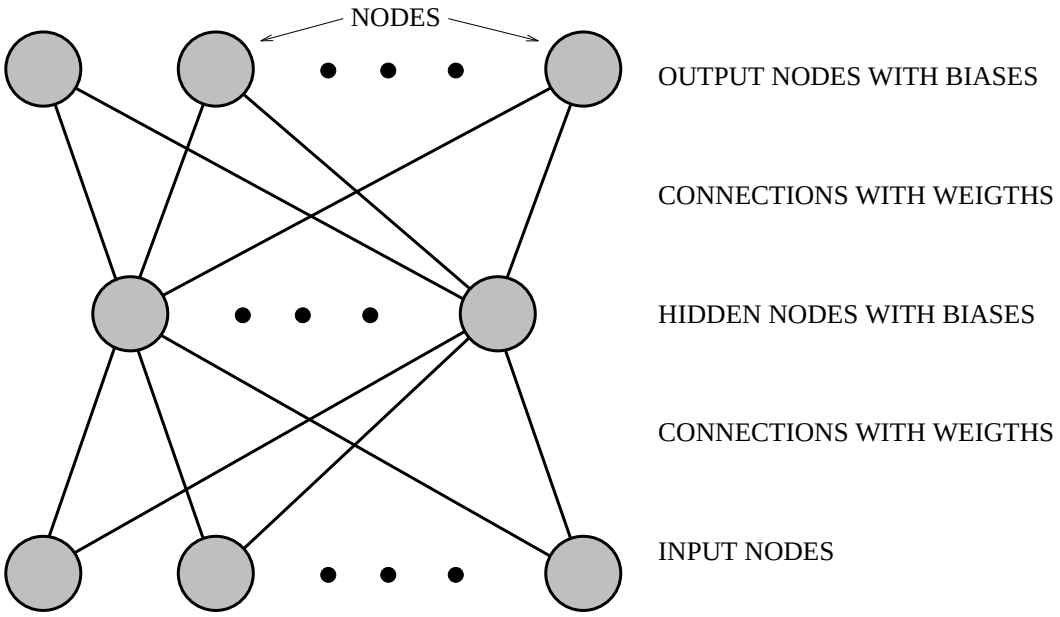


Figure 2.1: A fully-connected 3-layer, feed-forward neural network.

We also need an activation function $a_i : \mathbb{R} \longrightarrow \mathbb{R}$ which maps the potential to the activation of the node i. Generally, some sort of threshold function is used: a hard limiting threshold function (like the sign function), or a linear or semi-linear function, or a smooth limiting threshold function. Two examples of the latter are the tanh and the sigmoid (S-shaped), see figure 2.2:

- The hyperbolic tangens, tanh(x), yields output values in the range [-1,1].
- The sigmoid used in this work is

$$a_i(\phi_i) = \frac{1}{1 + e^{-\phi_i}}$$

yielding output values in the range [0,1].

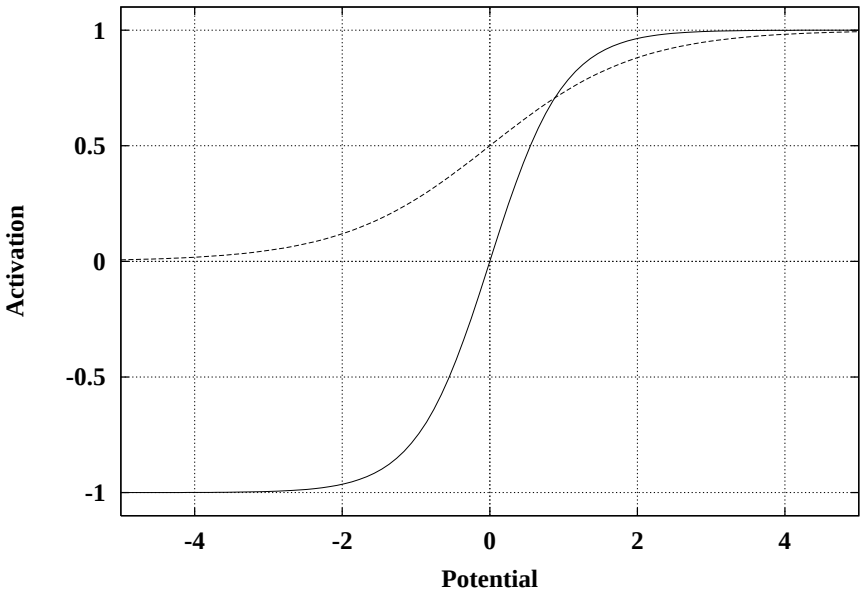


Figure 2.2: The tanh and sigmoid functions.

In the following a sigmoid activation function is assumed. In case the target patterns consist entirely of 0s and 1s it can be useful to interpret the continuous output in the range [0,1] as either 0 or 1 or *unknown*. This allows for a measure of percent correctly classified output activations. An interpretation of the continuous output from a sigmoid activation function called "40-20-40 percent criterion"[Alexis 1993] is:

$$I(a) = \begin{cases} 0, & \text{if } a < 0.4 \\ 1, & \text{if } a > 0.6 \\ unknown, & \text{otherwise.} \end{cases}$$

$I(a)$ is pictured in figure 2.3.

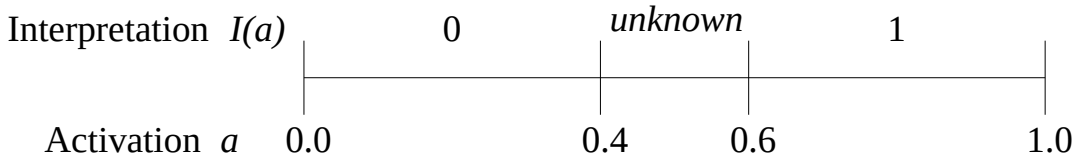


Figure 2.3: Interpretation of the output activation.

2.3 An equivalence relation on the space of networks

The interpretation I can be used as a mean to define an equivalence relation on the space of neural nets. For each inputvector in the training set, propagate the vector trough both $N1$ and $N2$. For each output node interpret the activation. If $N1$ and $N2$ agree on the interpretation on all outputnodes, on all training vectors, then $N1$ an $N2$ is defined to be functionally equivalent.

Formally define the equivalence relation $R_{networks}$ on the space of neural networks, as:

$$R_{networks}(N1, N2) \quad \leftrightarrow \quad \forall r \in \{1, ..., |\mathcal{T}|\} \; \forall j \in O : \; I(o1_j^{(r)}) = I(o2_j^{(r)})$$

2.4 Visualization of neural computation

To understand the behaviour of a neural network it is important to have some intuition of the neural model. The following section tries to impart it by using graphical examples in 2 and 3 dimensions. It is the author's opinion that an understanding gained from these examples is applicable to simple networks. The examples show some computational properties of a single node.

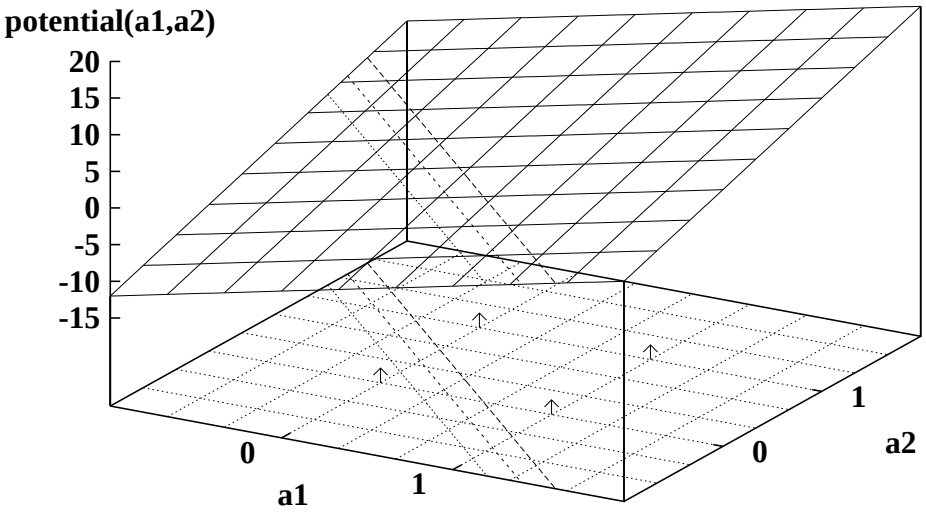
2.4.1 Node computation

The behaviour of a single node is determined by its potential function and its activation function. The very common summation node has a potential function which performs dotproduct between incoming activations and the weights associated with incoming links, inclusive bias.

The most common potential function performs dotproduct between incoming activations and the weights associated with incoming links, inclusive bias. This is a linear mapping from incoming values to a real value:

$$\phi = \sum_{i \in F} a_i w_i$$

In the case of just 2 incoming links with weights w_1 and w_2, bias b, and input values a_1 and a_2, the function can be viewed as a real function of a_1 and a_2. Projecting to the xy-plane, the contour lines, $\{(a_1, a_2) : \phi(a_1, a_2) = c\}, c \in I\!R$, are parallel and the distance from the zeroline $\{(a_1, a_2) : \phi(a_1, a_2) = 0\}$ to some contour line c, is proportional to c. The positive direction is perpendicular to the contour lines. Notice the sketch of contour lines, for $c = -1, 0, 1$, see figure 2.4.



1
0
-1

Figure 2.4: Surface of dotproduct $5a_1 + 5a_2 - 2$.

The contour lines indicates where the linear separation appears when applying the activation function. The sigmoid activation function takes the value 0.5 on the zeroline of the potential function. To illustrate how such a node classifies binary patterns, assume $n = 2$ binary inputs to the node. The input corresponds directly to a corner of a n dimensional (hyper)cube. Assuming the summation node should classify input $(0, 0)$ as class 0, and the remainder as class 1, the node could have $w_1 = 5.0$, $w_2 = 5.0$ and $b = -2$, see figure 2.5.

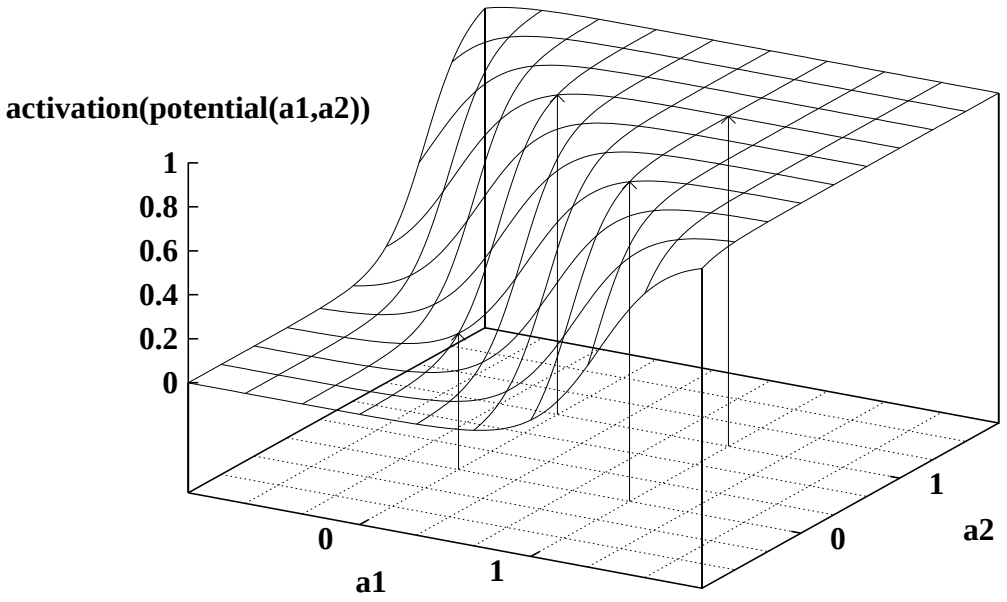


Figure 2.5: Sigmoid activation. Arrows show input (0,0),(0,1),(1,0) and (1,1).

The part of the sigmoid with highest slope gives a kind of fuzzy decision, with activation values neither close to 0 nor 1. This allows for uncertainty. The "width" of the fuzzy part depends on the number and absolute value of incoming weights. Therefore, with the correct bias, the overlap of the fuzzy part of activation function and the inputs can become arbitrarily small, theoretically. In practice this demands careful adjustment of the weights. The benefit is that as few patterns as possible are classified as 'unknown'.

The continuous output is compared to a binary target. The summation node linearly partitions its input into three sets, class 0, class 1, and class 'unknown', using the above interpretation of activation. The class 'unknown' is displayed as a "transparent 'belt", see figure 2.6. The arrows show input (0,0),(0,1),(1,0) and (1,1).

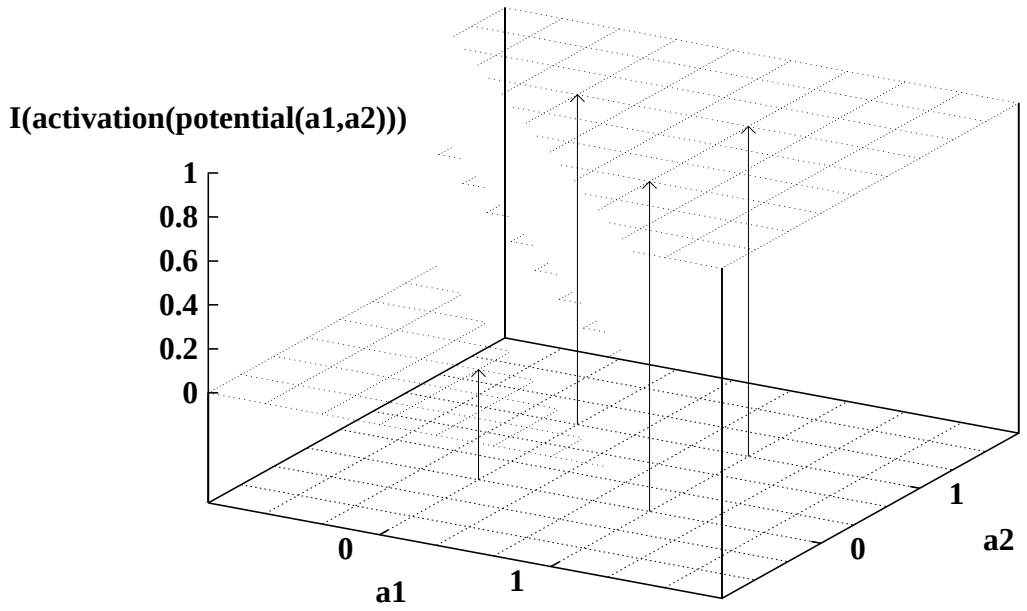


Figure 2.6: Binary interpretation.

Another 3-dimensional experience of the output of networks with just one output node is pictured in [Robbins 1993]. The networks solves the XOR task and some versions of the TwoSpiral task. The output is illustrated as a function of two reals in the interval [0;1], a unit square, with the function value as different shades of grey. Black and white represents values of near 0 and 1 respectively. The fuzzy part of the activation function comes out as smoothly varying grey.

2.5 Internal state of hidden layer

It is interesting to note that, as a node linearly separates its input, a two layer net can only handle problems which are linearly separable. For example, some variations of 0-9 digit recognition, are linearly separable[Harp et al 1989]. In a 3-layer net the hidden layer first linearly separates the input, and then the output layer linearly separates the activations of hidden layer to produce the final decision. In other words, the problems that can be handled by 3-layer nets must be "linearly linearly separable", assuming no connections from input to output layer. Introducing connections from input to output layer, obviously introduces additional computing power.

The activations of the hidden layer can be local or distributed. If, for all interesting patterns, the activations of hidden nodes are nearly 0 or 1 (the extreme values), the hidden nodes are said to act as feature detectors and the internal state of the hidden layer is said to be localized. Otherwise the internal state is said to be distributed.

A possible interpretation of internal state is to imagine the activations mapped to a unithypercube, with dimension equal to the number of hidden nodes. Then a localized internal state corresponds to a corner of the unithypercube and a distributed internal state corresponds to some point inside the unithypercube.

If the hidden representation is distributed, then the hidden nodes are highly dependent on each other, and no node can be removed or added without affecting the interpretation of the hidden state. "The behaviour of the hidden layer is more than the sum of its parts(nodes)".

If the hidden nodes act as feature detectors, it makes sense to think of the hidden node as an entity which can be removed or added to a net and each hidden node would still act as a detector, independent of the presence of other hidden nodes. "The behaviour of the hidden layer is the sum of its parts(nodes)". The complication of picking hidden nodes from two nets and thus building a new one is that the output layer makes its decisions on the basis of the hidden nodes. Removing or introducing feature detectors, without adjusting the connections and weights to the output nodes, inherits a risk of obstructing the output nodes classification of features detected by hidden nodes. Such an adjustment is exactly the same task as training (some of) the net.

Local as well as distributed states can be linearly separable. But in the task of combining two neural nets, it is perhaps impossible to make sensible combinations, if the nets implement distributed internal states. We must assume local internal states. But there is still a great danger that the cooperation of

hidden and output nodes is spoiled. This fact has great influence on combinations of genetic algorithms and neural networks discussed later.

2.6 The neural model

It is important to note, that whenever NNs are used to classify some data, the NN acts as an unknown model of the domain considered. The kernel of the neural model is a set of connected linear weighting and thresholdlike functions, and we should be worried if we have any suspicion if the domain modeled is intrinsically different from the neural model. Even if the net has succes learning some test data, we can not expect good generalisation, because of the intrinsic difference in the underlying model. This should be borne in mind whenever a problem is approached using neural nets.

Chapter 3

Genetics and Neural Nets

Combinations of GAs and NNs have been both supportive (they are used sequentially) and collaborative (they are used simultaneously). Using the terms introduced by [Yao 1992] the collaborative combinations of GAs and NNs can be divided into 3 major categories:

- Evolution of NN weights.
- Evolution of NN topology.
- Evolution of NN learning rules.

The list is now extended by:

- Evolution of both NN weights and topology.

The idea pursued in this thesis is to use a genetic algorithm for evolving both topology and weights. [Schaffer et al 1992]"Collaborative combinations typically involve using the genetic algorithm to determine the neural network weights or topology or both".

Evolution of NN weights. Given some mapping task, the topology is frozen and the GA is used to determine an optimal set of weights. The fitness is given by the error achieved. The approach can be hybridized, i.e. the global samples are optimized by local techniques. Studies in this categori includes [Miller 1989][Harp et al 1989][Whitley et al 1990]. [Whitley et al 1990] and many more, have tried using bitstrings with weights represented sequentially, and used wellknown bitstring crossover, sometimes with higher cross-propability at the boundary of weights. The topology is fixed and therefore also the mapping from bitstring to the corresponding network. They had succes on some toy problems. Using bitstrings and crossover only between weights is, of course, equivalent to regarding each weight as represented by a real number, of some precision. Considering that the weights in a network actually coorporate to implement some

functionality, it makes little sense to do plain crossover between weights or between bits representing a weight.

Therefore several attempts have been made to regard small groups of weights as basic entities which should not be disrupted. [Montana & Davis 1989] regard the fan-in connections, including bias, of non-input nodes to be entities, which is regarded atomic. They design a mutation operator which adjust all weights going in to some non-input node. Also [Hancock 1992] and [Radcliffe 1990] suggets treating all connections to a hidden node as an entity.

Then again, hidden nodes do also collaborate to fulfill the functionality of the net. Particular a net which distribute its internal state on several hidden nodes. Regarding hidden nodes as entities, it seems unreasonably to expect to develop nets which implement distributed internal states.

> [Yao 1992] "It is wellknown that nodes in a hidden layer of a multilayer feedforward network function basically as input feature detectors. Separation of encoded connection weights leading to the same node far apart in the binary representation will make the exploitation of non-linear interactions among them much more difficult because of the interruption of crossover[Belew et al 1991]. Hence, much more time will be needed to evolve useful feature detectors(connection weights). It has been demonstrated that the concatenation of encoded connection weights according to hidden nodes, is beneficial in exploiting and keeping useful functional blocks formed around hidden nodes."

It is thus commonly accepted that hidden nodes, including links and weights, constitutes building blocks. This is a reasonable assumption in nets where the hidden nodes act as feature detectors, each detecting some particular pattern(s) in the input. However, when the information in hidden layer is distributed over several nodes, it is difficult to interpret the interplay between hidden nodes. To discover correlations among activations in the hidden layer is timeconsuming and regarded unfeasible[Radcliffe 1990][Harp et al 1989]. It is thus expected that genetic algorithms treating hidden nodes as building blocks tends to develop featuredetecting hidden layers - a phenomenon not investigated here.

Evolution of NN topology. Given some mapping task, the GA is used to optimize topology. The nets are trained by local techniques like back propagation. The fitness is given by the achieved error and a penalty term on the complexity of the topology. Studies in this category includes [Whitley et al 1990] and [Montana & Davis 1989]. Use of blueprints by [Harp et al 1989], allows for a

compact representation of the topology, while the weights are found using back-propagation.

An important distinction is whether the topology and weights is encoded directly or indirectly. A direct encoding means that each connection and weight is represented on the genom. Whereas indirect encoding is inspired by the human genom which represent instructions for development of topology and weights.

Another distinction between the experiments is whether they allow dynamic development of neural topology or the topology is restricted, by some static datastructure. Of course, any implementation impose a physical upper limit on the size of nets. Indirect encodings can be static, and still represent dynamic networks, like the grammar coding in [Kitano 1990]. A peculiarity of dynamic systems fits in here. If the simulation has the possibility to grow infinitely, how do we avoid overflow on the physical computer: by an artificial upper limit or by the avaiable resources ? If our experiments is supposed to be compared by other experiments, a common limit would suffice. In real world tasks, the system should be allowed as much freedom as possible in order to explore wider.

Evolution of NN weights and topology. Some previous work in coevolution of topology and weights has been reported in [COGANN-92]:

Direct encodings that include both topology and weights have been reported in several papers. [Bornholdt and Graudenz 1991] used binary nodes and connections limited to ± 1 to learn boolean functions. They falsely claimed to use genetic techniques as they did not use any recombination whatsoever. It was just a parallel random hillclimber. [Collins and Jefferson 1991] created an "ant farm" task. [Martí 1992] found a winner-take-all network. [KrishnaKumar 1992] investigated control and modeling of dynamic systems. [Lindgren et al 1992] made a network learn regular languages. [Koza and Rice 1991] also used a direct encoding for both links and weights, but in terms of Lisp S-expressions. Their representation required a special recombination operator that preserves the syntax of proper S-expressions. [Elias 1992] used bitstrings to code adresses of node connections for a linked list representation. The addresses coded the node and also the location on its dendritic branch which simultaneously served to specify the weight of the connection. Also coding connections as adresses, [Das and Whitley 1992] focused on sparse distributed memories, which are equivalent to neural networks.

[Whitley et al 1990] and [Belew et al 1991] uses a direct representation - a wireframe, see figure 3.1

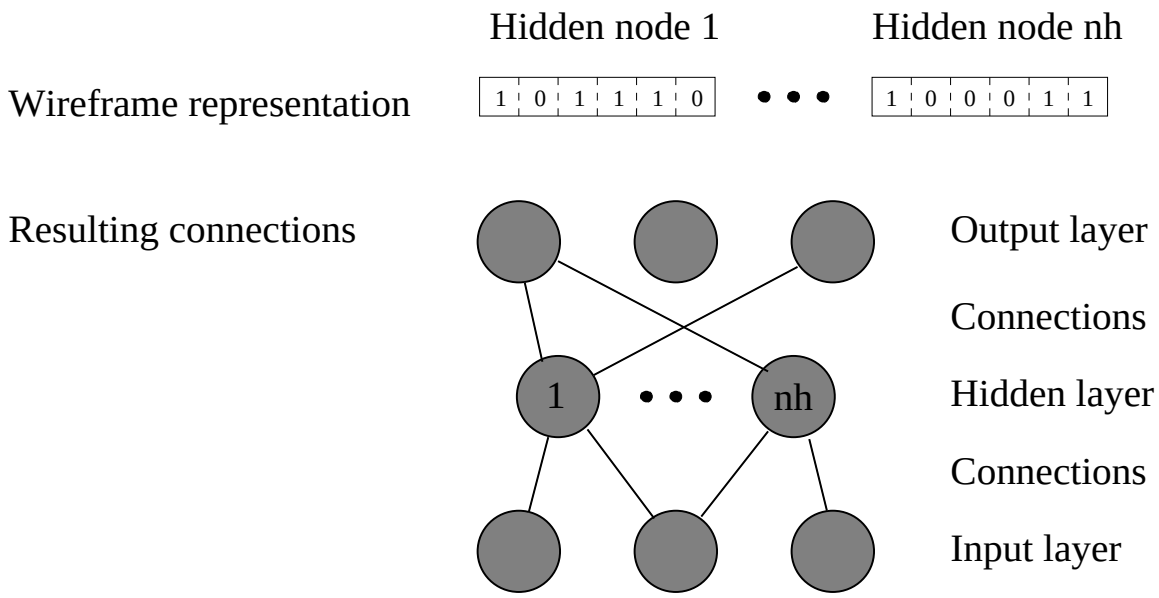


Figure 3.1: A wireframe representation

An initial, fixed architecture with all possible feed-forward connections, was trained to learn the task. This fully connected network was the "master" of the networks developed. The individuals contained a zero or one for each link in the trained network, which was used to mask out connections. The individual was evaluated by training a copy of this thin network. Part of the motivation for this particular study was to find smaller nets in order to reduce the number of connections needed for hardware implementations.

[Gruau 1993] used a sophisticated indirect representation which evolved nets for the 50 input parity and 40 input symmetry tasks. "Here we limit ourselves to binary weights (± 1). The algorithm that is put forward yields both the architecture and the weights by using chromosomes that encode an algorithmic description based upon a cell rewriting grammar. The developmental process interprets the grammar for l cycles and develops a neural net parametrized by l."

[Dasgupta & McGregor 1992] presents the Structured Genetic Algorithm (sGA). This encompasses a hierarchical representation of bitstrings, which should be biologically plausible. A bit in one layer activates or deactivates a substring of bits on the next level and any lower level substrings. So the dynamic behavior of bits is governed by higher level bits. The leaves need not be bits, but can be any

datatype. The number of bits on each level is fixed. This allows recombination and mutation to take place on specific levels. Notice, all hidden nodes thus have the same number of weights, despite some of the connections are deactivated. This allows recombination on the weight level without affecting the connection level and vice versa. The number of possible connections/weights remains fixed in all individuals throughout the population.

In the case of evolving both topology and weights they represent the topology as a wireframe on the first level. They allow all possible feedforward connections through the use of "upper-right triangle of the connectivity matrix". On the second level the weights are represented. They use 2pt crossover, and observed improved results on the XOR problem if recombination was restricted to occur between weights only. Fitness is sum-square-error plus penalty for many nodes/connections. In order not to favor too meager nets there is also penalty term for infeasibilty of nodes, that is, missing fan-in or fan-out connections. Furthermore, relatively high fitness nets are more likely to have changes occuring on the weight level, while lower fitness nets have changes on the topology level as well. This preserves promissing topologies for further training of the weights. A good idea, which may not work if a penalty of the topology cannot be defined. The sGA differs from a 'flat' representation, where only weights or connections are represented. In hierarchical representation even 'outmasked' connections are remembered, for possible later use. They report good results on function optimization over a simpler GA.

[Angeline 1993] argues that its too difficult to construct a reasonable recombination operator for neural net topology and weights, and presents GNARL, a parallel, random hillclimber, performing well on regular language tasks, requiring recurrent networks. There is probably no sensible way to recombine any two NNs, in general. Therefore GA is canceled for Evolutionary Programing with mutation operators for topology mutations and for weight mutations. Mutation is made stronger for low fitness nets through an intermediate definition of temperature. They argue to have minimal restriction on possible topologies, thus allowing for any, even recurrent, net. [Angeline 1993] outlines the GA and Evolutionary Programing(EP):

> "A classic GA consist of a Recombination Space (genospace) and an Evolution Space (phenospace) linked by an interpretation function, which makes Recombination Space independent on the current task. Both recombination and mutation are used, according to Hollands algorithm. Evolutionary Programing has only Evolution Space and taskdependent operators(...)
>
> Crossover tends to be most effective in environments where the

fitness of a member of the popoulation is reasonably correlated with the expected ability of its representational components (...) Environments where this is not true are called deceptive(...) There are three forms of deception when using crossover to evolve connectionist networks:

- competing conventions: equivalence modulo permutation of hidden nodes.
- a single topology affords multiple solutions for a task, each implemented by a unique representation spread across the hidden nodes.
- two nets with similar fitness can have highly different topologies."

3.1 A word on generalisation capability

Lets view the net as a function mapping input vectors to output vectors. The true power of artificial neural nets comes not from their ability to learn complex mappings from given set(s) of input/output vectors. But that the net generalize well on new data. This means that at a minimum, the learned mappingfunction can map input vectors in the neighbourhood of "learned" input vectors, to the neighbourhood of the corresponding output vector. To handle unseen neighbours gracefully, this requires the function to be smooth near the learned input vectors, though it is of no concern what the function looks like outside the corpus of interesting input vectors. [Schiffmann et al 1991] suggested using the performance of the trained network on a separate test set as a measure of generalizability of the topology, a method that was demonstrated by [Schaffer et al 1990]. The concept of generalisation is also addressable through minimization of the topology. It is researchers empirical observation that the smaller of two similar performing nets, has better generalisation performance. This suggests that smaller net be prefered and rewarded, fx. by means of a penalty term on bigger nets [Belew et al 1991]. Too few connections can prevent acceptable learning, while too many will tend to hinder generalisation through the phenomenon of overlearning[Harp et al 1989].

3.2 The permutation problem

There is a fundamental problem concerned with combination of 3-layered nets, whether combining topology or weights or both. Two identical nets, except for a permutation of hidden nodes, have identical functionality. This is called *competing conventions* [Schaffer et al 1992], *structural/functional mapping difficulty* [Whitley et al 1990], or simply *permutation problem*[Radcliffe 1990].

Specifically, in a fully connected, layered, feedforward network with a single hidden layer comprising of nh nodes, there is an $nh!$ potential redundancy associated with the indistinguishability of networks with relabelled hidden nodes. The precise level of redundancy depends on details of the representation used. This makes the genospace highly multimodal. If the recombination operator(s) uses the arbitrary labelling to guide combination, it is unlikely to be effective.

Example: Consider two nets $\mathcal{N}_1$ and $\mathcal{N}_2$ with with 2 hidden nodes each. Label the hidden nodes, inclusive connections, A and B in $\mathcal{N}_1$, and C and D in $\mathcal{N}_2$. Call $\mathcal{N}_1$ an "AB-network" and $\mathcal{N}_2$ a "CD-network", see figure 3.2

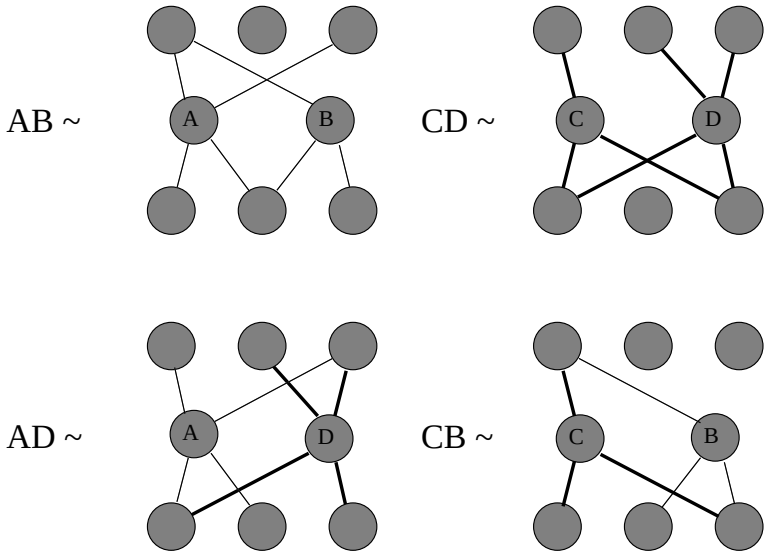


Figure 3.2: Parent networks AB and CD. Possible crossovers AD and CB.

Classic crossover allow children AD and CB, not AC and BD! Which in this example constitutes half of the obvious possible combinations.

> [Radcliffe 1993] "This is a phenomenon which is not seen in 'conventional' genetic search (for example, simple parameter optimization), and it has been strongly argued elsewhere that the problem is highly detrimental to the effectiveness of the search process, both in the specific context of neural networks ...[Belew et al 1990]..., and more generally..."

In spite of the growing concern with this problem, [Hancock 1992] present results that suggest it may be less of a problem than many suppose. These results

may be due to Hancock's use of small populations and high selection pressure leading to early convergence constituting of individuals with same permutation of the fittest hidden nodes [personal com. with NJ Radcliffe].

Chapter 4

A new paradigm

It is a common misunderstanding that strings of alleles are *the* representation, that single- or multi-point crossover are *the* way to combine individuals, and that the representation and the operators are necessarily independent on each other. The misunderstanding extends to cases when searching in more complicated domains, where it is common to adapt the complex domain into some string representation, and then leave the job to standard, classic crossovers. The new paradigm clarifies the concept of combination: The combination of individuals can be designed to suit the representation of the domain.

The advantage of this approach is that given any problem we can tailor the genetic operators to fit the task, usually doing better than some standard (bitswapping) operators. The disadvantages are that we loose the beautiful property that the GA is "inaware of" the domain of the searchspace - the GA use a standard representation, standard operators, and the user only has to deliver some interpretation and evaluation of individuals.

4.1 Problems with the genetic method

For implicit parallelism to work the schemata must contain individuals with correlated fitness. And this is the shortcoming of simple bitstrings, which presupposes correlations to be within fixed subsets of search space called schemata.

Further, the highly parallel search collapses if population converges to inherently similar individuals. In other words, if individuals are similar, then they have more schemata in common, and therefore the implicit parallelism is reduced to a few hyperplanes.

There is a general tendency in the GA community to move away from Hollands genetic model. Hollands model has a beatiful property, the user only has to provide an interpretation of a fixed length bitstring to the current domain and a

measure of goodness of instantiations of that domain. Then the genetic "engine" takes care of highly parallel search, without interference from user. The schemata theorem and the simplicity of a space of bitstrings gave faith to a robust, widely applicaple search method. But GA-hard problems[Goldberg 1989] occured, which sometimes resulted in performance at the level of random search. It is becoming clear that adapting any domain to bitstrings, and perform domain independent crossover on these, causes inefficiency. Domain specific representations and recombinators should be constructed. For instance [Whitley 1989][Radcliffe 1991] [Goldberg & Lingle 1985]([Radcliffe 1993]'s references) made tailored constructions for a famous graph problem, the Traveling Salesman Problem.

There is a need for a formal treatment of domain specific representation and recombination, and [Radcliffe 1993][Radcliffe 1993a] seems a carefully prepared attempt. These ideas are explained later in order to design a recombination operator used in the experiments.

4.2 Modern versus Classic genetic encoding

Once upon a time, everything was coded into fixed length strings of bits $B = \{0,1\}^l$ because genetic operators applied to strings of bits only. Actually it was the fundamental theorem of genetic algorithms, the schemata theorem, that limited the choice of representation space: only bitstrings belonging to the same schemata were considered to be correlated. Thus where genetic operators only considered appropriate, if they manipulated bitstrings, hopefully preserving properties of the parents. This led to the "building block hypothesis" - that good, short schemata juxtapoint to highly fit chromosomes[Goldberg 1989].

It is no longer necessary to create simple representation spaces, designed to be manipulated by standard operators. Rather the operators can be designed to manipulate instances of the domain in focus.

The forma-analysis (see section 4.3) by [Radcliffe 1993][Radcliffe 1993a] makes it possible to define the correlations among members in a much more flexible fashion. And to do the genetic search in S directly, formally at least. Forma-analysis will be reviewed in detail later.

C may be introduced, still, if this makes the implementation easier. C may also be introduced if it is much smaller than S caused by some biological plausible coding mechanism, like the blueprints of [Harp et al 1989].

The following is a summary of the authors view on the classic versus modern approach to use evolution in search and optimisation:

Classic evolution:

- *Phenospace* is domain dependent space over a fixed sized datastructure.
- *Genospace* of chromosomes with a fixed number of alleles - *classic representation*, fx. strings of bits.
- *Population* is a sample of points from genospace.
- Domain independent *crossover*, combining alleles of the chromosomes - *classic crossover*, fx. 1pt, 2pt, and Uniform crossover.
- *Coding function*: Genospace $\longrightarrow$ Phenospace.

Modern evolution:

- *Phenospace* is domain dependent space over any datastructure - *modern representation.* The data-structure must have some explicit extractable features, which can be combined/exchanged.
- No *genospace* is necessary, but some mapping or 'unwrapping' of the phenotype might be incorporated, like the representation in [Gruau 1993].
- *Population* is a sample of points from phenospace.
- Domain dependent *recombination*, combining features of the individuals - *modern recombination.*
- No *coding function* is necessary.

Whether to use Classic or Modern evolution depends on the task. If the task fits well into a fixed sized representation and no domain knowledge about possible interdependencies between solutions exists, we are better off using a classic representation. The Classic GA is wellfounded and investigated by numerious experiments. If domain dependent information helps to identify the features of solutions and suggests a sensible way to combine/exchange such features we are better off using a Modern GA.

A huge step towards a formal approach to the previous defined modern evolution is suggested by [Radcliffe 1993][Radcliffe 1993a], thoroughly generalizing schemata to formae. Formae are equivalence relations on the search space, dividing the space into reasonably correlated subspaces. The following section motivates and introduces formae.

It will be apparent that, using formae to define a recombination operator for 3-layer neural nets, it is possible to avoid the permutation problem. G iven two individuals then the probability distribution of the possible recombinations(children) is independent of permutation of the parents hidden nodes.

4.3 Formae

A schema creates an equivalence relation that partitions the search space into those individuals belonging to the schema, and those not. In order to do better than random search it is required that individuals in same equivalence class are somehow correlated: they must share some property or properties important to the optimum of focus. Schemata can only define such classes in terms of wildcards, dividing the highdimensional searchspace by hyperplanes. The intrinsic parallelism of processing schemata inherent conventional crossover has a generalisation, the intrinsic parallelism of processing formae. Formae, allow defining classes of correlating individuals as any subsets of the search space, providing flexibility far beyond schemata. Schemata are a special case of formae. The force of schemata is that they are already defined, but they may not fit the task. The force of formae is that you can optimize search by domain specific knowledge, allowing "reasonable" combinations only. This requires some knowledge of the task to be usefull.

The idea of allowing flexible describtion of correlated individuals has also come to [Vose 1991], but his genotypes was restricted to fixed lenght strings of bits. A quick comparison of **V**ose and **R**adcliffe:

Possible representations

V Fixed length strings of bits.

R Any datastructure, fx. fixed or variablesized multisets, sets, strings.

Abstract concept to define correlated individuals

V Matrix M induces "super-schemata", which are any subset of the search space.

R Equivalence relations Ψ on the search space, induces formae Ξ - the equivalence classes of the relations Ψ.

Construction of recombination operator

V Operator automatically follows from matrix M.

R Operator must be constructed and obey requirements of Respect, Gene Transmission, and Assortment, which are defined below.

The counting argument which is sometimes used to claim that binary representations are more powerful than those of higher cardinality is valid only if attention is restricted to traditional schemata. If more general formae are considered, the argument no longer holds. The degree of intrinsic parallelism which can be inferred is defined entirely by the selection of formae [Radcliffe 1990]. Radcliffe and Vose independently showed how the 'schema theorem' applies to general formae in exactly the same way as to schemata, given suitable expressions for the disruption coefficients.

4.3.1 An illustrative example

The presentation of formae given in [Radcliffe 1993][Radcliffe 1993a] is very formal and complete and with little use of comprehensive examples. This is appreciable, because overthrowing the well established paradigm of schemata (and theorem) naturally requires some persuasion. The following section is meant as a review and companion to the original introduction.

Assume the task is to search a set of possible boat configurations. Let S consist of boats with possibly no motor, left, right or both motors:

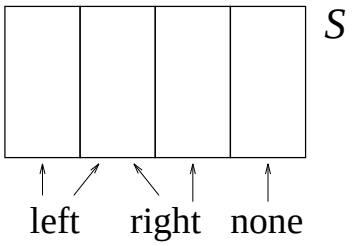


Figure 4.1: Space of boats

The criteria is to find a fast boat. An immediate coding of this space is

$$C = \{0,1\}^2 = \{00, 01, 10, 11\},$$

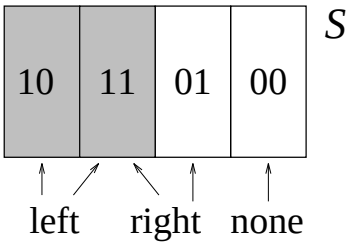
a bit for the left and the right motor.

The possible schemata are

$$\{0,1,*\}^2 = \{00, 01, 0*, 10, 11, 1*, *0, *1, **\},$$

inducing 9 equivalence relations. The relations possess two equivalence classes: the positive examples and the remaining.

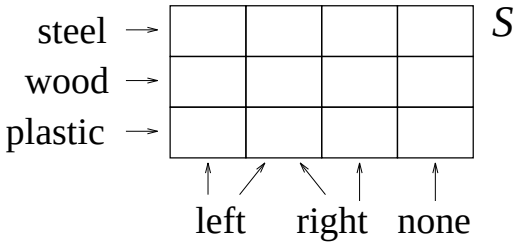
F.ex.: $\{1*\}$ induce the classes $\{10, 11\}$ and $\{00, 01\}$:



Conventional combination of 01 and 10 could yield either 00 or 11, i.e. either non or both motors - two completely opposite solutions in the context of boats. This means that two fairly fit boats, could yield a boat that can not move at all (00). The solution is either to make another coding of S or assure the combination of parents only yields children correlated with parents w.r.t. to some domain knowledge. This desire motivates formae.

The forma concept presumes that some of the correlations between elements in searchspace are known, at least partially. Then the elements which presumeably have something in common should belong to the same equivalence class. For every sensible correlation, thus forming a whole family of equivalence relations. Each equivalence relation partitions the search space w.r.t. a particular kind of correlation, independent of the other relations.

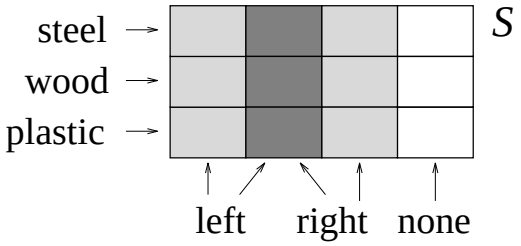
Example: To illustrate the force of formae, the boat example is extended such that the material is also a feature of boats: steel, wood, or plastic:



From the authors poor understanding of boats, it seems reasonable to focus on the following correlations to find rapid boats. I thus define 3 equivalence relations over the space of possible boats.

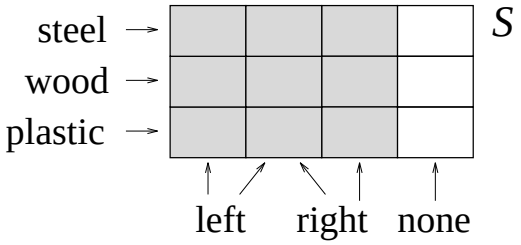
$R1$ is boats with the same number of motors, because they presumably have similar power:

$$R1 = \begin{cases} \{X \mid X \text{ has no motor } \} \\ \{X \mid X \text{ has a left or right motor } \} \\ \{X \mid X \text{ has both motors } \} \end{cases}$$



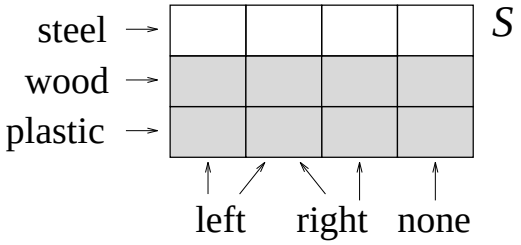
$R2$ is boats with or without motors, because they are essentially different:

$$R2 = \begin{cases} \{X \mid X \text{ has no motor } \} \\ \{X \mid X \text{ has at least 1 motor } \} \end{cases}$$



$R3$ is boats with presumable similar weight, because I assume this has some influence on speed:

$$R3 = \begin{cases} \{X \mid X \text{ is made of steel } \} \\ \{X \mid X \text{ is made of wood or plastic } \} \end{cases}$$



R1, R2, and R3 induce 7 formae where 6 are unique, as $\{X \mid X \text{ has no motor}\}$ occurs twice:

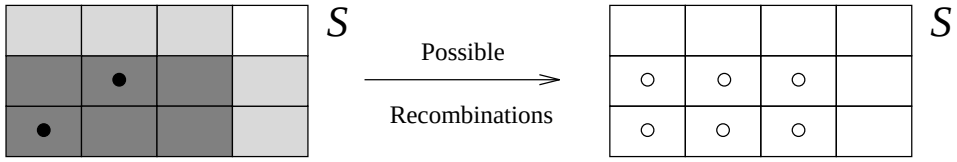
$\xi_1 = \{X \mid X \text{ has no motor }\}$
$\xi_2 = \{X \mid X \text{ has a left or right motor }\}$
$\xi_3 = \{X \mid X \text{ has both motors }\}$
$\xi_4 = \{X \mid X \text{ has at least 1 motor }\}$
$\xi_5 = \{X \mid X \text{ is made of steel }\}$
$\xi_6 = \{X \mid X \text{ is made of wood or plastic }\}$

4.3.2 Criteria for recombination

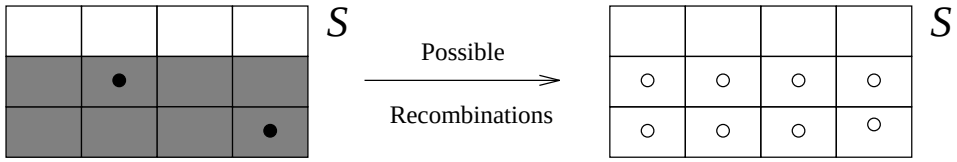
Just as conventional crossover operators "obey" schemata it is sensible to formulate some criteria for combination obeying formae. Guided by studies of the traditional crossover for conventional strings, three characteristics of recombination operators were suggested by [Radcliffe 1993] to be desireable: **Respect**, **Gene Transmission**, and **Assortment**.

Respect Respect requires that whenever two parents are both a member of some forma, all their offspring be members of that forma also. In other words a child must be member of all formae, that both its parents are. The formulation of the principle was motivated by a desire to ensure that in cases where the parents share some feature, children are guaranteed to inherit that feature also.

Example: Let parent η_1 be (left motor, plastic), parent η_2 (both motors, wood). Both parents belong to the formae $\{X \mid X \text{ has at least 1 motor }\}$ and $\{X \mid X \text{ is made of wood or plastic}\}$. Respect implies the child belongs to those formae, thus a member of $\{X \mid X \text{ has at least 1 motor and is made of wood or plastic }\}$.

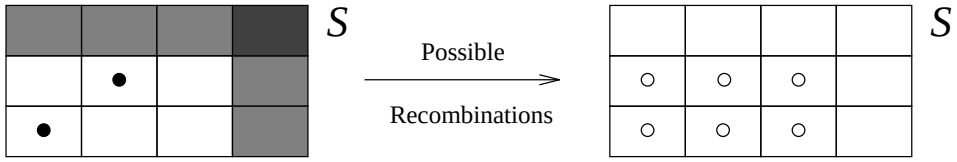


Example: Let parent η_1 be (no motor, plastic), parent η_2 (both motors, wood). Both parents belong to the formae $\{X \mid X \text{ is made of wood or plastic }\}$ only. Respect implies the child is of same material as either parent, but can have any number of motors.

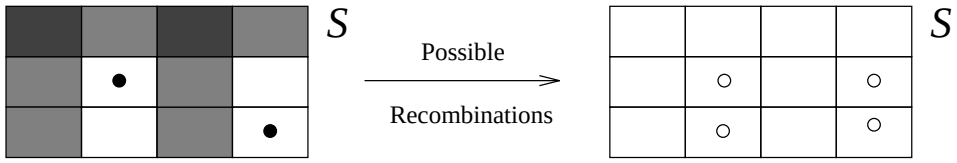


Gene Transmission Respect alone is not enough to ensure that each forma containing the child, also contains at least one of the parents. In other words, we do not want a child to possess properties (as defined by formae) which neither parent do. Notice gene transmission implies respect, because formae are equivalence classes.

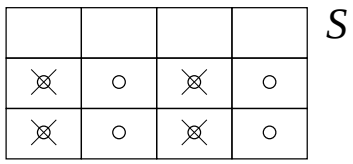
Example: Let parent η_1 be (left motor, plastic), parent η_2 (both motors, wood). None of the parents belong to $\{X \mid X \text{ has no motor }\}$ or $\{X \mid X \text{ is made of steel }\}$, so none of the children will either.



Example: Let parent η_1 be (no motor, plastic), parent η_2 (both motors, wood). None of the parents belong to $\{X \mid X \text{ is made of steel }\}$ or $\{X \mid X \text{ has a left or right motor }\}$, so no child will do so either, though parent η_2 is member of $\{X \mid X \text{ has at least 1 motor }\}$. This is a further restriction than implied by Respect.

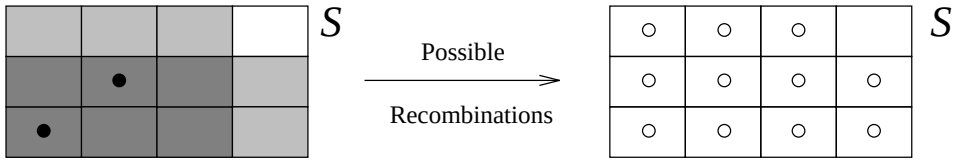


This last example shows that Respect does not imply Gene Transmission: The requirement of Respect on the combination of (no motor, plastic) and (both motors, wood) allow, for instance, the child (left motor, plastic). But this child is not among the possible children under Gene Transmission:

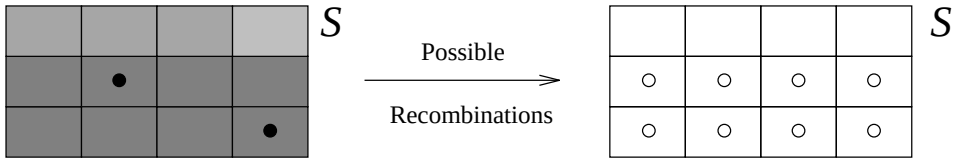


Assortment A recombination operator is said properly to assort a set of formae, if, whenever parent η_1 is member of some forma ξ_1, parent η_2 member of some forma ξ_2, and these formae have nonempty intersection, then it is possible that recombination of parent η_1 and η_2 yields a child which is a member of both formae. The notion of Assortment can be viewed as an extension and formalisation of the building-block hypothesis. If the choice of formae is such that the requirements of Respect and Assortment are incompatible, then the formae are said to be nonseparable. In such case [Radcliffe 1993] argues that Assortment should be given precedence over Respect.

Example: Let parent η_1 be (left motor, plastic), parent η_2 (both motors, wood). *Both* parents belong to the formae $\{X \mid X$ has at least 1 motor $\}$ and $\{X \mid X$ is made of wood or plastic $\}$. Assortment implies the child could be in any of these formae - thus violating Respect.



Example: Let parent η_1 be (no motor, plastic), parent η_2 (both motors, wood). Both parents belong to the formae $\{X \mid X$ is made of wood or plastic $\}$, η_1 belongs to $\{X \mid X$ has nomotor $\}$, η_2 belongs to $\{X \mid X$ has at least 1 motor $\}$. Assortment implies the child could be in the intersection of these formae - in correspondance with Respect.



To summarize the use of formae: It is our responsibility to choose domain-specific formae. If the formae happen to be nonseperable we can either redesign formae or make sure recombination gives precedence to Assortment. Then the recombination operator(s) must be designed to obey the 3 requirements.

To give the simplest example of nonseperable formae asume 3 formae A, B and C which (pairwise) intersects nonempty. Assume an individual η_1 belonging to A and η_2 to B. Assume further formae C such that η_1 and η_2 belongs to C and $A \cap B \setminus C \neq \emptyset$, see figure 4.2

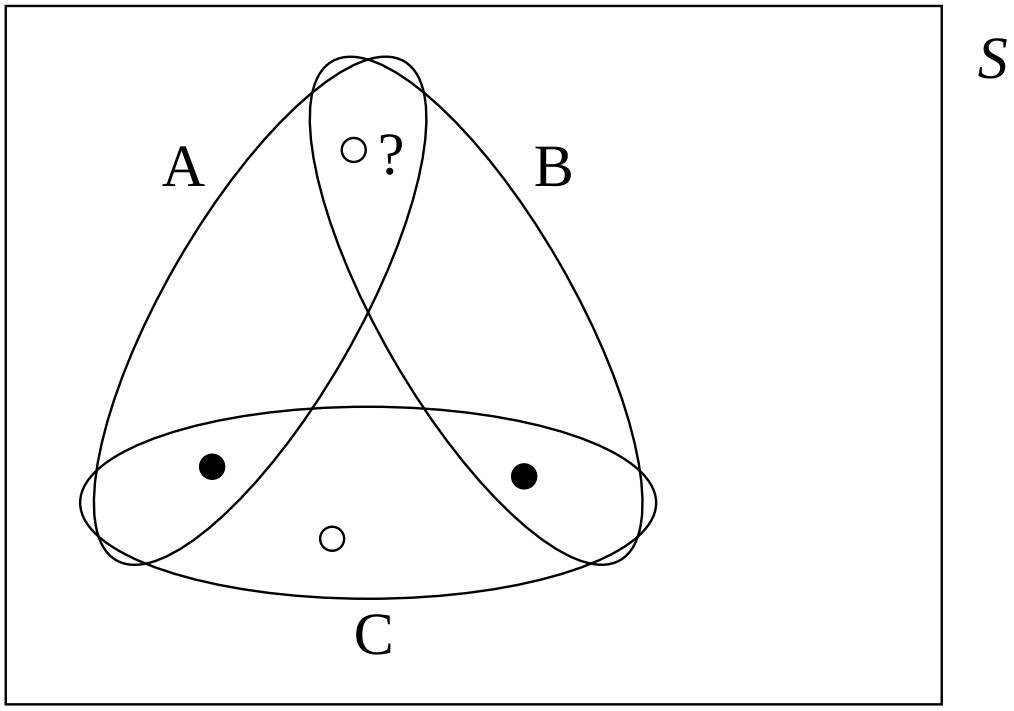


Figure 4.2: Three non-seperable formae in S

Let η_3 be a recombination of η_1 and η_2. Respect implies η_3 must belong to C and Assortment implies η_3 could belong to $A \cap B$. So Respect and Assortment are incompatible. Giving precedence to Assortment seems reasonable as Respect would limit the possible combinations to yield a child in formae C.

4.4 Use any knowledge of the search space

Several kinds of relations on the search space increases the advantage of the formae, as long as the relations actually describe classes which contain correlated individuals. The more kinds of relations which can be found to describe possible correlations points (individuals) in the search space, the less children which are uncorrelated with their parents will be created - thus the more precise the trajectories of search!

If *only a single* equivalence relation is used to induce formae (equivalence classes), then the formae would simply form a partition of the search space - they are disjoint. Therefore the requirements of Respect, Gene Transmission, and Assortment makes little difference to the possible outcome. Assume two individuals are selected for recombination, then:

- If these parents belongs to the same formae, then (due to Respect), the outcome must belong to the same schemata. Which is desireable.
- But if they belong to different formae, then there are no restrictions on the possible outcome whatsoever. The combination of two individuals from different formae could yield *any* individual in the search space.

Thus, just as schemata impose several relations on a bitstring space, several relations (like $R1$,$R2$,and $R3$) should be used to describe possible similarities between points (individuals) in the search space.

Chapter 5

NN Recombination

The intended meaning of the headline is recombination of both topology and weights. In order to give precedence to topology over weights it might seem good economy to have a GA evolving topologies which another GA train to find an optimized set of weights, in order to measure goodness of that particular topology. Promising methods in the litterature can be combined: One GA for toplogy evolution, one for weight optimisation. But during such a 2 step process there will of course be developed similar weights for similar topologies again and again. This is my motivation to have a GA managing entire nets in the population. [Radcliffe 1993] has used formae to combine topology of networks. His construction is carefully reviewed and then extended to include combination of weights.

5.1 Formae for networks

It seems appealing to define formae as equivalence classes of $R_{networks}$, but this is unfeasible for two reasons:

1. As only a *single* equivalence relation is used to induce formae, then the requirements of Respect, Gene Transission, and Assortment makes little difference on the possible outcome.

2. Combining two individuals from the same formae must thus (due to Respect) produce a child functionally similar to these parents. But this is exactly what all neural net optimisation is all about: to construct nets with a specific behaviour on a specific training set! Therefore this kind of formae is not attemted.

5.2 Formae for multisets

To optimize the topology of a feedforward net, with a fixed number of input and output nodes, n_i and n_o, and bias to all non-input nodes, the connections of each hidden node can be represented as 'wire frames'. Each bit indicate presence or absence of a link between the associated hidden node and a node in input or output layer. That is, bitstrings with lenght $n_i + n_o$. A 'wire frame representation' is a multiset in $\{0,1\}^{n_i+n_o}$. For example, if the net of concern has two input and two output nodes then the wireframe {1111, 0000, 1111} indicates 3 hidden nodes, the first and last is connected to all input/output nodes and the remaining is (unfortunately) not at all connected.

Thus the topology of any 3-layered feedforward network can be represented as a multiset of $\{0,1\}^{n_i+n_o}$. The optimization problem can now be formulated as a problem of finding optimal multiset(s). Of course each topology has to be evaluated by instantiating a weight to each connection and train the net to find an optimal set of weights. If the number of hidden nodes is also available to optimisation the size of the multiset is variable.

Consider the space S of multisets with elements from some universe ϕ. Formae should contain correlating elements. The multiplicity of an element x of a multiset ϕ is the number of occurences of x in ϕ. Assume the multiset is represented as a set of pairs of each unique element and its multiplicity in the multiset. Ex: {1111, 0000, 1111} can thus be represented as { (1111,2), (0000,1) }. It makes good sense to define two multisets to be correlated if the multiplicity of their elements, lie within some range.

Actually this is Radcliffes definition of formae for variably sized multisets:

> "A major aim of this paper is to suggest a possible representation for neural network topology optimisation which avoids permutational redundancy and allows formae defined in a way independent of hidden node labels to be respected and properly assorted. This formulation builds on the idea of regarding hidden nodes complete with their external connections as basic entities, and views a network topology as a collection of such hidden nodes. This is the reason for the concentration on set and multiset optimisation problems in the following sections."

Radcliffe's ideas are focused next. A forma is conveniently described by a set of 3-tuples of the form $(x, N_x\downarrow, N_x\uparrow)$ each of which is understood to specify that the forma contain multisets with multiplicity of x ranging between $N_x\downarrow$ and $N_x\uparrow$. The description specify a tuple for each element in universe ϕ, except those

with multiplicity zero.

Definition 5.2.1 *All 3-tuples of the form* $(x, N_x\downarrow, N_x\uparrow)$*, forms the formae for variabelsized multisets.*

These formae can be shown to be separaple. To construct a recombination operator it is convenient to define the similarity set of any two multisets:

Definition 5.2.2 Similarity set *of multisets* η *and* θ *from* S*, is the smallest formae containing* η *and* θ*.*

The smallest formae containing η and θ consists of multisets where multiplicity of each element x is within the multiplicity of x in η and θ. Radcliffe suggested the following recombination operator to obey these formae.

Definition 5.2.3 Random Respectful Recombinator $\mathbb{R}^3 : S*S \longrightarrow S$ *uniformly picks an element from the similarity set of the operands.*

Thus recombination of multisets η and θ is a new multiset where the multiplicity of an element x lie within the multiplicity in parents η and θ.

$\mathbb{R}^3$ can be shown to obey Respect, Gene Transmission and Assortment. In the case of fixeds ized sets and multisets more complex considerations are needed, leading to correspondingly more complex recombination operators, e.g. the assorting multiset recombinator.

In 3-layered feedforward networks there are $(2^{n_i}-1)*(2^{n_o}-1)$ different hidden node topologies, excluding nodes with missing links to input or output layer. Because there usually are many more possible topologies of a hidden node than there can possibly be in even large populations, Radcliffe argues that if the node types are considered to be atomic and not available for recombination, it will be extremely difficult for the genetic algorithm to make progress. The solution usually employed is to allow recombination to take place at the sub-parameter level. Therefore most of the nodes in the child (say all but one) can be generated using the proposed recombination operator. The final node can be constructed by directly crossing nodes from the two parents on the connection bit level. Radcliffe suggests using simple uniform crossover for this.

This ends the description of combination of 3-layer topologies. The following discusses the recombination of nets, that is, both topology and weights - 'weighted topologies'. Formae can be used to define a recombination operator for weighted topologies in two ways:

1. Formae can be defined for the space of weighted topologies. Either viewing combination of weighted topologies as a (multi)set optimisation problem as before, or as combination of neural nets - a more specialized datatype than multisets. The latter requires a deep insight in the effect of combining components of two arbitrary networks. A knowledge which is not well established.

2. A weighted topology can be considered a topology which happen to be accompanied by a set of weights, and use the above recombination operator. Simply keeping weights associated with their corresponding connections, during recombination. An example of this is the Structured Genetic Algorithm[Dasgupta & McGregor 1992], where the possible connections are predetermined by a fixedsized connection matrix. Each entrance is a bit indication presence(1) or absence(0) of the connections. Each possible connection has a weight. A connection can be present or absent without affecting the associated weight. This is even biologically plausible.

It is straightforward, though, to leave out the connection bit, keep the weight, and interpret a weight of zero as an absent connection. In the experiments a flat representation is used, because there was no immediate advantage of using a redundant connectionbit associated with each weight. The author was not aware of Dasguptas work, during the experimental phase.

The two approaches above turns out to be very similar. Of course we should try to use formae to construct an operator for nets, as in 1. above. It seems appealing to regard a hidden node (weights) as a feature detector which should be regarded as an entity. Then the hidden layer is a multiset with elements from the space of possible weight configuration for a hidden node. A weight of zero is equivalent to no connection. Of course there are more weight configurations for a node than topology configurations, but there can still be two identical weight configurations. Therefore the hidden nodes of a NN can still be regarded a multiset, and therefore the same recombination operator as for combination of just topologies is applicable. This ends the treatment of combination of weighted topologies.

5.3 Example of recombination

To illustrate the method for combining network topologies used in this paper, consider the two networks shown below. To construct a child, the first hidden node is constructed by combining a node from each parent using classic uniform crossover of the connectivity of nodes. The rest of the hidden nodes are chosen by picking each hidden node in the parents with probability 0.5, until there are no more nodes or the child has $MaxNoOfHidden$ nodes. Each weight is copied unaltered, with the corresponding connection, see figure 5.1

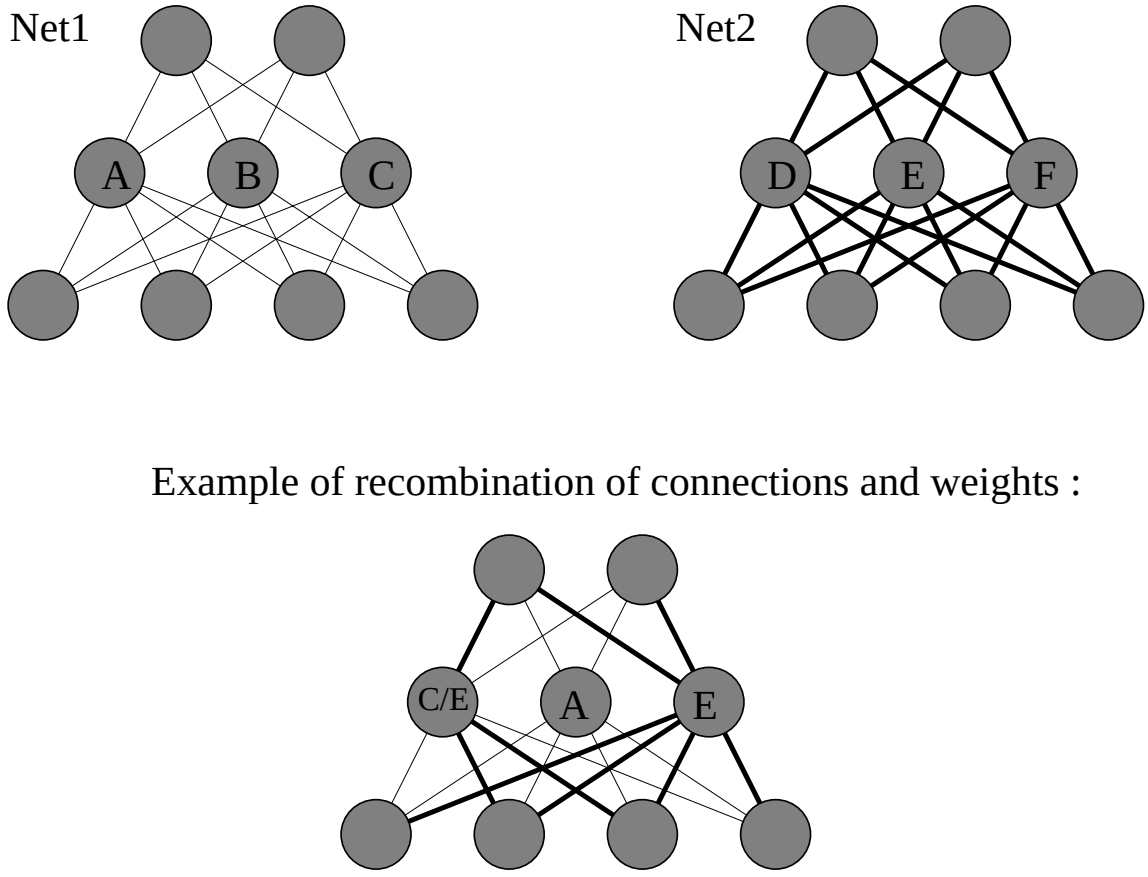


Figure 5.1: Above: Parent networks. Below: A possible recombination obeying formae for multisets.

Part II

The Cluster Analysis Tool

Chapter 6

Population dynamics analysis

6.1 Introduction to population analysis

Many GA practitioners use GAs as black box problem solvers, paying most or all attention to the final best result. This might origin in conservatism or blind trust in the robustness of GAs, settled by the fundamental theorem of genetic algorithms (the schemata theorem). But GA-hard problems[Goldberg 1989] or exotic representations and operators makes the GA behave differently. Therefore we must "open" the black box and analyze the internal behaviour of our algorithms.

GAs use probabilistic distributions to control the selection of parents, operators, and some even use selfadjusting parameter algorithms. All such phenomena can be analyzed to gain some understanding of different versions of GAs. GAs can, so to say, succeed in several different aspects:

- Operators succeed in sampling the attributes/characteristics from parent(s) to child(ren), as "required" by the schemata theorem and similar theorems. The succes of the operators is the responsibility of the designer.
- The individuals can be succesful: They can have high fitness relative to the global maximum in search space. The succes of the individuals is due to the robustness of the other aspects of the genetic algorithm used.
- The dynamics of the population can be succesful: not too big or too small clusters, thus allowing exploration and exploitation of search space. The internal behaviour of the population is governed by the properties of the GA, and the succes of this dynamic behaviour has great influence on the ability to find succesful individuals.

Here the focus is on the dynamics of the population. This can give important information of the behaviour of a particular GA or GAs in general as search methods. This is the aim of population analysis.

A population analysis tool can give an understanding of the actual effect of changing LifeCycle Administration, representation, operators or parameters, which is *not* available when simply observing fitness curves. It will also give us a mean to describe the occasionally chaotic behaviour of populations and thus the failure or succes of the solutions found. An explanation of the "flow" of the search - the answer to "how ?", has more than one kind of answer; we can focus on the whole run or on the changes from generation to generation. Possible kinds of population dynamics analysis, include:

1. A "map" or list of the search space points actually found. A list of several thousands individuals with their generation number (age) is quite enormous and incomprehensible.

2. A clustering of the search space points actually found, reporting size of clusters and a representative of each cluster, and the average "age" of the individuals belonging to that cluster, implying the longevity of the cluster. This would give a more comprehensive picture of the "flow" of search, but requires all individuals are saved during a run.

3. A clustering of the points in the population at each timestep, reporting the size of clusters and a representative of each cluster, and the average "age" of the individuals belonging to that cluster.

4. Some simple measure of the state of the population, yielding a single number, expressing the overall diversity of the population. The actual knowledge gained from such a measure is poor.

Only little attention has been paid to population analysis in the litterature. The popular GA simulator GENESIS is of kind 4 above. GENESIS measures the ratio of identical bits in the population at regular timestep intervals. This is a measure of convergence, which is appropriate only for GAs with classic representation and classic crossover.

A very sophisticated analysis tool, which not only delivers a snapshot of population clustering, but uses all individuals created throughout the run, is reported in [Louis et al 1993]. It is of kind 2 above. Their system extracts and processes historical information from the GA using knowledge acquisition and analysis tools developed for case-based reasoning(CBR).

> [Louis et al 1993] "After creating a sufficient number of individuals over a number of generations, the initial case data are sent on to a clustering program. A standard hierarchical clustering program is used, which clusters the individuals based on both the fitness and the alleles of the genotype. This clustering constructs a binary tree in which each leaf includes the data of a specific individual. The binary tree structure provides an index for the initial case-base. (...) Generation information can be used to determine when each part of the search space, covered by the GA, was considered. This information is very useful in detecting premature convergence, finding local minima (optima, ed.) and computing the relative longevity of a sub-space."

How powerful the CBR assisted explanation of GA results may seem, it is not obvious how the techniques can be applied to other representations than classic fixed length bit strings.

The idea of picturing the clustering in the population for every generation, is conceptually comprehensible. A simplified version of kind 3 above was used in the present experiments, to monitor the population dynamics and in particular monitor tendency to convergence.

Chapter 7

Cluster Analysis Tool

In this chapter an analysis tool is constructed which gains insight into any population dynamics phenomena, e.g. the 'Premature Convergence' phenomena, not just detection of convergence, but also how it evolved. "Similar individuals" are defined and the concept of similarity is used to group and monitor size of crowds as the genetic algorithm runs. This will shed light on the nature of convergence dynamics, which can be used in future investigations on controlling convergence. It is important to note that the following developed tool is only for analysis use, and is not intended to influence the population dynamics at all. It is also important to note that this chapter introduces yet another equivalence relation on the search space. This relation can, like the relations inducing formae, be defined completely arbitrarily in an 'ad hoc' manner. The particular equivalence relations used when defining formae for neural nets (multiset of hidden nodes) was motivated by a desire of constructing formae for multisets, in general. The equivalence relation used to cluster analyse the population of nets is motivated by a desire to express the functionality of the nets. This has certain advantages as discussed below.

In the case of co-evoling neural nets possible cluster concepts includes:

- identical functionality on the entire training set.
- identical functionality on specific subsets of the training set.
- number of hidden nodes.
- number of connections.
- topology of hidden nodes, preferably modolu permutations.

It is found appropriate to identify nets with identical functionality to be similar, because functionality of a network is of concern. The following illustrates the use of functional distance to monitor diversity in a population of neural nets.

Evaluation of an individual can be divided into 4 stages:

1) the genotype, which is translated into

2) the phenotype, which is input to

3) the evaluation process, which yields a

4) fitness value

Similarity can be measured in each of the 4 stages, with different advantages. The following comments assume the task is optimization of neural topology and weights. Each level is given a self-explanatory name - for reference:

1) genotype similarity: syntactic level. Weights or functionality are "invisible".

2) phenotype similarity: syntactic level. Weights can be compared, but there is no unambiguous way to pair off weights in any two nets. Thus functional comparison is still unpredictable.

3) evaluation similarity: functionality level. The outputvector corresponding to each inputvector can be compared by means of a functional metric or an equivalence relation.

4) fitness similarity: fitness level. Any inherent information of the individuals is "invisible". Fitness similarity can only be used to guide selection/reproduction.

To compare 2 nets intended to solve the same task, this view suggests using evaluation similarity. Such functional comparation would subsume trivial equivalences such as the equivalence between two networks that are the same up to a permutation of hidden node labels, and would further capture equivalence between two networks one of which had some completely redundant connections, or which implemented with several nodes a function which another network implemented with one node. Obviously we do not want to treat such "redundant" nets as special cases that deserves further investigation.

7.1 Formal definition of cluster

The first attempt is to define a metric between any two neural nets trained on the same training set. But clustering with respect to a metric inherits an uncertainty of the clustering. This metric is thrown away and the second attempt introduces an equivalence relation between neural nets. Equivalence relations are much simpler to deal with and possess no ambiguities.To cluster a set of individuals, we need a metric reflecting similarity. A possible definition is:

Definition 7.1.1 *Given two neural nets $N1$ and $N2$, with same number of input- and output-nodes, and trainingset $\mathcal{T}$, define* functional metric D_{func}*, as the sum over training patterns, over output nodes, the squared difference between output activations of $N1$ and $N2$:*

$$D_{func}(N1, N2, \mathcal{T}) = \sum_{r=1}^{|\mathcal{T}|} \sum_{j \in O} (o1_j^{(r)} - o2_j^{(r)})^2$$

where $o1_j^{(r)}$ is the output activation from net $N1$, r training pattern, j output node.

When $D_{func}(N1, N2, \mathcal{T}) = 0$, $N1$ and $N2$ are said to be functionally equivalent w.r.t. $\mathcal{T}$. If $D_{func}(N1, N2, \mathcal{T})$ is "small" $N1$ and $N2$ are said to be functionally similar or to have similar mapping behaviour w.r.t. $\mathcal{T}$. If for all pairs of individuals in a population, $D_{func}(N1, N2, \mathcal{T})$ is "large", the population is said to be functionally diverse. This leaves the difficult task of defining the concept of a cluster of similar individuals - i.e. determining when two clusters are close enough that they should be one cluster and so on. The problem of clustering "chains" of elements is: $N1$ and $N2$ is close, $N2$ and $N3$ is close but $N1$ and $N3$ is not close enough due to the choice of metric, threshold, or other fragile decisions, see figure 7.1.

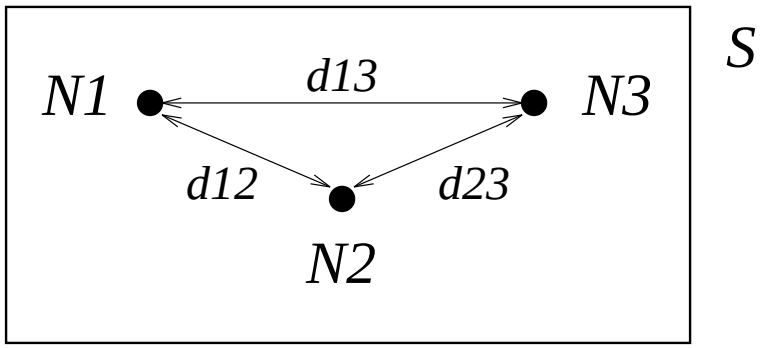


Figure 7.1: Search space S, with 3 individuals, $N1$, $N2$, and $N3$.

To avoid some of these subjective decisions, an interpretation of output activations is used to decide if two nets are functionally equivalent or not. This makes clustering an equivalence relation, avoiding the problem of clustering "chains" of elements, and thus a consistent cluster concept. The equivalence relation used in the experiments is $R_{networks}$ defined previously.

Definition 7.1.2 *Given an equivalence relation R on the search space S, each equivalence classes of R is called a* cluster in S, *and all individuals in the population B are said to constitute a* cluster *if and only if they belong to the same cluster in S.*

7.2 Clusters can be defined arbitrarily

But be wise. The equivalence relations used to determine clusters must not be too coarse or too fine grained. But they must be grained relative to size of search space. In the extreme coarse grained case the whole search space is considered a single equivalence class. In the extreme fine grained case each point in the space is considered equivalent with it self only. Clustering with respect to such extreme relations yields no extra information. It is not possible to give any general guidelines for cluster relations: how many clusters are appropriate or how many points in search space should be considered similar. The decisions to be made are similar to common classification tasks: how many classes are suitable, what kind of classes, and how could they be identified easily?

As a relaxing example lets classify all people into their geographic origin. People from the same origin are considered "equivalent" or "similar" and thus constitutes a cluster of people. This requires some division of the world which reflects the kind of information needed. If I say I was raised on planet earth or on the northern hemisphere you might think that it doesn't tell much about me. If I say I was raised in Svendborg you might think that it doesn't tell much about me, either. But if I say I was born and raised in Denmark, you instantly have a picture of a welfare society with healthcare and everything! You still don't know if I'm a good individual or have a high fitness, though.

If the cluster relation is too coarse grained then the appearence of all/most individuals within the same cluster does not imply convergence of the population, see figure 7.2.

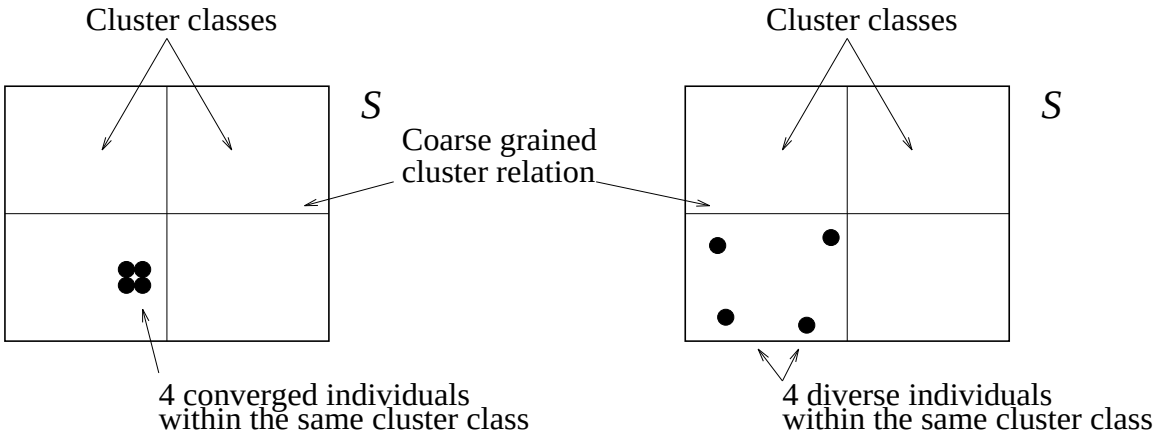


Figure 7.2: 2 examples of S with an cluster equivalence relation with only 4 classes. 4 individuals are within one of the classes.

If the cluster relation is too fine grained then the appearence of a diverse population, do not imply the population is diverse, because individuals might belong to clusters which, in a sense, is close together, see figure 7.3

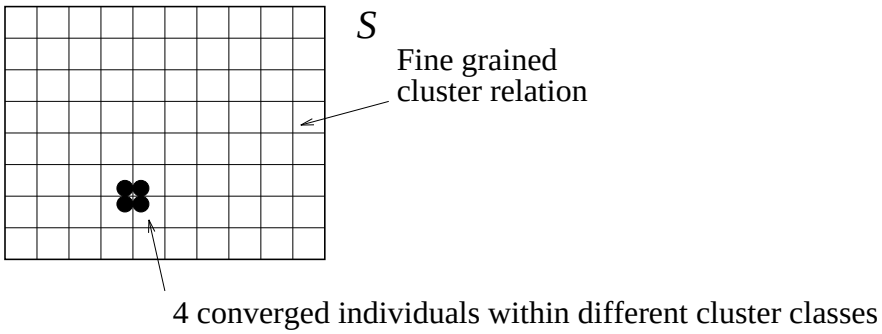


Figure 7.3: Example of S with an cluster equivalence relation with many classes. 4 individuals are within four of the classes.

We were better of joining some of these equivalence classes and start over with the new relation. Used correctly, the cluster analysis gives valueable information about the condition and dynamic behaviour of the population of a genetic algorithm. Notice, while Schemata and Formae are equivalence classes of several equivalence relations partioning the search space, then Clusters are equivalence classes of a single equivalence relation partitioning search space.

7.3 Cluster Analysis Bookkeeping algorithm

Given an equivalence relation R on the search space the Cluster Analysis Initialisation algorithm will divide the initial population into the respective clusters C_i of equivalent individuals.

Cluster Analysis Initialisation

R is an equivalence relation on the search space S.
C is a set of clusters C_i, $i \in \{1, ..., PopSize\}$.
C_i is a set (initially empty) of individuals equivalent under R.

for each individual I not clustered yet do
    assign I to an empty cluster C_n in C;
    for each individual J not clustered yet do
        if $R(I, J)$ then assign J to I's cluster;

Figure 7.4: Cluster Analysis Initialisation algorithm

There must also be some bookkeeping to maintain the set of cluster C when some individual is inserted into and some removed from the population. Cluster Analysis Bookkeeping algorithm mangage cluster relationship (equivalence class) of every new individual I, assuming K is the individual which is suppose to be removed. Assume the Replace Strategy is Replace Least Fit:

Cluster Analysis Bookkeeping

Let I be the child to be inserted in the population.
Let K be the individual to be removed from the population.

remove K from the population and remove K from its cluster;
insert I into the population;
if there exists C_i in C, such that I is equivalent to a representative for C_i
then assign I to C_i;
else find an empty cluster C_n in C;
    assign I to C_n;

Figure 7.5: Cluster Analysis Bookkeeping algorithm

Notice, this time consuming procedure must follow after every new individual created. The algorithm is simple and can be implemented more efficient than outlined. The number of clusters can never be larger than $PopSize$. The representative of a cluster C_i can be any individual belonging to C_i, since C_i is a subset of an equivalence class. The check for equivalence can be a computational expensive task though, as is the case for functional equivalence for neural nets.

7.4 Several Cluster relations simultaneously

More definitions can be used simultaneously independently of each other, without further complication! Actually I recommend this in order not to rely solely on a single kind of observation. Simply implement as many cluster definitions that might seem interesting to monitor and maintain each kind of cluster by the Cluster Bookkeeping algorithm. The following section gives means to display the information from a GA run with the Cluster Bookkeeping using a single cluster equivalence relation. Several relations should simply be displayed independently of each other. Using several clusters gives a more reliable population analysis. And a deeper understanding of the interplay between the GA and the task.

7.5 Cluster visualization

It was difficult to design graphs to show the size of clusters. The important thing is to be able to distinguish between a diverse population with many, small clusters and a population with a few, big clusters, and a (nearly) converged population constituting a single cluster, eventually plus a few, small clusters (a fraction of the "main" cluster).

The first idea of monitoring cluster sizes is to save and monitor all cluster sizes for every generation. But this results in enormous log files and in graphs with too many lines showing the smaller clusters, when the important information is whether a few clusters contain the entire population or the population is diverse.

The second idea is to save and monitor a subset consisting of the biggest cluster sizes. It turns out that it is difficult to preserve a link between a particular cluster in the population and the size of that cluster in the logfile. Using continuous lines, this results in graphs which simply do not connect correctly. Imagine a big cluster A and a few individuals compromising cluster B. B grows steady and ousts A. If this graduation should be expressed, then the appearence of an important cluster like B, must start a backtrace of the development of the cluster in order to truly monitor the dynamic process. This is possible, but a

much easier solution is to save the biggest cluster sizes unconditionally and plot the graphs using points, giving up any ordering, and rely on the human eye's superior capability to interpret continuety of a succession of points.

This was accomplished by plotting the 2 biggest clusters. An example of a graph showing the relative size of Clusters during a 20000 generation run, see figure 7.6.

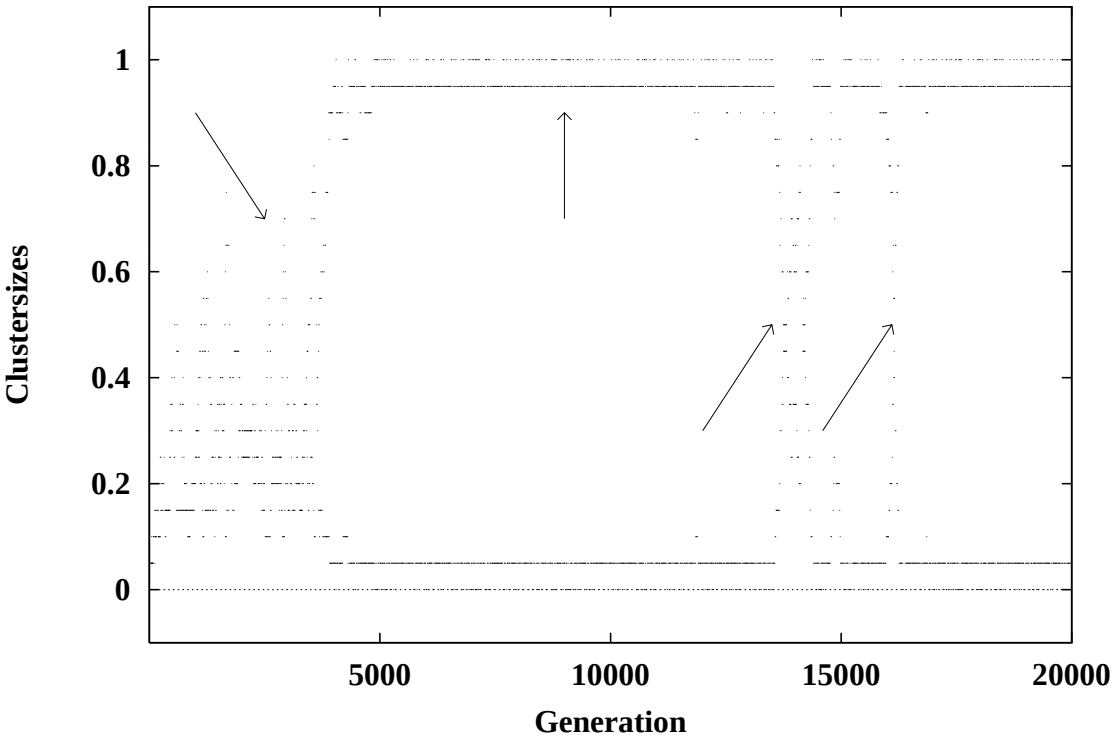


Figure 7.6: Explanatory Cluster graph.

The 1st arrow (from left) shows the biggest cluster constitutes 70 percent of the population. This leaves room for several other, smaller clusters, which are not displayed. The 2nd arrow shows the biggest cluster constitutes approximately 100 percent of the population. This leaves for room at most one individual outside that cluster. The 3rd and 4th arrows shows the phenomena of a huge cluster being "pushed out", by a new cluster of more fit individuals.

This inherits some information about the overall clustering, particular when the biggest cluster constitutes most of the population, that is, detection of convergence. But this kind of visualisation do lose information about the coming and going of *smaller*, thus less important, clusters. More sophisticated visualisation methods will reveal more detailed information. For the moment the above is sufficient.

Comparing the graphs of fitness of best individuals with the corresponding graphs of cluster sizes shows several interesting behaviours of the population. These includes coarse grained and fine grained search as well as convergence and a few examples of escape from converged state. This will be adressed in the chapter 11.

Part III

The Cluster concept in the future

Chapter 8

Population Diversity

Several methods have been developed with the purpose to avoid or delay Premature Convergence(PC). The following review of previous methods to avoid or delay convergence is partly due to [Jones 1993]:

The problem of premature convergence have been adressed by a number of authors using a diversity of techniques:

[Cavicchio 1970] (Jones reference), in his doctoral dissertation, suggested a preselction mechanism as a means of promoting genotype diversity. Preselection filters children generated, possibly picking the fittest, and replaces parent members of the population with their offspring.

[De Jong 1975] (Jones reference) crowding scheme is an elaboration of the preselection mechanism. In the crowding scheme, an offspring replaces the most similar string from a randomly drawn subpopulation having size CF (the crowding factor) of the current population. Thus a member of the population experiences a selection pressure in proportion to its similatity to other members of the population. Emperical determination of CF with a five function test set determined CF = 3 as optimal.

[Booker 1982] (Jones reference) implemented a sharing method in a classifier system environment which used the bucket brigade algorithm. If related rules share payments then subpopulations of rules will form naturally. However, it seems difficult to apply this mechanism to standard GAs. (...)

A straightforward approach to help maintain genotype diversity was introduced by [Maulding 1984] (Jones reference) via his uniqueness operator. The uniqueness operator helped to maintain diversity by incorporating a 'censorship' operator in which the insertion of an offspring into the population is possible only

if the offspring is genotypically different from all members of the population at a number of specified genotypical loci(alleles, ed.).

Somewhat later, [Goldberg et al 1987] (Jones reference) introduced the idea of shared value. The idea of sharing is very simple: if a number of strings in the current population cluster closely together, where 'closeness' can refer to the topology either of the space of phenotypes or that of genotypes, then their goal function values might be expected to be similar. If the goal function is large in this region, then a simple GA would give each string of the cluster a large selection propability, thus reinforcing the tendency for succeeding generations to further cluster in the same region - an effect which often leads not to an exhaustive examination of the optimal region, but to premature convergence. Goldberg proposed that if two or more strings were closer together than a specified 'niche width' then their goal function values should be shared between the group in some way. Let the ith member of the population have phenotype x_i and the population size be n. Then the niche count N_i of the ith member is a real number in the range $[1, n]$ computed on the basis of a characteristic function for the niche. This is one aspect of sharing which is rather *ad hoc* - specification of the niche width requires *a priori* knowledge of the search space, and to progressively refine the search the niche width must dynamically shrink at the right rate. Note that $N_i = 1$ if the string is isolated, as measured by the specified niche width, and $N_i = n$ if the niche width is taken so large that all members of the population are counted as being in the niche. The shared value of the ith string, x_i, is taken to be:

$$V_S(x_i) = \frac{v(x_i)}{N_i},$$

where $v(x)$ is the fitness function. It is the shared value which is used to compute the selection probabilities. Specifically, the probability of selecting the ith string is:

$$P_i = \frac{V_S(x_i)}{\sum_j V_S(x_j)}$$

Thus as more strings cluster in a specific region of a large goal function, their relative advantage in enhanced selection probability over other members of the population is correspondingly reduced. If a new good string is discovered in a hitherto unexplored region (thus having a low niche count), then its selection probability will be exceptionally high, thereby ensuring the rapid formation of a new, well populated niche. This ensures the rapid switching of attention, or adapbility, clearly a desirable property. (...)

In experiments[Jones 1993] one finds that after 100 generations with crossover only, good niche clustering has taken place and the selection probabilities based

on shared value are virtual equal across the population. At this point an extremely stable stochastic equilibrium is in force. The stability can be judged from the observation that if sharing is now removed ($N_i = 1, \forall i$), the overall distribution of the population in phenotype space remains essentially unchanged for several hundred generations. (...)

Mutation can be added to ensure detailed search of the established niches, and additional selection pressures such as replacing the least fit string can ensure that the least valuable niches are gradually evaluated. The actual effect of sharing can be simulated by less costly techniques such as Cavicchio preselection.

This ends the description of previous work in preventing or delaying premature convergence.

Some have tried to avoid premature convergence by posing a certain structure on the population, fx. a flat 2 dimensional array, and restrict recombination to certain neighbors. [Jones 1993] notice that the approach is biologically plausible and report several succesful applications. Jones also argues the selection is computational cheaper:

> "population members ... select their partner from a smaller group locally available ... no need to collect information globally to compute the selection probabilities ... A theoretical analysis of another diffusion GA using a 2-dimensional grid ... concludes that local dominance of highly fit schemata causes very fast initial niche formation and exploration of multiple peaks in the search space. These niches expand and compete, leading to one of the niches dominating and taking over the rest". Thus convergence is only delayed, not avoided.

Experiments with certain topological structures on the genetic population is important and interesting, because the results can be used to implement such structures using specialized hardware. For instance, a network of transputers or MIMD computers[Radcliffe 1990].

8.1 Any convergence is premature

This is the art of convergence control: we do not want the population to converge when the best individuals are not near global optimum, but when they are, we do want the population to be filled with individuals from the global valley, in order to finetune the search in that area.

> [Jones 1993]"In trying to counteract PC we are essentially trying to balance the exploitation of good solutions found so far against the exploration which is required to find hitherto unknown promising regions of the search space."

How do we know if some of the individuals are in the global valley (bassin of attraction containing the global optimum)? The answer is we can't know, so the only sensible strategy is to avoid convergence of the population at all. Therefore the label *'Any convergence is premature'.*

This inspired me to define the concept of clustering in a population. The idea is that a highly diverse population contain many clusters. And that a converged population, constitutes a single clusters, perhaps except a few individuals which have not been "pushed out" by the dominating cluster.

The following suggests a method developed by the author to gain some control with the convergence behaviour of genetic algorithms in general. It suggests a method to keep GA populations diverse upto certain subspaces defined by any equivalence relation over the search space. It is important to notice that the goal is not to keep the population as diverse as possible. Rather should the population be allowed to cluster some, but not all, of the individuals in order to finetune search in any area. Whether there is a certain 'critical number of individuals' needed to finetune search in certain areas is an open question. Keeping population diverse w.r.t. predefined subspaces does not exclude possibility of convergence within such a subspace though - a subject which is also discussed.

8.2 Artificial Diversity

To end the chapter of Population Diversity, a universal algorithm to keep GA populations diverse is described using pseudo-pascal. The kernel idea builds on the CAT.

Artificial Diversity(AD) - a simple solution to sustain diversity. While previous methods all attempts to keep a population diverse indirectly by means of splitting the population in several niches, or punishing close individuals by sharing, it is here suggested to keep populations diverse by the principle "There is only room for some of us here". The principle is explained using the neural net optimisation task adressed in the experiments, as an example. But it can easily be extended to any GA task.

A simple solution could be to put a limit on the number of similar individuals allowed in the population. Does this conflict with the fundamental theorem of genetic algorithms: "Individuals belonging to schemata with higher than average fitness will reproduce exponentially" ? No, putting a limit on number of similar individuals is no more restrictive than putting a limit on number of indiviuals in the population, $PopSize$. Introducing an upper limit on number of similar individuals will not allow any number of individuals to be searching the same neighbourhood and the best solution will not be "fine tuned" to the same extent. On the other hand, the risk of suboptimal convergence can be reduced.

As is the case in the Cluster Analysis Tool it is necessary to define an equivalence relation reflecting our intuition of similarity in the search space. A clusterdefintion can be used as a means to keep the population diverse up to the "graineess" of the clusterdefinition, i.e. number of equivalence classes[1]. The idea is to introduce an upper limit on the number of similar individuals allowed, $MaxClusterSize$. Then if a child to be inserted in the population belongs to the same equivalence class as some individuals already in the population and there are $MaxClusterSize$ of these similar individuals, then the new child should not allow the cluster to grow. Therefore a member of that cluster must be replaced - the least fit clustermember, for example. If the cluster has not reached maximum size or there is no individual similar to the new child, then the child simply replace some individual - the least fit populationmember, for example.

[1]The intuition is: a relation inducing many classes is more (fine) grained.

8.2.1 The algorithm

The following is a pseudo pascal version of the algorithm, very similar to the Cluster Bookkeeping algorithm. Assume the Replace Strategy is Replace Least Fit:

Artificial Cluster Diversity

C is a set of clusters C_i, $i \in \{1, ..., PopSize\}$.
C_i is a set of individuals belonging to the same equivalence class.
Let I be the child to be inserted in the population.

```
if there exists C_i in C, such that I is equivalent to a representative for C_i
then
    if |C_i| = MaxClusterSize
    then
        let the least fit individual in C_i be J;
        replace J with I in the population;
        replace J with I in C_i;
    else
        let the least fit individual in the population be J;
        replace J with I in the population;
        remove J from its cluster;
        insert I in C_i;
else
    find an empty cluster C_n in C;
    let the least fit individual in the population be J;
    replace J with I in the population;
    remove J from its cluster;
    insert I in C_n;
```

Figure 8.1: Artificial Cluster Diversity algorithm

8.2.2 Future extensions to Artificial Diversity

As is the case in the CAT, several cluster relations can be defined and run simultaneously. This complicates the algorithm slightly as follows. The tricky part is that each individual now belongs to several equivalence classes. As many as relations defined. None of the clusters must contain more than $MaxClusterSize$ individuals. As an example let there be two equivalence relations C' and C''. Assume a new child belongs to class C_i in C' and C_j in C''. If $|C_i|$ or $|C_j|$ is $MaxClusterSize$ then the child must replace an individual in $C_i \cap C_j$ to maintain the property of $MaxClusterSize$ individuals in each single cluster.

In effect this is exactly the same as taking the "intersection" of all the equivalence relations forming a "relation containing them all", R:

$$\forall x, y: \quad R(x,y) \iff C'(x,y) \wedge C''(x,y)$$

Maintaining the $MaxClusterSize$ property of R only, is thus equivalent to the more complicate case of maintaining several cluster definitions!

A very interesting extension of artificial diversity is to develop a method to allow dynamic populationsize, dependent on the population dynamics. If fx. all Clusters have the maximum allowed size, it might be time to extend the population. Otherwise if no cluster has reached the maximum allowed size, it might indicate that the population need to be shrinked to enhance clustering and thus fine tuning in some areas. The reader is encouraged to try this.

8.2.3 Cluster Analysis and Artificial Diversity

Cluster Analysis(CA) and Artificial Diversity(AD) origins from the same idea and are very similar, therefore a quick comparison is appropriate:

CA receipe:
- define equivalence relation(s);
- maintain cluster bookkeeping;
- report properties of clusters on the fly;

AD receipe:
- define equivalence relation(s);
- maintain cluster bookkeeping;
- restrict clustering to $MaxClusterSize$;

Part IV

Experiments and Results

Chapter 9

Visions and limitations

My vision is an automatic neural network generator which given sets of input and output patterns delivers a net with a user specified performance, within the constraints of a user specified time. We might want nets which are doing well on some training set(s), doing well on some unseen test set(s), is fast to use, requires a minimum of memory, etc. This possibly includes unlimited topology, any activation function etc.

The limitations on the experiments, are:

- 3-layered fully connected nets. No connection from input to output layer.
- Bias to hidden and output layer.
- Number of input and output nodes is fixed by the task at consideration.
- Number of hidden nodes is dynamically altered by the GA, only limited by a user-specified maximum - $MaxNoOfHidden$. Minimum 1 hidden node.
- Potential function is dotproduct, i.e. summation nodes.
- Activation function is the well known sigmoid $\frac{1}{1+exp(-x)}$.
- Optimization w.r.t. RMSE, thus learning is emphasized.
- No optimization w.r.t. ability to generalize, no penalty on size of nets.

They are chosen to limit the study reasonably. Though it can be argued that the ability to optimize w.r.t. number of connections or hidden nodes, is an important part of any neural network generator, it is omitted. Study of evolution dynamics is considered crucial here and has received corresponding effort.

9.1 Setup

This describes in detail the genetic algorithm used in the following experiments.

9.1.1 Choice of genetic algorithm

The desired approach is a recombination operator combining topology of 2 nets, simply copying the corresponding weights. Training of the nets is taken care of by mutation operators, which adjust certain groups of weights by a suitable random distribution, after an idea of [Montana & Davis 1989].

The Steady State reproduction method is used with the modification that whenever an operator creates more than one child, they are both evaluated and the least fit discarded. This modification was chosen due to empirical results not discussed here. If that child is ghenotypically different from both its parents, it is inserted, otherwise it is discarded. Note that every child generated counts as exactly one generation, even if it is discarded. This is a limited edition of what [Davis 1991] call 'steady-state without duplicates', which checks the entire population for a duplicate of the child(ren) to be inserted. Without this check, operators which tends to make identical copies, may cause premature convergence [Karlsen et al 1992]. Strategy of reproduction is chosen to be Steady State, because it highly facilitates the cluster analysis method, and because of its success in [Syswerda 1991] and in [Karlsen et al 1992]. As deletion method, exponential ranking is chosen, due to the results in [Syswerda 1991].

9.1.2 Modern Representation - Phenospace only.

Each individual represents the topology and weights of a fully connected, 3-layer perceptron, with a number of hidden nodes. No transformation from genotype to phenotype is done as each weight is represented as it is - a real number. Bias to all hidden and output nodes. No connection from input to output layer. The number of input and output nodes is fixed by the task, the number of hidden nodes is altered by evolution.

The actual arrangement of the weights is unimportant because each operator exactly specifies its operations in terms of which weights are manipulated and how . This allows for a well structured representation, unlike a flat string representation.

9.1.3 Evaluation of fitness

The process adopts socalled batch mode, i.e. all patterns have been presented to the network and RMSE is calculated. This is different from most sequential training algorithms, like sequential BP, where weights are updated after each training pattern is presented to the network. Generalisation has not been of concern in the experiments, as argumented below.

Several has reported succes with penalty on complexity of topology. The fitness function simply contains a penalty term, penalizing nets with relatively many connections/nodes. But the optimal nets happended to be small, caused by the simple tasks considered. It is unfortunately possible to have an inferior topology with a set of weights, such that it has lower error than a superior topology with arbitrary weights. Therefore it is possible for an inferior topology to outperform better topologies, just because it found good weights at an early stage. Here is a problem with penalization on number of connections. If the solution searched, happen to be a widely connected net, the GA may never discover this solution because it is outperformed by smaller nets with (nearly) optimal weights, rewarded by sparseness (and low error).

The author has no answer to this problem and has therefore decided not to include any kind of penalty on connectivity in the experiments. Therefore the algorithm is expected to find nets with more hidden nodes and non-zero connections than necessary to learn the training sets. To limit growth reasonably there is simply an artificial ceiling on number of hidden nodes associated with each experiment.

For simplicity the only property adressed is learning. Thus the only term in the fitnessfunction is RMSE.

9.1.4 Population initialization

The initialization process allows for different number of hidden nodes and randomly distributed weights. To obtain some initial diversity on the size of hidden layers, each individual is allocated room for 1 hidden node more than the previous, modulo the maximum allowed, $MaxNoOfHidden$. The number of hidden nodes, n_h, is attached to each individual. Thus there is a uniform distribution on number of hidden nodes in the initial population. $MaxNoOfHidden$ is specified by the user and task dependent, but could be estimated using resent theorems on computational strength of 3-layered nets. Such considerations are not within the aim of this thesis.

Each weight is set to $ln(U)$, where ln is the natural logarithm and U is uniformly distributed on the interval]0;1]. Then $ln(U)$ has probability distribution given by $f(x) = exp(-x)$. The sign of the weight is chosen fifty-fifty. "This is different from the initial probality distribution of the weights usually used in backpropagation', which is a uniform distribution between -1.0 and 1.0. The distribution reflects the emperical observation by researchers that optimal solutions tend to contain weights with small absolute values but can have arbitrarily large absolute values"[Montana & Davis 1989].

To make this less dependent on the random number generator, the actual implementation used an equivalent method. Let U be uniformly distributed on [-1,1]:

$$w_{ij} = V = \begin{cases} -ln(U), & U > 0 \\ ln(-U), & U < 0 \\ 0, & U = 0 \end{cases}$$

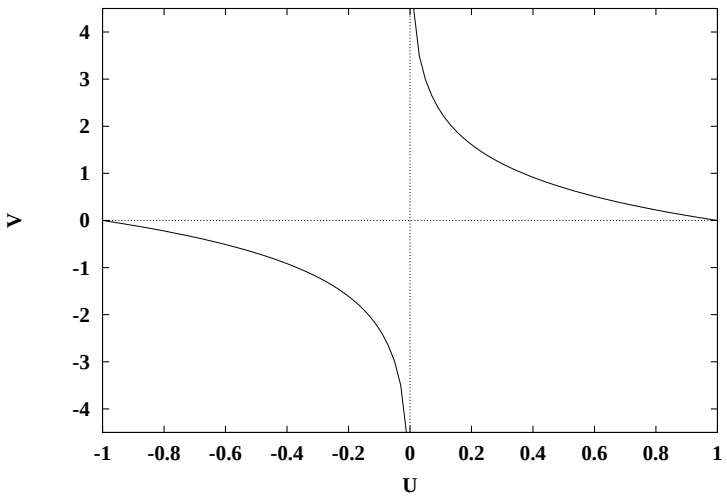


Figure 9.1: Weight initialisation function

This requires only one call to a random number generator, and has the same probability distribution as above.

9.1.5 Recombination explores weighted topologies

The recombination operator X is defined in chapter 5. The recombination rate P_{RECOMB} is part of the control set $\mathcal{K}_X$ for X.

9.1.6 Mutation explores weights

The operators are slight variations of operator MUTATE-NODES suggested by [Montana & Davis 1989]: The MUTATE-NODES is supposed to alter the weights in a feedforward net in order to improve mapping error.

> "This operator selects n non-input nodes of the network which the parent chromosome represents. For each of the in-going links to these n nodes, the operator adds to the link's weight a random value from the initialization probability distribution. It then encodes this new network on the child chromosome. The intuition here is that the in-going links to a node form a logical subgroup of all the links in terms of the operation of the network. By confining its changes to a small number of these subgroups, it will make its improvements more likely to result in good evaluation. In our experiments, $n = 2$."

Notice, MUTATE-NODES also alter the bias in the same manner. The probability of picking a hidden node is $n_h/(n_h + n_o)$, and $n_o/(n_h + n_o)$ for output nodes. In nets with many hidden nodes and few output nodes, the outputnode will rarely be "trained". It seems desireable to gain more control with the process of picking nodes. This can be done using 2 operators, which are exactly like MUTATE-NODES, but is only altering either a hidden node, or an output node: MUTATEOUT, MUTATEHIDD, see figure 9.2. This allows for experiments with different frequencies of application, by changing the weight associated with each operator, i.e. the operator rates. Such experiments are not part of this work, though.

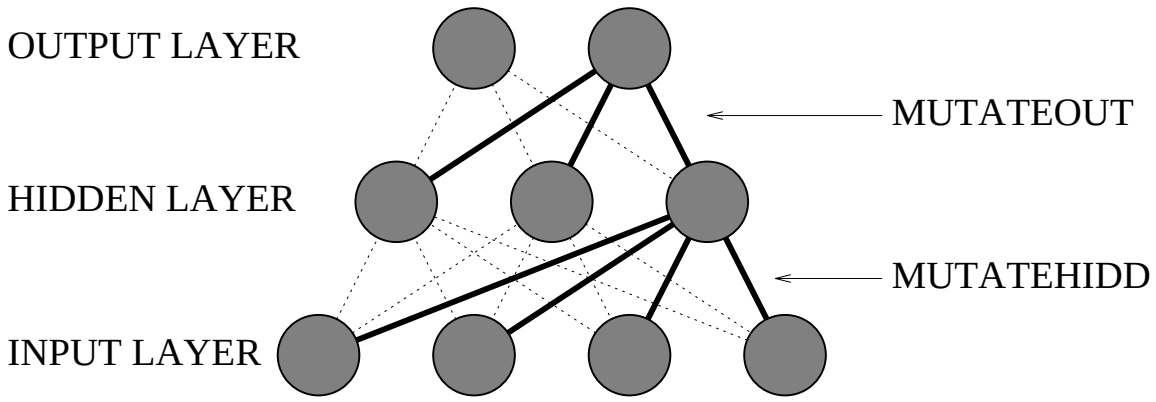


Figure 9.2: The weights mutated by the mutation operators.
Note: The biases (not showed) are treated like ordinary weights.

The mutation rate $P_{MUTATEOUT}$ are part of the control set $\mathcal{K}_{MUTATEOUT}$. The mutation rate $P_{MUTATEHIDD}$ are part of the control set $\mathcal{K}_{MUTATEHIDD}$.

The following 'Experiments' chapter reviews the exact contents of the tasks used in the experiments. Then the results are presented in 'Results' and thoroughly described in 'Presentation'. On this basis the experiments are reviewed and scientifically motivated in 'Interpretation'.

Chapter 10

Experiments

10.1 8-3-8 multiplexer experiments

The 8-3-8 task is straight forward and is mainly used to confirm the computational capability of a 3-layered feedforward neural net, or the performance of learning algorithms. The task is to map the 8 binary patterns consisting of a single 1 and seven 0s to the identity:

$$
\begin{aligned}
00000001 &\longrightarrow 00000001 \\
00000010 &\longrightarrow 00000010 \\
00000100 &\longrightarrow 00000100 \\
00001000 &\longrightarrow 00001000 \\
00010000 &\longrightarrow 00010000 \\
00100000 &\longrightarrow 00100000 \\
01000000 &\longrightarrow 01000000 \\
10000000 &\longrightarrow 10000000
\end{aligned}
$$

The topology was 8-3-8, 8 input, 3 hidden nodes, and 8 output nodes, see figure 10.1.

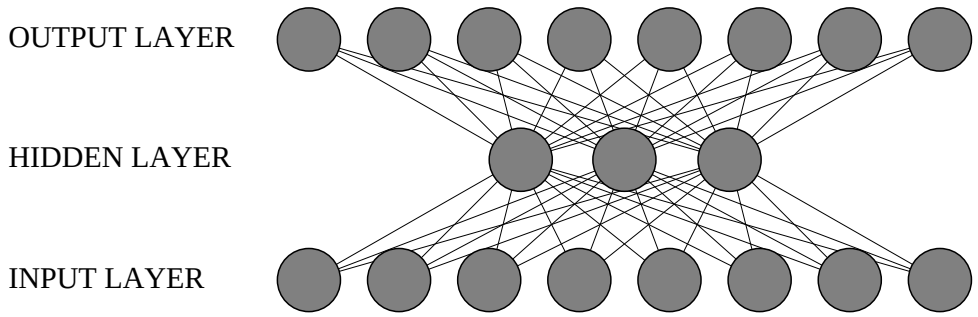


Figure 10.1: The 8-3-8 topology.

Exp 1
Popsize = 20. Generations = 20000. MaxNoOfHidden = 3.
$P_{RECOMB} = \frac{100}{120}$, $P_{MUTATEHIDD} = \frac{10}{120}$, $P_{MUTATEOUT} = \frac{10}{120}$.
Replacement Strategy is Replace Least Fit.

Exp 2
As Exp 1, but Replacement Strategy has changed for the mutation operators.
Popsize = 20. Generations = 20000. MaxNoOfHidden = 3.
$P_{RECOMB} = \frac{100}{120}$, $P_{MUTATEHIDD} = \frac{10}{120}$, $P_{MUTATEOUT} = \frac{10}{120}$.
Replacement Strategy for the mutation operators is Replace Parent!

Exp 3
As Exp 2, but Popsize is doubled.
Popsize = 40 !, Generations = 20000. MaxNoOfHidden = 3.
$P_{RECOMB} = \frac{100}{120}$, $P_{MUTATEHIDD} = \frac{10}{120}$, $P_{MUTATEOUT} = \frac{10}{120}$.
Replacement Strategy for the mutation operators is Replace Parent.

Each run in Exp 2 has the same initial population as the corresponding run in Exp 1, due to determent choice of randomseed and identical Popsize.

10.2 Two-Spiral experiment

For the two-spirals problem, the network has two continuous valued input nodes, one output node, and a number of hidden nodes, see figure 10.2.

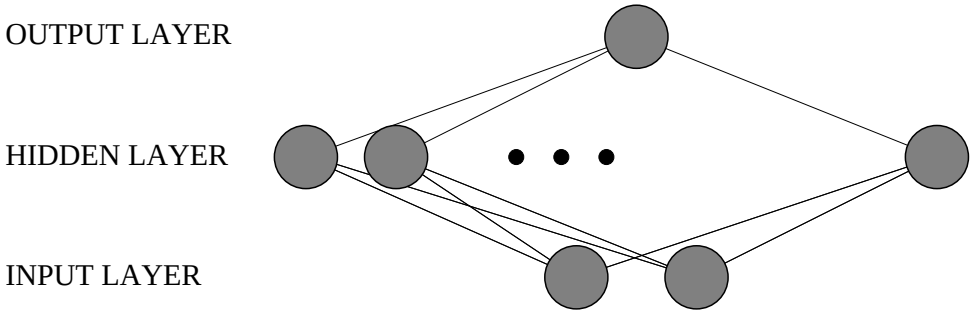


Figure 10.2: The topology used for the Two-Spiral task.

The training set consists of 192 pairs of X,Y coordinate values within a square. The input patterns are arranged in two interlocking spirals that circumscribe the origin three times. Exactly half of the input patterns are selected from one spiral with a 1 output, whereas the training patterns from the other spiral results in a 0 output, see figure 10.3.

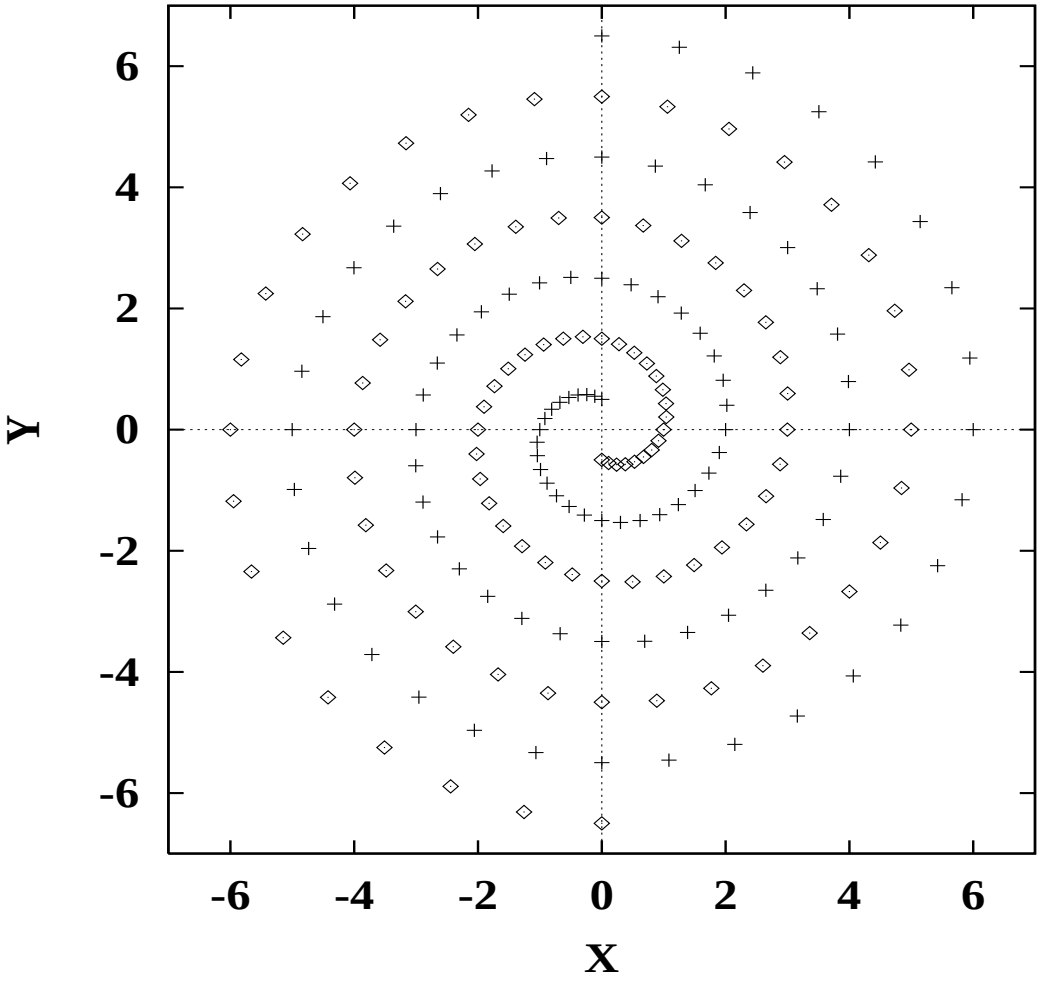


Figure 10.3: The Two-Spiral input patterns.

In general this problem is very hard to solve using traditional back-propagation. [Lang and Witbrock 1988] report obtaining a solution to the two-spirals problem using backpropagation on a network with 3 hidden layers, each with 5 hidden nodes. A salient feature of this particular network is that each node receives connection from every node in every earlier layer, not just from the immediately preceeding layer.

[Fahlman and Lebiere 1990] found that the Cascade-Correlation learning is able to solve the two-spiral problem consistently. [Karunanithi et al 1992] report a Genetic Cascade Learning with the advantage of training without use of gradient information. The algorithm was able to learn more than 96 percent of the training examples. The succesful, evolved nets had between 16 and 40 hidden nodes. Cascade-Correlation nodes usually have connections to all previous nodes. The maximum number of hidden nodes was arbitrarily set to 40. [Karunanithi et al 1992]"The computational time needed to solve the 2-spiral problem with Genetic Cascade Learning was much greater than that of Cascade Correlation algorithm".

[Robbins 1993] used a maximum number of 20 hidden nodes for optimizing topology on a reduced version of TwoSpiral. The maximum number of hidden nodes was set to 24 in Exp 4:

Exp 4
Popsize = 40. Generations = 20000. MaxNoOfHidden = 24.
$P_{RECOMB} = \frac{100}{120}$, $P_{MUTATEHIDD} = \frac{10}{120}$, $P_{MUTATEOUT} = \frac{10}{120}$.
Replacement Strategy for the mutators is Replace Parent.

Chapter 11

Results

Each experiment consist of 8 independent batch runs. For each 100th generation, that is 100 new created individuals, three kinds of measures were dumped. The current best individual in the population is called the elitist in the following.

The measures were: Root Mean Square Error(RMSE) of the elitist, percent correct output node activations($\%Correct$) of the elitist, and the size of the 2 biggest cluster at every generation(Clusters).

Informal testing showed that in all experiments most runs occasionally developed a hidden layer with MaxNoOfHidden hidden nodes and all connections had non-zero weights, thus the hidden nodes were fully connected. This is expected because the fitness function did not include any terms on connectivity.

To measure RMSE of a net the whole training set is propagated through the network and calculated by:

$$\epsilon(o,t) \ := \ \sum_{j \in O} (o_j - t_j)^2$$

$\mathcal{E}$ is defined as RMSE, over training patterns $\mathcal{T}$:

$$\mathcal{E} \ := \ \sqrt{\frac{1}{|\mathcal{T}|}(\sum_{r=1}^{|\mathcal{T}|} \epsilon(o^{(r)}, t^{(r)})}$$

Notice, fitness of an individual is RMSE.

An example of a graph showing RMSE of the best individual during a 20000 generation run is in figure 11.1. The RMSE graph is 'stalling' if part of the graph shows little or no progress, as indicated by the arrow.

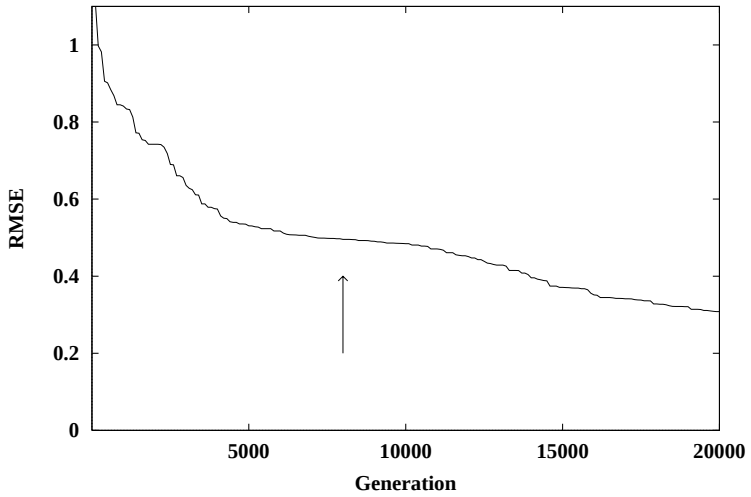


Figure 11.1: The RMSE of the best network during a Run.

To measure $\%Correct$ the whole training set was propagated through the best network and the number of correct output activations was calculated by:

$$\#Correct = \sum_{r=1}^{|\mathcal{T}|} \sum_{j \in O} Correct(o_j^{(r)}, t_j^{(r)}),$$

where

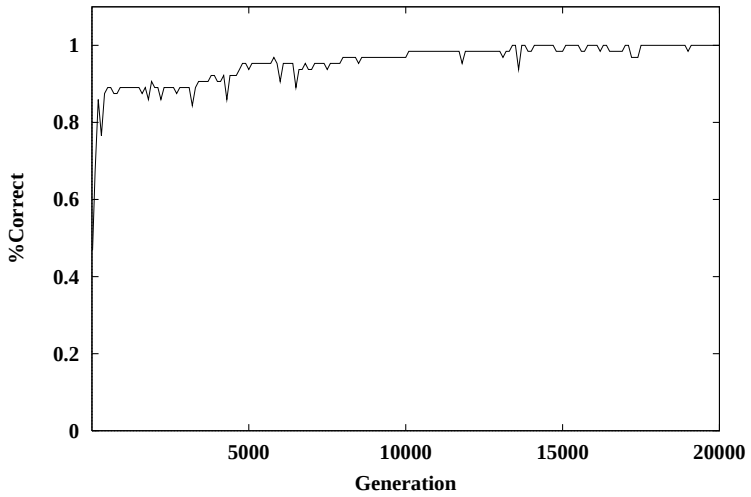
$$Correct(Output, Target) = \begin{cases} 1, & \text{if } (Target = 1 \text{ and } Output > 0.6) \\ 1, & \text{if } (Target = 0 \text{ and } Output < 0.4) \\ 0, & \text{otherwise.} \end{cases}$$

$\%Correct$ is defined as

$$\%Correct = \frac{\#Correct}{|\mathcal{T}| * |\mathcal{O}|},$$

where $|\mathcal{T}|$ is number of patterns and $|O|$ is number of output nodes. We say the network has learned the entire training set if and only if $\%Correct = 1.0$.

An example of a graph showing *%Correct* of the best individual during a 20000 generation run is:



To measure Clusters the Cluster Bookkeeping algorithm, described previously, was used to maintain a set of cluster sizes, which was updated for each new individual created.

For each experiment RMSE, %Correct, and Clusters are showed. To compare different runs of the same experiments, all 8 independent runs are graphed on one page. This allows direct comparison, instead of simply averaging the results or similar kinds of presentation. Visual comparison of RMSE, %Correct, and Clusters is easily done for a particular experiment, and also inbetween experiments.

11.1 Presentation

This section gives the reader a good feeling for the interpretation of the graphs and an overview of the output. The section can be skipped on a first reading. The presentation circumscribe every experiment, the 3 kinds of measures, RMSE, %Correct, and Clusters, and gives a short comment on the graphs.

All Clusters graphs have one thing in common, which is not obvious from the graphs: in the initial population there was never two individuals belonging to the same cluster. This is partly a matter of coincedence, and partly reflects that the current choice of cluster relation is so fine grained, that 20 or 40 randomly created individuals hardly belongs to the same class.

Exp 1

RMSE graphs.
All graphs show that the initial best individual has a RMSE >= 1.0. Run 1, 6, and 7 are stalling at the end. At 20000 generations Run 1 is worst with RMSE > 0.5, Run 4 and 8 are the best with RMSE < 0.1, and the remaining have a RMSE between 0.2 and 0.3.

%Correct graphs.
Very rapidly all Runs show $\%Correct > 80\%$. Finally, Run 1 is worst with $\%Correct$ about 95%, Run 4 and 8 are the best with $\%Correct = 100\%$, and the remaining have $\%Correct$ about 97.5%.

%Correct and RMSE of the best individual for all Runs of Exp 1:

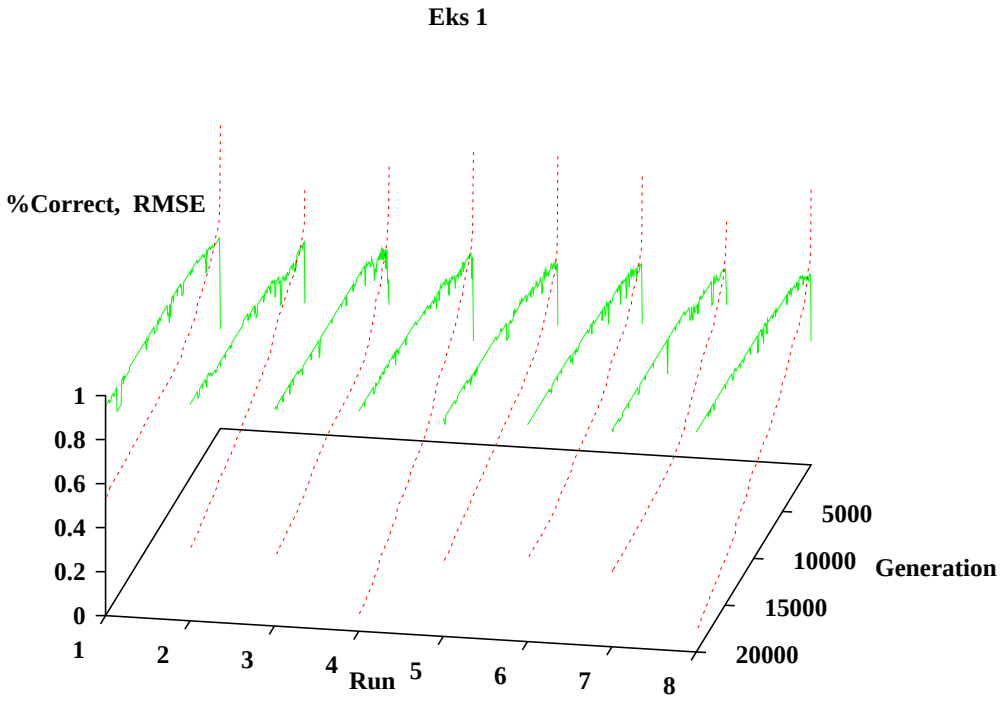


Figure 11.2: %Correct and RMSE of Exp 1.

Cluster graphs.
Run 1 show that after only a few hundred generations a cluster is formed which dominates the population. By dominant is meant 19 or 20 individuals in a population of size 20. Then there is some shifting of dominant clusters until the 7000th generation where a single cluster is dominant for the remaining of the run, though a few individuals outside the dominant cluster have limited succes around the 8300th, the 18000th, and the 20000th generation.

Run 2 never form any dominant cluster for 8000 generations, though a few generations show cluster sizes above half the population size. Then some shifting of dominant clusters sustain the run.

Run 3 show a lot of shifting in the search space, without convergence until the 4000th generation where so many individuals are attracted to the same functionality that they become a dominant cluster. The cluster is dominant until a shift to another dominant cluster occurs around generation 14000 and again 16000.

Run 4 has no dominant clusters until the 5000th generation. This lasts until the 6000th generation, where an oscillation between dominant and nondominant starts. This is replaced by a dominant cluster after the 14000th generation.

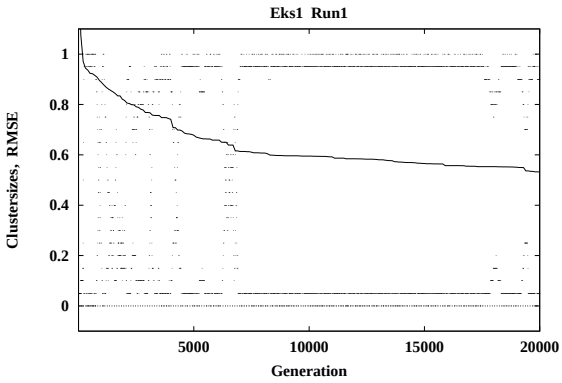
Run 5 is similar to Run 2.

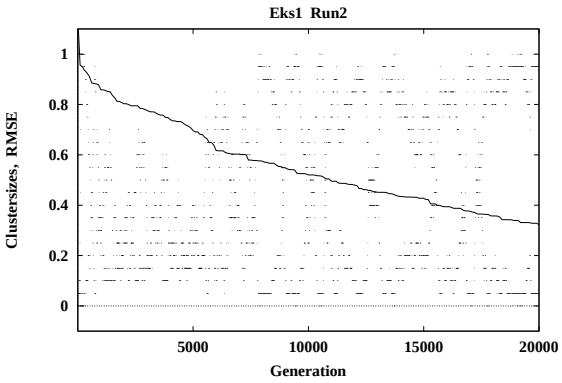
Run 6 is similar to Run 4.

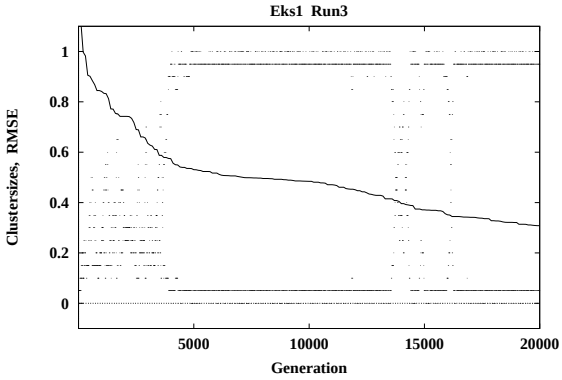
Run 7 is similar to Run 1, but only one cluster ever becomes dominant.

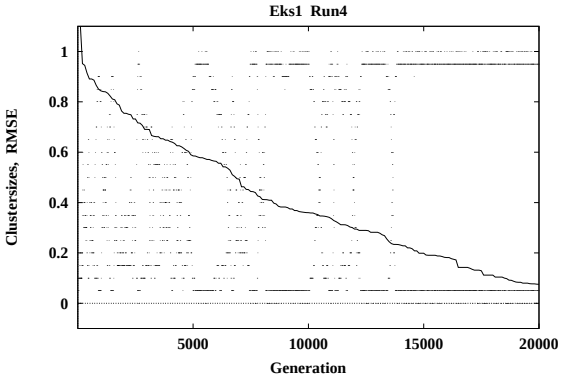
Run 8 is similar to Run 4.

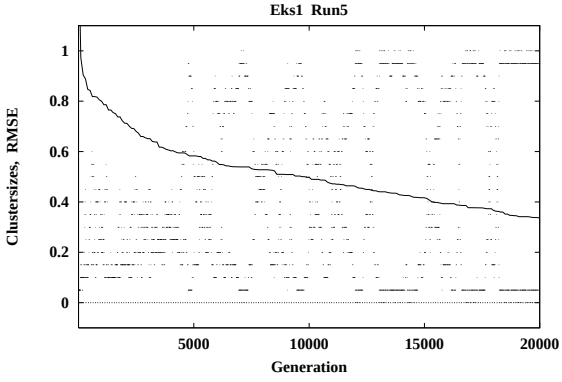
Turn to the next page for an overview of the Cluster graphs and RMSE of Exp 1:

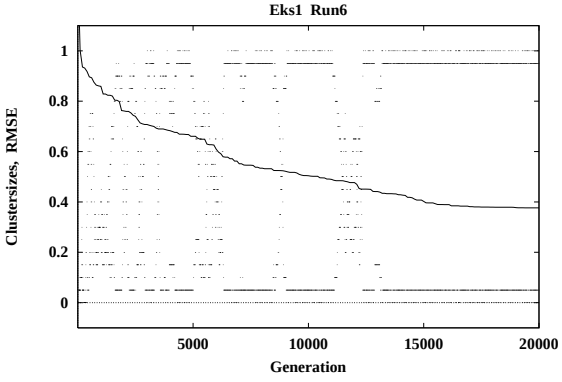

Eks1 Run1
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000

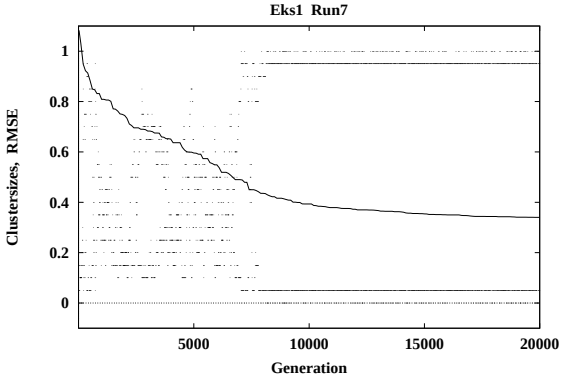

Eks1 Run2
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000

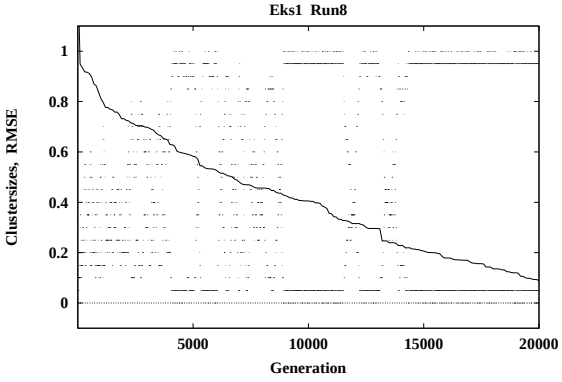

Eks1 Run3
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks1 Run4
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks1 Run5
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks1 Run6
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks1 Run7
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks1 Run8
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000

Exp 2

RMSE graphs.
All graphs show that the initial best individual has a RMSE >= 1.0. Run 1 and 5 are stalling intermediately. Run 8 is stalling in the end. After 20000 generations Run 1 is worst with RMSE > 0.5, Run 2 is best with RMSE < 0.25, and the remaining have a RMSE between 0.3 and 0.5.

%Correct graphs.
Very rapidly all Runs show $\%Correct > 80\%$. Finally, Run 2 and 4 are best with $\%Correct = 100\%$, and the remaining have $\%Correct$ about 97.5%.

%Correct and RMSE of the best individual for all Runs of Exp 2:

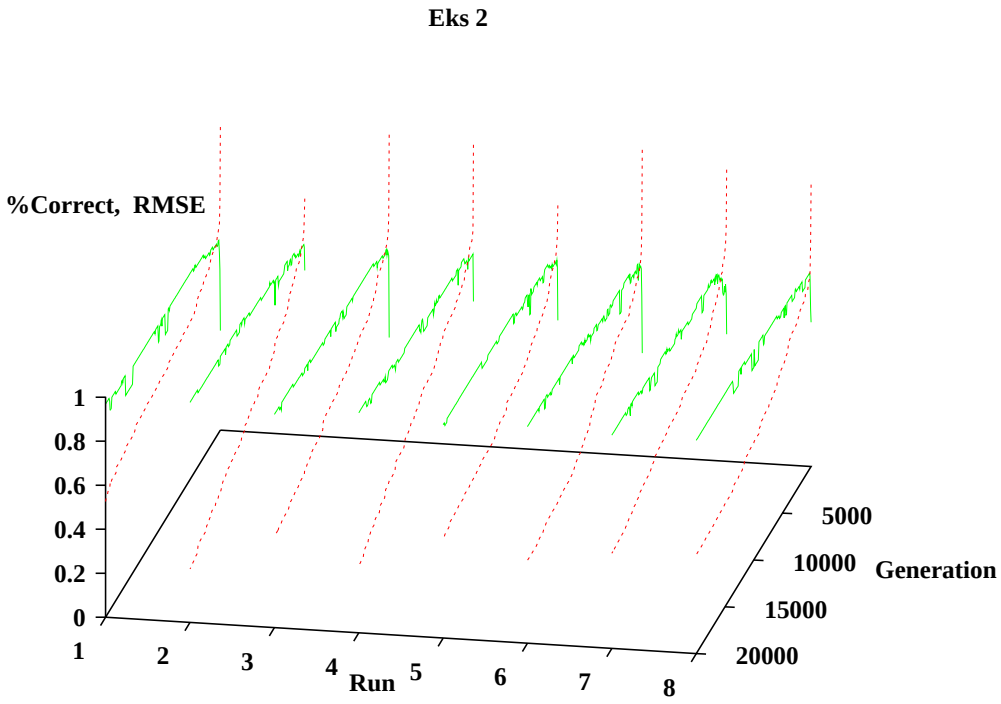
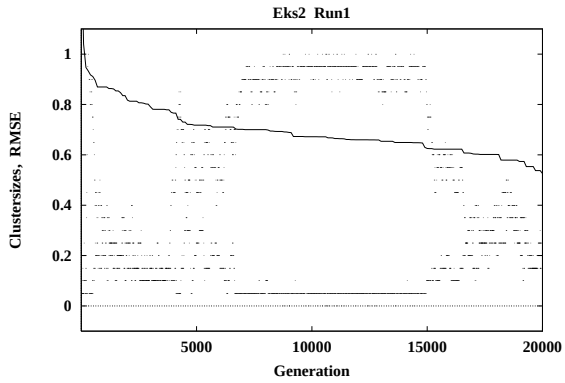
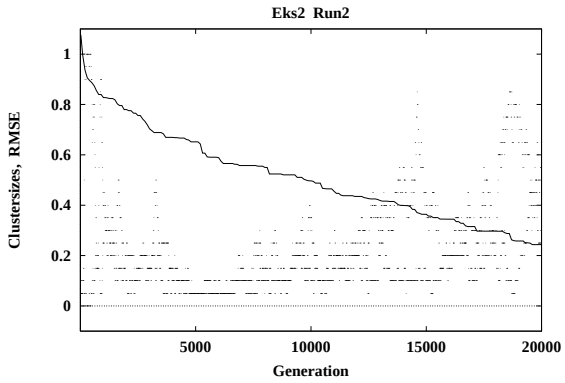
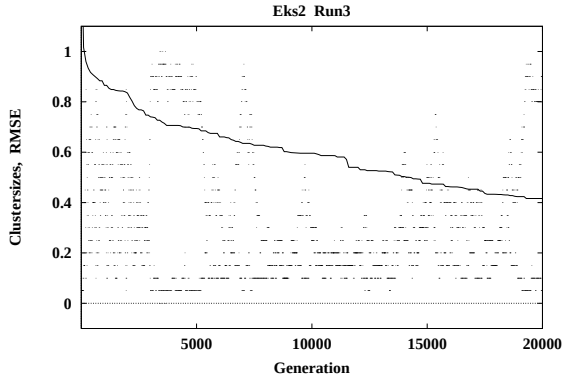


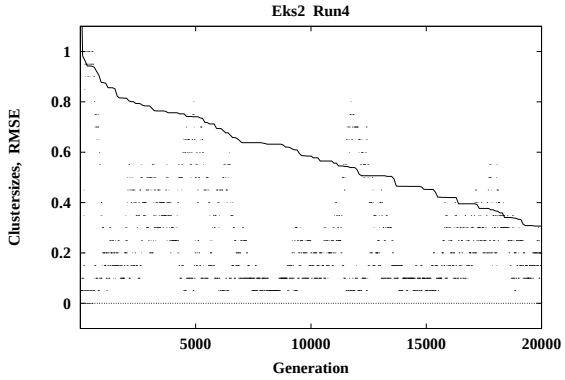
Figure 11.3: %Correct and RMSE of Exp 2.

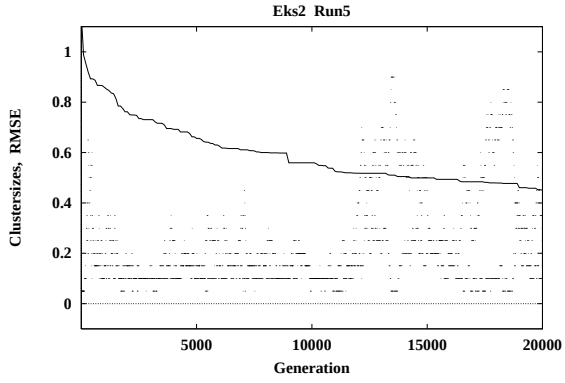
Clusters graphs.
Only Run 1 and 3 shows occurrence of a dominant cluster which only sustain for some period. The other Runs show varying tendency to cluster:

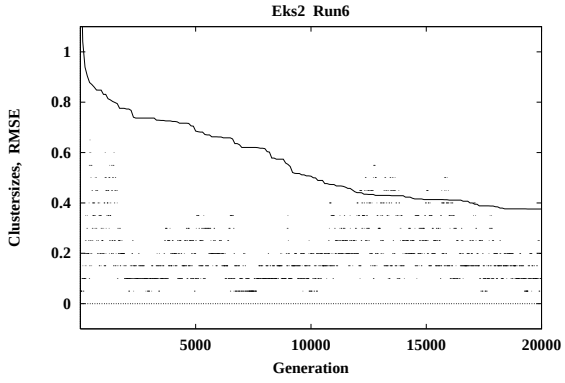

Eks2 Run1
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000

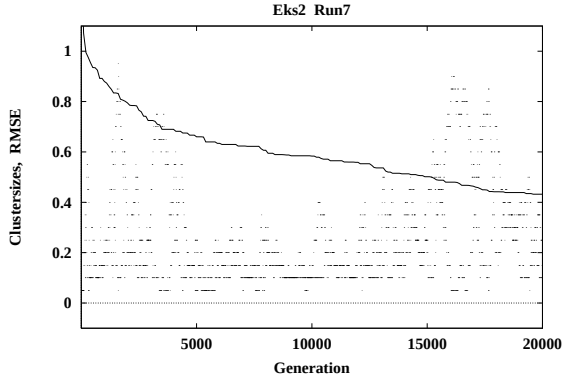

Eks2 Run2
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000

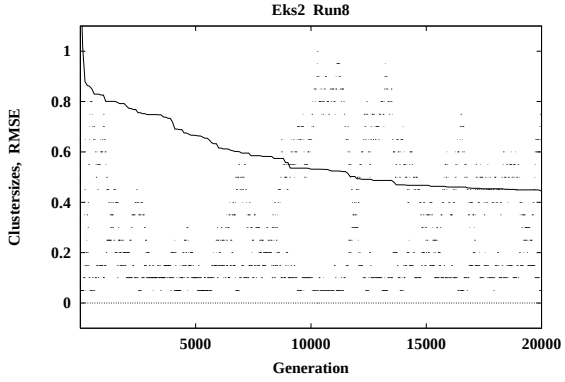

Eks2 Run3
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks2 Run4
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks2 Run5
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks2 Run6
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks2 Run7
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks2 Run8
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000

Exp 3

RMSE graphs.
All 8 graphs show that the initial best individual has a RMSE > 1.0. None of the RMSE graphs seem to stall. After 20000 generations all graphs range between 0.4 and 0.6.

%Correct graphs.
Very rapidly all Runs show $\%Correct > 80\%$. Finally all Runs have $\%Correct$ between 90% and 95%.

%Correct and RMSE of the best individual for all Runs of Exp 3:

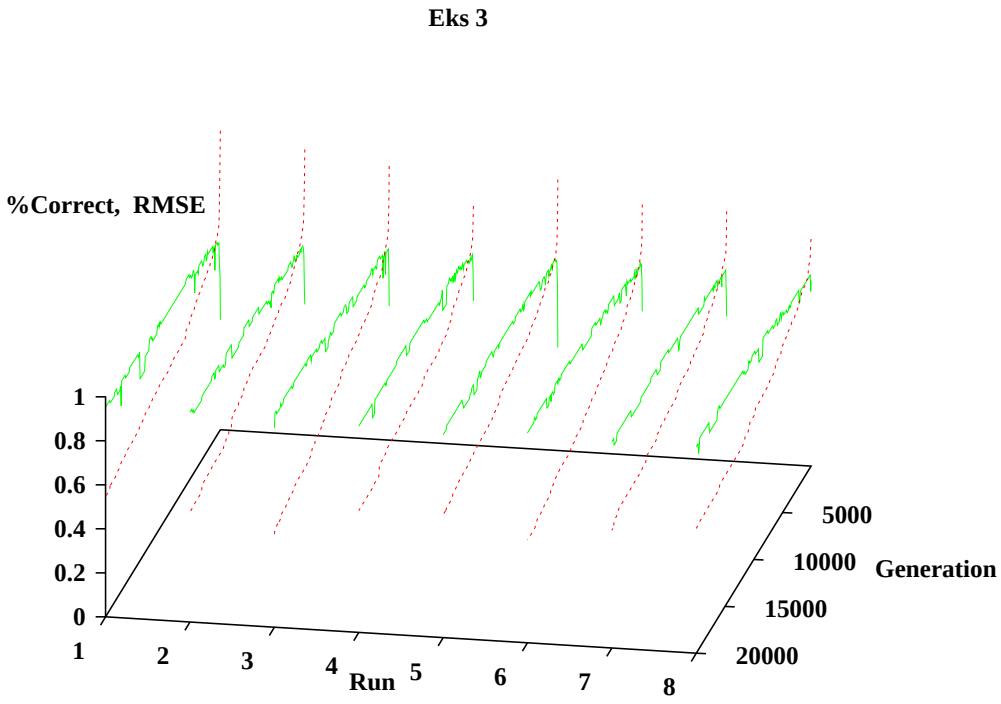
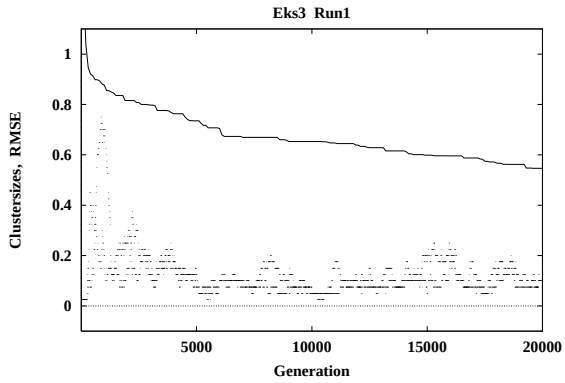
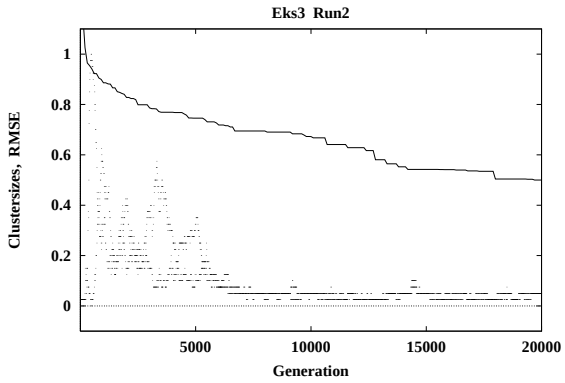
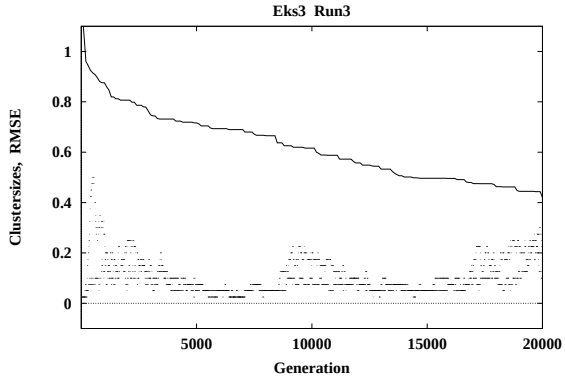


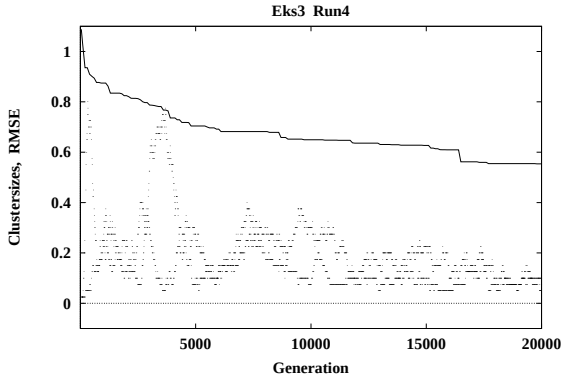
Figure 11.4: %Correct and RMSE of Exp 3.

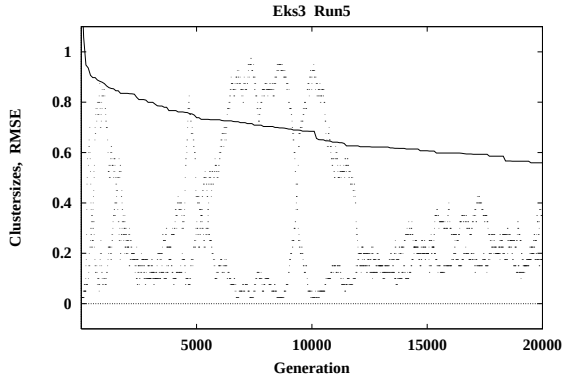
Clusters graphs.
All graphs show cluster sizes below half of population size, except at the beginning of runs. Run 5 seems unusual by showing a nearly dominant cluster from generation 6000 which is taken over by another cluster from generation 9000 to 12000:

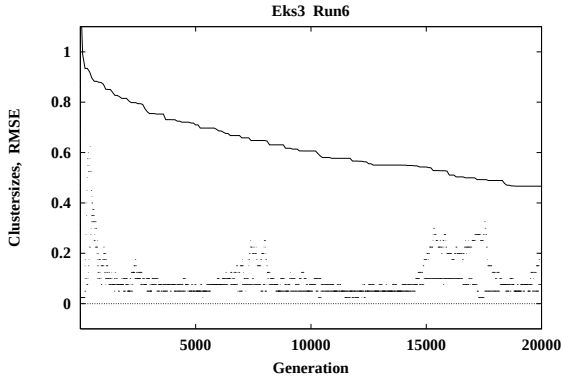

Eks3 Run1
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000

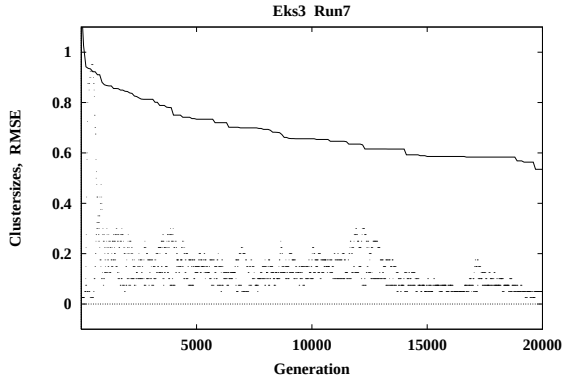

Eks3 Run2
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000

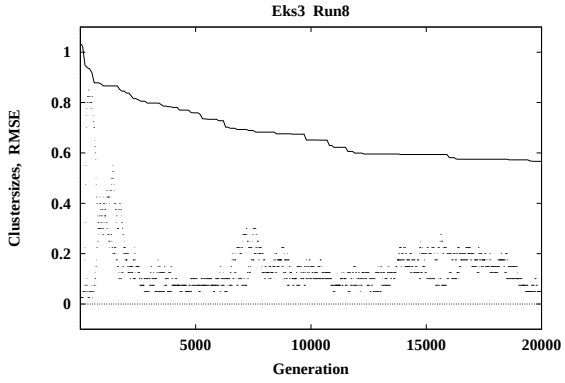

Eks3 Run3
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks3 Run4
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks3 Run5
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks3 Run6
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks3 Run7
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks3 Run8
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000

Exp 4

RMSE graphs.
All 8 graphs show an approximately linear descenting RMSE from slightly above 0.5 to slightly below 0.5.

%Correct graphs.
Run 2, 3, 4, 6, and 7 shows some random "bursts" of $\%Correct$ classification, ranging between 20 and 40 percent. The final $\%Correct$s are all below 40%.

%Correct and RMSE of the best individual for all Runs of Exp 4:

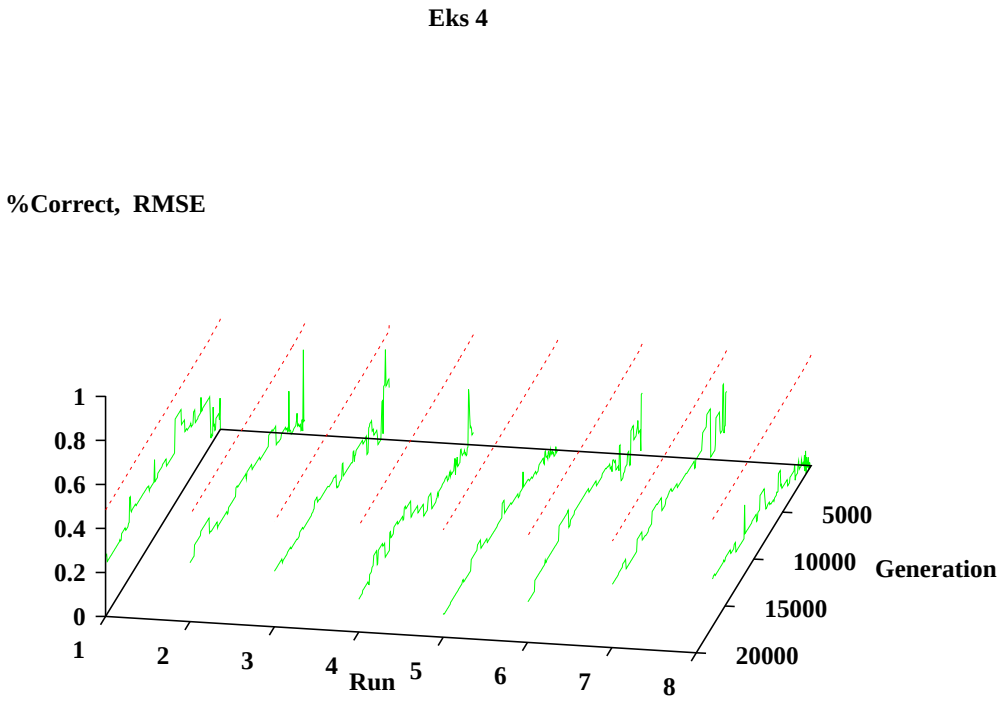
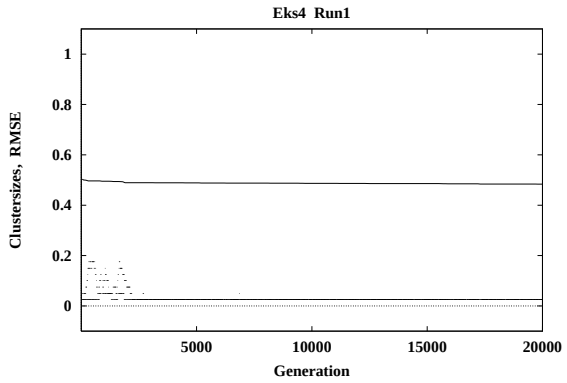
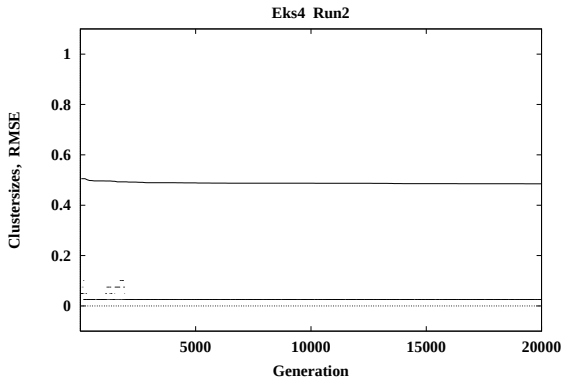
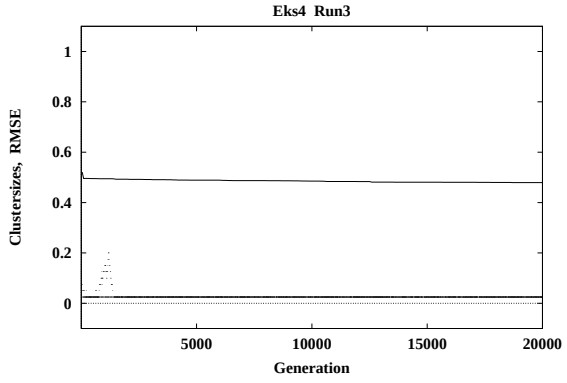


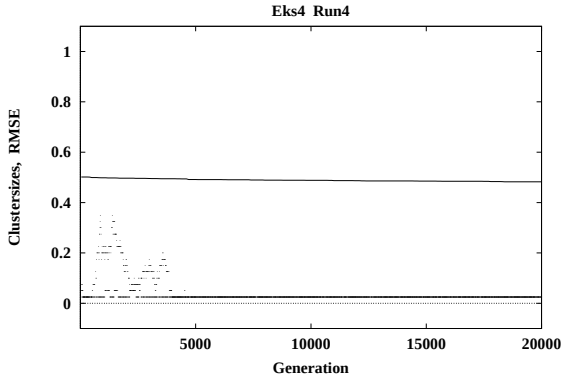
Figure 11.5: %Correct and RMSE of Exp 4.

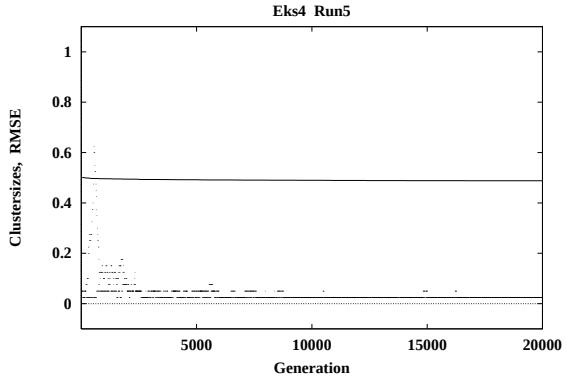
Cluster graphs.
Most runs shows that the population has very small cluster sizes, coresponding to, at most, two or three similar individuals in the population at any time. This means hardly no individuals become functionally equivalent with the currently defined equivalence concept:

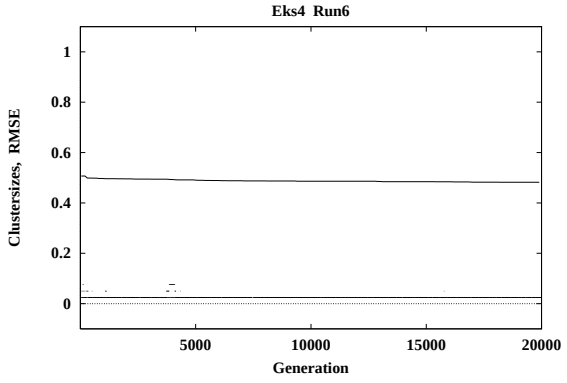

Eks4 Run1
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000

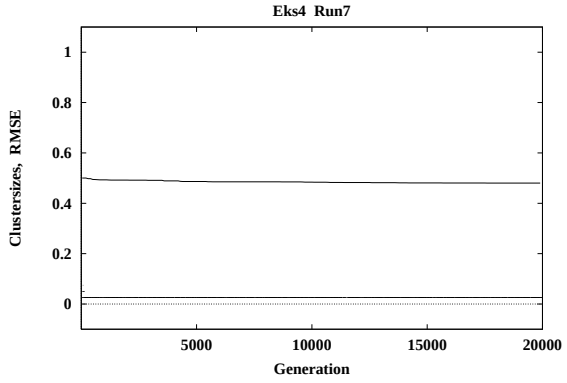

Eks4 Run2
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000

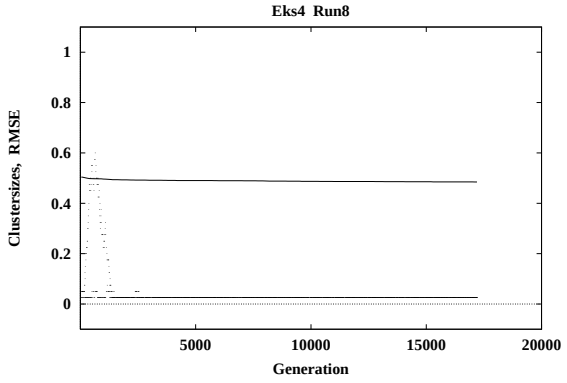

Eks4 Run3
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks4 Run4
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks4 Run5
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks4 Run6
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks4 Run7
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000


Eks4 Run8
Clustersizes, RMSE
Generation
0
0.2
0.4
0.6
0.8
1
5000
10000
15000
20000

Chapter 12

Interpretation

This section describes the motivation and order of the experiments. The experiments have two purposes:

- to test the usefulness of the cluster analysis tool on these particular tasks and demonstrate that clustermonitoring gives a great understanding of the dynamic behaviour of genetic populations in general.

- to test the usefulness of the algorithm (operators and Lifecycle administration) on a toy problem and a commonly accepted hard problem.

The first experiment should be simple in order to test simple aspects of the GA and the Cluster Analysis Tool(CAT). A common choice is the 2bit parity, or XOR, problem. But these problem are too simple to illustrate the global sampling capability of GAs [Pers. comm. with N.J. Radcliffe], hence also not fruitful to picture population dynamics with the CAT. Some toy problems are n-parity, n+n-adders, and n-log_2(n)-n-multiplexers.

Multiplexing has the advantage of predefining the number of hidden nodes. Therefore the dynamic development of hidden layer can be tested by setting the maximum number of hidden nodes to log_2(n) and the minimum to one. From a weight training point of view, the task is learnable by local optimisation techniques like backpropagation. The 8-3-8 task seems "big" enough to allow for some topological development and requires learning capabilty beyond the XOR task. It is also "small" enough to test the CAT within reasonable time. The task was also chosen because of its simplicity, in order to test the training operators MUTATEHIDD and MUTATEOUT. Furthermore, the topology is not fixed to 3 hidden nodes, but is allowed to develop dynamically between 1 and 3.

[Robbins 1993] report population sizes of 10 to 32 to optimize topology for nets on the XOR task and two reduced Two Spiral tasks (1 and $1\frac{1}{2}$ turn spirals). I decided to start with a small population size, 20, because the CAT must

compare every new individual to every individual in the population. Which is computational demanding as population size grows. The replacement strategy was to replace the least fit always. The total number of new individuals (= generations) was set to 20000, which was high enough to reach 100% correct classification, for at least some runs.

The $\%Correct$ results were nonconsistent, as only 2 out of 8 runs succeeded to learn the task completely, as seen on the graphs. The RMSE graphs shows that the 6 failing runs all seem to stall in the end, as if the population has converged to some suboptimum. In the Cluster graphs there are no outstanding difference between the 2 succesful runs (Run 4 and 8) and the remaining.

The 2 succesful runs have some shifting clusters until generation 14000, where a single cluster seems to have entered a valley in search space with RMSE around 0.2, which is fine tuned to less than 0.1. The same picture is seen in Run 6, but it is not as succesful. Most Cluster graphs show convergence to a single cluster (functionality) in the population, but some are not succesfull, they have Premature Converged.

A closer look at RMSE and Clusters of Run 1, 3, 6, and 7 shows that an early dominant cluster tends to stall progress, while the dominant cluster at the end of Run 4 and 8, seems to exploit the optimum rapidly.

This is consistent with commonsense understanding of genetic population dynamics. It is proposed that the population is rapidly filled with similar individuals, created by MUTATEOUT and MUTATEHIDD, which only affects the incomming weights of a single node at a time. The recombination operator, on the other hand, disrupts at the topological level, thus creating children further from the parents, functionally.

The proposal can be tested. In order to avoid the mutation operators filling the population with nets with slightly different weights, the replacement strategy for mutation was enhanced: the child always replaces its parent. This is the only difference from Exp 1, yielding Exp 2.

Notice, I also tried to enhance the replacement strategy for mutation further by only allowing replacement with the parent, if the child was more fit than its ancestor. This consistently stalled every run at an early stage! The reason for this might be that it is impossible to wander around in search space if only 'greediness' is allowed. There are no graphs illustrating this, though.

Turning directly to the Clusters graphs of Exp 2, the enhanced replacement worked just fine: except for Run 1, no runs were dominated by a single cluster.

This suggests that the population is diverse, and thus less likely to get stuck in some local optima. This is confirmed by the RMSE graphs, where only Run 1 and 5 seems to stall in the middle of the run. The remaining runs progresses consistently, though not as fast as the best runs in Exp 1 !

It is therefore concluded that the mutation operators tend to fill the population with functionally similar individuals. This conclusion rely mainly on the results of the Cluster graphs.

The recombination operator, on the other hand, is not likely to produce a child functionally equivalent to either of its parents: despite the facts that $P_{RECOMB} = \frac{100}{120}$ and Replacement Strategy for recombination is Replace Least Fit, Exp 2 shows that recombination is not likely to produce an outcome functionally similar to any of its parents. This is not surprising, though, as the recombination operator changes the topology and thus has a great influence on the functionality of the resulting net.

The difference between Exp 1 and Exp 2 illustrates the tradeoff between coarse grained and fine grained search (exploration and exploitation [Jones 1993]. The best RMSE results are not as good as in Exp 1, but this is the price for the expected robustness of the algorithm. In the following experiments the Replacement Strategy for the mutation operators is to replace parent.

To get a picture of the influence of population size on the performance on the resulting net(s) and of the population dynamics in a bigger population, indeed, Exp 3 is the same as Exp 2 except that the population size is doubled to 40.

The RMSE graphs of Exp 3 range from 0.4 to 0.6 at the end. Compared with 0.25 to 0.55 in Exp 2. The impression is that Exp 3 progresses slower than Exp 2. On the $\%Correct$ graphs it is obvious that the task is not learned in any of the runs, as all graphs end up with less than 97% correct classification.

Turning to the Cluster graphs of Exp 3 the picture is quite different from Exp 2: Only in the very begining of the runs are clusters above half of populations size, except for Run 4 and 5. Cluster size is mainly below a quarter of populations size.

This indicates that clustering is not prominent in a population of double the size, compared to Exp 1 and 2, where cluster size is often above half of population size. Refer to 'Discussion' for a detailed treatment.

Despite that the genetic algorithm can not consistently learn the 8-3-8 task, a more demanding task i now investigated: The Two-Spiral task. Using the parameters from Exp 3.

Turning directly to the $\%Correct$ graphs, none of the runs ever reached 40% correct classification. This is highly unsatisfactory compared to the Genetic Cascade net in [Karunanithi et al 1992], which learned 190 of the 192 patterns, though requiring 300000 evaluations (new individuals) on average.

While $\%Correct$ graphs shows some limited progress over a run, the RMSE graphs shows a strange behaviour: The RMSE is approximately constant 0.5 throughout every run. How come that the ratio of correct classified outputactivations shows progress while the error is almost constant ?

The target values of the patterns consist of equally many 0s and 1s, representing either spiral. Therefore the output of the best net is either constant or 'uniformly' distributed on the interval $[0; 1]$. The output is not constant because, the $\%Correct$ graphs shows little progress. What seems to happen is that a few outputvalues drift from slightly below 0.6 to slightly above 0.6 and some from slightly above 0.4 to slightly below 0.4, thus classifing more patterns correct, while the effect on the error is minimal.

The population dynamics analysis (the Cluster graphs) show nearly no individuals having identical functionality. This conforms with the hypothesis that the recombinator has difficulty preserving functionality, particularly on nets involving many hidden nodes (up to 24).

Learning seems stalled right from the beginning. This suggests that more attention should be given to learning. There are three ways to do this.

- Either to adjust the probability of the mutation operators.
- Or turn replacement strategy to Replace Least Fit for recombination also.
- Or shrink population size.

The first would give more precedence to the mutation operators, thus increasing probability to make a succesful weight change. The second would bias the population to contain several topological nets with slightly different weights, also called fine grained search. Having several similar individuals makes the population less vulnerable to a few unsuccesful weight adjustments. This involves a risk of convergence however. The third should confirm the results from the 8-3-8 experiments: smaller populations tend to cluster more.

This suggests three experiments using the same settings, except:

1. Turning $P_{MUTATEHIDD}$ and $P_{MUTATEOUT}$ from $\frac{10}{120}$ to $\frac{50}{120}$, and adjusting P_{RECOMB} accordingly from $\frac{100}{120}$ to $\frac{20}{120}$.

2. Changing replacement strategy for the mutation operators.

3. Halving population size from 40 to 20.

If some of the approaches make progress they could be combined. Also it would be interesting to see if the population dynamics develop as proposed. These experiments do not fit into the schedule of this study, but they might very well be part of future investigations.

Part V

Discussion and Conclusion

Chapter 13

Discussion

13.1 Training results

Training results refer to the MSE and %Correct of the best developed nets during a run. Is the task fully learned? Is the algorithm robust over several runs? How does learning speed compete with results reported elsewhere?

The 838 task As the maximum number of hidden nodes was set to three this experiment mainly investigates the algorithms ability to train the net, and can be compared to other approaches, both genetic and local techniques. The experiments are compared to [Dasgupta & McGregor 1992] on another small multiplexer task.

Did the GA succeed to learn 838 multiplexing? Two runs in Exp 1 and 2 succeded to learn 838 with 100 %Correct classification.

Is learning speed competitive? [Dasgupta & McGregor 1992] report genetic development of topology and weights of a 4-X-4 network which learns the task completely in 400 to 600 generational replacement generations, population 100, hence 40000 to 60000 new individuals! This should be compared to 20000 generated individuals. Thus compared to Dasgupta the algorithm compete well (Exp 1 and 2) on a small multiplexer task.

What topologies was developed ? The best individuals always had 3 hidden nodes, that is topology 8-3-8. Informal testing showed that in all experiments most runs developed a hidden layer with MaxNoOfHidden hidden nodes. All connections had nonzero weights, thus the hidden nodes were fully connected. This was expected because the fitness function does not include any terms on connectivity.

The TwoSpiral task The TwoSpiral problem has recently been used in [Karunanithi et al 1992][Potter 1992] and [Robbins 1993], which all manage to classify most points of the spirals correctly. Therefore a %Correct less than 40 percent is unsatisfactory. And MSE nearly constant 0.5 which corresponds to completely arbitraty classification even more. A discussion of the failure of the algorithm is the aim of the next section.

13.2 Simultaneous optimisation of topology and weights

To summarize previous approaches to evolve topology and weights, the following gives a good overview. It is based on the general framework in [Yao 1992]:

> It can be viewed as a hierarchical adaptive system with three levels. At the highest level, architectures evolve on the slowest time scale in an environment decided by the task to be accomplished by the system. (The next level is learning rule evolution, and the last is weight evolution. ed.) An analogy to the solar system could be drawn here. When sets of connection weights (moons) are evolving, where C stands for a particular set (moon), they are in an environment decided by the learning rule B (planet), the architecture A (solar system) and the task (Milky Way galaxy) (See figure 13.1, ed.)

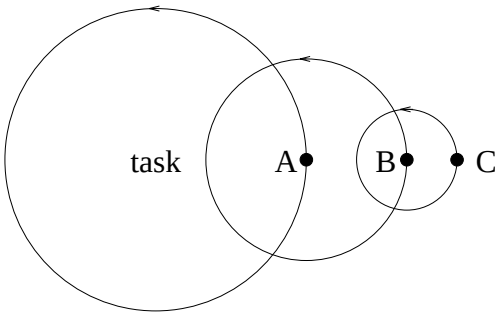
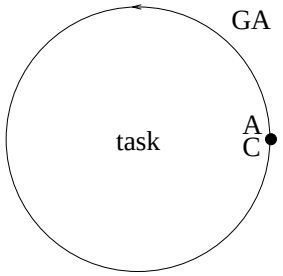


Figure 13.1: The Milky Way Galaxy and a 3 level adaptive system. Three level adaptive system evolving architectures, learning rules and weights, respectively. The circles represents any kind of optimisation technique.

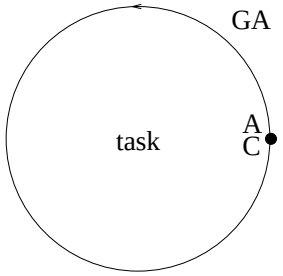
> [Yao 1992] The framework could be viewed as a hierachical model of a general adaptive system if we do not constrain ourselves to GA-based evolutionary search procedures,.... Simulated annealing, gradient descent searches, evolution strategies, evolutionary programming, and even one-shot (only-one-candidate) searches can all be considered as some types of evolutionary search procedures.

As I see it, previous approaches to evolution of topology and weights can be conceptually divided in three cases:

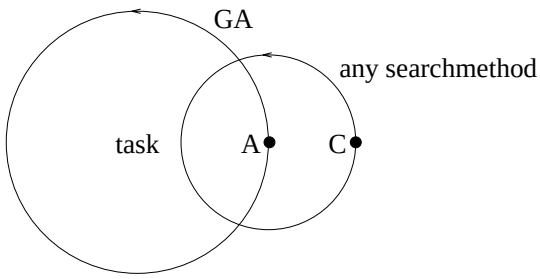
1. A one-level adaptive system, a GA, which discovers both topology and weights. This is the approach in this thesis. Also in [Karunanithi et al 1992] [Potter 1992] [Das and Whitley 1992]. GA which evolve both topology and weights at the same level. One level adaptive system evolving architectures and weights. The circle represent a GA *with* recombination:



2. A one-level adaptive system without recombination, that is as case 1, but with recombination rate set to zero. Selective random hillclimber discovers topology and weights. One level adaptive system evolving architectures and weights. The circle represent a GA *without* recombination - a selective hillclimber:



3. A two-level adaptive system, separating the discovery of topology and weights. Discovery of computational capability The highest level discovers topologies by a GA. The second level discovers weights by a GA or other perhaps local techniques. Two level adaptive system evolving architectures and weights:



Case 1 and 2 in a sense coevolves topology and weights, while Case 3 subsequently evolves topology and weights. The cases 1 and 2 can be interpolated between by varying the operator rates.

In the case of 3-layered feedforward nets there is a big difference between the task of combining topologies and combining topologies with associated weights. The former is usually used when interweaving a GA search for topology with another GA or a traditional parametric training algorithm searching over weights (Case 3). The latter is coevolution of both the topology and the weights values within a single GA (Case 1 and 2).

Radcliffes recombination operator was originally formulated for combination of hidden nodes in 3-layered nets (topology was defined as multisets of bitstrings representing the connections of hidden nodes) using an assumption of correlation between connections of hidden nodes. It does seem fair to believe that the computational capability of a topology generated this way is correlated with its parents[Angeline 1993]. But this assumes near optimal weights are found for the topology, as in Case 3. A two-level adaptive system inherents the possibility of computing the same sets of weights for similar topologies over and over again, though.

The idea of coevolving topology and weights are therefore appealing. The hurdle is to find a recombination operator which presumably generates a child correlated with its parents. The only sensible correlation is functional behaviour. The recombination operator used in these experiments is an extension of Radcliffes operator. It is based on a desire to recombine variable sized multisets and then a 3-layered net is viewed as a multiset of its hidden nodes. The problem with this point of view is that this does not emphasis on the functional behaviour of the outcome, only on the set theoretic properties.

The following argues that coevolving topology and weights should use a small recombinationrate (a special case of Case 1) or no recombination at all (Case 2). The reason being that no efficient recombination operator for this task has ever been developed, including the operators used in these experiments. The difficulty is partly due to the competing conventions problem (permutation problem), and partly because the coorporation between hidden nodes is not fully understand: the behaviour of the neural net is more complicated than the sum of its parts (hidden nodes). Therefore it has been difficult to design a proper way of combining any two neural networks. A proper recombination of two neural networks should(must) preserve some functionality of the parents. I propose such an operator does not exist. The reason being that the desire of searching and finding a certain functionality of a neural network is exactly the purpose of all neural network optimisation attempts. It means not the failure of the experiments, but

rather gains new insight in the growing field of using genetic algorithms to build neural networks efficiently.

The succes of any GA depends on the ability to investigate correlations of individuals in the population[Goldberg 1989]. The observation that an efficient recombinator for entire neural nets might not exist, thus suggest to reduce recombination to a minimum, or perhaps exclude recombination:

A relative low recombination rate implies a relative high mutation rate. High mutation rates has been reported by [Montana & Davis 1989][Whitley et al 1990] [Davis 1991] and [Angeline 1993].

An alternative approach to coevolve topology and weights without the hurdles of finding appropriate recombinators is to exclude recombination. This is done in [Angeline 1993] which argues that an evolutionary program (EP) is more suitable than a GA: "Because EP models evolution at the level of the behaviour of unmixing species, recombination of distinct population members is not considered. EP's commitment to mutation as the sole reproductive operator for searching over a space is preferable when there is not sufficient calculus to guide recombination by crossover, or when separating the search and evaluation spaces does not afford an advantage."

Summary: If we cannot develop a recombinator for neural networks which preserves functionality, then the recombination rate should be small or no recombination should be used at all, i.e. a parallel random hillclimber with pressure on the better individuals. Or alternatively, use a two level adaptive system, evolving topologies and, subsequently, weights for these topologies.

13.3 Population Dynamics

Population Dynamics refer to the information from the Cluster graphs. Is the population too diverse? Has it converged? Or is it efficiently explorering and explorating the search space?

This section discusses the information from the cluster monitoring on the experiments. It is also a demonstration of the use of cluster monitoring as a tool for understanding the dynamic behaviour of genetic populations in general.

13.3.1 Why is clustering more prominent in the small population ?

The following general discussion arrives at the conclusion that smaller populations tends to cluster more than bigger ones, based on simple probabilistic considerations similar to the schemata theorem.

Dependend on the nature of the operators and the cluster definition, operators might produce children in the same cluster as its parent(s). If all operators are consistently closed under the current cluster definition, this alone explains the cluster tendency of the smaller population. There is no build-in guarentee that operators are closed w.r.t. an arbitrary cluster defintion, though. In the same manner as forma or schemata have disruption coefficients for the current operators, clusters can be said to have disruption coefficients as well. Like the schemata theorem this gives a model explaining that good clusters grow dependend on the operators ability to correlate children with parents w.r.t. the chosen cluster definition. It even explains why clusters with highfitness individuals has greater probability of growing as populations size shrinks:

Given two populations Pop1 and Pop2 where Pop2 is bigger than Pop1 and contain the same individuals as Pop1 plus the remaining. Thus the relative fitness of an individual in Pop1 is bigger than the same in Pop2. Imagine a cluster of some fixed size in Pop1. The probability of selecting an individual from a certain cluster is the sum of probabilities of selecting each individual in that cluster. Therefore the same cluster have less chance of having a member selected, in Pop2. Given that operators are likely to produce children in the same cluster as either parent, clustering is thus more probable in the smaller population. Very much like the schemata theorem this explains why superior clusters tends to grow. Even more important, it explains why small populations are more haunted by clustering than bigger populations.

The above is confirmed by several researchers emperical observations when using GA's to optimize topology/weights of neural nets: small populations tends

to be filled with similar individuals. [Jones 1993]"For small populations the schemata theorem is actually an explanation for premature convergence (i.e. the failure of the algorithm) rather than a result which explains succes".
[Whitley et al 1990]"Our analysis indicate that in these small populations (50 compared to 3000!, ed.) only a single solution is being pursued", on training a fixed topology of over 300 connections. Several experiments by [Whitley et al 1990] confirms the tendency to cluster as population size shrinks.

[Hancock 1992] presents results that suggests the permutation problem can be overcome. [Schaffer et al 1992] explains: "These results may be attributable to Hancock's use of small populations (100, ed.) and high selection pressure leading to early convergence to a single convention (permutation of hidden nodes, ed.)".

There is no build-in guarantee that operators are closed w.r.t. an arbritrary clusterdefintion. And this is not the case in my experiments either:

- the cluster equivalence relation used in my experiments cluster w.r.t. functionality.
- the recombinator has been designed to fullfill certain set properties, which in some cases might preserve the functionality of the parents. Replacement strategy is Replace Least Fit.
- the mutation operators produce a child with slightly different weights, and are thus likely to preserve functionality. Replacement strategy is Replace Parent. An operator with Replacement strategy Replace Parent, does not contribute to the extension of any clusters, obviously.

13.3.2 Escape of Premature convergence

Premature convergence in a classic GA with a population of fixed sized bitstrings, multipoint crossover and bitmutation is easily detected as a population consisting of nearly identical bitstrings. In this case crossover is likely to produce individuals which are already in the population, and mutation is likely to produce similar or disrupted, unfit individuals. The population has converged and the operators can't help it.

In the case of more sophisticated representations, like reals, and specialized datastructures we need more sofisticated operators. Measuring convergence have become difficult and we must determine what we actually want to monitor. Our choice may reflect our intuition of similar individuals or the formae that the recombination operator obey. The ideal recombinator would preserve functionality when combining functionally equivalent parents - as we emphasize functionality of our neural nets. The cluster defintion used in the experiments reflects an intuition of functionally equivalent individuals. But the operators does not guarentee

to be closed under functionality. Therefore the detection of convergence in a single cluster does not imply that the operators can not create children outside that cluster, as the following graphs show. Some runs shows dominant clusters, i.e. clusters that constitutes most of the population. The following Clusters graph shows convergence from generation 7000 to 8000 and from 9000 to 11000. This does not mean the population has converged, the convergence is escaped around generation 11500, but a new cluster dominates the population from generation 14000 and sustains to the end of the run, see figure 13.2. Other runs do sustain

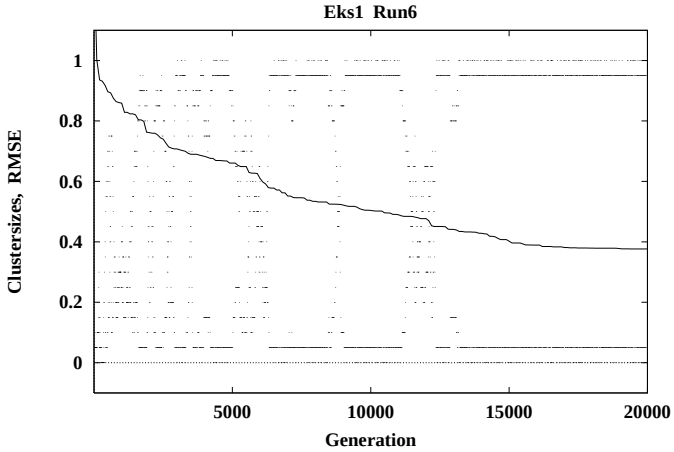


Figure 13.2: Cluster sizes and RMSE of Exp1, Run 6.

a single cluster to the end, see figure 13.3. Notice that the RMSE graph tends to

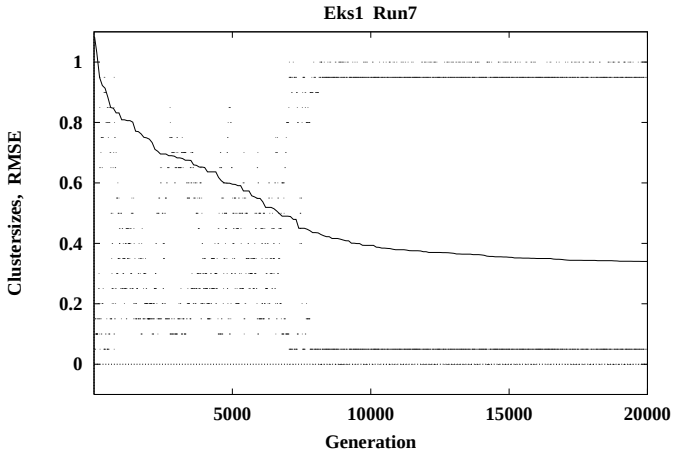


Figure 13.3: Cluster sizes and RMSE of Exp1, Run 7.

stall when the largest cluster is dominating the population. This seems to happen because the convergence appears to early and are allowed to sustain infinitely.

Chapter 14

Conclusion

14.1 Experiments

Exp 1 and 2 indicates the recombinator often disrupts functionality. Turning replacement strategy of the mutators from Replace Least Fit to Replace Parent (Exp 1 to 2) and observing fewer (8 to 1) runs with dominating clusters lead to the conclusion that the recombinator is much less likely to preserve functionality to the outcome than the mutators. This is not surprising because the mutators slightly adjust a few weights, while the recombinator disrupts topology.

Compare the conclusion drawn from Exp 2, with Exp 3: Doubling the population size implies the ratio $MaxClusterSize/PopSize$ is roughly halved. This lead to a discussion involving the schemata theorem, concluding that the observed behaviour is governed by the inherent mathematics of a genetic population.

14.1.1 When is search most succesful?

An idealised behaviour of the GA can be described on the experience gained by studing the Cluster graphs: It seems that the most succesful run starts with a diverse population and it should stay diverse until an individual is in the global valley of attraction. Then the population should cluster around that individual. Here they finetune the best solutions.

When all individuals have stabilised on the same sets of weights for the hidden nodes, the intrinsic disruptive recombination operator have only a few different hidden node types to combine and are thus less likely to produce children which are not functionally equivalent with individuals already in the population. That prevents the operator from creating too many children outside the valley. The mutation operators slightly adjust weights, and due to pressure on the better individuals, steadily climb to the optimum. Good examples are Exp 1 Run 4 and 8, see figure 14.1.

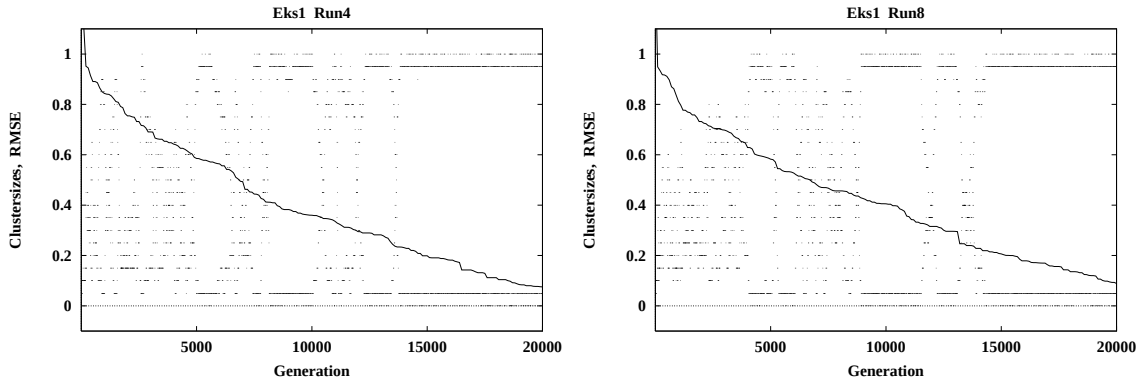


Figure 14.1: RMSE and Cluster sizes of Exp 1, Run 4 and 8

14.1.2 When is search most robust?

To predict what setup of Lifecycle Administration, operator parameters, and PopSize is most robust for any given neural net mapping task, cannot be determined from Exp 1 to 4 alone, and is not within the range of this paper. More work will be necessary to develop that kind of knowledge. The Cluster Analysis Tool is ideal for such analysis.

14.2 CATs are widely applicable and useful

The CAT gave insight in the phenomena of convergence and effects of changing the size of population. It helped explaining the failure of the recombination operator constructed for entire neural networks (topology and weights).

Beside enriching experiments, exploring the basic GA characteristics, like Life-Cycle Administration, operators, parameters and more, CATs can also investigate the effect of applying diversity methods, like crowding and sharing. Even nicher can be monitored: as several independent subpopulations, or as a single, big population. Monitoring cluster graphs for a multiniche GA would immediately elucidate the effect of using nicher on any particular task: While each niche seems to contain a few interesting clusters, they might in fact be exploring the very same areas of the search space. The latter phenomena is unveiled by the cluster graph which consider the subpopulations as a single, big population. The Cluster Analysis Tool is applicable for analysis, understanding, and development of genetic algorithms.

14.3 Is there a long way yet before the vision comes true?

My vision of an automatic universal net constructor was described in a previous chapter. Whether the GA/NN community will ever discover such a "philosophers' stone" is not possible to predict at this time, but if such a constructor can be developed, I propose it will contain the following characteristics: direct specification of topology and weights, low recombination rate, thus mainly mutation.

It would also be interesting to see Artificial Diversity and dynamic population size in any future investigations of genetic algorithms.

Appendix A

Using the simulator

This appendix gives instructions on how to run the simulator used in the experiments. This allows for further experiments with the build-in Cluster Analysis Tool. The source code of the simulator can be found in directory:

```
/a/home/rubus/kimkorte/GA/FGA
```

To illustrate how the simulator can be invoked, the 838 multiplexer task is used as an example. Substitute '838' for '*task*' in the following:

At Daimi, Computer Science Department, Aarhus University, login on the net of workstations and type:

```
cd /a/home/rubus/kimkorte/GA/838ex
```

4 input files are needed: *task*.in, *task*.data, *task*.teach, and *task*.net:

The file 838.net specifies the number of input nodes, the $MaxNoOfHidden$, and the number of output nodes:

```
8
3
8
```

The file 838.data contains the input patterns. The first line must be 'distributed'. The second line must specify the number of patterns. The remaining must contain the patterns in the following format:

```
distributed
8
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0
```

```
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
```

The file 838.teach contains the output patterns. The first line must be 'distributed'. The second line must specify the number of patterns. In a multiplexer task like 8-3-8, the *task*.teach file is equal to the *task*.data file.

```
distributed
8
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
```

The file 838.in specifies the parameters of the genetic algorithm:

```
#      Experiments = 8
#     Total Trials = 20000
#  Population Size = 40
#      MaxNoOfEach = 50
#      MinNoOfEach = 0
# Structures Saved = 0
#    Dump Interval = 100
#      Dumps Saved = 1
#          Options = "bK"
#             Seed = 0
#         InitSeed = 1
#        StartSeed = 0
#            Range = 5.0
#            Shake = 0.0
#            Creep = 0.4
#           Creep2 = 0.3
#Uniform_crossover = 00.0  0.0  0 2
#Removeweight      = 0.0  0.0  0 0
#Mutate           = 0.0  0.0  0 0
#Large_Creep   = 0.0  0.0  0 1
#Radclif3_crossover= 100.0  0.0  0 0
#Mutate_Hidd       = 10.0  0.0  0  1
```

```
#Mutate_Out         = 10.0  0.0  0  1
#Mutate_Both    = 0.0  0.0  0  1
#SomeCreep2         = 0.0  0.0  0  1
#Mutate_New         = 0.0  0.0  0  0
```

'Experiments'	specifies the number of Runs.
'Total Trials'	the number of Generations created in each Run.
'Population Size'	Popsize.
'MaxNoOfEach'	must be bigger than 'Population Size'.
'MinNoOfEach'	must be 0.
'Structures Saved'	must be 0.
'Dump Interval'	how often the %Correct and RMSE are dumped, in Generations.
'Dumps Saved'	must be 1.
'Options'	explained below.
'Seed'	not used.
'InitSeed'	random seed for the initialisation of population in every Run.
'StartSeed'	random seed for the start of the first Run.
'Range'	not used.
'Shake'	not used.
'Creep'	not used.
'Creep2'	not used.
'Uniform_crossover'	not used.
'Removeweight'	not used.
'Mutate'	not used.
'Large_Creep'	not used.
'Radclif3_crossover'	the Recombination rate, '0.0', '0', Replacement Strategy.
'Mutate_Hidd	the Mutate_Hidd rate, '0.0', '0', Replacement Strategy.
'Mutate_Out	the Mutate_Out rate, '0.0', '0', Replacement Strategy.
'Mutate_Both'	not used.
'SomeCreep2'	not used.
'Mutate_New'	not used.

The Replacement Strategy is selected, as:

0 Replace Least Fit Individual.
1 Replace Parent.
2 Exponential rank deletion.

The options have the following meaning:

b	Bestflag	print final best value.
C	Collectflag	performance data in outfile.
d	Dumpflag	dump after each evaluation.
D	Distflag	dump distmatrix.
e	Evolutionflag	Show Optr,P1,P2,Child,Rates in Evolution file.
i	Initflag	read initial structures.
f	Fitnessflag	print fitness gains.
K	Clusterflag	dump Clusters.
L	Lastflag	dump the last generation.
l	Logflag	log starts and restarts.
m	Monitorflag	Show Parents and Childs in Monitorfile.
p	Populationflag	Show Perf and nh of all individuals in the end.
s	Shakeflag	mutate all new individauls.
S	Simflag	dump similarity of pop.
t	Traceflag	trace program execution, for debugging only.
o	Onlnflag	print final Online value.
O	Offlnflag	print final Offline value.

The simulator is invoked by typing the name of the simulator, 'fga', followed by: the directory containing the *task*-files and a '*label*' to follow the output created, thus:

```
/a/home/rubus/kimkorte/GA/FGA/fga /a/home/rubus/kimkorte/GA/838ex eks1
```

The output is files with filenames *task.label*.out.*run* For example would the first Run of the previous invokation produce a file, '838.eks1.out.1'.

The output files contain contain output of the form:

$$Generation, \%Correct, \%Wrong, \%Unknown, RMSE, Online, Offline.$$

The important columns are 'Generation', '%Correct', and 'RMSE'.

The only option recommended are: K - dump Clusters. This option cause the simulator to invoke the Cluster Initialisation algorithm and *each* time a new individual is created, the Cluster Analysis Bookkeeping algorithm. The size of the 3 (not 2, unfortunately) biggest Clusters are dumped to the file *task.label*.kliker.*run*, for example '838.eks1.kliker.1'. The computational time is increased multiple times, when invoking the 'K' option.

The cluster files contains output of the form:

$$Generation, \#1, \#2, \#3,$$

where #1, #2, and #3 symbolises the size of the 3 biggest clusters. To produce the relative size of the 2 biggest clusters, the smallest must be filtered out and the remaining must be divided by Popsize. It is also convenient to sample every 10th line in order to save diskspace.

The UNIX command 'awk' can be used to do this. Assume Popsize = 20:

```
awk '{if($1>0 && $1 % 10 == 0)
{if($2<$3 && $2<$4){print $1,"1",$3/20,$4/20}else{if($3<$4)
{print $1,"1",$2/20,$4/20} else {print $1,"1",$2/20,$3/20}}}}'
838.eks1.kliker.1 > 838eks1klik.1
```

To produce the graph of cluster sizes and RMSE from Exp 1, Run 1, 'gnuplot' was used:

```
gnuplot 838eks12d.gnu
```

- where '838eks12d.gnu' contained:

```
set term post port "Times-Bold" 20
set ylabel "Clustersizes,  RMSE"
set xlabel "Generation"
set nokey
set yt 0,.2,1
set yr [-0.1:1.1]
set xr [0:20000]
set xt 5000,5000,20000
set out "838eks12d.1.ps"
set title "Eks1  Run2"
plot"838eks1klik.1" u 1:3 w d 3,"838eks1klik.1" u 1:4 w d 3,
"838.eks1.out.1" u 1:5 w l 1
```

The actual batch file used to preprocess the %Correct and RMSE information in the file 838.eks1.out.1 was:

```
#!/bin/csh -f
####  error -> 838eks1error  ####
awk '{if(NR>2){print $1,"1",$5}else{print}}' 838.eks1.out.1 > 838eks1error
cat 838eks1error | tail -1 | cat >> 838eks1error
echo "" | cat >> 838eks1error
```

```
awk '{if(NR>2){print $1,"2",$5}else{print}}' 838.eks1.out.2 >> 838eks1error
echo "" | cat >> 838eks1error
...
```

(- and so on for Exp 3 to 8.)

```
awk '{if(NR>2){print $1,"1",$2}else{print}}' 838.eks1.out.1 > 838eks1rigtig
cat 838eks1rigtig | tail -1 | cat >> 838eks1rigtig
echo "" | cat >> 838eks1rigtig

awk '{if(NR>2){print $1,"2",$2}else{print}}' 838.eks1.out.2 >> 838eks1rigtig
echo "" | cat >> 838eks1rigtig
...
```

(- and so on for Exp 3 to 8.)

The use of

```
cat 838eks1error | tail -1 | cat >> 838eks1error
```

is to duplicate a single line of measures. This trick causes gnuplot to treat the data as separate lines, not as a surface plot.

To produce the graphs of %Correct and RMSE from Exp 1, 'gnuplot' was used:

```
gnuplot 838out.gnu
```

- where '838out.gnu' contained:

```
set term post land color "Times-Bold" 16
set parametric
set view 60,101,,
set xt 5000,5000,20000
set zt 0.0,0.2,1.0
set zr [0:1.5]
set ticslevel 0
set tics out
set nokey
set yl "Run
set xl "           Generation
set zl "%Correct,  RMSE
set title "Eks 1
set out "838eks1out.ps
set out "838eks1out.ps
splot "838eks1rigtig" w l,"838eks1error" w l 3
```

The following briefly describes the equipment used during this work.

- For development of the simulator the compiler 'gcc' and debugger 'gdb' were used on a Hewlett Packard, apollo, Series 700 workstation.
- The text was written using 'emacs' with 'auc-tex'.
- Preprocessed using 'latex'.
- Previewed using 'ghostview'.
- The pages containing the color graphs were printed individually using a feature in 'ghostview'.
- Printed on 'Apple Laserwrite II' and 'HP InkJet'.
- The figures were created using 'xfig'.
- The graphs were created using 'gnuplot'.

Bibliography

[COGANN-92] *Combinations of Genetic Algorithms and Neural Networks*, Schaffer JD, Whitley D, eds. IEEE Computer Society Press, 1992.

[FGA] *Foundations of genetic algorithms* , Gregory J. E. Rawlings, ed. , San Mateo (CA): Morgan Kaufmann, 1991.

[IJCAI-89] *Proceedings of eleventh international joint conference on artificial intelligence* , San Mateo, CA: Morgan Kaufmann..

[ICGA-3] *Proceedings of the Third International Conference on Genetic Algorithms* , San Mateo (CA): Morgan Kaufmann, 1989.

[ICGA-4] *Proceedings of the Fourth International Conference on Genetic Algorithms* , San Mateo (CA): Morgan Kaufmann, 1991.

[NC&A] *Neural Computing and Applications*, Vol 1, Number 1, 1993, Springer Verlag London Limited.

[Alexis 1993] Alexis Wieland of MITRE Corporation on "connectionists mailing list" on the Internet.

[Angeline 1993] Angeline PJ (1993), PhD thesis *Evolutionary algorithms and emergent intelligence*, Chapter 4, 65-85: *The Emergence of Task-Specific Structures* , Ohio State University.

[Baker 1987] Baker EJ (June 1987) *Reducing bias and inefficiency in the selection algorithm*, Proceedings of the Second Int'l. Conference on Genetic Algorithms, 14-21, John, J. Greffenstette, ed., Morgan Kaufmann.

[Belew et al 1990] Belew RK, McInerney J, Schraudolph NN (June 1990) *Evolving Networks: Using the Genetic Algorithm with Connectionist Learning*, CSE Technical Report #CS90-174, Computer Science & Eng. Dept. (C-014), University of California, San Diego.

[Belew et al 1991] Belew RK, McInerney J, Schraudolph NN (Feb 1991) *Evolving Networks: Using the Genetic Algorithm with Connectionist Learning*,

CSE Technical Report #CS90-174 (Revised), Computer Science & Eng. Dept. (C-014), University of California, San Diego.

[Booker 1982] Booker LB (1982) *Intelligent behaviour as an adaption to the task environment* [dissertation]. Ann Arbor (MI): Univ of Michigan.

[Bornholdt and Graudenz 1991] Bornholdt S, Graudenz D (1991) *General Asymmetric Neural Networks and Structure Design by Genetic Algorithms* , DESSY 91-046 Hamburg, Germany: Deutsches Electronen-Synchrotron.

[Cavicchio 1970] Cavicchio DJ (1970) *Adaptive search using simulated evolution* [dissertation]. Ann Arbor (MI): Univ of Michigan (unpublished).

[Collins and Jefferson 1991] Collins R, Jefferson D (1991) *An artificial neural network representation for artificial organisms.*, In H.-P. Schwefel & R.Männer (Eds.), Parallel problem solving from nature, 269-263. Berlin: Springer-Verlag.

[Das and Whitley 1992] Das R, Whitley D (1992) *Genetic sparse distributed memory* , [COGANN-92] 97-107.

[Dasgupta & McGregor 1992] Dasgupta D, McGregor DR (1992) *Designing Application-Specific Neural Networks Using the Structured Genetic Algorithm* , [COGANN-92] 87-96.

[Davis 1991] Davis L (1991) *Handbook of Genetic Algorithms*, Von Nostrand Reinhold, New York.

[De Jong 1975] De Jong K (1975) *An analysis of the behavior of a class of genetic adaptive systems* [dissertation]. Ann Arbor (MI): Univ of Michigan, Dept. Computer and Communication Sciences. Diss. Abstr. Int 36(10), 5140B

[Elias 1992] Elias JG (1992) *Genetic generation of connection patterns for a dynamic artificial neural network* , [COGANN-92] 38-54.

[Eshelman & Schaffer] Eshelman LJ, Schaffer JD *Real-Coded GA and Interval-Schemata*, Philips Laboratories, NY.

[Fahlman and Lebiere 1990] Fahlman S, Lebiere C (1990) *The Cascade-Correlation Learning Architecture* , CMU-CS-90-100, Pittsburgh: PA: Carnegie-Mellon University.

[Goldberg 1989] Goldberg DE (1989) *Genetic Algorithms in Search, Optimisation and Machine Learning* , MA: Addison-Wesley.

[Goldberg & Lingle 1985] Goldberg DE, Lingle R (1985) *Alleles, loci, and the Traveling Salesman Problem* , In: Proceedings of the International Conference on Genetic Algorithms and their Applications, Hillsdale: Lawrence Erlbaum Associates.

[Goldberg et al 1987] Goldberg DE, Richardson J (1987) *Genetic algorithms with sharing for multimodal function optimization* In: Proceedings of the Second International Conference on Genetic Algorithms; 41-49; Cambridge (MA).

[Gruau 1993] Gruau F (1993) *Genetic Synthesis of Boolean Neural Networks witha Cell Rewriting Developemental Process* In: [COGANN-92] 55-74.

[Hancock 1992] Hancock PJB *Genetic Algorithms and Permutation Problems: A Comparison of Recombination Operators for Neural Net Structure Specification* In: [COGANN-92] 108-122.

[Harp et al 1989] Harp SA, Samad T, Guha A (1989) *Towards the genetic synthesis of neural networks* , [ICGA-3] 360-369.

[Holland 1975] Holland JH (1975) *Adaption in Natural and Artificial Systems* , Ann Arbor, MI: University of Michigan Press.

[Jones 1993] Jones AJ (1993) *Genetic Algorithms and Their Application to the Design of Neural Networks* , [NC&A] 32-45.

[Karunanithi et al 1992] Karunanithi N, Das R, Whitley D (1992) *Genetic Cascade Learning for Neural Networks* , [COGANN-92] 134-145.

[Kitano 1990] Kitano H (1990) *Designing neural network using genetic algorithm with graph generation system*, Complex Systems 4, 461-476.

[Koza 1991] Koza JR (1991) *A Hierarchical Approach to Learning the Boolean Muliplexer Function* In: [FGA] 171-192.

[Koza and Rice 1991] Koza JR, Rice JP (1991) *Genetic generation of both the weights and architecture for a neural network* , IEEE international joint conference on neural networks II-397 - II-404, Seattle, WA:IEEE.

[KrishnaKumar 1992] KrishnaKumar K (1992) *Immunized Neurocontrol - Concepts and Initial Results* , [COGANN-92] 146-168.

[Lang and Witbrock 1988] Lang KJ, Witbrock MJ (1988) *Learning to Tell Two Spirals Apart* , Proc. of the 1988 Connectionist Models Summer School, Morgan Kauffman.

[Lindgren et al 1992] Lindgren K, Nilsson A, Nordahl MG, Råde I (1992) *Regular Language Inference using evolving neural networks* , [COGANN-92] 75-86.

[Louis et al 1993] Louis S, McGraw G, Wyckoff RO (1993) *Case-based reasoning assisted explanation of genetic algorithm results*, J. Expt. Theor. Artif. Intell. 5, 21-37, Taylor & Francis Ltd.

[Martí 1992] Martí L (1992) *Genetically generated neural networks II: Searching for an optimal representation* IEEE international joint conference on neural networks, II-221 - II-226, Baltimore, MD:IEEE.

[Maulding 1984] Maulding ML (1984) *Maintaining diversity in genetic search* In: Proceedings of the National Conference on Artificial Intelligence; 247-250.

[Miller 1989] Miller GF, Todd PM, Hedge SU *Designing Neural Networks using Genetic Algorithms* , In: [ICGA-3] 379-384.

[Montana & Davis 1989] Montana DJ, Davis L (1989) *Training Feedforward Neural Networks Using Genetic Algorithms* , [IJCAI-89] 762-767.

[Potter 1992] Potter MA (1992) *A genetic cascade-correlation learning algorithm*, [COGANN-92] 123-133.

[Radcliffe 1990] Radcliffe NJ (1990) *Genetic neural networks on MIMD computers* [dissertation]. Edinburgh (UK): Univ of Edinburgh.

[Radcliffe 1991] Radcliffe NJ (1991) *Forma analysis and random respectful recombination* In: [ICGA-4] 222-229.

[Radcliffe 1993] Radcliffe NJ (1993) *Genetic set recombination and its application to neural network topology optimisation* [NC&A] 67-90.

[Radcliffe 1993a] Radcliffe NJ (1993) *The algebra of genetic algorithms.* Edinburgh Parallel Computing Centre, Univ of Edingburgh, King's Buildings, Scotland.

[Robbins 1993] Robbins GE, Plumbley MD, Hughes JC, Fallside F, Prager R (1993) *Generation and Adaption of Neural Networks by Evolutionary Techniques (GANNET)* , [NC&A] 23-31.

[Schaffer et al 1990] Schaffer JD, Caruana RA, Eshelman LJ (1990) *Using genetic search to exploit the emergent behaviour of neural networks* In: Forrest S (Ed.) *Emergent computation* 244-248, Amsterdam: North Holland.

[Schaffer et al 1992] Schaffer JD, Whitley D, Eshelman LJ *Combinations of Genetic Algorithms and Neural Networks: A Survey of the state of the art* , [COGANN-92] 1-37.

[Schiffmann et al 1991] Schiffmann W, Joost M, Werner R (1991) *Performance Evaluation of Evolutionary Created Neural Network Topologies* In: H.-P. Schwefel & Männer (Eds.), *Parallel problem solving from nature* 274-283. Berlin: Springer-Verlag.

[Syswerda 1991] Syswerda G (1991) *A Study of Reproduction in Generational and Steady State Genetic Algorithms* , In: [FGA].

[Vose 1991] Vose MD (1991) *Generalizing the notion of schema in genetic algorithms* , University of Tennessee, Artificial Intelligence 50, Elsevier.

[Whitley 1989] D Whitley, T Starkweather, D Fuquay (1989) *Scheduling problems and traveling salesmen: The genetic edge recombination operator* , [ICGA-3].

[Whitley et al 1990] Whitley D, Starkweather T, Bogart C (1990) *Genetic algorithms and neural networks: Optimizing connections and connectivity* , Parallel Computing, 14, 347-361.

[Karlsen et al 1992] Karlsen BL, Tatari E, Hansen KK (august 1992) *Genetiske Algoritmer: Analyse og anvendelse af to principielle strukturer*, written examn (unpublished), Computer Science Department, Aarhus University.

[Yao 1992] Yao X (1992) *A review of evolutionary artificial neural networks*, Technical report. Victoria, Australia: Commonwealth Scientific and Industrial Research Organisation.