Pernambuco Federal University
Informatics Center

Model Selection of RBF Networks via Genetic Algorithms

By
E. G. M. de Lacerda

THESIS

Recife, PE - Brazil
March, 2003

Pernambuco Federal University
Informatics Center

Model Selection of RBF Networks Via Genetic Algorithms

By
E. G. M. de Lacerda

THESIS

Submitted to the Informatics Center of Pernambuco Federal University in partial fulfillment of requirements for degree of doctor of Philosophy, 2003.

Abstract

One of the main obstacles to the widespread use of artificial neural networks is the difficulty of adequately defining values for their free parameters. This work discusses how Radial Basis Function (RBF) neural networks can have their free parameters defined by Genetic Algorithms (GAs). For such, it firstly presents an overall view of the problems involved and the different approaches used to genetically optimize RBF networks. It also proposes a genetic algorithm for RBF networks with a nonredundant genetic encoding based on clustering methods. Secondly, this work addresses the problem of finding the adjustable parameters of a learning algorithm via GAs. This problem is also known as the model selection problem. Some model selection techniques (e.g., crossvalidation and bootstrap) are used as objective functions of the GA. The GA is modified in order to adapt to that problem by means of occam's razor, growing, and other heuristics. Some modifications explore features of the GA, such as the ability for solving multiobjective optimization problems and handling objective functions with noise. Experiments using a benchmark problem are performed and the results achieved, using the proposed GA, are compared to those achieved by other approaches. The proposed techniques are quite general and may also be applied to a large range of learning algorithms.

Resumo

Um dos principais obstáculos para o uso em larga escala das Redes Neurais é a dificuldade de definir valores para seus parâmetros ajustáveis. Este trabalho discute como as Redes Neurais de Funções Base Radial (ou simplesmente Redes RBF) podem ter seus parâmetros ajustáveis definidos por algoritmos genéticos (AGs). Para atingir este objetivo, primeiramente é apresentado uma visão abrangente dos problemas envolvidos e as diferentes abordagens utilizadas para otimizar geneticamente as Redes RBF. É também proposto um algoritmo genético para Redes RBF com codificação genética não redundante baseada em métodos de clusterização. Em seguida, este trabalho aborda o problema de encontrar os parâmetros ajustáveis de um algoritmo de aprendizagem via AGs. Este problema é também conhecido como o problema de seleção de modelos. Algumas técnicas de seleção de modelos (e.g., validação cruzada e bootstrap) são usadas como funções objetivo do AG. O AG é modificado para adaptar-se a este problema por meio de heurísticas tais como narvalha de Occam e growing entre outras. Algumas modificações exploram características do AG, como por exemplo, a abilidade para resolver problemas de otimização multiobjetiva e manipular funções objetivo com ruído. Experimentos usando um problema benchmark são realizados e os resultados alcançados, usando o AG proposto, são comparados com aqueles alcançados por outras abordagens. As técnicas propostas são genéricas e podem também ser aplicadas a um largo conjunto de algoritmos de aprendizagem.

For God

and

for my father.

Acknowledgments

I would like to express my gratitude to my supervisors, Dra. Teresa Ludemir and Dr. André de Carvalho, for their permanent guidance and patience throughout my research. I also thank to them, Dr. Aluísio Araújo and Dr. Francisco Carvalho for revising the manuscript of the thesis. I thank my Parents for their love and support and my brothers for encouragement. Finally, I wish to acknowledge CNPq, FAPESP and FACEPE for their support.

Contents

List of Figures

Notation of Chapter 2

S	feasible set
D	solution set
N	population size
l	bit string (chromossome) length
f_i	fitness
$\bar{f}$	average fitness
$f(\cdot)$, $g(\cdot)$	objective (or fitness) functions
H	schema
$O(H)$	order of a schema H
$\delta(H)$	defining length of a schema H
$U(x,y)$	uniform distribution (being x and y the lower and upper limits of this distribution)
$N(\mu,\sigma)$	normal distribution with mean μ and standard deviation σ
σ	standard deviation
$r \sim F$	it indicates that r is a random number drawn from a distribution F
$\mathbf{c}_j = [c_{j1}, c_{j2}, \ldots, c_{jn}]^\mathsf{T}$	jth offspring (real-coded GA)
$\mathbf{p}_j = [p_{j1}, p_{j2}, \ldots, p_{jp}]^\mathsf{T}$	jth parent (real-coded GA)

Notation of Chapter 3

$\mathcal{D}$	dataset
$(\mathbf{x}_i, y_i)$	ith example of the dataset
$\mathbf{x}_i$	input vector
y_i	desired output (output for short)
p	number of examples in dataset
$\mathcal{H}$	hypothesis space
h	hypothesis
$e(h)$	true error of hypothesis h
$\widehat{e}(h)$	estimate of true error of hypothesis h
m	number of hidden units
$\lVert\cdot\rVert$	euclidean norm
$\langle\cdot\rangle$	average
$\mathbf{c}_j = [c_{1j}, c_{2j}, \ldots, c_{nj}]^\mathsf{T}$	vector center of the jth hidden unit
n	dimension of the input space (or the number of input units)
$\mathbf{w} = [w_1, \ldots, w_m]^\mathsf{T}$	Weight vector. w_j is the weight connecting the jth hidden unit and the output unit
w_0	bias
σ_j	width of the jth hidden unit
α	overlap factor for widths
$z_j(\cdot)$	activation function (or basis function) of the jth hidden unit
$\mathbf{Z}$	design matrix, which is a matrix with the jth column $\left[z_j(\mathbf{x}_1), z_j(\mathbf{x}_2), \ldots, z_j(\mathbf{x}_p)\right]^\mathsf{T}$
$\mathbf{Z}^+$	pseudo-inverse of $\mathbf{Z}$
$\mathbf{y} = [y_1, y_2, \ldots, y_p]^\mathsf{T}$	desired output vector
β	ridge or regularization parameter
$\mathbf{I}$	identity matrix

Notation of Chapters 4 and 5

$\mathcal{D}$	decoding function
$\mathcal{E}$	problem-specific genetic encoding
$\mathcal{F}$	feasible set
$\mathcal{P}$	phenotype space
$\mathcal{G}$	genotype space
$\mathcal{L}$	legal set
$\mathbf{P}$, $\mathbf{Q}$	chromossome (or individual)
$\mathbf{p}_i$, $\mathbf{q}_i$	ith component of a chromossome (in general, encodes parameters associated with a basis function)
S_i	ith cluster (K-means algorithm)
R_i	ith region of the input space (associated with cluster S_i)
K	number of regions (or clusters)
r_j	identifier indicating that the jth center is inside the region R_{r_j} of the input space
$\mathbf{l}_i = [l_{i1}, \ldots, l_{in}]^\mathsf{T}$	lower limits of region R_i
$\mathbf{u}_i = [u_{i1}, \ldots, u_{in}]^\mathsf{T}$	upper limits of region R_i
b_j	boolean flag: if b_j = TRUE then $\mathbf{p}_j$ is valid, otherwise $\mathbf{p}_j$ is discarded during the decoding

Notation of Chapter 6

$\mathcal{L}$	learning algorithm
$\mathcal{E}$	example space ($\mathcal{X} \times \mathcal{Y}$)
$\mathcal{X}$	input space
$\mathcal{Y}$	output space
θ	set of parameters for a learning algorithm $\mathcal{L}$
τ	training parameters
λ	it represents both adjustable parameters and chromossome
$h(\lambda, \mathcal{D})$	hypothesis h built by learning algorithm $\mathcal{L}$ with adjustable parameters λ and dataset $\mathcal{D}$
$h(\mathbf{x}; \lambda, \mathcal{D})$	represents the prediction of $h(\lambda, \mathcal{D})$ for the data point $\mathbf{x}$
$e(\lambda)$	true error of hypothesis with adjustable parameter λ
$\widehat{e}(\lambda, \mathcal{D})$	estimate of the $e(\lambda)$ using the information from the dataset $\mathcal{D}$
$\delta(x, y)$	loss function (e.g., for regression $\delta(x, y) = (x - y)^2$)
$\mathcal{D}_t$	training set
$\mathcal{D}_h$	holdout or validation set
k	occam elitism size
n	elitism size
p	number of examples in D (i.e., $p = \|D\|$)
B	number of bootstrap datasets (or samples)
$f(\lambda)$	fitness of chromossome λ
$\mathbf{a} <_p \mathbf{b}$	$\forall i(a_i \leq b_i) \land \exists i(a_i < b_i)$, the vector $\mathbf{b}$ is said to be dominated by $\mathbf{a}$
s	selection pressure (best/median fitness ratio)

Chapter 1

Introduction

1.1 Motivation

"Your requests shed light
upon your objectives."
(Book: Christian Agenda,
André Luiz - medium F.C.Xavier.)

Artificial Neural Networks (ANNs) are computational tools inspired by biological nervous system with applications in science and engineering. This work is about a kind of ANN for applications in multivariate nonlinear regression, classification and times-series called Radial Basis Function (RBF) Network. Although ANNs have usually achieved good performances in several domains, those performances and the ANN training process are directly influenced by an appropriate choice of the network architecture. Unfortunately, the space of network architectures is infinite and complicated and there is no general purpose, reliable, and automatic method to search that large space.

Several alternative approaches have been proposed to search the space of network architectures. These approaches may roughly be grouped into four categories:

- Trial and Error;
- Constructive Algorithms;
- Pruning Algorithms;

- Modern Optimization Metaheuristics.

When the Trial and Error method [15, 75] is employed, different values for the network parameters must be selected, trained and compared before the choice of an ultimate network. The disadvantage of this method becomes more apparent if, after the choice of the best values, the patterns set is changed, making necessary to restart the design process. This search can be done more efficient if heuristics are used to guide it.

In the constructive approach [19, 78, 80], a network starts its training with a minimal architecture and, according to the problem complexity, new units and connections are inserted, aiming to improve the network performance. Often the constructive approach is combined with pruning techniques [106, 21] which optimizes the networks' performance by removing units and connections that are irrelevant or redundant. Despite being fast, the constructive approach are based on hillclimbing methods and because of this they only produce suboptimal solutions.

Metaheuristics are the most exciting fields in approximate optimization techniques of the last two decades. They have had successes in solving several difficult optimization problems that arise in many practical areas. Commonly used metaheuristics are simulated annealing [50], tabu search [33] and genetic algorithms [34]. Genetic algorithms, GA, (the metaheuristic used in this work) aim at solving search and optimization problems by simulating biological evolution. GAs works with a population of individuals. An individual can be seen as a state (or a point) in the search space. A traditional GA carried out the search or optimization by means of three biologically-inspired operators namely *selection*, *crossover* and *mutation*. The GA "selects" individuals, "combines" (by means of crossover) individuals with each other and "mutates" them in order to produce a new generation of individuals. In this way, populations evolve through successive generations in the direction of the best solution. The selection operator drives the population to regions of better individuals. Mutation and crossover operators drive it to explore unknown regions of the search space. Eventually, the population converges to the best chromosome. When a GA is applied to RBF networks, it generates several networks' variations (new individuals) and combines and mutates their features, thus generating new networks with improved performances through a number of generations.

GAs are one of the standard techniques of searching in complicated search spaces. So one of the reasons to apply GAs to the RBF Networks is due to the complex nature of their optimization which involves aspects of both numerical and combinatorial optimization with a complicated multimodal cost surface.

Although most of the GAs for ANN optimization has been focused on Multilayer Perceptron, MLP, networks [105] (a very popular kind of ANN), their long training time is a strong negative factor concerning the design efficiency. RBF networks are known for requiring a much shorter training period. To take advantage of this feature, novel methods for optimizing RBF Networks via GAs are proposed in this work.

In fact, the search of network architectures (i.e., the search by the best number of hidden units of the network) is an instance of a generic problem known as model selection. The model selection problem arises repeatedly in machine learning. It is the problem of estimating the true error of different hypotheses (also called models) in order to choose the (approximate) best one. Other instances of this same problem follow:

- The search by the best amount of pruning of a decision tree;
- The search by the best degree of a polynomial fit to a set of points;
- The search by the best subset of variables of a multivariate linear regression model;
- The search by the best value of ridge (or regularization) parameter of the ridge regression.

Unlike other metaheuristics, GAs provide, in our opinion, a better framework for the model selection problem, owing to its *peculiar* capacity of handling a lot of hypotheses simultaneously and multiples (and noise) objective functions. Nevertheless, the use of this capacity for model selection has been less studied than other aspects of GAs such as encoding and genetic operators.

In spite of GA produce, in general, good results when it optimizes machine learning models, the computational cost is most of times expensive. Thus, the motivations of this work, in short, it is to explore the fast training of RBF networks to obtain a less expensive genetic optimization and explore the potential of GAs for model selection.

1.2 Objective

This work deals with the optimizing RBF networks via GAs in two aspects:

1. The genetic encoding of RBF networks (chapters 5 and 4).

   This work studies encoding issues such as redundancy and illegality (that becomes the encoding of RBF networks a no trivial one) and proposes operators and encodings to an efficient combination of RBF networks and GAs.

2. The model selection problem via GA (chapter 6).

   This work tackles the model selection problem by exploring and modifying the GA search mechanism itself. This search mechanism enables GAs to use (and to create) a lot of model selection specific heuristics (e.g., occam elitism, growing and shuffling), to cope with noise estimation of the true error and to be used for multiobjective optimization. The aim of this research intends to improve the network performance and complexity as well as to study the use of traditional model selection methods (such as crossvalidation and bootstrap) by GAs and to propose less expensive alternatives to them (but as efficient as them) whose applications are also useful for other machine learning models optimized by GAs.

1.3 Organization

This text is organized as follows.

Chapter 2 is a tutorial on Genetic Algorithms, which is a class of Evolutionary Algorithms with emphasis over the crossover operator. Chapter 3 introduces learning concepts and gives a brief introduction to RBF networks. Chapter 4 gives an introduction to the optimization of RBF networks via GAs and presents review of previous works and related problems. Chapter 5 shows a proposed genetic encoding and the respective operators. Chapter 6 describes the model selection problem and a GA approach for this problem.

Chapter 2 is based on the works [57, 58]. Chapter 4 is adapted from [60]. There are two genetic encodings presented in chapter 5, named model I and model II, respectively. The model I is in [55, 56, 60] and Model II is in [61, 59]. Finally, Chapter 6 is based on works [63, 62].

Chapter 2

Genetic Algorithms

"The capacity to proceed
joyfully through life is
a blessing. Nevertheless,
remember to watch the
direction that your feet are
taking along the way."
(Book: Christian Agenda,
André Luiz - medium F.C.Xavier.)

"To be born, to die, to be reborn again,
and to always progress. That is the Law."
(Allan Kardec's comment on
the evolution of the spirit)

2.1 Introduction

Since the 1960s, computational techniques inspired in evolutionary processes of living beings have been developed as a metaphor of the Darwinian principles of natural evolution and survival of the fittest. Such techniques are termed *Evolutionary Computation*. Genetic Algorithms, GAs (an evolutionary algorithm) have been used broadly in search and optimization.

Optimization is the search for a better solution to solve a given problem. It consists of trying several solutions and to use information collected in this process in order to improve the quality of the solutions. A simple example of optimization is the improvement of the picture of a TV set. New solutions for TV images are found by adjusting the antenna. The picture of TV gets better and better and, eventually, the best (optimal) solution is found. In general, an optimization problem has:

- A search space, which contains all possible solutions of the problem.
- An objective function, used to evaluate the solutions, associates, to each solution, a value (a figure of merit, also named fitness in the GA literature).

In mathematical terms, optimization consists of searching a solution that maximizes (or minimizes) the objective function. For example, consider the following function to be maximized [71]:

$$\begin{array}{ll} \text{Maximize} & f(x) = x\sin(10\pi x) + 1 \\ \text{Subject to} & -1 \leq x \leq 2 \end{array} \qquad (2.1)$$

Although Problem 2.1 is a simple one, it is not easy to find its highest value (the so-called *global maximum*) because there is a number of *local maxima* in this function, as shows Figure 2.1. A local maximum is the highest value of a function in a finite neighborhood but not on the boundary of that neighborhood. The maximum global for this problem is at $x = 1.85055$ whose value is $f(1,85055) = 2.85027$. Section 2.2 shows that Problem 2.1 cannot be solved by several optimization methods.

To solve Problem 2.1, a simple GA randomly generates an initial population of individuals. Each individual of the population represents a potential solution by encoding it into a data structure called *chromosome*. Each individual is evaluated and given a measure of how good it is to solve the problem at hand. This measure is named *fitness*.

The next steps produce, in successive generations, new populations. Three basic operations are used to transform a current population into a new population: *selection*, *crossover* and *mutation*. The selection operator selects a sample of individuals favoring the high-fitness individuals[1]. The crossover takes two selected individuals (called parents) and combines its parts creating new ones (called offspring). Mutation takes an

[1]The purpose here is to mimic the Darwinian principle of natural selection which high-fitness individuals must have greater survival probably.

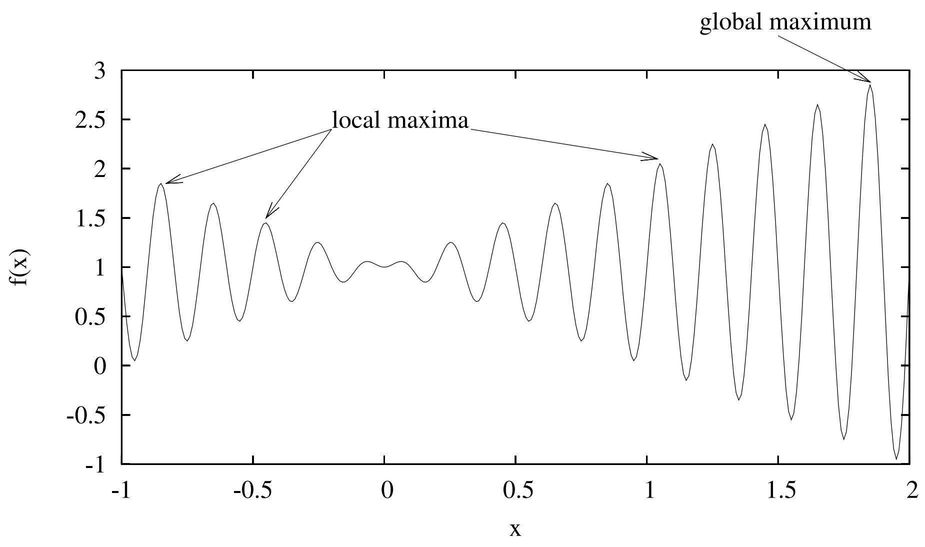


Figure 2.1: Function $f(x) = x \sin(10\pi x) + 1$

offspring and creates a transformed individual by modifying its parts randomly. The transformed individuals will compose the new generation of individuals. That procedure is repeated until a satisfactory solution is found. Figure 2.2 shows a simple GA. The following section shows each step of a simple GA in details.

2.1.1 The Binary Chromosome

The first step to solve the Problem 2.1 is to represent the parameter x of the problem as a chromosome. Here, a string with 22 bits will be used (more bits increase the numerical precision). Thus, an example of a typical chromosome could be

$$s_1 = 1000101110110101000111 \tag{2.2}$$

```
*"Let $P(t)$ be the population at generation $t$."*
$t \leftarrow 0$
initialize $P(t)$
evaluate $P(t)$
**WHILE** the stopping criterion is not satisfied **DO**
    $t \leftarrow t + 1$
    select parents $P(t)$ from $P(t-1)$
    apply crossover over $P(t)$
    apply mutation over $P(t)$
    evaluate $P(t)$
**END WHILE**
```

Figure 2.2: A Simple Genetic Algorithm

In order to know what solution chromosome s_1 represents, it is necessary to decode it. Firstly, the conversion from the binary base to the decimal base is carried out[2]:

$$d = (1000101110110101000111)_2 = 2.288.967 \tag{2.3}$$

Because d is a number in the interval $[0, 2^l - 1]$ (where l is the string size), one must map it to the range of the problem. To do this, the following formula may be used

$$x = \min + (\max - \min)\frac{d}{2^l - 1} \tag{2.4}$$

thus,

$$x_1 = -1 + (2+1)\frac{2.288.967}{2^{22} - 1} = 0,637197 \tag{2.5}$$

represents the solution coded in chromosome s_1.

It is worth noting that objective functions with several parameters have them represented in the same chromosome in such a way that each parameter takes a segment of the string. For each chromosome s_i, a fitness f_i is assigned (a measure of how good it

[2]The value of a bit string $s = (b_1, b_2, \ldots, b_l)$ (where $b_i \in \{0, 1\}$) in the decimal base is computed as follows: $d = \sum_{i=1}^{l} 2^{l-i} s_i$.

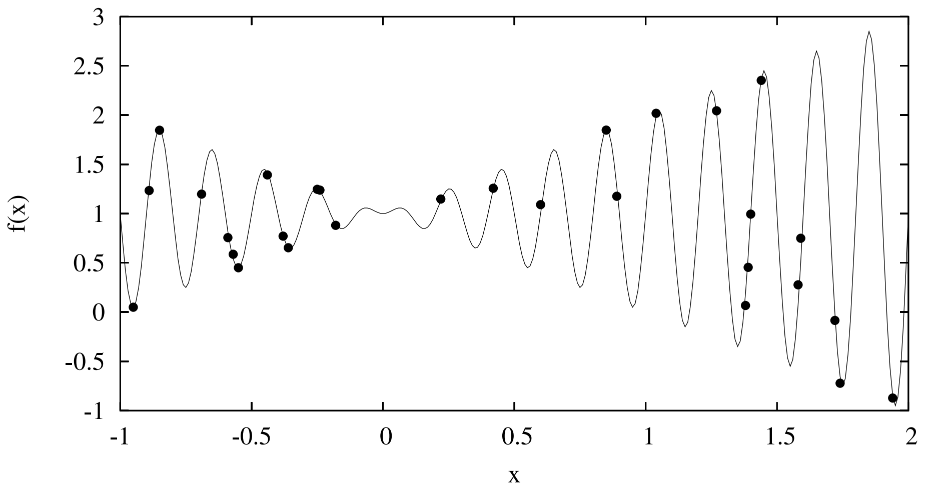


Figure 2.3: Initial population

is to solve the problem). The fitness is based on the objective function value, as shown in the next section.

2.1.2 Selection

In general, a GA starts with a random initial population. The initial population for Problem 2.1 with 30 chromosomes is shown in one of the columns of the Table 2.1. This population is sorted in decreasing order of objective function value. Table 2.1 also shows the value of the variable x represented by a chromosome, the objective function value and the fitness. The chromosomes were randomly generated because there is no prior knowledge about the region of the search space where the solution of the problem can be found. Figure 2.3 graphically shows the initial population.

Inspired in natural selection process of living beings, the GA selects the best chromosomes (i.e., the high-fitness chromosomes) from the initial population in order to generate offspring by means of the crossover and mutation operators. An intermediate population, called *mating pool*, is then created by allocating these selected parent chromosomes.

The selection algorithm most used is the *roulette wheel selection* [34]. Its basic idea is to determine the selection probability for each chromosome proportional to its fitness value. If f_i is a fitness value of i^{th} chromosome, then its selection probability p_i is given

Table 2.1: Initial Population

Rank	**Chromosome**	x_i	**Objective Function**	**Fitness**	**Accumulated Fitness**
i	s_i		$f(x_i)$	f_i	$\sum_{k=1}^{i} f_k$
1	110100000001111011011 1	1,43891	2,35251	2,00000	2,00000
2	1100000110100100011111	1,26925	2,04416	1,93103	3,93103
3	1010111001010110010000	1,04301	2,01797	1,86207	5,79310
4	1001111000011001000101	0,85271	1,84962	1,79310	7,58621
5	1001110110111000011100	0,84829	1,84706	1,72414	9,31035
6	0000110011111010010110	-0,84792	1,84610	1,65517	10,96552
7	0011000000100111010010	-0,43570	1,39248	1,58621	12,55172
8	0111100101000001101100	0,42098	1,25777	1,51724	14,06897
9	0100000000110011101000	-0,24764	1,24695	1,44828	15,51724
10	0100000010001111011110	-0,24343	1,23827	1,37931	16,89655
11	0000100101000000111010	-0,89156	1,23364	1,31035	18,20690
12	0001101001100010101111	-0,69079	1,19704	1,24138	19,44828
13	1010000110011000011011	0,89370	1,17582	1,17241	20,62069
14	0110100001011011000100	0,22292	1,14699	1,10345	21,72414
15	1000100011110001000011	0,60479	1,09057	1,03448	22,75862
16	1100110011001010001110	1,39988	0,99483	0,96552	23,72414
17	0100011001000100011101	-0,17655	0,88140	0,89655	24,62069
18	0011010011110100101000	-0,37943	0,77149	0,82759	25,44828
19	0010001101001100101100	-0,58633	0,75592	0,75862	26,20690
20	1101110101101111111111	1,59497	0,74904	0,68966	26,89655
21	0011011011001101110110	-0,35777	0,65283	0,62069	27,51724
22	0010010001001111100111	-0,57448	0,58721	0,55172	28,06897
23	1100101110110011111000	1,38714	0,45474	0,48276	28,55172
24	0010011001100110100111	-0,54999	0,45001	0,41379	28,96552
25	1101110010010100100001	1,58492	0,27710	0,34483	29,31035
26	1100101011000111010011	1,37631	0,06770	0,27586	29,58621
27	0000010000100100110001	-0,95144	0,04953	0,20690	29,79310
28	1110100001000000010001	1,72169	-0,08458	0,13793	29,93103
29	1110101000111100000000	1,74494	-0,72289	0,06897	30,00000
30	1111101100000001010111	1,94147	-0,87216	0,00000	30,00000

by

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \tag{2.6}$$

The roulette wheel algorithm mimics a weighted roulette wheel. The roulette wheel is built in such a way that the size of its slots is proportional to the fitness value of its corresponding chromosome. One selects a chromosome by spinning the wheel. By repeating this process a number of times equal to the population size, the set of the selected parent chromosomes is produced.

The roulette wheel method can be performed by the following practical procedure: one computes a column of accumulated fitness in a table as, for example, in the Table 2.1. Next, one generates a random number r (from an uniform distribution) between 0 and SumFitness, where SumFitness is the sum of the fitnesses for the whole population. Finally, the chromosome to be selected is the first (from top to bottom) one that has the accumulated fitness larger than r. For example, if $r = 28.131$ then the chromosome in the line 23 of the Table 2.1 is selected and its copy is allocated in the mating pool. The same steps are repeated until the mating pool is filled with a number of individuals equal to the population size. The roulette wheel algorithm is showed in Figure 2.4.

A way of obtaining the fitness values is to set them equal to an objective function value. Nevertheless, the roulette wheel algorithm does not work with negative objective function values. Moreover, an objective value too large can take a very large slot of the wheel causing convergence problems like the ones showed in Section 2.4.5. It is possible to leave the roulette wheel algorithm and to use the *n-way tournament selection* algorithm instead. In this case, n chromosomes are chosen with same probability, and the chromosome with highest fitness is selected and then allocated in the mating pool. The same steps are repeated until the mating pool is filled with a number of individuals equal to the population size. The 2-way tournament selection (also called binary tournament selection) is commonly used. Another way of avoiding those problems is to re-scale the value of objective function suitably (as shown in Section 2.4.6) or to use the so-called *ranking* method [6].

In Table 2.1, the fitnesses were defined by the ranking method. The first (the best) chromosome in the ranking had an arbitrary fitness equal to 2.0, and, for the last (the worst) chromosome the value 0.0 was assigned (a better way to do this is shown in

```
$\mathrm{SumFitness} \leftarrow \sum_{i=1}^{N} f_i$          /* sum up all fitness values in the population */
$\mathrm{Rand} \leftarrow \mathrm{random}(0, \mathrm{SumFitness})$   /* generate a random number between 0 and SumFitness */

$\mathrm{PartialSum} \leftarrow 0$
$i \leftarrow 0$
REPEAT
    $i \leftarrow i + 1$
    $\mathrm{PartialSum} \leftarrow \mathrm{PartialSum} + f_i$
UNTIL PartialSum $\geq$ Rand
RETURN $i$                            /* return the $i^{th}$ chromosome */
```

Figure 2.4: Roulette wheel parent selection algorithm

Section 2.4.6). The remainder fitnesses were set by interpolating those two extremes using a straight line, that is, $f_i = 2(N-i)/(N-1)$, where N is the population size.

2.1.3 Crossover and Mutation

Crossover and mutation are search mechanisms employed to scan unknown regions of the search space. Crossover is the main genetic operator and mutation is, in general, regarded as a background operator. Crossover takes two selected chromosomes (called parents) from the mating pool and swaps their parts creating a new one as follows: an integer number k, called cut point, is selected uniformly from the interval $[1, l-1]$ where l is the string length. Next, two of the selected individuals swaps their parts between the position $k+1$ and l, inclusively. For example:

```
parent_1  (00101010111|100000111111)        (00101010111|010010101100)  offspring_1
parent_2  (00111110101|010010101100)  ==>   (00111110101|100000111111)  offspring_2
                       |                                 |
                       cut point                         cut point
```

Crossover is applied with a given probability called the *crossover rate*. Typically, the crossover rate is between 0.6 and 1.0. If crossover does not pass in the probability test, then the offspring are formed by identical copies of the parents. This fact allows that some solutions are not destroyed by crossover. The probability test can be implemented by generating a random number r between 0 and 1. If $r <$ crossover rate then the crossover is performed.

After crossover, *each* bit of the offspring may be mutated (i.e., flipped) with a low probability, called *mutation rate*. For example:

$$\begin{array}{ccc} \text{offspring} & & \text{mutated offspring} \\ (0010\boxed{\mathbf{1}}01011010\boxed{\mathbf{0}}10101100) & \Longrightarrow & (0010\boxed{\mathbf{0}}01011\ 010\boxed{\mathbf{1}}10101100) \end{array}$$

Note that mutation may destroy relevant information of the chromosome. On the other hand, mutation also allows new information to be created into a chromosome. Hence the use of the mutation is useful but must be moderated by using a low mutation rate (typically 0.0001). Figure 2.5 shows the first generation of the chromosomes for Problem 2.1. The crossover was applied to each pair of chromosomes of the mating pool and mutation was applied to every bit of the offspring. The first generation, graphically showed in Figure 2.6, presents very little improvement, that is, several chromosomes are still far from the global maximum.

Next, the GA continues creating new generations for a fixed number of generations. Figure 2.7 and 2.8 show the eighth and the last (or twentieth-five) generations where the largest part of the population is close to the global maximum. Most of the times, there is not clear stopping criterion for GA, however, if 95% of the population is representing the same objective function value then it is possible to say that the GA converged. When the maximum value of the objective function is known, then this value may be used as a stopping criterion.

2.1.4 Elitism

Figure 2.9 shows, for each generation, the value of the objective function for the best chromosome and the average value of the objective function over the whole population. It is worth noting that the best chromosome may be lost, when the GA goes from the current generation to the next generation, due to either crossover or mutation. Therefore, it is interesting to transfer the current best chromosome to next generation without

Ranking	**Intermediate Population** (**m***ating pool*)	**Crossover and Mutation**	**First Generation** (the underline indicates mutation)	**Cut point**
1	1101000000011110110111		1101000000011000011100	12
5	1001110110111000011100		1001110110111110110111	12
12	0001101001100010101111		000110100110**0**010010110	12
6	0000110011111010010110		00**1**01100111100**0**0101111	12
19	0010001101001100101100		0**1**10001101000100011101	9
17	0100011001000100011101	●	0100011001001100101100	9
15	1000100011110001000011	●	1000100011110001010010	17
7	0011000000100111010010	●	00**0**1000000100111000011	17
10	0100000010001111011110		0100000010001111101100	16
8	0111100101000001101100		0111100101000001011110	16
6	0000110011111010010110		0000110011111010010110	21
18	0011010011110100101000		001101001111010010**0**000	21
9	0100000000110011101000		0100**1**00000**0**10010101111	13
12	0001101001100010101111		0001101001100011101000	13
17	0100011001000100011101		0100011001000100010010	17
7	0011000000100111010010		001100**1**000100111011101	17
11	0000100101000000111010	●	000011100101011001000**1**	3
3	1010111001010110010000	●	10101001010**1**0000**0**1101**1**	3
3	1010111001010110010000	●	1010111001010110010111	19
1	1101000000011110110111		1101000**1**00011110110000	19
15	1000100011110001000011		1000100010011001000101	9
4	1001111000011001000101		1001111001110001000011	9
13	1010000110011000011011		1010000110011000011010	20
6	0000110011111010010110		0**1**0011**1**01111101001011**0**	20
15	1000100011110001000011		1000010000100100110001	3
27	0000010000100100110001		0000100011110001000011	3
8	0111100101000001101100		011**00**001010000**1**1101100	19
19	0010001101001100101100		**1**010001101001100101**0**00	19
8	0111100101000001101100		0111100101000**1**00111010	5
11	0000100101000000111010		00001001010000**1**1101100	5

Figure 2.5: Chromosomes of the first generation

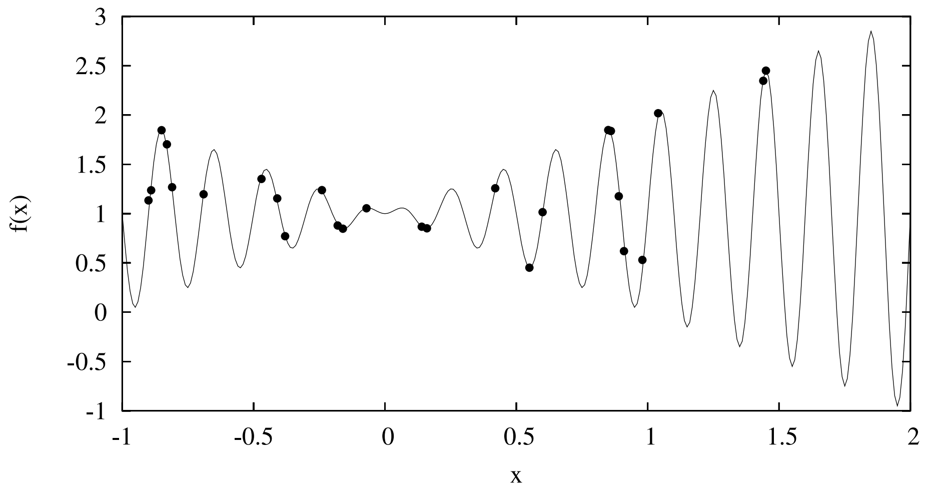


Figure 2.6: First generation

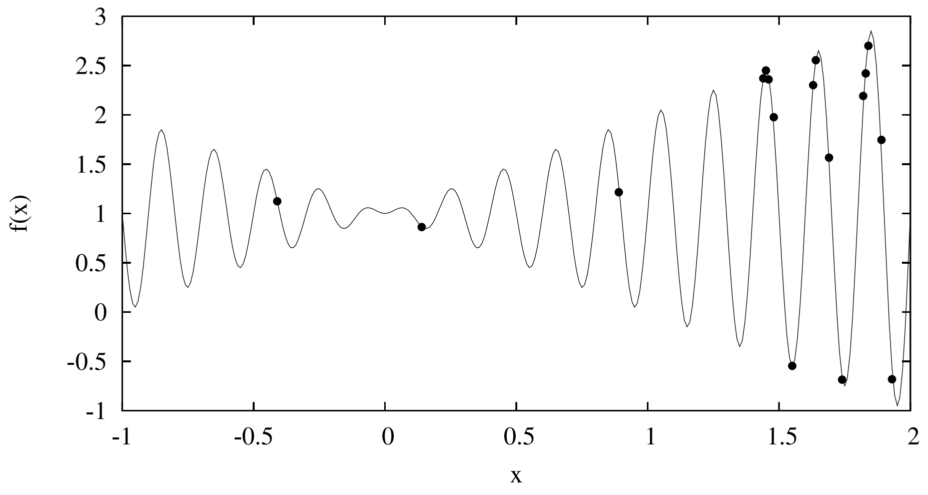


Figure 2.7: Eighth generation

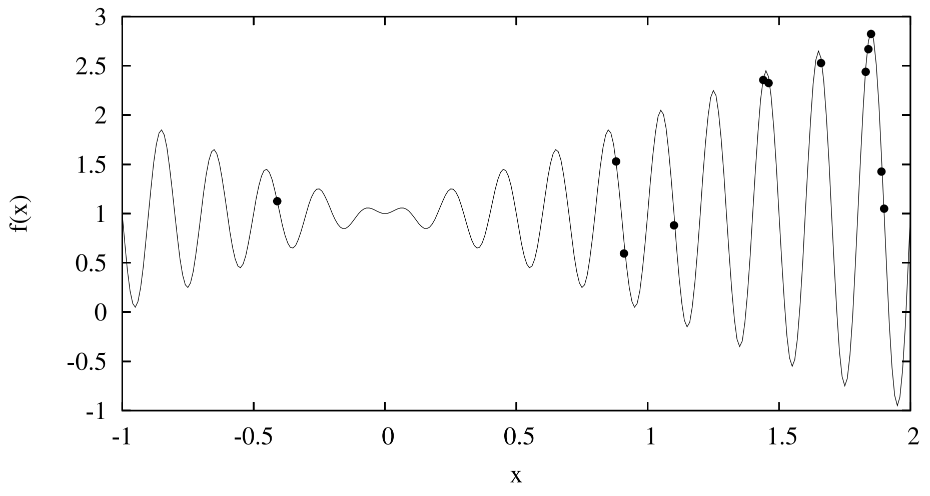


Figure 2.8: Twentieth-five generation

modifications because, in principle, there is no reason to lose the best solution. This strategy is called *elitism* which is very common in traditional GAs. The elitism was proposed by DeJong (1975) [25] in one of the pioneering works about GAs.

Figure 2.10 shows the performance of the best chromosome through generations, using the GA with and without elitism. The values plotted in the graph represent the average over 100 runs of the GA. In this problem 2.1, it is clearly shown that the GA with elitism found the solution faster than the GA without elitism. Because GAs sometimes find local maxima instead of global maximum, for both GAs the average falls bellow the value of the global maximum.

2.1.5 n-Point and Uniform Crossovers

The types of crossover operators more broadly used for bit string representation are the n-point and uniform crossovers. The 1-point crossover is the same crossover presented in Section 2.1.3. The 2-point crossover is shown in Figure 2.11. Two cut points are randomly chosen and the section between the two cut points are interchanged. The 4-point crossover is shown in Figure 2.12. The most used n-point crossover had been the 2-point crossover.

In the uniform crossover, the offspring takes each bit from one of the parents with equal probability. This can be performed as follows. For each pair of parents, a mask

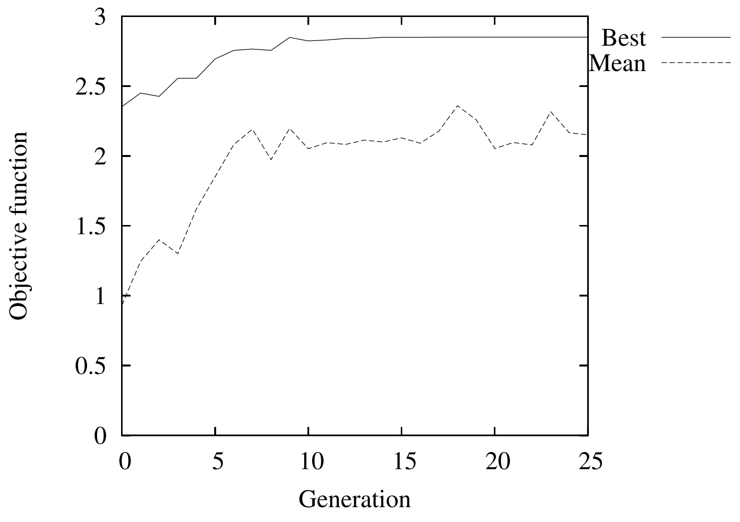


Figure 2.9: The maximum and mean value of objective function as a function of generation

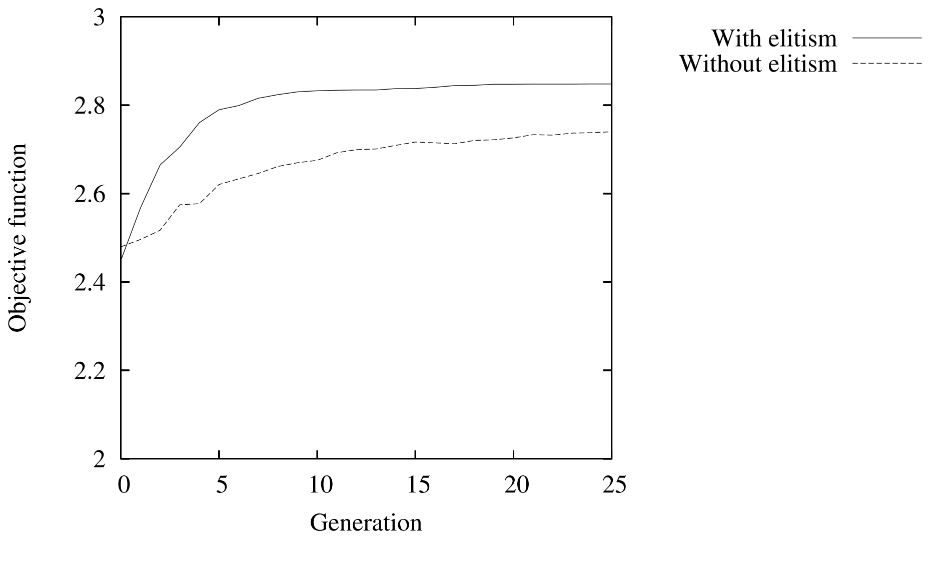


Figure 2.10: GA with and without elitism

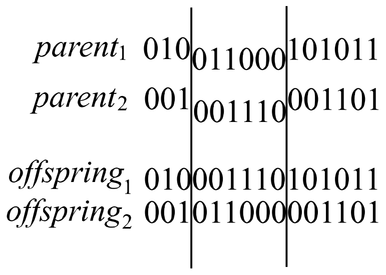

parent1 010 011000 101011
parent2 001 001110 001101
offspring1 010 001110 101011
offspring2 001 011000 001101

Figure 2.11: 2-point-crossover

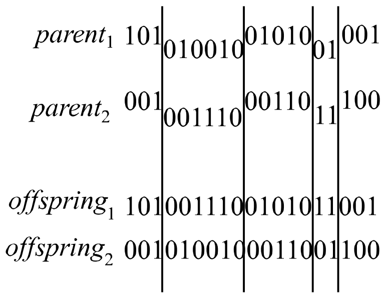

parent1 101 010010 01010 01 001
parent2 001 001110 00110 11 100
offspring1 101 001110 01010 11 001
offspring2 001 010010 00110 01 100

Figure 2.12: 4-point-crossover

of random bits is generated, as can be seen in Figure 2.13. If the first bit of the mask is equal to one then the first bit of the parent$_1$ is copied to the first bit of the offspring$_1$. Otherwise, the first bit of the parent$_2$ is copied. This procedure is then repeated for the remainder bits. In the creation of the offspring$_2$ the procedure is inverted: if the first bit of the mask is equal to one then the first bit of the parent$_2$ is copied to the first bit of the offspring$_2$ and so on. Note that the uniform crossover is not the same thing as the $(l-1)$-point crossover, where l is the bit string size, once $(l-1)$-point crossover always takes half of the bits from each parent.

In [27], the performance of several crossovers was studied. The conclusion, according to [11], is that there is no large difference among them. Moreover, the GA is, in general, robust in such a way that even under a certain range of parameter variation (e.g., mutation and crossover rate, number of cut points), its performance is not significantly modified [37]. More discussion is shown later in Section 2.1.8.

Next section presents the relationship between terms used in the GA literature and Biology.

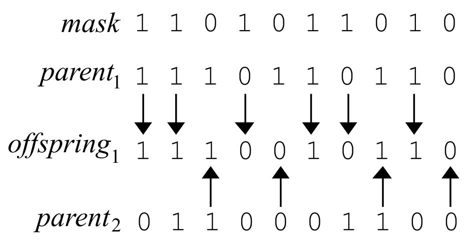

mask 1 1 0 1 0 1 1 0 1 0
parent$_1$ 1 1 1 0 1 1 0 1 1 0
offspring$_1$ 1 1 1 0 0 1 0 1 1 0
parent$_2$ 0 1 1 0 0 0 1 1 0 0

Figure 2.13: Uniform crossover

2.1.6 GA Terminology

The GA literature is rich in terms from Biology. Assuming that a potential solution may be represented as a set of parameters, some definition follows.

Chromosome. The data structure in which a potential solution is encoded.

Gene. A parameter of a potential solution. Genes are joined to form a chromosome.

Individual. This is formed by a chromosome (or more than one) and its fitness.

Alleles. The values that a gene can take. For example, if a gene represents colors then its alleles are blue, yellow, green and so on.

Genotype. The terms genotype and chromosome are often used interchangeably in GA literature. But there is a distinction between them. In Biology a genotype is formed by one or more chromosomes. Genotype is the total genetic information of an organism or phenotype. As in most of the GA applications an unique chromosome contains the total genetic information of the organism, then this unique chromosome also represents the genotype of the organism.

Phenotype. The solution or organism built from the genotype. For example, consider a chromosome which codes parameters (such as dimensions of beams) of the design of a bridge. The encoded parameters represents the genotype of the bridge. Its respective phenotype is the real bridge over the river (built from this genotype).

Epistasis. In Biology, epistasis is the iteration between genes in which the presence of a gene suppresses the effect of other genes. Consider, for example, the optimization of the combination of colors of a doctor's office. Genes are the color of clothes, furniture and wall. White is, in general, the color used for clothes. White is always the good color for clothes, it doesn't matter what the colors of the others genes are. Thus, clothes is a non-epistatic gene once it does not iterate, in general, with others genes. Furniture and wall are clearly epistatic genes, once a good color for the wall can lose its effect in the presence of a strange color for the furniture. Mathematically, epistasis means nonlinearity.

To a complete description of the GA terms, see [45].

2.1.7 Schema Theorem

The Holland schema theorem aims to give a theoretical foundation for GAs. However, its understanding can improve the constructing of practical GA applications. According to this theory, GA handles certain parts of the bit string called schemata. A schema is a string formed by symbols 0, 1, and *. The star * is the a "don't care" symbol which matches either a 0 or a 1.

Examples of schemata are showed in Figure 2.14. The schemata H_1 = 1****, H_2 = **10* and H_3 = *0*01 contain the same string 10101 (which is contained in $2^5 = 32$ schemata). Strings 11001, 11011, and 10101 are contained in the same schema 1****. The following definitions are useful:

Definition 2.1 **Order** *of a schema H, in symbols $O(H)$, is the number of 0s and 1s present in H.*

Definition 2.2 **Defining length** *of a schema H, in symbols $\delta(H)$, is the distance between the first 0 or 1 symbol and the last 0 or 1 symbol (see Figure 2.14).*

One can see, in Figure 2.15, that there are a relationship between certain schemata and the fitness value. For example, strings that start with 1 seem to have higher fitness. In other words, strings that are contained in the schema 1**** seem to have higher fitness. Figure 2.15 also suggests that strings with many 1's have higher fitness. In other words, strings contained in schemata like 111*1, 1*11*, 111**, 111*1 seen to have higher fitness than strings contained in schemata with many 0's as 000*0, 0*00*, 000** and 000*0.

	H_1	H_2	H_3
	1****	**10*	*0*01
11001	$\surd$		
11011	$\surd$		
10101	$\surd$	$\surd$	$\surd$

$$\begin{array}{ll} \delta(H_1) = 0 & O(H_1) = 1 \\ \delta(H_2) = 1 & O(H_2) = 2 \\ \delta(H_3) = 3 & O(H_3) = 3 \end{array}$$

Figure 2.14: Strings and Schemata

Given a schema H, an interesting issue is to know the ratio between the number m of strings in the current population belonging to the schema H and the number m' of strings in the next population belonging to the schema H. To know this ratio, suppose a GA with roulette wheel selection, but without crossover and mutation. Let b be the average fitness over all strings of the current population and let a to be the fitness of the schema H. The fitness of the schema H is defined as being the average fitness over all strings of the current population belonging to H. Thus the desired ratio between m and m' is given by

$$m' = \frac{a}{b} m \tag{2.7}$$

Equation 2.7 says that the number of schema H increases in next population if H is an *above-average schema* (i.e., if $a > b$). In fact (and this can be easily proved [34]), the number of above-average schemata increases *exponentially* in the subsequent populations. This claim is *almost* the schema theorem, because this analysis does not considered the crossover and mutation effects. These operators disrupt the schemata before they pass to next population. For example, consider the following crossover operation:

Schema in parent p_1	$(01 * \mid * * 10)$
Schema in parent p_2	$(* * * \mid * 101)$
Schema in offspring c_1	$(01 * \mid * 101)$

The schema in p_2 (which has a *short* defining length) pass to the offspring c_1, however, the schema in p_1 (which has a *long* defining length) do not pass to the offspring. Note that long schemata are easily disrupted by crossover. However, there is a kind of schema which is not easily disrupted by crossover. These very important schemata are called building blocks.

> **Building block**. A schema is a building block if it is short, above-average and of low order.

Even though these are a lot of obstacles to the passing traffic of schemata from the current population to the next one, the Holland schema theorem says that [34]:

> **Holland schema theorem**. The number of building blocks increases exponentially in the subsequent populations.

According to schema theorem, GAs samples large rates of building blocks to following populations. Is this a good strategy to a GA find the optimal? Yes, but only if following hypothesis holds:

> **Building block hypothesis**. Joining building blocks together in the same chromosome gives higher fitness.

If the above hypothesis holds then GAs are working correctly by favoring the building blocks and by eliminating the bad schemata. Functions in which the building block hypothesis fails are called GA-deceptive. Fortunately, the building block hypothesis holds for many of objective functions of the real world. GA-deceptive functions tend to contain a isolated optimal surrounded by bad points. Finding such isolated optimal is like to find a needle in a haystack. It worth noting that GA-deceptive functions are also difficult to any optimization method.

Other important theoretical issue is on the quantity of information handled by GAs. In spite of handling only a population of n chromosomes, the actual quantity of information handled by GA is much larger. It can be proved that GAs handles $O(n^3)$ schemata! [34]. This large quantity of information handled with only n chromosomes is termed *implicit parallelism* and it is one of the explanations for good performance of GAs.

Keeping in mind that GAs depends on recombination of buildings blocks, then the encoding of the solution must encourage the formation of building blocks. Per example, if two genes are related (epistatic) then they must be close to each other in the encoding in order to avoid easy disruption of these genes.

String	Fitness
10010	300
00110	5
11100	400
00111	100

Figure 2.15: A population

2.1.8 Which Crossover is the Best?

In [27], crossovers operators were compared empirically. However, this work did not achieve a conclusive answer, about which crossover is the best, because the differences of performance among them were not large enough. It follows that, in practice, to know which is the best crossover appears to be no critical issue (at least for traditional crossovers).

Theoretically, under the light of the schemata, it is possible to analise the crossovers. For example, the schema 1*******11 is disrupted by 1-point-crossover in any cut point. There is not this problem with the 2-point-crossover. Note that there are several cut points in which 2-point crossover does not disrupt this building block (e.g., 1**|****|*11). In theory, 2-point-crossover is better than 1-point-crossover. This claim is visualized when the chromosome is interpreted as a ring. See Figure 2.16. Under this view point, 2-point-crossover can become a 1-point-crossover if its first cut point is always fixed on the junction of the ring. Therefore 1-point-crossover is merely one particular case of the 2-point-crossover. Furthermore, 2-point-crossover allows the formation of building blocks on a junction of the ring whereas 1-point-crossover does not allows this formation. So 2-point-crossover handles more building blocks than 1-point-crossover.

N-point-crossovers suffer from *linkage* phenomenon in which adjacent genes tend to be inherited together as a group. As an example, consider an objective function whose genes a, b, c, d, e, f, g, h, i, and j are coded into a string such as

1010	11011	1011	11101	10101	10101	1010	1101	1110	0101
a	b	c	d	e	f	g	h	i	j

The adjacent genes (such as e and f) tend to be inherited together. Genes coded apart (such as b and g), by contrast, tend to be separated by 1-point-crossover or 2-

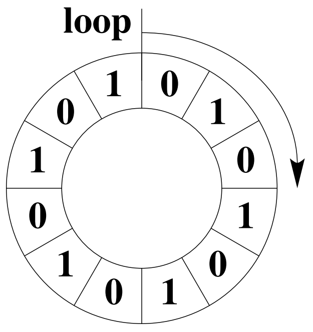


Figure 2.16: Chromosome interpreted as a ring

point-crossover. This fact seen to indicate that the ordering of genes can influence the evolution. So a few techniques have been proposed for automatic reordering the positions of genes during the evolution (e.g., the inversion operator [47]). But there is a drawback here: if, besides optimizing the genes, GA optimized the ordering of genes then a search space would increase enormously. This is one of the reasons because techniques as the inversion operator have been rarely used.

Uniform crossover does not suffer of the linkage phenomena since ordering of genes is irrelevant for it. Uniform crossover has also other advantages: 2-point-crossover can not combine any material of the parents but uniform crossover can. For example the offspring 11111 can be produced by parents 01001 and 10110 using uniform crossover. But there are not a way to produce offspring 11111 from 01001 and 10110 using 2-point-crossover. Moreover, uniform crossover disrupts schemata of same order with same likelihood. Under N-point-crossover, such likelihood varies according to the defining length of the schema. Table 2.2 summarizes the comparison between uniform crossover and 2-point-crossover.

2.2 Optimization via GA

2.2.1 A Brief Introduction

In general, a numeric function optimization problem has the form

$$\begin{array}{rl} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in S \end{array}$$

Table 2.2: Uniform crossover vs. 2-point-crossover

Uniform crossover	2**-point-crossover**
Combines any material of the parents.	Only a limited quantity of material of parents can be combined. That is, an offspring produced by 2-point-crossover can be also produced by uniform crossover. But the reciprocal claim does not hold.
Disrupts schemata of same order with same likelihood.	Does not disrupt schemata of the same order with same likelihood (depends on the defining length).
Short schemata are easily disrupted.	Short schemata are difficult to be disrupted.
Long schemata are easily disrupted, but less likely than under 2-point-crossover.	Long chromosome are easily disrupted.
The ordering of genes is irrelevant.	The ordering of genes is relevant.

where $f : D \to \Re$ is known as *objective function*. D is called *search space*, or *solution space*. Commonly $D = \Re^n$. S is called *feasible set* and $S \subseteq D$. In what follows, some examples are given:

Example 1

$$\begin{aligned} \text{minimize} \quad & f(x_1, x_2) \\ \text{subject to} \quad & x_1^2 + x_2^2 = 5 \end{aligned}$$

Example 2

$$\begin{aligned} \text{minimize} \quad & f(x_1, x_2) \\ \text{subject to} \quad & x_1^2 + 2x_2^2 \leq 5 \end{aligned}$$

Example 3

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in \Re^n \end{aligned}$$

There are two types of solution for an optimization problem:

1. **Local minimum**: a point $\mathbf{x}^* \in S$ is a local minimum of f over S if there is an $\epsilon > 0$ such that $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ for all $\mathbf{x} \in S$ within a distance ϵ of $\mathbf{x}^*$.

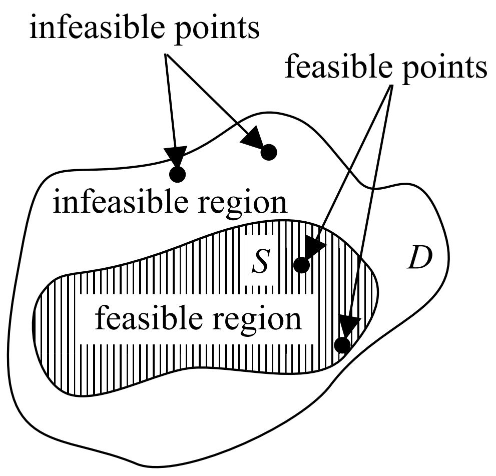


Figure 2.17: Infeasibility

2. **Global minimum**: a point $\mathbf{x}^* \in S$ is a global minimum of f over S if $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ for all $\mathbf{x} \in S$.

There are two kinds of objective functions: *unimodal functions*, which have an unique minimum, and *multimodal functions*, which have multiple local minima. Optimization problems can be classified into categories as follows:

1. **Constrained vs. unconstrained**. An optimization problem is said be constrained if its parameters **x** have constraints or limits (see examples 1 and 2). Figure 2.17 shows a constrained problem. The feasible set contains all feasible solutions (i.e., the solutions that satisfy all constraints). A problem is said be unconstrained if its parameters can take any value (see example 3). Most optimization techniques work better with unconstrained optimization.

2. **Linear vs. nonlinear programming**. A problem is known as a linear programming problem when it is formulated in terms of linear objective functions and linear constraints. The standard technique to solve linear programming problems is the Simplex Method [23]. On the other hand, if the objective function and/or constraints are nonlinear, then the problem is said be a nonlinear programming problem. There is not a general technique to solve nonlinear programming problems. However, for some particular cases of nonlinear problems (e.g., the so-called quadratic programming), there are efficient computational techniques.

3. **Continuous vs. discrete**. Parameters can be continuous or discrete. Problems with continuous parameters have an infinite number of solutions. Discrete parameters occur in a special kind of optimization known as *combinatorial optimization* and have only an finite number of potential solutions. In general, combinatorial optimization involves a certain combination of the parameters as, for example, find the best ordering of a list of tasks or actions.

2.2.2 GA and the Others Methods

In order to place GAs in the context of optimization consider some kinds of optimization methods.

1. **Generate and test search** (also known as *exhaustive* or *random search*) uses two modules. A generator module produces systematically or randomly potential solutions. A test module evaluates every potential solution and then accept or reject it. The generator module could generate all solutions before the test module run. However, it is more common an interchanged usage of the two modules. The generate and test search is an unintelligent approach, rarely used in practice.

2. **Analytical methods** use calculus techniques to obtain the minimum of a function. They have drawbacks: they give no information whether the minimum is either local or global and they require derivatives. Furthermore, if the number of parameters is large, then it is very hard to find every minimum, making analytical methods impracticable for real world problems.

3. **Downhill methods** starts the search at some random point of the search space. Next, “downhill” moves are made on the objective function surface. Eventually the bottom of the surface is reached. Several methods use the information from derivatives (gradient) to make intelligent “downhill” movements. Nevertheless, this approach does not guarantee that the reached bottom of the surface is the global minimum. For example, Figure 2.18 shows the case where “downhill” moves, starting at a random point A, lead the algorithm to point B. And, because no further downhill moves can be made from B, the algorithm is stuck in the local minimum B. State of the art downhill methods that use derivatives include the conjugate-gradient family methods (e.g., Fletcher-Reeves algorithm) and the

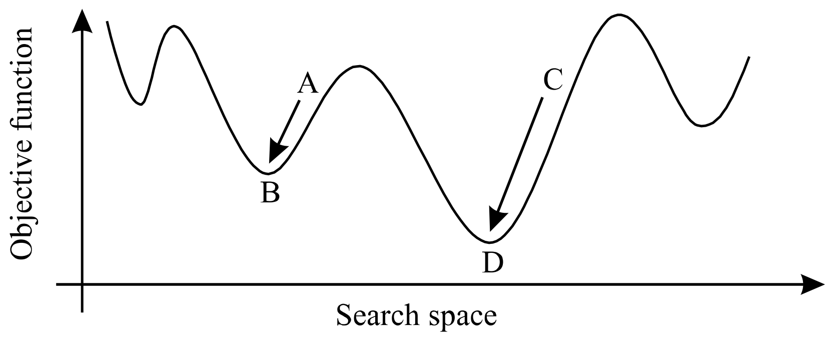


Figure 2.18: The downhill method

quasi-newton family methods (e.g., BFGS algorithm). Simplex Nelder-and-Mead downhill method is an interesting alternative because it does not use derivatives. The downhill methods are efficient and fast. Their drawback is the incapacity to find the global minimum, unless the function is well-behaved and unimodal [66, 9, 82].

GAs have been used in complicated optimization problems, where downhill and analytic methods fail. Some advantages and features of the GAs follow:

- GAs works with both continuous and discrete parameters or combinations of them.
- GAs execute simultaneous searches over several regions of the search space, once GAs work with a population instead of an unique point.
- GAs use a payoff (objective function) information, instead of derivatives or other auxiliary knowledge.
- GAs do not need a profound mathematical knowledge about the problem at hand in order to solve it.
- GAs optimize a large number of parameters.
- GAs work with a coding of parameters, not the parameters themselves.
- GAs can optimize several objective functions (multiobjective optimization) providing a list of solutions, not a simple solution.

- GAs are flexible to deal with arbitrarily constrained optimization problems.
- GAs are stochastic, not deterministic.
- GAs have successfully found global minimum even on very complex and complicated objective function surfaces.
- GA computer implementations are portable and modular. In the sense of that their search mechanism do not depend on problem-specific parts and they can be ported to other applications.
- GAs are tolerant to incomplete and noise data.
- GAs are easily hybridized (or work cooperatively) with others techniques.

GAs require a very large number of objective function evaluations. If such evaluations are expensive, then GAs can become computationally impracticable, that is, GAs cannot arrive at a good solution in a reasonable time. Furthermore, GAs converge slower than downhill methods. Thus GAs should not be used if a problem can be solved by a fast downhill method.

2.2.3 Exploration-Exploitation Trade-off

A characteristic of the generate-and-test search methods is the search for unknown points of the search space. This characteristic is called *exploitation*. On the other hand, the downhill methods characterize themselves by using information gained about the points previously visited. This characteristic is called *exploration*. Any efficient method that searches for a global minimum must use both exploitation and exploration [10].

GAs combines both exploitation and exploration. A factor that influences the quantity of exploration and exploitation in the GA search is the selection pressure. Informally, the term selection pressure is widely used to characterize either the strong (called high selective pressure) or the weak (called low selection pressure) emphasis of selection of the best individuals [5]. Informally, the selection pressure is given by the expected number of copies, in the mating pool, of the best individual[3]. In fitness-proportionate selection, the selection pressure is given by $f_{\max}/\overline{f}$ (ratio of the value of

[3]Nevertheless, more formal studies have been made (e.g.,[35, 13]).

the best fitness to the value of the average fitness). In the n-way tournament selection, the selection pressure is given by n.

Crossover and mutation lead the GA to unknown points of the search space (exploitation). The selection operator uses the fitness information of the points previously visited and drives the GA to the best regions of the search space (exploration). Eventually, the GA converges to the global minimum.

The crossover can combine the good parts of the parent chromosomes (i.e., the building blocks). Consequently, the offspring may have (in principle) higher fitness than their parents. Mutation also plays an important role in the search once it allows that any point of search space can be visited. The selection operator discards low fitness chromosomes and, thus, several genes are discarded too. It is worth noting that when the mutation is absent (low exploitation), the discarded genes cannot be recovered because the crossover does not create new genes. The crossover only combines existent genes.

When the selection pressure is very high, the GA has a behavior similar to the downhill methods (too much exploration). The best individuals have very high fitness. Such high fitness individuals tend to dominate the following populations once that their genes spread through generations with very high probability. Hence the GA converges fast (possibly to a local minimum) without exploiting unknown points of the search space.

On the other hand, when the selection pressure is low, the GA has a behavior similar to the random search (so much exploitation) because the fitness is approximately equal for all individuals of the population. Here GA becomes closely related with the so-called random walk stochastic process. The increasing of diversity is other factor that increase the exploration.

In short, if the selection pressure is low then the GA converges slowly, however, the search space is quite exploited. When the selection pressure is high, the GA converges quickly but it does not exploit unknown points.

All these factors lead the conclusion that the GA convergence depends on a compromise between exploitation and exploration. This compromise is arbitrarily regulated either by the selection pressure or by the diversity. In fact, many parameters of the GA (as shown later) are different ways of increasing either the selection pressure (exploitation) or the diversity (exploration).

Besides the selection operator, another factor may cause the loss of diversity: the *genetic drift*. The genetic drift is the variation that happens by chance in the frequence of genes. It affects mainly small populations. Obviously, the loss of diversity impedes that the GA exploits the search space. As a result, the GA may converge to a local minimum. This problem is called *premature convergence*. Adequate mutation rates can keep a good diversity of genes and therefore to combat the genetic drift.

2.3 RCGA - The Real-Coded Genetic Algorithm

This section presents the Real-Coded Genetic Algorithm, RCGA, compares the binary encoding with the real encoding and presents a list of operators for the real encoding.

2.3.1 Binary vs. Real Encoding

To illustrate this section, consider the following function to be maximized:

$$\begin{aligned} &\text{Maximize} && f(x_1,x_2) = 0.5 - \frac{\left(\sin\sqrt{x_1^2+x_2^2}\right)^2 - 0.5}{\left(1.0 + 0.001\,(x_1^2+x_2^2)\right)^2} \\ &\text{Subject to} && -100 \le x_1 \le 100 \\ & && -100 \le x_2 \le 100 \end{aligned} \tag{2.8}$$

This function is a benchmark function known as F6 in the GA literature [24]. The maximum global of F6 is at (0.0), whose value is $f(0.0) = 1$.

Binary Encoding

The Problem 2.8 has two parameters or genes (x_1 and x_2). Each gene is coded into a segment of the bit string chromosome. The bit string length depends on the required numerical precision. Each decimal point of accuracy requires 3.3 bits. If the user wants eight decimal positions then $8 \times 3.3 = 26.4 \approx 27$ bits are required for each gene. It follows that the chromosome needs 27 bits $\times$ 2 genes $=$ 54 bits for eight decimal positions. For example:

$$011010010010011010000010110100011100010000111001011000$$

In order to decode this chromosome, firstly, it is divided into two strings of 27 bits:

$$011010010010011010000010110$$
$$100011100010000111001011000$$

Next, the conversion from the binary base to the decimal base is carried out:

$$\begin{aligned} d_1 &= (011010010010011010000010110)_2 = (55129110)_{10} \\ d_2 &= (100011100010000111001011000)_2 = (74518104)_{10} \end{aligned}$$

Finally, the two strings are mapped to the interval of the problem using the formula:

$$x_i = a_i + (b_i - a_i)\frac{d_i}{2^l - 1} \tag{2.9}$$

where $x_i \in [a_i, b_i]$ and d_i is the value of the bit string, associated with x_i, converted to the decimal base. Thus

$$x_1 = -100 + (100 - (-100))\frac{55129110}{2^{27} - 1} = -17.851224 \tag{2.10}$$

$$x_2 = -100 + (100 - (-100))\frac{74518104}{2^{27} - 1} = 11.040629 \tag{2.11}$$

Real Encoding

In the real encoding, the chromosome is a vector of floating point numbers in which each component of the vector is a parameter of the problem. The solution given in (2.10) and (2.11) is straightforwardly coded into the vector:

$$[-17.851224, 11.040629]^{\mathsf{T}}$$

which is more friendly for human than a bit string. Moreover, the nature of the real encoding makes easier to propose new genetic operators (as showed later). The real encoding is also more compatible with traditional optimization methods, once the chromosome is already in the form (a vector of floating point numbers) commonly used by these methods. Therefore, the real encoding is simpler to combine or hybridize with traditional optimization methods. Besides these advantages, real encoding has shown better performance than binary encoding, mainly in optimization problems with continuous parameters. Some researchers pointed out that binary encoding is prone to generate large chromosomes [72]. For example, a problem with 100 parameters may require a string of at least 2700 bits for good accuracy. Large chromosomes make the search space large, causing loss of efficiency in the search.

The binary encoding is more portable than the real encoding, because many problems can not be coded into a vector of floating point numbers. The binary encoding is historically important. It was used in the pioneers works of GAs by Holland (1975). It is simple for theoretical analysis. Some theoretical arguments have been favorable to binary encoding, but counter arguments have also been given. Therefore, there are advocates for both binary and real encodings (see [46] for a discussion).

2.3.2 Genetic Operators

This section shows a (non exhaustive) list of operators for real encoding. Throughout this section the following notation is used. Parents are represented by vectors

$$\begin{aligned}\mathbf{p}_1 &= [p_{11}, p_{12}, \ldots, p_{1l}]^\mathsf{T} \\ \mathbf{p}_2 &= [p_{21}, p_{22}, \ldots, p_{2l}]^\mathsf{T}\end{aligned}$$

and the offspring by:

$$\mathbf{c} = [c_1, c_2, \ldots, c_l]^\mathsf{T}$$

where $p_{ij}, c_i \in \Re$. If more than one offspring exists, then the i^{th} offspring is represented by $\mathbf{c}_i = [c_{i1}, c_{i2}, \ldots, c_{il}]^\mathsf{T}$. Genes can have arbitrary constraints. But, for simplicity, in this section the gene c_i of the offspring $\mathbf{c}$ is in the interval $[a_i, b_i]$ where $a_i, b_i \in \Re$. If the gene c_i does not satisfies this constraint (i.e., $c_i \notin [a_i, b_i]$) then the offspring is said to be infeasible. $U(x, y)$ denotes an uniform distribution being x and y the lower and upper limits of this distribution. $N(\mu, \sigma)$ denotes a normal distribution with mean μ and standard deviation σ. The notation $r \sim F$ indicates that r is a random number drawn from a distribution F (for example, $r \sim U(x, y)$).

Most of the genetic operators for RCGA may roughly be put into four classes [32]:

1. Conventional operators;

2. Arithmetical operators;
3. Direction-based operators;
4. Mutation operators.

Conventional Operators

Conventional operators are adaptations from the operators for binary encoding (e.g., n-point and uniform crossovers). Conventional operators work well with binary encoding, but with continuous encoding they merely exchange genes values without create new information (i.e., new continuous numbers). It is better then to use arithmetical operators.

Arithmetical Operators

Arithmetical operators perform a linear combination of the parent vectors.

Average or intermediate crossover [24]. Given two parents $\mathbf{p}_1$ and $\mathbf{p}_2$, the offspring $\mathbf{c}$ has the form

$$\mathbf{c} = \frac{1}{2}(\mathbf{p}_1 + \mathbf{p}_2)$$

Arithmetical crossover [71]. Given two parents $\mathbf{p}_1$ and $\mathbf{p}_2$, the offspring $\mathbf{c}_1$ and $\mathbf{c}_2$ have the form:

$$\begin{aligned} \mathbf{c}_1 &= r\mathbf{p}_1 + (1-r)\mathbf{p}_2 \\ \mathbf{c}_2 &= (1-r)\mathbf{p}_1 + r\mathbf{p}_2 \end{aligned}$$

where $r \sim U(0,1)$.

The arithmetical crossover is equal to the average crossover if $r = 0.5$. It produces an offspring enclosed in the segment of line I joining the parent points (see Figure 2.19). It follows that arithmetical crossover is not able to find the minimum if it is not enclosed in the initial population. Hence it is interesting to have a crossover that extrapolates the segment of line joining the parent points. This can be carried out by the blend crossover.

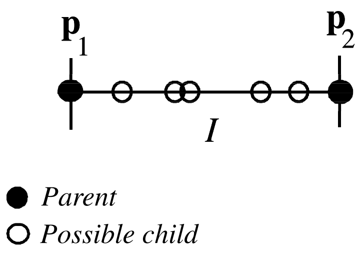


Figure 2.19: Arithmetical crossover

Blend (BLX-α) crossover [28]. Given two parents $\mathbf{p}_1$ and $\mathbf{p}_2$, the offspring $\mathbf{c}$ has the form:

$$\mathbf{c} = \mathbf{p}_1 + \mathbf{r}^\mathsf{T}(\mathbf{p}_2 - \mathbf{p}_1)$$

where $\mathbf{r} = [r_1, \ldots, r_l]^\mathsf{T}$ with $r_i \sim U(-\alpha, 1+\alpha)$.

The BLX-α is shown in Figure 2.20 for the unidimensional case and in Figure 2.21 for the multidimensional case. In the unidimensional case, if $\alpha = 0$ the offspring lies within the segment of line I joining the parent points $\mathbf{p}_1$ and $\mathbf{p}_2$. Parameter α extends the segment I. For example, if $\alpha = 0.5$, then the segment I is extended in $0.5I$ for each extreme. In the multidimensional case, BLX-α creates the offspring randomly within a hyper-rectangle defined by parents points. Unlike the arithmetical crossover, BLX with $\alpha > 0$ has an extrapolating property.

The BLX-0.5 is often used because the offspring may be either inside or outside the segment of line I with the same probability. The BLX-α had been successfully applied in many problems. It is perhaps the most used crossover for RCGAs.

Example. Consider the following two parents for the Problem 2.8:

$$\begin{aligned}\mathbf{p}_1 &= [30.173, 85.342]^\mathsf{T}\\ \mathbf{p}_2 &= [75.989, 10.162]^\mathsf{T}\end{aligned}$$

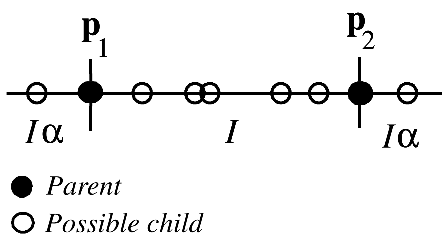


Figure 2.20: BLX-α applied to an unidimensional space

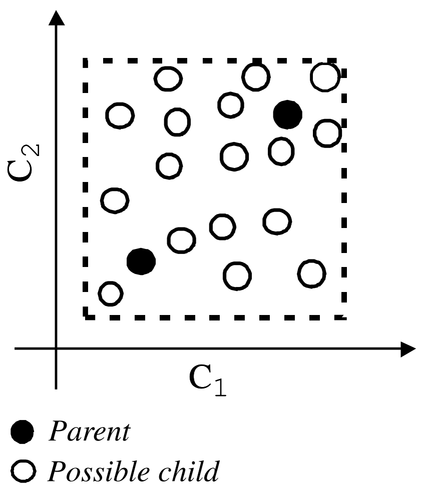


Figure 2.21: BLX-α applied to a multidimensional space

Applying BLX with extended 0.5 (BLX-0.5) and the random vector $\mathbf{r} = [r_1, r_2]^\mathsf{T} = [1.262, 0.234]^\mathsf{T}$ with $r_1, r_2 \sim U(-0.5, 1 + 0.5)$ results in:

$$\begin{aligned} c_1 &= 30.173 + 1.262(75.989 - 30.173) = 87.993 \\ c_2 &= 85.342 + 0.234(10.162 - 85.342) = 67.750 \end{aligned}$$

Thus, the offspring is given by:

$$\mathbf{c} = [87.993, 67.750]^\mathsf{T}$$

Another crossover follows.

Linear Crossover [104]. Given two parents $\mathbf{p}_1$ and $\mathbf{p}_2$, the offspring $\mathbf{c}_1$, $\mathbf{c}_2$ and $\mathbf{c}_3$ have the form:

$$\begin{aligned} \mathbf{c}_1 &= 0.5\mathbf{p}_1 + 0.5\mathbf{p}_2 \\ \mathbf{c}_2 &= 1.5\mathbf{p}_1 - 0.5\mathbf{p}_2 \\ \mathbf{c}_3 &= -0.5\mathbf{p}_1 + 1.5\mathbf{p}_2 \end{aligned}$$

Direction-based operators

Direction-based crossovers are formed by introducing fitness or gradient information into the crossover in order to determine the direction of the search.

Heuristic crossover [71]. Assuming that the fitness function $f(\cdot)$ is to be minimized, then, given two parents $\mathbf{p}_1$ and $\mathbf{p}_2$, the offspring $\mathbf{c}$ has the form:

$$\mathbf{c} = \begin{cases} \mathbf{p}_1 + r(\mathbf{p}_1 - \mathbf{p}_2) & \text{if } f(\mathbf{p}_1) \leq f(\mathbf{p}_2) \\ \mathbf{p}_2 + r(\mathbf{p}_2 - \mathbf{p}_1) & \text{if } f(\mathbf{p}_1) > f(\mathbf{p}_2) \end{cases}$$

where $r \sim U(0, 1)$ (if fitness function $f(\cdot)$ is to be maximized then above relational operators $\leq$ and $>$ must be inverted). See Figure 2.22.

Another example of an operator that uses the fitness information is the quadratic crossover [1] in which performs a numerical fit to the fitness function using three par-

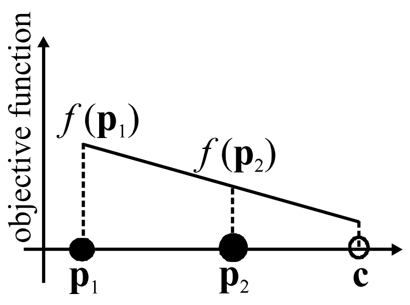


Figure 2.22: Heuristic crossover

ents [42]. In [32], there are examples of operators that use the gradient information to determine the direction of the search.

Mutation operators

Mutation operators create offspring by altering parents with a random number taken from some distribution.

Uniform mutation. It replaces the gene of the parent by a uniform random number. That is, if the k^{th} gene of $\mathbf{p}$ is selected for mutation, the offspring $\mathbf{c}$ has the form:

$$c_i = \begin{cases} U(a_i, b_i), & \text{if } i = k \\ p_i, & \text{otherwise} \end{cases}$$

where $[a_i, b_i]$ is the feasible range of the gene c_i.

Gaussian mutation. It replaces the gene of the parent by a Gaussian random number. That is, if the k^{th} gene of $\mathbf{p}$ is selected for mutation, the offspring $\mathbf{c}$ has the form:

$$c_i = \begin{cases} N(p_i, \sigma), & \text{if } i = k \\ p_i, & \text{otherwise} \end{cases}$$

Real number creep [24]. It adds a small random generated number to a gene. The random generated number can have a variety of distributions (uniform, normal, etc). So, this operator is like the uniform and Gaussian mutations, except that the randomly generated number must be small. The underlying idea is to generate similar chromosomes by adding small numbers to their genes. This is very useful when the chromosome is close to a minimum because the creep operator can quickly lead the GA to the minimum by generating chromosomes around the minimum.

Boundary mutation [71]. It replaces the gene of the parent by either its lower or its upper bound. That is, if the k^{th} gene of $\mathbf{p}$ is selected for mutation, the offspring $\mathbf{c}$ has the form:

$$c_i = \begin{cases} a_i, & \text{if } i = k \text{ and a binary random digit is 0} \\ b_i, & \text{if } i = k \text{ and a binary random digit is 1} \\ p_i, & \text{otherwise} \end{cases}$$

where $[a_i, b_i]$ is the feasible range of the gene c_i.

Non uniform mutation [71]. It replaces the gene of the parent by a random number taken from a non uniform distribution. That is, if the k^{th} gene of $\mathbf{p}$ is selected for mutation, the offspring $\mathbf{c}$ has the form:

$$c_i = \begin{cases} p_i + \Delta(t, b_i - p_i), & \text{if } i = k \text{ and a binary random digit is 0} \\ p_i - \Delta(t, p_i - a_i), & \text{if } i = k \text{ and a binary random digit is 1} \\ p_i, & \text{otherwise} \end{cases}$$

where t is the generation number. The function $\Delta(t, y)$ has the following property: it gives a value in the range $[0, y]$, such that a probability of $\Delta(t, y)$ returning a value close to zero increases as t increases. Initially (as t is small) this operator searches the space uniformly. At later generations (as t is large), it searches the space locally. In [71], the following function was used:

$$\Delta(t, y) = yr\left(1 - \frac{t}{t_{\max}}\right)^b,$$

where $r \sim U(0, 1)$, $t_{\max}$ is the maximal generation number and b is a parameter determining the degree of nonuniformity (in [71], the authors used $b = 6$ as default).

2.4 Practical Aspects

This section makes some comments about practical aspects of GA which the authors consider useful or interesting. Such comments were taken from the literature on Genetic Algorithms (especially from [42]).

2.4.1 Initial Population

Some practical comments to generate the initial population are:

1. **Prior knowledge**. Information about promising regions of the search space can be used to generate initial populations. For example, individuals can be generated with bias to the promising regions.

2. **Seeding**. Solutions obtained by other optimization methods can be inserted in the initial population. This makes sure that the GA does not perform worse than the other methods. Furthermore, GA can converge faster if the inserted solution is close to the global minimum.

3. **Dealing with small populations**. The following can be useful for very small populations which presumably do not cover some regions of the search space.

    - To generate an initial population so that it is larger than the next populations in order to improve the search space covering.
    - Initial population is generated in such a way that it represents points in the form of a grid over the search space. Hence it can uniformly cover the search space.
    - Generate the first half of the population. Next, invert every bit of the first half, then set the second half equal to the inverted first half. This insures that every bit have both the values 0 and 1 within the population. Furthermore this can improve the diversity.

2.4.2 Objective Function

The objective functions in some real-world problems can be very complicated, consuming a lot of computer time. Some problems require, in each individual evaluation, a complete simulation of a process, which may take a long time. Some ways to deal with time-consuming objective functions are suggested.

1. **Avoid revaluating duplicated individuals** by using the fitness of previously evaluated individuals. Some precautions can be useful as well:

    - Avoid generating identical chromosomes in the initial population.
    - Check whether the offspring is equal to the parents.
    - Check whether crossover and/or mutation were applied to the parents. If they were not applied, then the offspring are equal to the parents.
    - Keep the population always with distinct chromosomes. This also helps to keep the diversity.
    - Before evaluate a new chromosome, check whether it is already in some place within population. In extreme cases, it can be worth storing all chromosomes of the preceding generations in order to check whether the offspring has been evaluated in a past generation.

2. **Simplify the objective function**. In early generations of the GA, it is only necessary a rough estimation of the fitness to make the GA able to find the promising regions of the search space. Then, a simplified and faster version of the procedure that calculates the fitness could be used in early generations. In the last generations, where the individuals are similar to each other, the original objective function should be used to make an exact distinction among the individuals.

3. **Add a downhill method in the end**. A GA is fast in finding the region of the global minimum. However, the GA is slow to go down this region. In the last generations, the GA can be suspended and replaced by a downhill method (which quickly goes to the bottom of the region) to complete the search.

2.4.3 Stopping Criteria

Some usual stopping criteria for GAs are:

- **Maximum number of generations**.

- **Correct answer**. Stop if the GA found the minimum value of the objective function.
- **Convergence**. If no improvement happens during several generations, then stop. The GA can have found the global minimum, but be sure here because GAs sometimes converge to a local minimum. Alternatively, the following methods can be used:
    - A gene is said to have converged when 95% of the population share the same value. A population is said to have converged when all its genes have converged [10].
    - If either the average fitness or its standard deviation has not changed significantly under several generations, then stop the GA.

2.4.4 Generational and Steady State Replacement

GAs work by replacing individuals in a population with new ones. Common types of replacement schemes are:

- Generational replacement;
- Generational replacement with elitism;
- Steady state replacement;
- Steady state replacement without duplicates.

Consider a population with N individuals. In the generational replacement, the whole population is replaced in each generation, that is, N offspring are produced to replace N parents. Alternatively, one may replace the whole population by the N best individuals taken from the union among all parents and all offspring. However, in this case, the selection pressure is increased.

In the case of the generational replacement with elitism, the k best individuals (typically $k = 1$) in the population are never replaced by inferior individuals. This type of replacement was used in Section 2.1.4. Note that the higher the value of k, the higher the selection pressure.

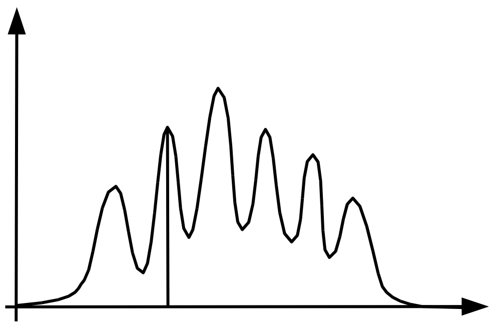


Figure 2.23: Premature convergence

In the steady state replacement, only two (or one) offspring are produced in each "generation". Then these two offspring replace the two worst individuals in the population. Alternatively, the two oldest individuals of the population may be replaced by the new ones. This is based on the idea that old individuals already have spread out their genes over the population and hence they can be discarded. The generalized form of the steady state replacement creates $k < N$ offspring to replace k parents. The crossover rate is often higher (≈ 1) in the steady state replacement than in the generational replacement.

The steady state replacement often produces many duplicate individuals in the population [24]. The steady state replacement without duplicates is an alternative scheme which does not replace an individual with an offspring if there is already a duplicate of that offspring in the population.

2.4.5 Convergence Problems

The premature convergence is a classical GAs problem. It occurs when individuals with high fitness (but not optimal) appear in the population, while the true optimal individuals have not risen yet. Such high fitness individuals (known as superindividuals) generate an excessive number of offspring. Thus, the superindividuals spread out their genes throughout the population, whereas other genes disappear. So the population is dominated by superindividuals' genes, once the population is finite. As a result, the GA converges to a local maximum/minimum, as shown in Figure 2.23.

One can prevent the premature convergence by limiting the number of offspring for each individual. This limitation can be carried out by scaling, ranking, tournament selection or other methods described in sections 2.4.6 and 2.4.7. The maintenance of the diversity of individuals also helps to combat the premature convergence, once this problem is caused by the loss of diversity itself. The increasing of the mutation rate may improve the diversity (more genes are created). Another alternative to improve the diversity is to avoid duplicate individuals in the population.

Another convergence problem in GAs is due to the overcompression of the fitness range. This is explained as follows. Consider changing the objective function of the optimization problem given by Equation 2.8 to the following new objective function:

$$f(x_1, x_2) = 2000 - \frac{\left(\sin\sqrt{x_1^2 + x_2^2}\right)^2 - 0,5}{\left(1,0 + 0,001\,(x_1^2 + x_2^2)\right)^2} \tag{2.12}$$

According to the Equation 2.12, the values of the objective function become almost the same for all population, as shown in the Table 2.3. These situations reduce the selection pressure. In other words, they practically eliminate the GA selection mechanism, since all individuals have the same selection probability. It follows that the GA loses its search capacity, becoming merely a random walk process.

A related convergence problem is known as slow finishing [10]. Because almost all individuals are similar to each other near the end of a GA run, almost all individuals have similar fitness. Again, this is a case of overcompression of the fitness range, and so the GA converges slowly. Overcompressed range problems can be combated with the same methods used to combat premature convergence: scaling, ranking or other methods shown in sections 2.4.6 and 2.4.7.

2.4.6 Remapping the Objective Function

The value of the objective function is not always appropriated to the fitness value. For example, the value of the objective function may:

- be negative (the roulette wheel selection does not work);
- lie in an overcompressed range (eliminates the selection);

Table 2.3: Overcompression of the fitness range

Individual	Objective function	Selection probability
A	2,000.999588	20.004%
B	2,000.826877	20.002%
C	2,000.655533	20.001%
D	2,000.400148	19.998%
E	2,000.102002	19.995%

- be very high with respect to others points of the function (causes premature convergence);

Because of this, remapping the objective function to an appropriated value can then be necessary. There are several ways to do this as discussed next.

Ranking

Basically, the ranking methods sort the population by the value of objective function and assign the fitness according to rank. The following paragraphs show some ranking methods.

In the linear ranking, the fitness is given by [6, 102]:

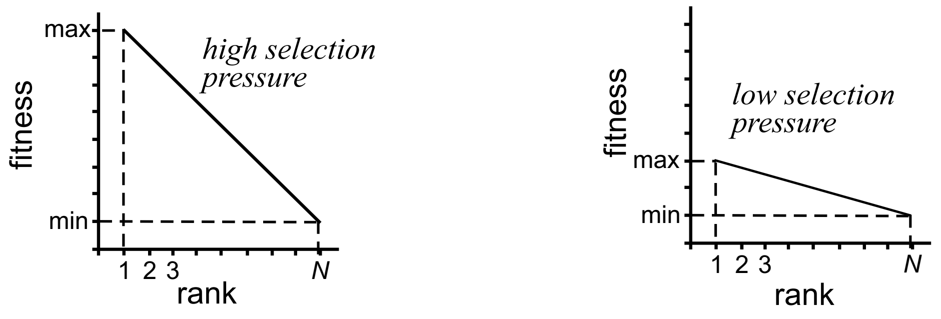
$$f_i = \text{min} + (\text{max} - \text{min})\frac{N-i}{N-1} \tag{2.13}$$

where i is the index of the individual in a list sorted in order of decreasing value of the objective function, and N is the population size. Linear ranking requires:

$$1 \leq \text{max} \leq 2 \tag{2.14}$$

$$\text{max} + \text{min} = 2. \tag{2.15}$$

It is worth noting that the linear ranking fitness represents the expected number of copies of an individual in the mating pool. Therefore, max is the expected number of copies of the best individual in the mating pool. So max can be interpreted as the selection pressure (see 2.2.3). The ranking solves the problem of overcompression of



(a) (b)

Figure 2.24: Ranking and the selection pressure

range (see Table 2.3) discussed in Section 2.4.5 by expanding the fitness range, as shown in the Table 2.4.

Table 2.4: Linear ranking

Chromosome	Objective function	Rank	Fitness	Selection probability
A	2,000.999588	1	2.0	40%
B	2,000.826877	2	1.5	30%
C	2,000.655533	3	1.0	20%
D	2,000.400148	4	0.5	10%
E	2,000.102002	5	0.0	0%

By adjusting the parameter max from the Equation 2.13, the selection pressure can be controlled. In Figure 2.24(a), the high selection pressure strongly favors the best individuals and then drives the search to the highest fitness regions of the search space (much exploitation). In Figure 2.24(b), the low selection pressure weakly favors the best individuals, driving the search to unknown regions of the search space (much exploration).

In the exponential ranking, the fitness is given by [71]

$$f_i = q(1-q)^{i-1} \tag{2.16}$$

where $q \in [0, 1]$ and i is the index of the individual in a list sorted in order of decreasing value of the objective function. Alternatively, the fitness may be normalized by dividing the Equation 2.16 by $1 - (1 - q)^N$. The exponential ranking allows a larger selection pressure than the one allowed by linear ranking.

Linear Scaling

In the linear scaling, the fitness is obtained by the following linear mapping:

$$f = ag + b$$

where g is the value of the objective function and f is the scaled fitness (see Figure 2.25). The coefficients a and b are determined by limiting the expected number of copies, in the mating pool, of the best individual (because an excessive number of copies causes loss of diversity). The linear scaling [34] transforms the fitness in such a way that the average fitness becomes equal to the average value of the objective function:

$$\bar{f} = \bar{g}, \tag{2.17}$$

and the maximum fitness becomes equal to C times the average fitness

$$f_{\max} = C\bar{f} \tag{2.18}$$

The parameter C (typically between 1.2 and 2.0) can be used to control the selection pressure. The coefficients a and b are determined as follows. We have

$$\bar{f} = \frac{1}{N}\sum_{i=1}^{N}(ag_i + b) = a\bar{g} + b \tag{2.19}$$

From 2.17 and 2.19, we have

$$\bar{g} = a\bar{g} + b \tag{2.20}$$

Note that $f_{\max} = ag_{\max} + b$. Replacing this equation and 2.19 in 2.18, we have

$$ag_{\max} + b = C(a\bar{g} + b) \tag{2.21}$$

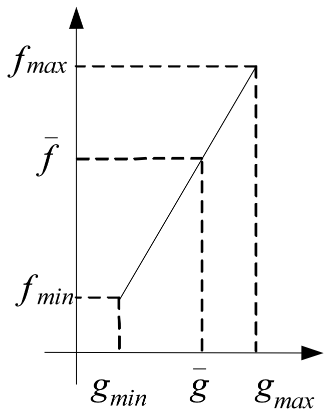


Figure 2.25: Linear scaling

From 2.21 and 2.20, we have

$$a = \frac{\bar{g}(C-1)}{g_{\max} - \bar{g}}, \qquad b = \frac{\bar{g}(g_{\max} - C\bar{g})}{g_{\max} - \bar{g}}$$

When the scaling produces negative fitnesses, the coefficients a and b are computed using another method (by imposing $f_{\min} = 0$). These two methods are in the procedure in the Figure 2.26, where the test $g_{\min} > (C\bar{g} - g_{\max})/(C-1)$ verifies whether there are negative fitnesses.

2.4.7 Selection Methods

The Darwinian natural selection is one of the biological evolution principles that GAs attempts to mimic. It provides the force that drives a GA towards the best regions of the search space. The following selection methods are broadly used:

- Roulette wheel selection;
- n-Way tournament selection;
- Stochastic universal sampling.

The roulette wheel selection is described in the Section 2.1.2. Next, the ramainder methods are described in this section.

```
$\bar{g} \leftarrow \frac{1}{N}\sum_{i=1}^{N} g_i$
**IF** $g_{\min} > (C\bar{g} - g_{\max})/(C-1)$ **THEN**
    $\Delta \leftarrow g_{\max} - \bar{g}$
    $a \leftarrow (C-1)\bar{g}/\Delta$
    $b \leftarrow \bar{g}(g_{\max} - C\bar{g})/\Delta$
**ELSE**
    $\Delta \leftarrow \bar{g} - g_{\min}$
    $a \leftarrow \bar{g}/\Delta$
    $b \leftarrow -\bar{g}g_{\min}/\Delta$
**END IF**
**RETURN** $a$ and $b$
```

Figure 2.26: Procedure for calculating the linear scaling coefficients a and b

n-Way Tournament Selection

The n-way tournament method selects each individual as follows: one randomly choose, with the same probability, n individuals and the best one (the winner) among them is selected. This procedure is repeated until to select (in general) N individuals where N is the population size. When n is equal to two, this method is called binary tournament selection.

A generalization of this method is the probabilistic n-way tournament selection. Whether or not an individual wins the tournament depends on its victory probability. The best individual of the tournament set wins the tournament with probability q (where $0,5 < q < 1$), the second one wins with probability $1 - q$, the third one wins with probability $q(1-q)^2$, and so on. The higher the tournament size n or victory probability q, the higher the selection pressure. Note that if $n = N$, the probabilistic tournament selection is equivalent to the exponential ranking, while if $n = 1$, it is equivalent to a random selection. Note also that neither scaling nor ranking is necessary for the n-way tournament selection.

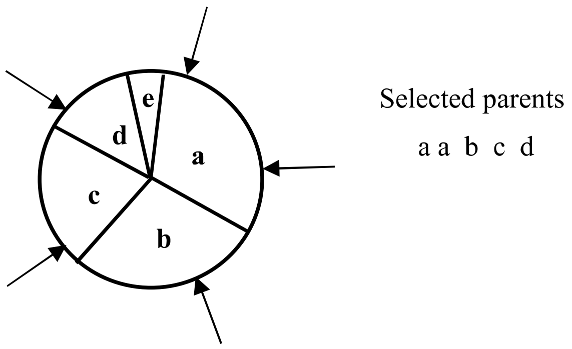


Figure 2.27: Stochastic Universal Sampling

Stochastic Universal Sampling

Both roulette wheel selection and stochastic universal sampling [6], SUS, are fitness-proportionate schemes. However, the roulette wheel selection presents a large variance for the expected number of copies (in the mating pool) of an individual. The SUS corrects this problem by using a scheme theoretically as perfect as possible.

The population is shuffled and a pie graph is built by associating each slice to an individual. The slice size is proportional to the individual's fitness. Next, N equally spaced pointers are placed around the pie where N is the population size. Finally, every pointed individual is selected, as shown in Figure 2.27.

2.5 Summary

The following are some of major points of the chapter:

- The main GA operators are three ones: crossover, mutation and selection. By favoring the high-fitness individuals, the selection operator drives the GA to high-fitness regions of the search space. Whereas crossover and mutation allow GA explore unknown regions of search space. The simultaneous combination of the three GA operators will make GA converge to the highest fitness individual.

- In general, the GA convergence depends on two aspects: exploitation and exploration. Equivalently, GA convergence depends on selection pressure and the diversity once high selection pressure provokes high exploration. Whereas high diversity provokes high exploitation.

- GA is a productive technique to deal with complex optimization problems that cannot be solved by conventional optimization methods.
- Nontraditional GAs (such as the real coded GA) are most of the time better to solve real world problems than the binary GA.
- Despite GA being straightforward to apply to lot of optimization problems, some practical aspects must be considered in the optimization via GAs. The main practical aspects to be considered to make the GA suitable to solve a specific optimization problem are:

  1. The genetic encoding;
  2. A way to create the initial population;
  3. The selection pressure and other user-controlled parameters;
  4. The design of the genetic operators;
  5. The choice of the objective function.

Chapter 3

Learning and RBF Networks

“ Give your lessons wisely
in the school of life while
the book of trials still rests
in your hands. Learning is a
blessing, and there are thousands
of brothers and sisters nearby
awaiting a scholarship
in reincarnation.”
(Book: Christian Agenda,
André Luiz - medium F.C.Xavier.)

This Chapter introduces learning concepts and gives a brief introduction to RBF networks.

3.1 The Learning Problem

Frequently, in machine learning, one looks for a function that estimate the output y of a system for an input $\mathbf{x}$. Examples of these systems arise in many different settings:

Example 1. A system for image recognition. For example, a system to classify a digital image whose input is a vector $\mathbf{x}$ containing pixels/colors of the image and the output y is the identification of the image.

Example 2. A system for illness diagnosis. For example, a system classifies a tumor by responding whether it is either benign ($y = 0$) or malignant ($y = 1$), where the input is a vector $\mathbf{x}$ containing features of the patient.

Example 3. A system for prediction of consumption of electrical energy. For example, a function whose output y is the hourly consumed energy and the input vector $\mathbf{x}$ contains the date, time of day, outside temperature, and speed wind.

Our effort in this work is to find an approximation or a prediction h for a future output y of the system. The process in which an approximated function h is built is called *learning*. The learning process has three components [97]:

1. A generator of random input vectors $\mathbf{x}$, drawn independently from a fixed but unknown distribution $P(\mathbf{x})$.
2. A supervisor which returns an output y to every input vector $\mathbf{x}$, according to a conditional distribution function $P(y|\mathbf{x})$, also fixed but unknown.
3. A learning algorithm capable of representing a set of functions $h \in \mathcal{H}$.

A learning algorithm is an algorithm to build functions $h \in \mathcal{H}$. A function h is called a *hypothesis* and the set of all hypothesis h is called the *hypothesis space* and it is denoted by $\mathcal{H}$. There are learning algorithms that represent their hypotheses as boolean functions [96], neural networks [88], decision trees [83, 14], decisions lists [86, 22], inference rules [73, 84], linear discriminant functions [29, 103], splines [99], hidden Markov models [85], Bayesian networks [44] and stored list of examples [93, 2].

The learning problem is formulated as follows.

The (supervised) learning problem is that of choosing from the given set of hypothesis $h \in \mathcal{H}$ the one which approximates best the supervisor's response. The selection is based on a dataset of p independent examples:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i); i = 1, \ldots, p\} \tag{3.1}$$

but the joint probability distribution $P(y, \mathbf{x}) = P(y|\mathbf{x})P(\mathbf{x})$ is unknown and the only available information is contained in the dataset $\mathcal{D}$.

3.2 The True Prediction Error

In order to select the best hypothesis, one estimates the *true prediction error* [26] (true error for short), denoted by e, in which is a measure of how good a hyphotesis f is at preticting the supervisor response y for input vector $\mathbf{x}$.

In regression problems, the true error $e(h)$ is given by expected squared difference between a supervisor response and a hypothesis response:

$$e(h) = \mathrm{E}\langle y - h(\mathbf{x})^2 \rangle \tag{3.2}$$

where the expectation E refer to repeated sampling of examples drawn from an unknown joint probability distribuition $P(\mathbf{x}, y) = P(y|\mathbf{x})P(\mathbf{x})$. Equation 3.2 can also be written as

$$\begin{aligned} e(h) &= \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} (y_i - h(\mathbf{x}_i))^2 & (3.3) \\ &= \int (y - h(\mathbf{x}))^2 P(\mathbf{x}, y) d\mathbf{x} dy & (3.4) \end{aligned}$$

The standard tool to estimate the true error is the k-fold-crossvalidation method [95, 51]. Other estimates of true error are holdout and bootstrap methods (see Chapter 6). Next section shows how to estimate the true error using k-fold-crossvalidation over a simple regression problem.

3.3 Estimating the True Prediction Error

Consider a learning problem whose dataset $\mathcal{D}$ is obtained as follows. The input examples x are drawn from uniform distribution $U(-4, 4)$ and the output y is drawn from conditional distribution:

$$y \sim P(y|x) = f(x) + \epsilon \tag{3.5}$$

where:

- The noise ϵ is drawn from normal distribution with mean zero and variance $\sigma^2 = 0.03$;

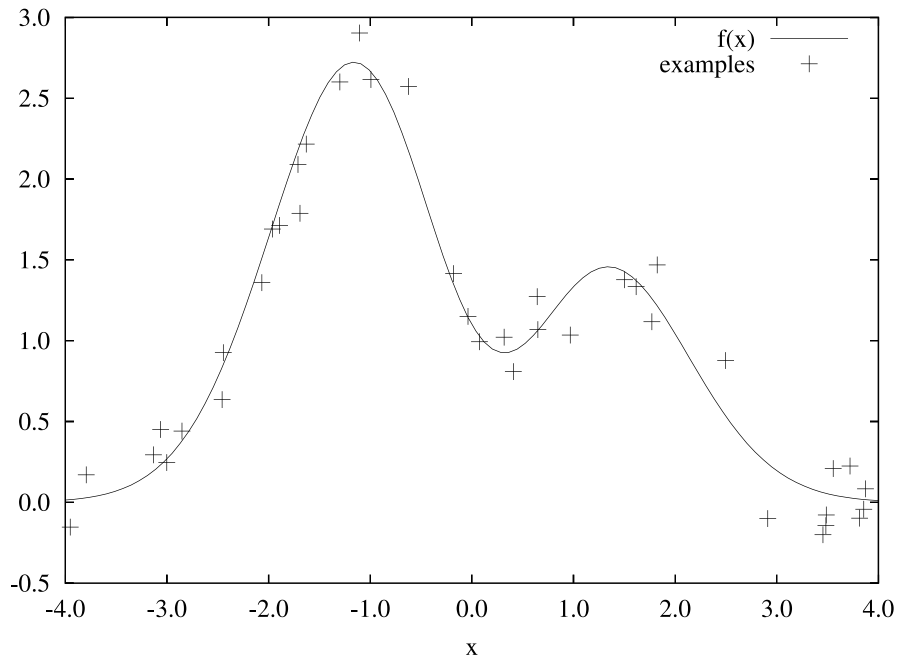


Figure 3.1: A dataset

- The function $f(x)$ is a hermite polinomial [68]:

$$f(x) = 1.1(1 - x - 2x^2)\exp\left(-\frac{x^2}{2}\right) \tag{3.6}$$

In Figure 3.1 is shown the dataset $\mathcal{D}$ in which the solid line is the function f (Equation 3.6).

In this example, the learning algorithm represents hypotheses by means of Radial Basis Function (RBF) Networks (described in Section 3.4). The learning algorithm builds a RBF network by determining its parameter setting. Unfortunately, the learning algorithm[1] is unable to determine all parameters of RBF networks (it is unable to determine the number of hidden units of the RBF network). Therefore, the number of hidden units will be determined by trial and error. In Table 3.1, it is shown 10 possible choices of hidden units and the true error of the respective hypothesis built by learning algorithm.

[1]This learning algorithm is described in Section 3.5 with overlap factor $\alpha = 1.5$.

Table 3.1: True error for several RBF network hypotheses

Number of Hidden Units	True Error
5	0.0605944
6	0.0495772
7	0.0296295
8	0.0147486
9	0.0112011
10	0.0100640 *
11	0.0104499
12	0.0188118
13	0.0151860
14	0.0186200
15	0.0222238

The best hypothesis (from Table 3.1) has 10 hidden units because it minimizes the true error. However, the true error is unknown by user and because of this the user cannot claim that the best hypothesis has 10 hidden units. Fortunately, the user may estimate the true error by k-fold-cross-validation and to use the estimates (instead of the true errors) in order to choose a hypothesis.

The k-fold-crossvalidation method divides the dataset $\mathcal{D}$ in k subsets (also named *folds*): $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$. The folds have equal size and are mutually exclusive. It produces k hypotheses $h_1, \ldots, h_k$, where each one is built by learning algorithm from the dataset $\mathcal{D} \setminus \mathcal{D}_j$ (see Figure 3.2). The performance $\widehat{e}_j$ obtained by the hypothesis h_j is measured on the dataset D_j:

$$\widehat{e}_j = \frac{1}{|\mathcal{D}_j|} \sum_{(\mathbf{x},y)\in\mathcal{D}_j} (y - h_j(\mathbf{x}))^2 \tag{3.7}$$

Let $\widehat{e}(h)$ to be the k-fold-crossvalidation estimate of the true error for hypothesis h. The estimate $\widehat{e}(h)$ is equal to average of the performances of the k hypotheses:

$$\widehat{e}(h) = \frac{1}{k} \sum_{j=1}^{k} \widehat{e}_j \tag{3.8}$$

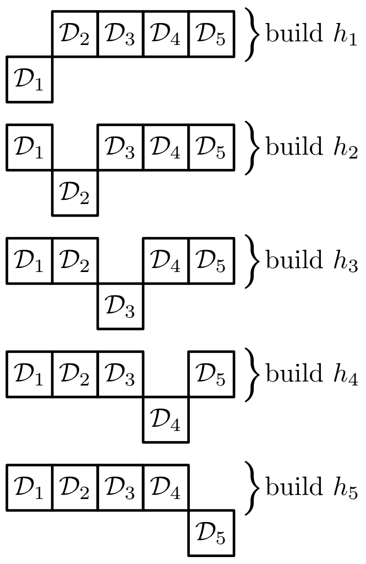


Figure 3.2: The k-fold-crossvalidation method for $k = 5$

$$= \frac{1}{k}\sum_{j=1}^{k}\left(\frac{1}{|\mathcal{D}_j|}\sum_{(\mathbf{x},y)\in\mathcal{D}_j}(y - h_j(\mathbf{x}))^2\right) \tag{3.9}$$

Because $|\mathcal{D}_1| = |\mathcal{D}_2| = \ldots = |\mathcal{D}_k| = |\mathcal{D}|/k$ then

$$\widehat{e}(h) = \frac{1}{\mathcal{D}}\sum_{j=1}^{k}\sum_{(\mathbf{x},y)\in\mathcal{D}_j}(y - h_j(\mathbf{x}))^2 \tag{3.10}$$

In Table 3.2, it is shown the 10-fold-crossvalidation ($k = 10$) estimate for each hypothesis from Table 3.1. According to this estimate (Equation 3.10), the user chooses the hypothesis with 11 hidden units. This hypothesis (plotted in Figure 3.3) is similar to the best hypothesis from Table 3.1 (in terms of number of hidden units). This result shows that 10-fold-crossvalidation provided a good estimate for this example.

Note that there are two kinds of parameters in a hypothesis: the parameters (also called training parameters) that are determined automatically by learning algorithm and the parameters (also called adjustable or tunning parameters) that are not determined by learning algorithm. In this example, the adjustable parameter is the number of hidden units of the RBF network.

Table 3.2: Estimate of true error for several RBF network hypotheses

Number of Hidden Units	True Error	Estimate of True Error
5	0,0605944	0,1488140
6	0,0495772	0,1193860
7	0,0296295	0,0866174
8	0,0147486	0,0587643
9	0,0112011	0,0871740
10	0,0100640 *	0,0719187
11	0,0104499	0,0387574 *
12	0,0188118	0,0445568
13	0,0151860	0,0505755
14	0,0186200	0,0949899
15	0,0222238	0,0458567

The problem of estimating the true error for a hypothesis using different values of adjustable parameters in order to choose the approximate best one is known as *model selection*[41].

The model selection process in this example was done by trial and error. To search by trial and error the set of values of adjustable parameters is impracticable if such set is large. In this situation, a genetic algorithm can be used to search the set of values of adjustable parameters. This issue is addressed later (in Chapter 6). Next the hypothesis representation used throughout this text (the RBF Network) is presented.

3.4 Introduction to Radial Basis Function Networks

RBF Networks have their origin in the solution of the multivariate interpolation problem [81, 15]. These networks have traditionally only one hidden layer (see Figure 3.4). Properly trained, they can approximate an arbitrary function $f : \Re^n \rightarrow \Re$ by mapping:

$$f\left(\mathbf{x}\right) \approx h(\mathbf{x}) = w_0 + \sum_{j=1}^{m} w_j z_j\left(\mathbf{x}\right) \tag{3.11}$$

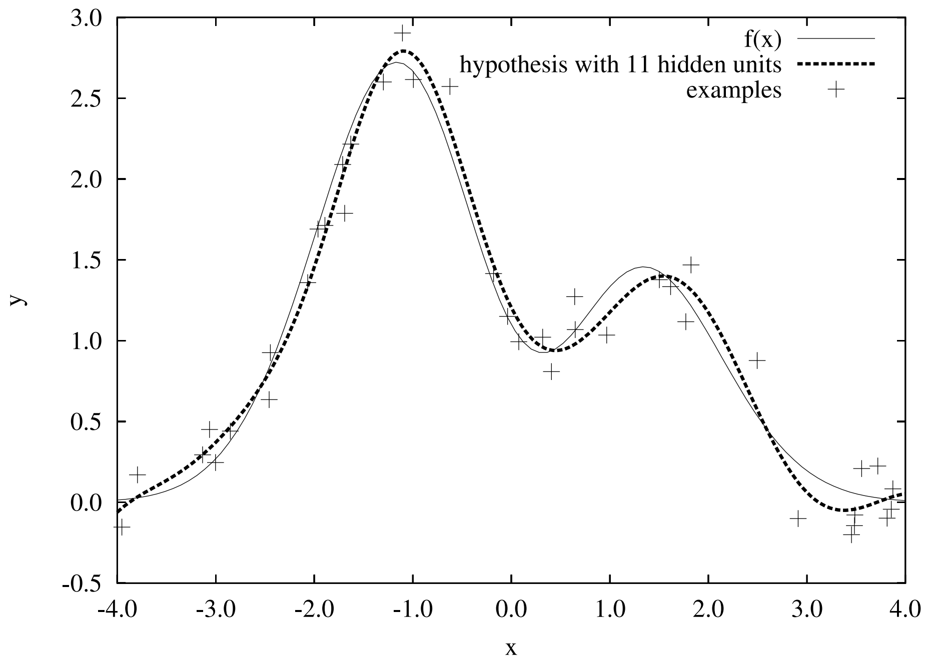


Figure 3.3: Hypothesis selected by k-fold-crossvalidation with $k = 10$

where, $\mathbf{x} \in \Re^n$, $\{w_i;\ i = 1, \ldots, m\}$ denotes the *weights* coefficients, w_0 is the *bias* and $z_j(\mathbf{x})$ represents the *activation function* (also known as radial basis function), which is given by:

$$z_j(\mathbf{x}) = \phi\left(\frac{\|\mathbf{x} - \mathbf{c}_j\|}{\sigma_j}\right) \tag{3.12}$$

where $\|\cdot\|$ is the Euclidean norm, $\mathbf{c}_j = [c_{j1}, c_{j2}, \ldots, c_{jn}]^{\mathsf{T}}$ is the *center vector*, σ_j is the *width*, which is a scaling factor for the radius $\|\mathbf{x} - \mathbf{c}_j\|$, and $\phi(\cdot)$ is a non-linear function that monotonically decreases (or increases) as **x** moves away from $\mathbf{c}_j$. A common example of a radial basis function is the Gaussian function $\phi(v) = \exp(-v^2/2)$. Others examples are: $\phi(v) = v$ (linear); $\phi(v) = v^3$ (cubic); $\phi(v) = v^2 \log v$ (thin plate spline); $\phi(v) = \sqrt{v^2 + 1}$ (multiquadratic); and $\phi(v) = 1/\sqrt{v^2 + 1}$ (inverse multiquadratic).

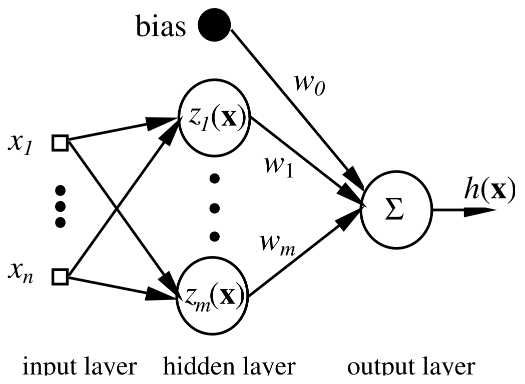


Figure 3.4: A Radial Basis Function Network

3.5 Hybrid Learning of RBF networks

Several training techniques have been proposed to train RBF networks. A well known training technique[75] employs a hybrid approach that combines unsupervised and supervised learning. In order to see how it works, let

$$\{(\mathbf{x}_i, y_i)\,;\ i = 1, \ldots, p\} \tag{3.13}$$

be the set of the training examples where $\mathbf{x}_i$ is an *input vector* and y_i its *desired output*.

The unsupervised learning stage defines the center and width of the radial basis functions. Simple methods make $\mathbf{c}_i = \mathbf{x}_{\alpha_i}$, for $i = 1, \ldots, m$, where $\alpha_i \in \{1, \ldots, p\}$ is randomly chosen. Usually, the number of centers, m, is determined by trial and error. Nevertheless, this approach is prone to generate large networks, overfitting, and numerical problems (mainly when the data set is noisy)[78].

A more efficient approach employs a clustering algorithm, such as K-means[4] or self-organizing feature map[52]. Roughly speaking, the K-means starts by randomly assigning the p input vectors $\mathbf{x}_j$ to K sets $S_1, \ldots, S_K$. Next, it computes the mean vectors of each set as:

$$\mathbf{m}_i = \frac{1}{|S_i|} \sum_{\mathbf{x}_j \in S_i} \mathbf{x}_j \tag{3.14}$$

In the following steps, it re-assigns all input vectors $\mathbf{x}_j$ to the nearest cluster S_i (i.e., nearest mean vector) and recalculates the mean vector for each cluster. This two steps procedure is repeated until there is no further change in the mean vectors. These vectors become the centers (i.e., $\mathbf{c}_i = \mathbf{m}_i$ for $i = 1, \ldots, K$). Another alternative is to partition the input space in regions using a decision-tree [53].

The widths are usually defined by computationally inexpensive heuristics [89]. Moody and Darken, in [75], suggest that a single value σ for all basis functions gives good results. They used $\sigma = \langle \|\mathbf{c}_i - \mathbf{c}_j\| \rangle$, where $\mathbf{c}_j$ is the nearest center from $\mathbf{c}_i$ and $\langle \cdot \rangle$ indicates the average over all such pairs. Others methods use a different value σ_i for each basis function. In [89], each width σ_i is defined as

$$\sigma_i = \alpha \|\mathbf{c}_i - \mathbf{c}_j\|, \tag{3.15}$$

where α is an overlap factor and $\mathbf{c}_i$ and $\mathbf{c}_j$ are defined as before.

The Least Squares Problem

In the supervised learning stage, the RBF network with fixed centers and widths can be interpreted as a case of multivariate linear regression on the training set:

$$\mathbf{y} = \mathbf{Z}\mathbf{w} + \mathbf{e} \tag{3.16}$$

where $\mathbf{y} = [y_1, y_2, \ldots, y_p]^\mathsf{T}$ is the desired output, $\mathbf{Z}$ is the *design matrix*, which is a matrix with the j th column $[z_j(\mathbf{x}_1), z_j(\mathbf{x}_2), \ldots, z_j(\mathbf{x}_p)]^\mathsf{T}$, $\mathbf{w} = [w_1, w_2, \ldots, w_m]^\mathsf{T}$ is the output layer weight vector and $\mathbf{e}$ is the error. The vector $\mathbf{w}$ is determined minimizing the sum of squared errors:

$$\text{Find } \mathbf{w} \text{ that minimizes } \mathrm{SSE} = \mathbf{e}^\mathsf{T}\mathbf{e} \tag{3.17}$$

A simple method to find the solution of this linear least squares problem may be obtained solving the well-known linear system (called *normal equations*):

$$\left(\mathbf{Z}^\mathsf{T}\mathbf{Z}\right)\mathbf{w} = \mathbf{Z}^\mathsf{T}\mathbf{y} \tag{3.18}$$

Nevertheless, this simple method to solve the least squares problem is prone to numerical problems as shown in the next section.

3.6 Computational Considerations

The three following methods are usually used to solve a least squares problem (described by the Equations 3.16 and 3.17):

- Cholesky decomposition;
- QR decomposition;
- Singular value decomposition (SVD).

The Cholesky decomposition of the matrix $\mathbf{Z}^\mathsf{T}\mathbf{Z}$ is the faster way to solve the linear system 3.18 (i.e., the normal equations). However, the solution of a least squares problem directly from normal equations 3.18 via Cholesky decomposition is susceptible to roundoff errors. Because of this, a method called QR decomposition (of the matrix $\mathbf{Z}$) may be better than Cholesky decomposition due to its numerical stability [64, 41].

Another numerical problem, called ill-conditioning[2], can arise due to large and noise training datasets. In general, Cholesky and QR decompositions fail to provide satisfactory results if there are ill-conditioned matrices (e.g., if $\mathbf{Z}^\mathsf{T}\mathbf{Z}$ is ill-conditioned) in the least square problem. In this case, the use of SVD has been recommended [82]. SVD does not directly use the normal equations, instead of this, SVD computes the pseudo-inverse matrix $\mathbf{Z}^+$. Thus, the weight vector $\mathbf{w}$ is given by

$$\mathbf{w} = \mathbf{Z}^+\mathbf{y} \tag{3.19}$$

where

$$\mathbf{Z}^+ = \left(\mathbf{Z}^\mathsf{T}\mathbf{Z}\right)^{-1}\mathbf{Z}^\mathsf{T} \tag{3.20}$$

Another way to deal with ill-conditioning problems by using regularization (see section 3.7).

3.7 Ridge Regression

Ridge regression (also called weight decay) is a particular type of regularization. The regularization technique is often used to avoid overfitting in neural networks [43]. Penalty

[2]Ill-conditioned matrices are close to singular.

or regularization functions are added to the SSE in order to control the smoothness properties of the network. The ridge regression minimizes the cost function:

$$\text{Find } \mathbf{w} \text{ that minimizes } C = \mathrm{SSE} + \beta \mathbf{w}^{\mathsf{T}} \mathbf{w} \tag{3.21}$$

where β is the regularization (or ridge) parameter, which control the smoothness of the RBF network. The solution to this least squares problem is obtained solving the linear system:

$$(\mathbf{Z}^{\mathsf{T}} \mathbf{Z} + \beta \mathbf{I}) \mathbf{w} = \mathbf{Z}^{\mathsf{T}} \mathbf{y} \tag{3.22}$$

where $\mathbf{I}$ is the identity matrix. The SVD is not necessary to solve the system (3.22), since regularization itself avoids numerical problems. Thus, faster algorithms (such as Cholesky or LU decomposition [82]) able to solve linear systems can be used instead of SVD.

This fast approach (i.e., the use of regularization combined with either Cholesky or LU decomposition) is very useful to design RBF networks using Genetic Algorithms, once short computational time to evaluate RBF networks is critical to the overall performance. Because of this, this fast approach is used in all the experiments in this work.

Chapter 4

Combining RBF Networks and Genetic Algorithms

"Do not run away from the lessons
to be learnt along your evolutionary
path, however difficult or painful
they may be, so that later on life may
open up the sanctuary of wisdom to you.."
(Book: Christian Agenda,
André Luiz - medium F.C.Xavier.)

This Chapter shows several ways to combine RBF networks and GAs. Encoding issues and some problems involved in the combination of RBF Networks with Genetic Algorithms are also described (mainly the redundancy problem). Finally, a review of previous works is given.

4.1 Combining Neural Networks and Genetic Algorithms

Roughly speaking, ANNs can be combinated with GAs in three different ways:

- **Evolutionary design**. It is the evolution of neural architectures and of the algorithmic parameters. The parameters to be optimized may include the number of layers, the number of hidden units, the activation function and the algorithmic parameters, such as learning rate.

- **ANN training**. A GA may be used as a training algorithm to optimize the values of the network weights and centers.

- **Evolution of learning rule**. Given an ANN, this method looks for an efficient learning rule.

In the case of evolutionary design, in general, each individual can be seen as a state of the space of neural architectures. Beginning the process with a population of genotypical representations of neural architectures, usually randomly generated, a generator reconstructs each network (phenotype) from its representation (genotype), according to a chosen encoding procedure. Then, in order to evaluate their performances, all the networks are trained, in general, with the same dataset. Next an evaluation function determines the current state of the population, establishing the fitness (an aptitude grade) for each architecture according to its performance or generalization over the dataset.

Through a selection method, such as the roulette wheel, the architectures are selected according to a relative probability associated to their fitnesses. The selection continues until the next population of candidates is completed. Later, the candidates go through a reproduction stage guided by mutation and crossover genetic operators. By means of these operators, a new generation of neural architectures is constructed. In order to avoid the possible disappearance of the best architectures from the population, an elitist policy can also be used, which automatically sends best networks to the next generation.

This cycle is repeated and the population evolves gradually towards genotypes that correspond to phenotypes with higher performances. This cycle is carried out a certain number of times until the algorithm finds appropriate solutions to the problem.

The evolutionary design also includes the evolution of algorithmic parameters. Examples of algorithmic parameters are the learning rate and the number of epochs of the Backpropagation algorithm or the regularization parameter of the ridge regression method used in the RBF Network training.

In ANN training, the GA is used as a training algorithm to optimize the centers and widths (and optionally the weights) of RBF networks. In the evolution of learning rule, given an ANN, this method looks for an efficient learning rule [18, 105]. In [18], learning rules, competitive with the Delta Rule (also known as LMS algorithm [103]), were evolved.

Although most of the evolutionary approaches for ANN design have been focused on MLP networks [105], their long training time is a strong negative factor concerning the chromosome evaluation efficiency. RBF networks are known for requiring a much shorter training period. To take advantage of this feature, a few methods have also been proposed to optimize the parameters of RBF networks.

RBF networks training optimization has been pursued by other approaches like OLS (Orthogonal Least Squares) [19, 78, 79] and RAN (Resource Allocating Network) [80, 48, 106]. Despite being very fast, these methods perform a local search; thus they can easily fall in local minima and produce sub-optimal solutions. GAs, on the other hand, are global search methods (see section 2.2). They can provide an efficient alternative for the optimization of RBF networks. When RBF networks are genetically optimized, several parameters may be considered, such as:

- *Number of hidden units*: m;
- *Basis functions*: $z_1, \dots, z_m$ used by each unit;
- *Centers*: $\mathbf{c}_1, \dots, \mathbf{c}_m$ where $\mathbf{c}_j = [c_{j1}, c_{j2}, \dots, c_{jn}]^\mathsf{T}$ is a vector center of the basis function z_j;
- *Widths*: $\sigma_1, \dots, \sigma_m$ where σ_j is the width of basis function z_j;
- *Weights*: $\mathbf{w} = [w_1, \dots, w_m]^\mathsf{T}$ where w_j is the weight connecting the jth hidden unit and the output unit.

The current evolutionary optimization approaches usually optimize a subset of these parameters. The weights, for example, are most of the time determinated by least squares methods. In next section, some problems involved in the combination of RBF Networks with GAs are described.

4.1.1 Encoding Issues

The choice of the adequate encoding for the chromosomes is a central issue for the optimization of RBF Networks through GAs. The encoding defines the class of neural architectures that can be evolved. Moreover, the definition of genetic operators is, in general, based on the encoding chosen. These factors contribute directly or indirectly to

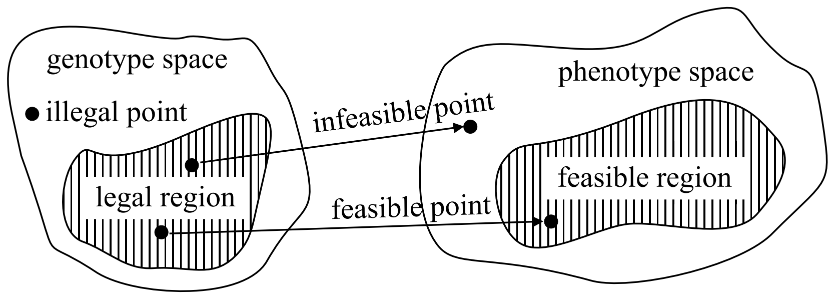


Figure 4.1: Genotype to phenotype mapping.

the efficiency (with respect to processing time and fitness values obtained) of the genetic optimization [7].

Traditional encodings use binary string. However, in order to provide a representation more suitable to the characteristics of the problem being solved, a large range of encodings have been proposed [32]. Encodings have varied from real strings (used mostly in numerical optimization) and integer permutation encodings (used in some combinatorial optimization problems) to general data structures, often used in engineering problems.

To evaluate a chromosome, GAs map a point from the genotype space to the phenotype space. From this mapping, many important issues concerning genetic encodings may arise. Here, it is useful to distinguish two important concepts on the genotype-phenotype mapping: infeasibility and illegality (Figure 4.1).

A phenotype is infeasible if it lies outside the feasible region of the optimization problem. A genotype is illegal if it cannot be mapped to the phenotype space. Note that infeasibility is derived from the nature of the constrained optimization problems whereas illegality is derived from the nature of problem-specific encoding. Therefore, infeasibility and illegality are unrelated concepts. In order to better explain these concepts, two examples of illegality are given.

Example 1: consider the Traveling Salesman Problem, TSP: a seller visits N cities (e.g., cities A, B, C, D, and E), returning to the first city. Each city is visited only once. In this example, two possible tours are (BACDE) and (EBDAC). The application of a two-point crossover [34] to these tours results in:

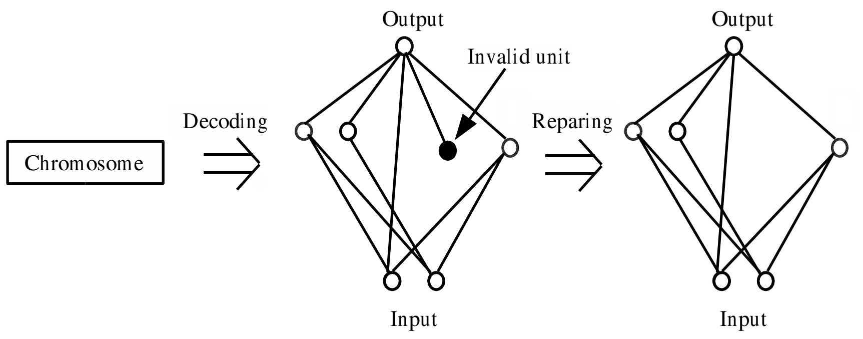


Figure 4.2: Reparing an illegal network

Tour 1	(BA|CD|E)
Tour 2	(EB|DA|C)
Offspring	(BA|DA|E) $\Rightarrow$ illegal tour

Clearly, this offspring is illegal because its tour is invalid, once the city A is visited twice. Repair techniques are usually employed to convert a illegal chromosome to a legal one. For example, the well-know PMX crossover [34] often used in the TSP is essentially a two-point crossover combined with a repair procedure to fix the illegal chromosome produced by the two-point crossover.

Example 2: some encodings used for ANNs optimization [40] can produce invalid networks. For example, the ANN from Figure 4.2 has one hidden unit without input connections. Thus, the chromosome that produced this network is illegal. A repair procedure could delete that invalid unit.

4.1.2 Desirable Properties of Genetic Encodings

Consider the following notation used througout this section.

- $\mathcal{E}$ denotes a problem-specific genetic encoding.
- $\mathcal{G}$ denotes the genotype space, i.e., the set of all genotype representable in the chosen encoding $\mathcal{E}$.
- $\mathcal{P}$ denotes the phenotype space (or solution set), i.e., the set of all solutions for the otimization problem.

- $\mathcal{F} \subseteq \mathcal{P}$ denotes the feasible set, i.e., the set of all phonotype in $\mathcal{P}$ that is not infeasible.
- $\mathcal{L} \subseteq \mathcal{G}$ denotes the legal set, i.e., the set of all genotypes in $\mathcal{G}$ that is not illegal.
- $\mathcal{D} : \mathcal{L} \rightarrow \mathcal{P}$ denotes the decoding function, i.e., the function that produces a phenotype p corresponding to a genotype g. Note that the domain of $\mathcal{D}$ is $\mathcal{L}$, once illegal genotypes cannot be mapped to the phenotype space.

Based on the works of [32] and [7], some desirable properties of genetic encodings are listed as follows.

1. *Nonredundancy*. $\mathcal{D}$ is injective. The mapping between encodings and solutions must be injective.

   If a non injective mapping occurs, the GA waste time in searching because one or more individuals may be duplicated in the genotype space. Hence, the injective mapping (nonredundancy) is a desirable property for an encoding. Next section details this property for RBF networks.

2. *Legality*. $\mathcal{G} = \mathcal{L}$. Any instance of an encoding corresponds to a solution.

3. *Feasibility*. Range $\mathcal{D} = \mathcal{F}$.

4. *Completeness*. $\mathcal{D}$ is surjective. Any solution has a corresponding encoding.

   This property guarantees that any point of the search space is accessible by the GA search.

5. *Goldness*. Let $p^* \in \mathcal{P}$ to be the best solution. If there exist a $l \in \mathcal{L}$ such that $p^* = \mathcal{D}(l)$. That is to say, the best solution has a corresponding encoding.

6. *Nonredundant completeness*. $\mathcal{D}$ is bijective. The encoding is nonredundant and complete.

7. *Strong casuality (or well-conditioning)*. Small variations on the genotype space due to mutation imply small variations in the phenotype space.

   This focus whether the neighborhood of a chromosome (in the genotype space) is also preserved in the corresponding phenotype space. A search process is strongly casual (or well-conditioned) if it do not destroy the neighborhood of a chromosome. Weak casuality (or ill-conditioning) refer to the case where small variations on the genotype space imply large changes in the phenotype space, and vice versa [32]. According to [91], strong casuality is a desirable property of genetic encodings.

4.1.3 Redundancy and Illegality in RBF Network Encodings

The encoding of RBF networks may suffer from a problem named redundancy. In the literature of genetic optimization of ANNs, redundancy is also known by different names: functional equivalence problem [77], competing conventions problem [90] and permutation problem[38]. Redundancy occurs if the mapping from chromosomes (genotype space) to the RBF Networks (phenotype space) is not an injective mapping.

Two chromosomes are redundant if their associated RBF networks perform the same input-output mapping[1]. An example of redundant encoding is shown as follows. Consider a generic chromosome of the form:

$$\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m) \tag{4.1}$$

where $\mathbf{p}_i$ encodes parameters (e.g., centers and widths) associated with the basis function z_i. By using this encoding, the RBF networks on the left and right sides of Figure 4.3 could be encoded by the chromosomes $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ and $(\mathbf{c}, \mathbf{a}, \mathbf{b})$, respectively. Although these networks can perform the same input-output mapping (since they have the same units), they have distinct chromosomes. This may significantly increases the search space.

According to [38], the traditional crossover operator is not appropriated for redundant encoding, once it may generate offsprings with duplicated basis function (illegality). For example:

Parent 1	(a	b	c)	
Parent 2	(c	a	b)	
Offspring	(a	a	b)	$\Rightarrow$ duplicated basis function

[1]In section 4.2.3, there is a formal definition of redundancy for RBF networks.

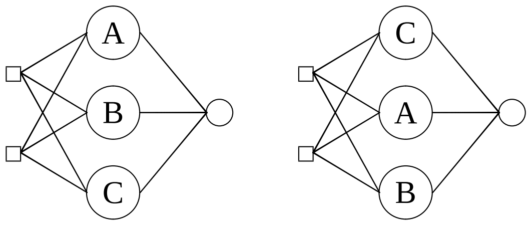


Figure 4.3: Redundant RBF networks.

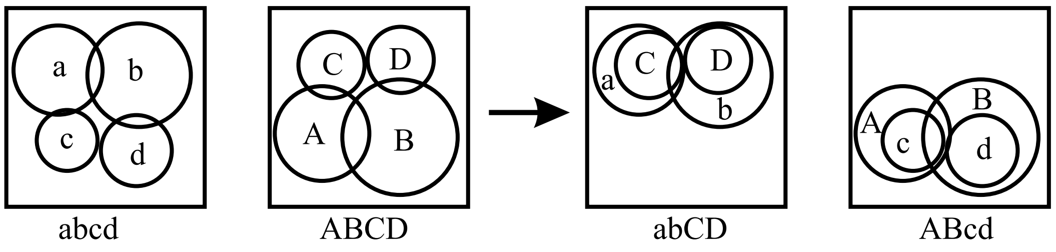


Figure 4.4: Overlapped Gaussian Functions.

Although two identical basis function into the same chromosome is unlikely, it is possible to have two similar basis function, as can be seen in Figure 4.4 [38]. In this figure, the basis functions a and C are not identical, but they are very similar because they play similar roles in the network (once they have Gaussian functions overlapping each other). Such problems make the design of the crossover operator very difficult and may significantly increase the search time.

4.2 Review of Previous Works

Three works were relevant for this research. They are described in the following sections. Other works are briefly described.

4.2.1 Selecting Centers from Patterns

Billings and Zheng [12] addressed the combinatorial aspect of RBF networks optimization [12]. In their work, a GA selects a subset of the input patterns to become the center vectors. Each chromosome is a variable-length string representing a subset of patterns.

For example, the chromosome P.

$$P = (100 \quad 7 \quad 411 \quad 286)$$

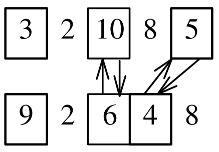
represents a RBF network with four centers placed on the patterns labeled 100, 7, 411 and 286. The authors used thin-plate-spline basis function and employed the genetic operators proposed by Lucasius and Kateman [65]. These operators are suited to solve a combinatorial optimization problem known as *subset selection problem*.

Two types of crossovers were used: the fixed length crossover and the variable length crossover. The lenghts of the parents are preserved in the fixed length crossover, while they are changed in the variable length crossover.

Let P1 and P2 be the parents. In the fixed length crossover, the common genes in both parents are first searched and two binary template strings, T1 and T2, are created to mask the common genes in both parents. The bits belonging to T1 and T2 are set to one if the corresponding gene is a common gene and zero otherwise. For example,

$$\begin{array}{llllll} P1 = & 3 & 2 & 10 & 8 & 5 \\ P2 = & 9 & 2 & 6 & 4 & 8 \end{array} \longrightarrow \begin{array}{llllll} T1 = & 0 & 1 & 0 & 1 & 0 \\ T2 = & 0 & 1 & 0 & 0 & 1 \end{array}$$

In this case, the fourth bit of T1 is set to one because the fourth gene of P1 (namely 8) is present in both parents. Next, the operator selects a random number of distinct genes from the end of P1 and exchanges the same number of distinct genes with P2. The common genes in both parents are preserved. The following example shows the offspring obtained by exchanging two genes of P1 with two genes of P2:



$$\longrightarrow \begin{array}{llllll} F1 = & 3 & 2 & 6 & 8 & 4 \\ F2 = & 9 & 2 & 10 & 5 & 8 \end{array}$$

The variable length crossover is quite similar to the fixed length crossover, but it exchanges a random number of distinct genes from the end of P1 with a random number of distinct genes from the end of P2. Figure 4.5 shows the offspring obtained by exchanging two genes of P1 with three genes of P2.

In this work, the Lucasius and Kateman's trade mutation was also used. Consider a complementary training set defined as the difference between training set and the

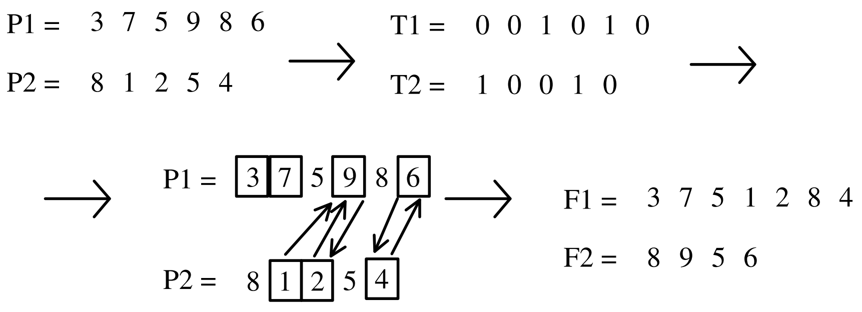


Figure 4.5: Lucasius and Kateman's variable length crossover

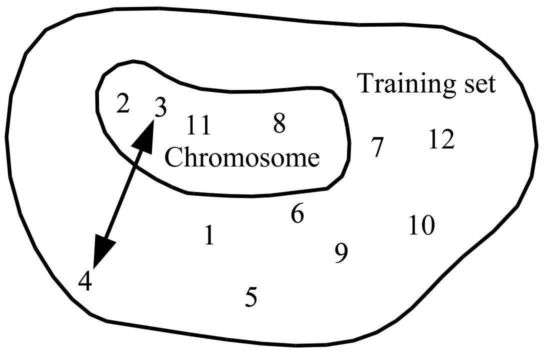


Figure 4.6: Trade mutation

set of patterns coded in the chromosome. The trade mutation replaces the genes of a chromosome with patterns randomly selected from the current complementary training set in an import-export fashion (see Figure 4.6) [31].

Addition and delete operators were also used. According to the authors, they help to keep the population diversity. The addition operator concatenates a random number of genes to the end of a chromosome. The delete operator deletes a random number of genes from a chromosome, starting from a randomly defined chromosome position. The networks were evaluated using a data set from a liquid level system. The authors used 500 training patterns, 500 validation patterns, a population of 60 individuals and each algorithm run for 400 generations. The authors used a multiobjective genetic algorithm with two objective functions: the Akaike Information Criterion (AIC) [3] over both training set and validation set in which improved the generalization.

This work, published in 1995, presents the following interesting ideas:

1. The use of a multiobjective genetic algorithm.

2. Commonly, previous GA models for neural networks minimized simple error functions such as Sum-Squared Error, SSE (or similar functions such as Mean Squared Error, MSE) whereas this GA minimizes a model selection criterion function (namely AIC).

3. In general, previous GAs models for neural networks used only the training set to compute the objective function whereas this model uses simultaneously both training and validation sets in order to compute the objective function.

4. The encoding is quite simple. However, this simplicity restricts the centers to the input training patterns causing too large regions of search space be skipped during the searching process.

4.2.2 Crossing Hypervolumes

Carse and Fogarty [17] proposed a method to genetically optimize centers by crossing hypervolumes of the input space. In this work, a chromosome $\mathbf{P}$ is represented by a list of tuples as follows

$$\mathbf{P} = (\mathbf{p}_1, \ldots, \mathbf{p}_m)\,. \tag{4.2}$$

where the tuple $\mathbf{p}_i$ is given by:

$$\mathbf{p}_i = (c_{1j}, \sigma_{1j}, c_{2j}, \sigma_{2j}, \ldots, c_{nj}, \sigma_{nj})\,. \tag{4.3}$$

A tuple $\mathbf{p}_i$ encodes the parameters of the following basis function:

$$z_i(\mathbf{x}) = \prod_{j=1}^{n} \exp\left(-\frac{(x_j - c_{ij})^2}{\sigma_{ij}^2}\right) \tag{4.4}$$

which may have a different width for each component of the center vector. This encoding has the same form of the generic (and redundant) encoding showed in section 4.1.3 (Equation 4.1). It follows that it is prone to redundancy too. This drawback was tackled by authors by means of a modified 2-point crossover which exchanges hypervolumes of the input space instead of chunks of the chromosome structure. This hypervolume is determined by two crosspoint vectors $\mathbf{a}, \mathbf{b} \in \Re^n$, whose elements are given by:

$$a_j = \mathrm{min}_j + (\mathrm{max}_j - \mathrm{min}_j)\, r_1 \tag{4.5}$$

$$b_j = a_j + (\mathrm{max}_j - \mathrm{min}_j)\, r_2 \tag{4.6}$$

where r_1 and r_2 are randomly selected from the range $[0, 1]$ with uniform probability density and $[\mathrm{min}_j, \mathrm{max}_j]$ is the allowed range for the component x_j of the input vector $\mathbf{x}$.

After the modified 2-point crossover, the first offspring contains all the tuples $\mathbf{p}_i$ from the first parent which satisfy:

$$\forall j, ((c_{ij} > a_j) \wedge (c_{ij} < b_j)) \vee ((c_{ij} + \mathrm{max}_j - \mathrm{min}_j) < b_j) \tag{4.7}$$

together with all the tuples $\mathbf{p}_i$ from second parent which do not satisfy this condition. The second offspring contains the remaining tuples $\mathbf{p}_i$ from both the parents. It is shown an example of this crossover in Figure 4.7 by crossing hypervolumes on a 2-dimensional input space.

This work suggested that redundancy problem affects the GA performance by showing experiments which the performance of the modified 2-point crossover is better than the one of the conventional 2-point crossover. This work also presented an interesting idea of crossing hypervolumes of input spaces. This idea is quite general because it can be adapted, as shown in Chapter 5, to other types of crossovers in order to cross hypervolumes in different ways.

4.2.3 Functional Equivalence of RBFs

This section shows other approach to deal with the functional equivalence problem. In [77], the functional equivalence between chromosomes is formally expressed as:

Definition 4.1 *Let* $\mathbf{p}_i = (w_i, \sigma_i, c_1, c_2, \ldots, c_n)$*. Two chromosomes* $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m)$ *and* $\mathbf{P}' = (\mathbf{p}'_1, \mathbf{p}'_2, \ldots, \mathbf{p}'_m)$ *are* functionally equivalent *if and only if there exists a permutation* π *of the set* $(1, \ldots, m)$*, such that* $\mathbf{p}_i = \mathbf{p}'_{\pi(i)}$*, for each* $i \in \{1, \ldots, m\}$*.*

Neruda [77] proposes a unique encoding, named *canonical parameterization*, to represent a class of functionally equivalent chromosomes. For such, he uses a lexicographic

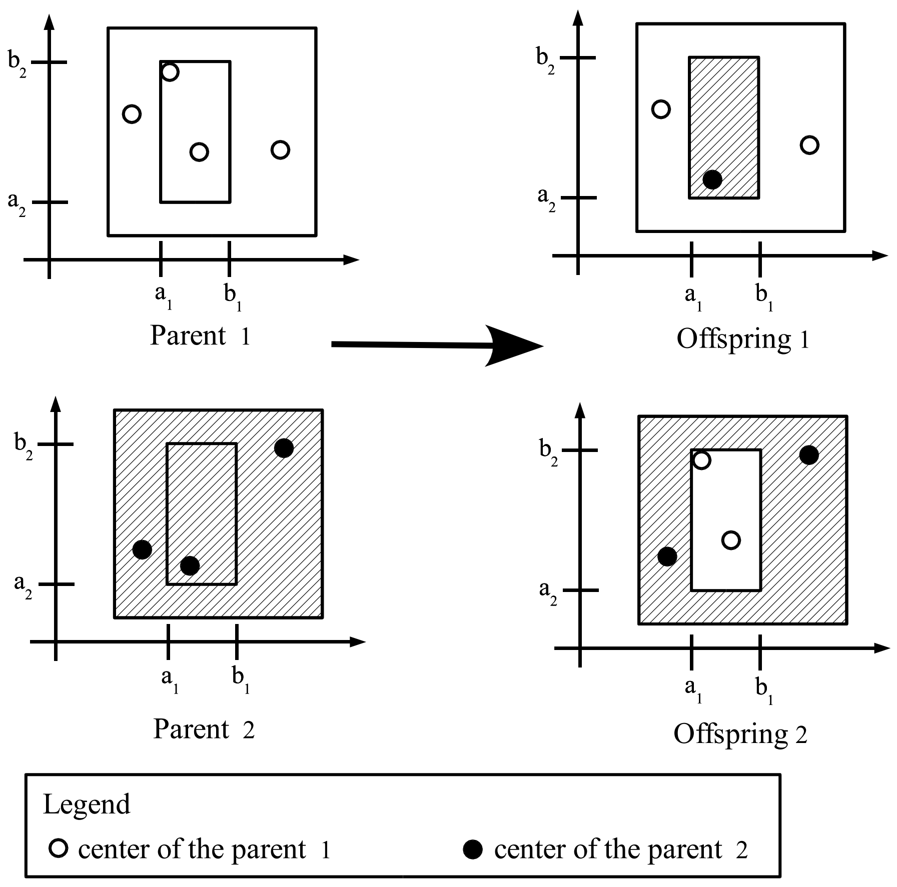


Figure 4.7: Crossing Hypervolumes

ordering for the tuples $\mathbf{p}_i$ as follows.

Consider the $(n+2)$-tuples $\mathbf{p}$ and $\mathbf{q}$. One can say that $\mathbf{p}$ precedes $\mathbf{q}$ (in symbols $\mathbf{p} \prec \mathbf{q}$) if there exists an index $k \in \{1, \ldots, n+2\}$ such that $p_j = q_j$, for $j < k$ and $p_k < q_k$. The chromosome $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m)$ is a canonical parameterization if:

$$\mathbf{p}_1 \prec \mathbf{p}_2 \prec \ldots \prec \mathbf{p}_m \tag{4.8}$$

The GA, proposed in this article, requires that the chromosomes be canonical parameterizations. The genetic operators were adapted to preserve this property. Mutation is applied on elements of a randomly chosen tuple $\mathbf{p}_i$ generating a new tuple $\mathbf{p}'_i$, which must be restricted to the limits:

$$\mathbf{p}_{i-1} \prec \mathbf{p'}_i \prec \mathbf{p}_{i+1} \qquad (4.9)$$

The application of a 1-point-crossover on the parents $\mathbf{P} = (\mathbf{p}_1, \dots, \mathbf{p}_m)$ and $\mathbf{Q} = (\mathbf{q}_1, \dots, \mathbf{q}_m)$, with cut point in the position i, produces the offspring:

$$(\mathbf{p}_1, \dots, \mathbf{p}_i, \mathbf{q}_{i+1}, \dots, \mathbf{q}_m)$$

which is valid only if $\mathbf{p}_i \prec \mathbf{q}_{i+1}$; otherwise another cut point must be chosen.

By eliminating structurally different chromosomes representing networks with the same functionality (i.e., redundancy), this model clearly reduces the search space. It would be interesting to see its performance. However, in the article consulted, the author did not present any experimental results for his model.

4.2.4 Other Models

Maillard and Gueriot[69] modified the model described in Section 4.2.1 by allowing the centers to assume other points besides the training input vectors. In this model, the authors also investigated the use of several types of basis functions in the same network. According to the authors, networks with different basis functions presented a smaller number of hidden nodes and achieved lower error rates than those using only Gaussian functions. The chromossome is a variable length list of five gene sequences. Each 5-gene sequence codes the characteristics of a basis function as shows Figure 4.8. The center of a basis function is defined as the weighted barycenter of two patterns labeled as Id1 and Id2 in Figure 4.8.

Whitehead and Choate [100] proposed a genetic approach that evolves space-filling curves to set the center vectors. The underlying idea involves the mapping of the centers from a n-dimensional region of the input space (defined by such space-filling curves) to a unidimensional space in which the chromosome is encoded. This reduces the number of degrees of freedom of the genetic encoding. In another article, the same authors evolved the centers and widths of the radial basis functions through a cooperative-competitive GA [101]. In this method, each individual encodes only one hidden unit. The whole population represents a unique RBF network. The individuals compete and

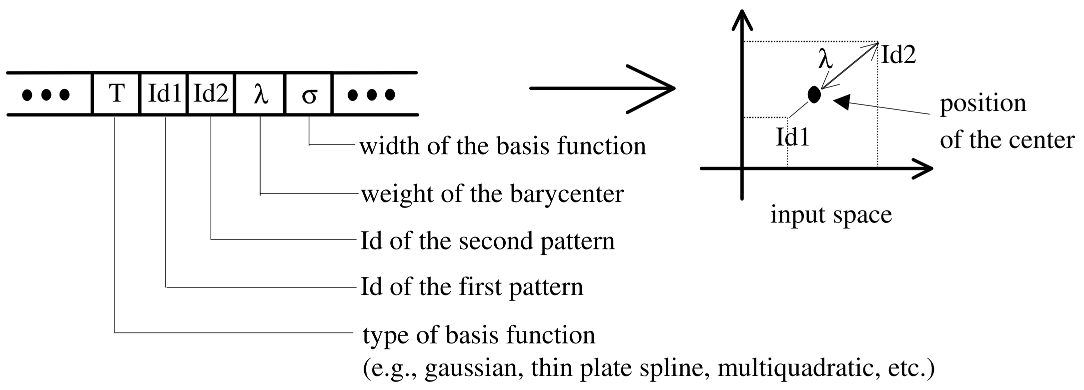


Figure 4.8: Decoding the chromossome formed by a list of 5-gene sequences

cooperate among themselves to improve the overall performance of the network represented by the population.

Chen, Wu and Alkadhimi, in [20], train RBF networks with a combination of GAs and the ROLS algorithm (Recursive Orthogonal Least Squares). Firstly, GA evolves the widths and a regularization parameter of the ROLS algorithm. Next, the ROLS algorithm defines the number and position of the center vectors. Others works involving GA and RBF networks are [8, 92, 16, 39, 98, 54, 70].

4.3 Comments

Three works [12, 77, 17] served as a starting point for ideas in this research:

- In [12] (see section 4.2.1), a multiobjective optimization GA method for RBF networks was presented in which the objective functions were model selection criteria (namely the Akaike information criterion). Inspired on this approach, it is proposed, in Chapter 5, a multiobjective GA using other different model selection criteria.

- In [77] (see section 4.2.3), it was given a light on the redundant problem in RBF networks by presenting a theoretical treatment on this problem.

- In [17], it was shown how deal with redundant problems in RBF networks by crossing hypervolumes on input space instead structural chunks of chromosome.

Inspired on this approach, it is proposed, in the Chapter 6, a crossover that also cross hypervolumes on input space.

Despite of being interesting, other works, shown in section 4.2.4, do not directly influence this research. In fact, most of these works empathize the encodings itself. Whereas this research empathizes the objective functions (mainly the model selection criteria). Moreover, some works do not deal with redundant problem. Whereas this research consired the redundant problem in order to design the proposed RBF encoding in Chapter 5.

Chapter 5

The Proposed Genetic Encodings and their Operators

"If the question is excessively
complex, wait one more day or
one more week in order to resolve it.
Time does not pass in vain."
(Book: Christian Agenda,
André Luiz - medium F.C.Xavier.)

This chapter presents the GA proposed in this research by describing the encoding and the genetic operators. The objective functions and others components are described in the next chapter. Two new genetic encodings are proposed here:

1. Model I (also called the model with multiple centers per cluster);
2. Model II (also called the model with one center per cluster).

They are described as follows.

5.1 Model I - Multiple Centers per Cluster

5.1.1 Encoding

In the model I, the chromosome is a variable length list of tuples given by:

$$\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{m_\mathbf{P}}) \tag{5.1}$$

where $m_\mathbf{P}$ is the number of hidden units coded in the chromosome $\mathbf{P}$. The tuple $\mathbf{p}_j$, $1 \leq j \leq m_\mathbf{P}$, codes a basis function and is given by:

$$\mathbf{p}_j = (r_j; \sigma_j; c_{j1}, c_{j2}, \ldots, c_{jn}) \tag{5.2}$$

The parameters of the basis function coded into $\mathbf{p}_j$ are:

- $\sigma_j \in [0, 1]$ is the coded width.
- $c_{j1}, \ldots, c_{jn} \in [0, 1]$ are the coded coordinates of the center.
- r_j is a integer identifier indicating that the center is inside the region R_{r_j} of the input space (this region is discussed below).

5.1.2 Partitioning the Input Space

The partition of the input space creates a set of K regions $\{R_1, \ldots, R_K\}$, which are placed in the areas where there is high density of training input patterns, as showed in Figure 5.1. The width of the region R_i along each coordinate direction is determined by the corresponding components of the vectors $\mathbf{l}_i = [l_{i1}, \ldots, l_{in}]^\mathsf{T}$ and $\mathbf{u}_i = [u_{i1}, \ldots, u_{in}]^\mathsf{T}$. That is,

$$R_i = \{\mathbf{x} \in \Re^n : l_{i1} \leq x_1 \leq u_{i1}, \ldots, l_{in} \leq x_n \leq u_{in}\} \tag{5.3}$$

In other words, the components l_{ik} and u_{ik} are the lower and upper limits, respectively, of the region R_i along a coordinate direction.

In this work, the vectors $\mathbf{l}_i$ and $\mathbf{u}_i$ are obtained from the clusters of training input patterns generated by the K-means clustering algorithm [4]. Consider S_i a cluster of

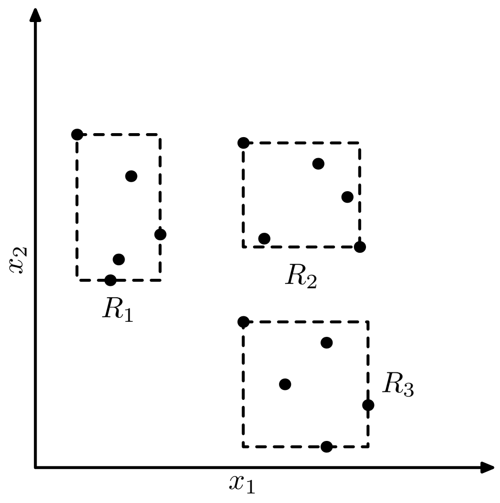


Figure 5.1: Partioning the input space by means of clusters of patterns

input training patterns. The vectors $\mathbf{l}_i$ and $\mathbf{u}_i$ are obtained from cluster S_i as follows:

$$l_{ik} = \min_{\mathbf{x} \in S_i} x_k \tag{5.4}$$

$$u_{ik} = \max_{\mathbf{x} \in S_i} x_k \tag{5.5}$$

for all $k = 1, \dots, n$. By using the above procedure, the region R_i encompass all patterns of the cluster S_i.

5.1.3 Decoding

Consider the following chromosome to be decoded: $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_m)$ where $\mathbf{p}_j = (r_j; \sigma_j; c_{j1}, c_{j2}, \dots, c_{jn})$. The parameter $r_j \in [1, K]$ is coded as a integer. The parameters $\sigma_j, c_{j1}, c_{j2}, \dots, c_{jn}$ are coded as floating-point values and normalized as: $\sigma_j, c_{j1}, c_{j2}, \dots, c_{jn} \in [0, 1]$. The decoding is carried out as follows

$$c'_{jk} = l_{r_j,k} + c_{jk}(u_{r_j,k} - l_{r_j,k}), \text{ for } k = 1, \dots, n. \tag{5.6}$$

$$\sigma'_j = s\sigma_j \tag{5.7}$$

where the coefficient s is a scaling factor (in this work s is equal to the half the maximum distance separating pairs of training input patterns). Thus, the center of basis function

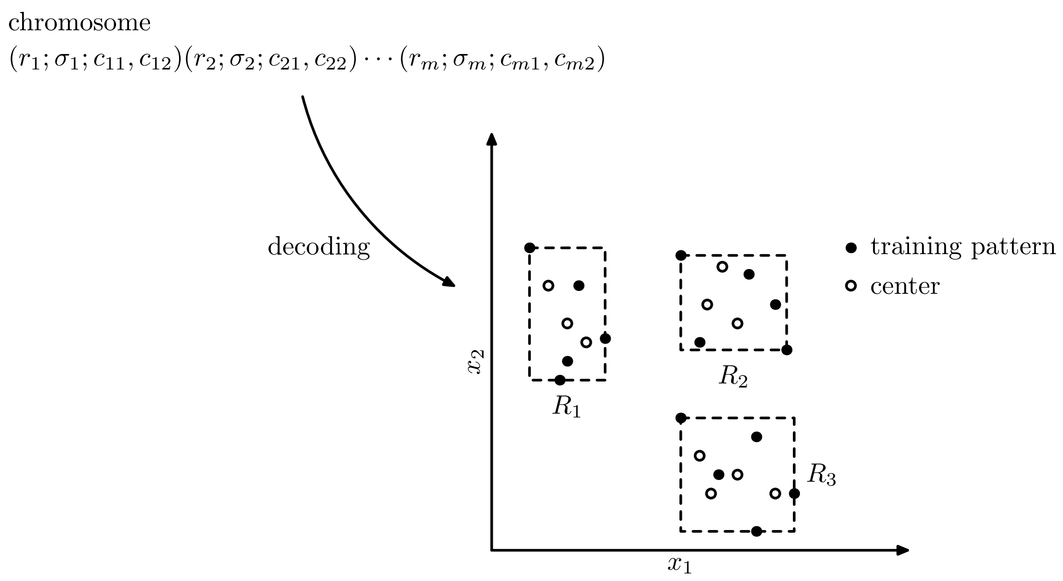

chromosome
$(r_1;\sigma_1;c_{11},c_{12})(r_2;\sigma_2;c_{21},c_{22})\cdots(r_m;\sigma_m;c_{m1},c_{m2})$
decoding
● training pattern
○ center
R_1
R_2
R_3
x_1
x_2

Figure 5.2: Chromosome decoding

coded into $\mathbf{p}_j$ is given by

$$\mathbf{c}_j = \begin{bmatrix} c'_{j1} \\ c'_{j2} \\ \vdots \\ c'_{jn} \end{bmatrix} \tag{5.8}$$

A decoding example is shown in Figure 5.2.

5.1.4 The Cluster Crossover

The proposed GA uses a modified crossover operator, here named cluster. It is described as follows. First, consider the parents:

$$\begin{aligned} \mathbf{P} &= (\mathbf{p}_1, \ldots, \mathbf{p}_{m_\mathbf{P}}) \\ \mathbf{Q} &= \left(\mathbf{q}_1, \ldots, \mathbf{q}_{m_\mathbf{Q}}\right) \end{aligned}$$

Second, create a template bit string randomly $(b_1, b_2, \ldots, b_K)$, $b_i \in \{0,1\}$, for $i = 1, \ldots, K$. Finally, perform the cluster crossover as follows:

```
for $i = 1$ to $K$ do
    if $b_i = 1$ then
        Move all genes $\mathbf{p}_j$ that are member of region $R_i$ to the 1st offspring
    else
        Move all genes $\mathbf{q}_j$ that are member of region $R_i$ to the 1st offspring
    endif
    Move the rest of the genes to the 2nd offspring.
endfor
```

In Figure 5.3 is shown a example of the cluster crossover applied to the regions illustrated in Figure 5.1.

Cluster Crossover vs. Traditional Crossovers

The use of the traditional crossover operators might produce duplicated genes into the same chromosome, as shown in the following example.

Consider a case in which the centers are on the one-dimensional space. In this simple case, the chromosomes are formed by a variable list of 3-tuples (region, width, center). Figure 5.4 shows the traditional crossover producing an offspring with duplicated genes. By crossing hypervolumes, the cluster crossover avoids the illegality problem in this example as illustrated in Figure 5.5.

The cluster crossover works like the traditional uniform crossover but crosses regions R_i instead of structural chunks of chromosomes (as is usually performed by standard uniform crossover). To put in another way, the cluster crossover is a method for performing the crossing of hypervolumes on phenotype space, whereas the standard uniform crossover operators are performed on the genotype space. It is similar in spirit to the Carse and Fogarty's crossover (see Section 4.2.2).

5.1.5 Mutation Operators

The following mutation operators are used:

- The uniform mutation replaces widths, centers and region identifiers by uniform random numbers (see Section 2.3.2 for details).
- The creep mutation adds a Gaussian noise to the value of widths and centers. The added noise is small, so creep mutation plays the role of a local search (see Section 2.3.2 for details).

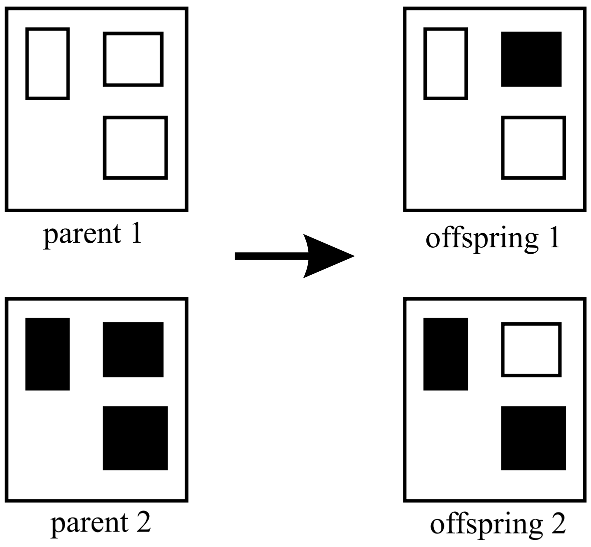


Figure 5.3: The cluster crossover. The offspring 1 randomly takes a region (cluster) from one of the parents. The offspring 2 takes the rest of the regions.

- The addition and delete operators add and delete randomly chosen hidden units.

5.2 Model II - One Center per Cluster

In this section, it is presented the model II. The main difference between the model I and the model II is that the model II contains only one center per cluster rather than multiple centers per cluster.

5.2.1 Encoding

In the model II, the chromosome is a fixed length list of tuples given by:

$$\mathbf{P} = (\alpha, \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_K) \tag{5.9}$$

where α is the overlap factor (from the Equation 3.15), K is the number of regions. The tuple $\mathbf{p}_j$ codes a basis function, and is expressed by:

$$\mathbf{p}_j = (b_j, c_{j1}, c_{j2}, \ldots, c_{jn}) \tag{5.10}$$

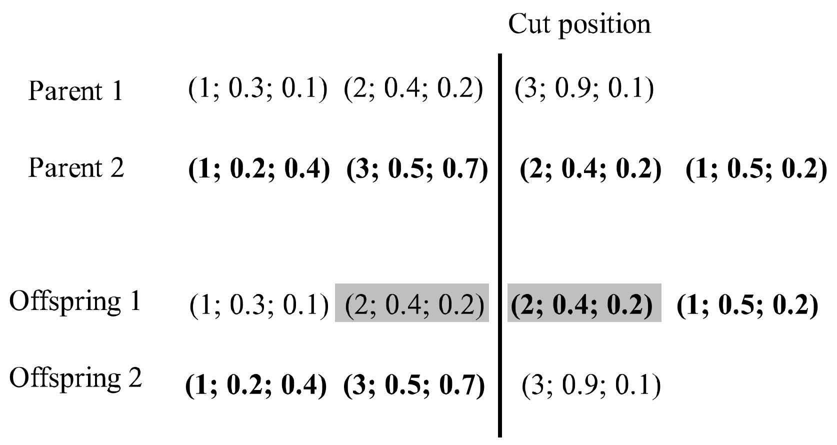


Figure 5.4: Traditional crossover generates duplicated genes

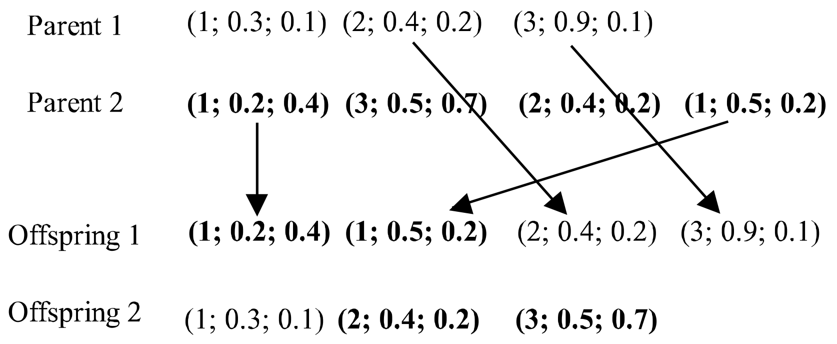


Figure 5.5: Cluster crossover using the template bit string (0,1,1)

The parameters of the basis function coded into $\mathbf{p}_j$ are:

- $c_{j1}, \ldots, c_{jn} \in [0, 1]$ are the coded coordinates of the center $\mathbf{c}_j$. The center $\mathbf{c}_j$ is inside the region R_j (i.e., $\mathbf{c}_j \in R_j$).
- b_j is a boolean flag: if b = TRUE then $\mathbf{p}_j$ is valid, otherwise $\mathbf{p}_j$ is discarded during the decoding.

In spite of having fixed length, the chromosome may produce network architectures with variable sizes because some basis functions will be discarded (depending on whether b_j is TRUE or not). The procedure used to generate regions is the same one discussed in the section 5.1.2.

5.2.2 Decoding

Consider the following chromosome to be decoded: $\mathbf{P} = (\alpha, \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_K)$. The parameters $\alpha, c_{j1}, c_{j2}, \ldots, c_{jn}$ were coded as floating-point values and normalized in the interval $[0, 1]$. The decoding is caried out as follows:

$$c'_{jk} = l_{r_j,k} + c_{jk}(u_{r_j,k} - l_{r_j,k}) \tag{5.11}$$

$$\alpha' = s\alpha \tag{5.12}$$

for $k = 1, \ldots, n$. The coefficient $s = 5$ is a scaling factor. Thus, the center of basis function coded into $\mathbf{p}_j$ is given by

$$\mathbf{c}_j = \begin{bmatrix} c'_{j1} \\ c'_{j2} \\ \vdots \\ c'_{jn} \end{bmatrix} \tag{5.13}$$

The overlap factor α' is used in the Equation 3.15 to determine the width´s σ_i (i.e, $\sigma_i = \alpha' \|\mathbf{c}_i - \mathbf{c}_j\|$).

5.2.3 Genetic operators

Two crossovers are proposed for the model II:

1. Cluster crossover;

2. Cluster crossover combined with the blend crossover.

The cluster crossover, described in section 5.1.4, is adapted for the model II as follows. Let $\mathbf{P}_1 = (\mathbf{p}_{11}, \ldots, \mathbf{p}_{1K})$ and $\mathbf{P}_2 = (\mathbf{p}_{21}, \ldots, \mathbf{p}_{2K})$ be the parents, where $\mathbf{p}_{kj} = (b_{kj}, c_{jk1}, c_{jk2}, \ldots, c_{kn})$. Let $\mathbf{Q} = (\mathbf{q}_1, \ldots, \mathbf{q}_K)$ be their child. The cluster crossover for model II is then given by:

```
for $j = 1$ to $K$ then
    $\mathbf{q}_j = \mathbf{p}_{1j}$ or $\mathbf{p}_{2j}$ with equal probability.
endfor
```

Note that the cluster crossover itself does not change the value of the centers. To do this, the cluster crossover is combined with the blend crossover (a well-known crossover for real-coded GAs described in section 2.3.2) as follows:

```
for $j = 1$ to $K$ then
    if ($b_{1j} = 1$) and ($b_{2j} = 1$) then
        $\mathbf{q}_j$ = BlendCrossover($\mathbf{p}_{mj}, \mathbf{p}_{nj}$)
    else
        $\mathbf{q}_j = \mathbf{p}_{1j}$ or $\mathbf{p}_{2j}$ with equal probability.
    endif
endfor
```

in which the subroutine BlendCrossover() refers to the blend crossover.

The combination of cluster and blend crossovers for model II is straightforward because each region has only a center. For model I, however, the combination of cluster and blend crossovers is not obvious because there are multiple centers of the parents inside the same region. It follows that the model I rely only on the mutation greep operator to change the float-point value of the centers.

As before, the traditional mutation operator and some other variations are used. The creep mutation adds a noise with normal distribution to widths and centers. Addition and Deletion operators add and delete a hidden unit randomly chosen.

5.3 Comments

A lot of traditional RBF network learning methods used to find the position of the gaussians (e.g., the k-means [75] and others [87, 76]) are based on assumption that gaussians should be on regions of clustering of training set points because a Gaussian outside of such regions is prone to be spurious once its response tends to be zero. The proposed encodings also impose the gaussians to remain in such regions by restricting the gaussians to hyper rectangular regions on clustering regions. There are some promising ideals in the proposed encoding:

- By restricting the gaussians to clustering regions, the GA keeps only (approximately) good networks in the population once avoid handling spurious gaussians. It follows that the search space is also reduced.

- Unlike traditional learning methods, the GA allows that the gaussians are evolved in any place on clustering regions [1].

- The crossover is only performed between clustering regions. So the encoding is nonredundant and legal (except in the atypical case when the corners of the hyper-rectangles overlap) because the crossover between two legal chromosomes does not produce an illegal offspring.

- It is possible to add more parameters to the chromosome (e.g., widths, basis functions) if this is needed. Moreover, in the proposed encoding both network complexity and performance can be evolved simultaneously.

Some comments follow by comparing the proposed encoding and the encodings given by Billings and Zheng [12] (see section 4.2.1) and Carse and Fogarty [17] (see section 4.2.2). Despite the search space of the Billings and Zheng's encoding to be smaller than the proposed encoding, it is too restrictive because it limits the centers to the input pattern positions whereas, in the proposed encoding, the centers may lie on any place within a cluster. In Carse and Fogarty's encoding, the crossover is performed between random regions of the input space. But, it seems be a better idea to perform the crossover between clustering regions because clusters are regions with higher probability of finding the best centers. Hence the search space is reduced.

It is worth noting that the encoding is only a component of the GA. Therefore, the encoding itself is not the unique responsible for evolving good networks because the choice of objective function also plays an important role in the GA search as shown in next chapter. Furthermore, some strategies and modifications in the traditional GA can influence strongly the search. Such issues are also addressed in next chapter.

[1] In the traditional k-means, per example, the gaussians are always fixed on mean of the cluster.

Chapter 6

Model Selection via Genetic Algorithms

"Keep your balance. Unbridled
passions and desires
are forces of destruction
within the Divine Creation."
(Book: Christian Agenda,
André Luiz - medium F.C.Xavier.)

This chapter addresses the problem of finding the adjustable parameters of a learning algorithm using Genetic Algorithms, GAs. This problem is also known as the model selection problem. Some model selection techniques (e.g., crossvalidation and bootstrap) are used as objective functions of the GA. The GA is modified in order to adapt to that problem by means of occam's razor, growing, and other heuristics. Some modifications explore features of the GA, such as the ability for handling multiple and noise objective functions. This chapter addresses GA search mechanism, instead of other more studied aspects like encoding and genetic operators (mainly for neural networks [105]). It is organized as follows.

Section 6.1 gives some useful definitions. Section 6.2 describes formally the model selection problem. Section 6.3 presents a simple GA method with occam elitism for model selection and some experimental results. Crossvalidation, bootstrap, and other objective functions are presented in Section 6.4, together with some experimental results. Section 6.6 evaluates a multiobjective GA employing two heuristics (growing and shuffling) that aim to improve the quality of the genetic hypothesis. Finally, in section 6.7, the GA is compared to two constructive algorithms used to determinate the RBF network architecture.

6.1 Training and Adjustable Parameters

Most Machine Learning methods focus the problem of approximating a target function $f : \mathcal{X} \rightarrow \mathcal{Y}$ by using the information from a dataset (i.e., a sample of examples $(\mathbf{x}_i, y_i)$, for $i = 1, \ldots, p$, where $\mathbf{x} \in \mathcal{X}$ and $y_i = f(\mathbf{x}_i)$). In principle, a learning algorithm $\mathcal{L}$ builds a hypothesis $h \in \mathcal{H}$ that approximates the target function f by determining an set of parameters θ. Most of the time, however, there are some parameters in the set θ in which the learning algorithm itself is unable to determine: the adjustable parameters. Thus, for a particular learning algorithm $\mathcal{L}$, there are two kinds of parameters in the set of parameters θ:

Definition 6.1 *A set of parameters is said be* ***training parameters*** *(denoted by τ) if it is determined automatically by $\mathcal{L}$.*

Definition 6.2 *A set of parameters is said be* ***adjustable parameters*** *(denoted by λ) if it is not determined by $\mathcal{L}$.*

Note that $\theta = \tau \cup \lambda$ and $\tau \cap \lambda = \emptyset$. The adjustable parameters are typically determined by human subjective judgment based on previous experience, rule of thumbs or heuristics provided by authors and practitioners of the learning algorithm.

In other words, the adjustable parameters are determined by minimizing an estimate of the *true prediction error* [26] (true error for short) over a set of adjustable parameters that is known to work well on the dataset. A common example of this procedure is to train a neural network using the backpropagation learning algorithm [88]. The backpropagation sets the weight parameters of the network, but is unable to obtain the number of hidden units (which are the adjustable parameters of the problem). The search carried out by the user in order to find the best adjustable parameter is essentially the trial-and-test optimization method, which becomes inefficiency if the search space is too large. Hence, it is necessary the human subjective judgment and heuristics to make the trial-and-test method more efficient. Others common examples of adjustable parameters are the amount of pruning of a decision tree, the degree of a polynomial fit to a set of points, the ridge parameter of the ridge regression, and the best subset of variables of a multivariate linear regression model.

6.2 Model Selection

The problem of estimating the true error of hypothesis using different adjustable parameters in order to choose the (approximate) best one is known as **model selection**[41]. The purpose of this work is to present Genetic Algorithms as an alternative to the trial-and-test optimization method for model selection.

More formally, the optimization problem to be solved by GA is described as follows. A hypothesis $h : \mathcal{X} \to \mathcal{Y}$ (where $\mathcal{X} \times \mathcal{Y} = \mathcal{E}$ is the example space) is built by the learning algorithm $\mathcal{L}$ from a choice of adjustable parameter λ and a training data set $\mathcal{D}$. It is assumed that the learning algorithm is deterministic. That is, for a particular choice of λ and $\mathcal{D}$, the learning algorithm always builds the same hypothesis. Thus, the hypothesis h can be represented as $h(\lambda, \mathcal{D})$. The notation $h(\mathbf{x}; \lambda, \mathcal{D})$ represents the prediction of $h(\lambda, \mathcal{D})$ for the data point $\mathbf{x}$ (i.e., $h(\mathbf{x}; \lambda, \mathcal{D}) = h(\lambda, \mathcal{D})(\mathbf{x})$).

The true error measures how well a hypothesis $h(\lambda, \mathcal{D})$ predicts the response value of a future example (see also Sections 3.2 and 3.3). The true error is defined as

$$e(\lambda) = \mathrm{E}\langle \delta(y, h(\mathbf{x}; \lambda, \mathcal{D})) \rangle \tag{6.1}$$

where $\delta(x, y)$ is a loss function. The expectation in (6.1) refers to repeated sampling of examples randomly drawn from the example space $\mathcal{E}$ with the same probability that was used to select the examples from training dataset $\mathcal{D}$. Regression problems usually employ the quadratic loss function $\delta(x, y) = (x - y)^2$ (see an example in Sections 3.2 and 3.3). Classification problems usually make use of the $0/1$ loss function: $\delta(x, y) = 1$ if $x = y$, otherwise $\delta(x, y) = 0$. By using the $0/1$ loss function, the function $e(\lambda)$ represents the probability of correctly classified examples.

The problem of choosing the adjustable parameter (i.e., the model selection problem) can be defined as:

$$\text{Find } \lambda \text{ that minimizes } e(\lambda) \tag{6.2}$$

The true error $e(\lambda)$ is unknown, once the only available information on the target function is contained into the dataset $\mathcal{D}$. Because of this, a number of methods for estimating the true error have been proposed [26]. The most commonly used methods for true error estimation are:

- Holdout;
- Crossvalidation;
- Bootstrap.

Consider that $\widehat{e}(\lambda, \mathcal{D})$ is an estimation of the $e(\lambda)$ using the information from the dataset $\mathcal{D}$ by means of those methods. The problem of optimizating the adjustable parameter could be reformulated as

$$\text{Find } \lambda \text{ that minimizes } \widehat{e}(\lambda, \mathcal{D}) \tag{6.3}$$

The best estimations of the true error are usually obtained by the crossvalidation and bootstrap methods, which transform this optimization problem into a difficult problem. Crossvalidation and bootstrap are computer procedures, but may be interpreted as simple functions from the optimization view point. As objective functions, they have the following drawbacks:

1. These functions are not easy to be handled algebraically and hence derivatives are not easy to calculate;
2. These functions are also multimodals (i.e., there exist various points λ that minimizes them locally);
3. For a given point λ, to obtain the response of the functions crossvalidation and bootstrap, a large amount of CPU time may be needed, making their minimization hard for any optimization technique (including GAs).

Because of the complex nature of this optimization problem, GAs are a natural candidate for solving it. Unlike other optimization metaheuristics, GAs provide a suitable framework for the model selection problem, due to its peculiar search mechanism able to handle many hypotheses simultaneously. This search mechanism enables GAs to use (and to create) new heuristics (as shown later), to cope with noise estimation of the true error and to be used for multiobjective optimization.

6.3 A Simple GA for Model Selection

The application of GAs for model selection involves the optimization of the adjustable parameters. It is worth noting that the adjustable parameters of a hypothesis do not only depend on the features of the hypothesis, but also on the learning algorithm. Different learning algorithms can produce different adjustable parameters for the same hypothesis.

To avoid proliferating notation, the symbol λ will represent both the chromosome and the adjustable parameters it encodes. Throughout the text, the dataset

$$\mathcal{D} = \{(\mathbf{x}_i, y_i); \quad i = 1, \ldots, p\} \tag{6.4}$$

represents the set of all the examples avaliable for the problem at hand.

A typical GA for model selection has the following characteristics:

1. The chromosome encodes only the adjustable parameters.
2. A learning algorithm $\mathcal{L}$ embedded in the procedure that evaluate the chromosome. The learning algorithm $\mathcal{L}$ is used to determine the training parameters.
3. A method for estimating the true error.

A simple GA for model selection is illustrated as follows:

```
main program

"Let $P(t) = \{\lambda_1, \ldots, \lambda_N\}$
be the population at generation $t$."

$t \leftarrow 0$
initialize($P(t)$)
evaluate($P(t)$)
while the stopping criterion is not satisfied do
  $t \leftarrow t + 1$
  $P(t) \leftarrow$ select($P(t-1)$)
  $P(t) \leftarrow$ crossover($P(t)$)
  $P(t) \leftarrow$ mutation($P(t)$)
  evaluate ($P(t)$)
end while
```

```
procedure evaluate($P$)

for each $\lambda_i$ in the population $P$ do
  Estimate the true error $\widehat{e}(\lambda_i, \mathcal{D})$
  Set fitness $f_i$ of $\lambda_i$ equal to $\widehat{e}(\lambda_i, \mathcal{D})$
end for
```

The above procedure evaluate() computes the fitness function, that is, the estimation of the true error $\widehat{e}$ for each chromossome by using some traditional methods for model selection, such as holdout, crossvalidation and bootstrap. In the next section, the holdout function is presented.

6.3.1 The Holdout Fitness Function

One of the most used methods for model selection, the holdout method [51], as can be seen in Figure 6.1, divides the dataset $\mathcal{D}$ into two parts: the training set $\mathcal{D}_t$, on which the hypothesis is built, and the holdout set $\mathcal{D}_h$ (also known as validation set), on which its performance is measured. Thus, $\mathcal{D}_h = \mathcal{D} \setminus \mathcal{D}_t$. Let $h(\lambda, D_t)$ be the hypothesis built from the dataset $\mathcal{D}_t$ using the adjustable parameter λ. The fitness of the chromosome λ is given by:

$$f(\lambda) = \frac{1}{|\mathcal{D}_h|} \sum_{(\mathbf{x},y)\in\mathcal{D}_h} \delta(y, h(\mathbf{x}; \lambda, D_t)) \tag{6.5}$$

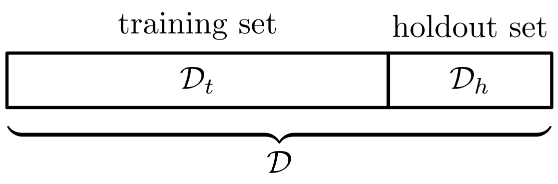


Figure 6.1: The data set partition by the holdout method.

In order to improve the simple GA illustrated in section 6.3, the use of the occam elitism is proposed.

6.3.2 The Occam Elitism

Occam's razor is a principle attributed to the 14th century philosopher William of Occam. The principle states that "*entities should not be multiplied unnecessarily*". With this "razor", Occam cut out all superfluous, redundant explanations. Scientists have reintepreted the Occam's razor. A useful statement of the principle for scientists is, "*when you have two competing theories which make exactly the same predictions, the one that is simpler is the better*." Machine learning scientists [74] have stated that principle as "*prefer the simplest hypothesis that fits the data*". For this work, the following statement is used:

Occam's razor: given two hypothesis with the same estimation of true error, the one that is simpler is the better because it is likely to have lower true error.

The traditional elitism is a well known strategy in GAs which works by never replacing the n best chromosomes in a population with inferior solutions. That is,

Elitism: the n best chromosome are kept from generation to generation.

The occam elitism is based on the assumption that the n best chromosomes in the population are equivalent in terms of estimation of true error (at least under some statistical confident limit). Because (according the Occam's razor) the k simplest hypotheses among the n best chromosomes are better than the $n - k$ remaining hypotheses, those k simplest hypotheses shouldn't be replaced with inferior solutions. In other words,

Occam elitism: the k simplest hypothesis among the n best chromosomes are kept from generation to generation.

Next section shows experiments comparing the traditional elitism with the occam elitism.

6.3.3 Experimental Studies

In the experiments performed, the proposed GA (model II) was applied to a benchmark dataset: a Hermite polynomial approximation [68] (see Figure 6.2), which is defined by:

$$f(x) = 1.1(1 - x - 2x^2)\exp\left(-\frac{x^2}{2}\right) \tag{6.6}$$

The datasets used in the experiment were generated under the same conditions adopted in [78]. Each dataset has 40 examples randomly chosen in the range [-4, +4] with added gaussian noise. The following noise variances were used: 0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, and 0.1. For each noise variance, the GA program was executed, at least, 50 times (where each execution used a different random dataset) and the results were averaged[1]. The GA program used the parameters in table 6.1.

The holdout method generated the holdout dataset with 50% of the original dataset. The performance of the hypotheses was measured using the Root-Mean-Square Error (RMSE = $p^{-1}\sqrt{\mathrm{SSE}}$) over a test set with 200 uniformly spaced noiseless examples in the range [-4, +4]. The occam elitism was used with $n = 5$ and $k = 1$. Thus, the smallest network among the five best chromosomes was kept from generation to generation.

According to the results of the Figure 6.3.3, which shows the number of centers as a function of noise level, the occam elitism produced notably smaller networks than the traditional elitism. Moreover, GA with occam elitism also produced a smaller true error, as shows the Figure 6.3.3. These results show that the occam elitism seems to be an effective method. Because of this, all the following experiments in this work will use the occam elitism (with $n = 5$ and $k = 1$).

[1]GA may output outliers (spurious networks with large errors) due to premature convergence. The average value is strongly influenced (become biased towards) by the outliers. Because of this it was used the 5% trimmed mean instead of the mean. The 5% trimmed mean means that both the top 5% and the bottom 5% of a ranked sample are discarded and the mean is calculated for the rest of the sample.

Table 6.1: GA parameters.

Population	500
Generations	500
Number of regions	15
Crossover rate	0.60
Mutation rate	0.05
Addition rate	0.3
Deletion rate	0.3
Basis function	Gaussian

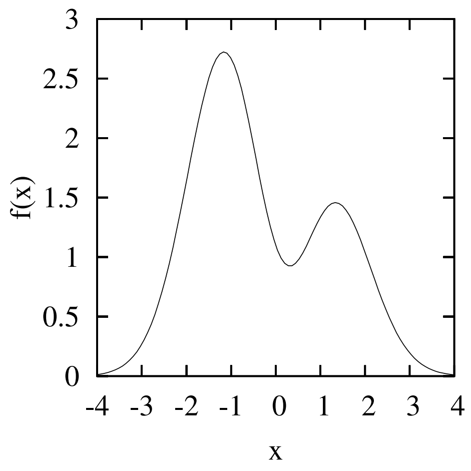


Figure 6.2: Mackay's Hermite polynomial.

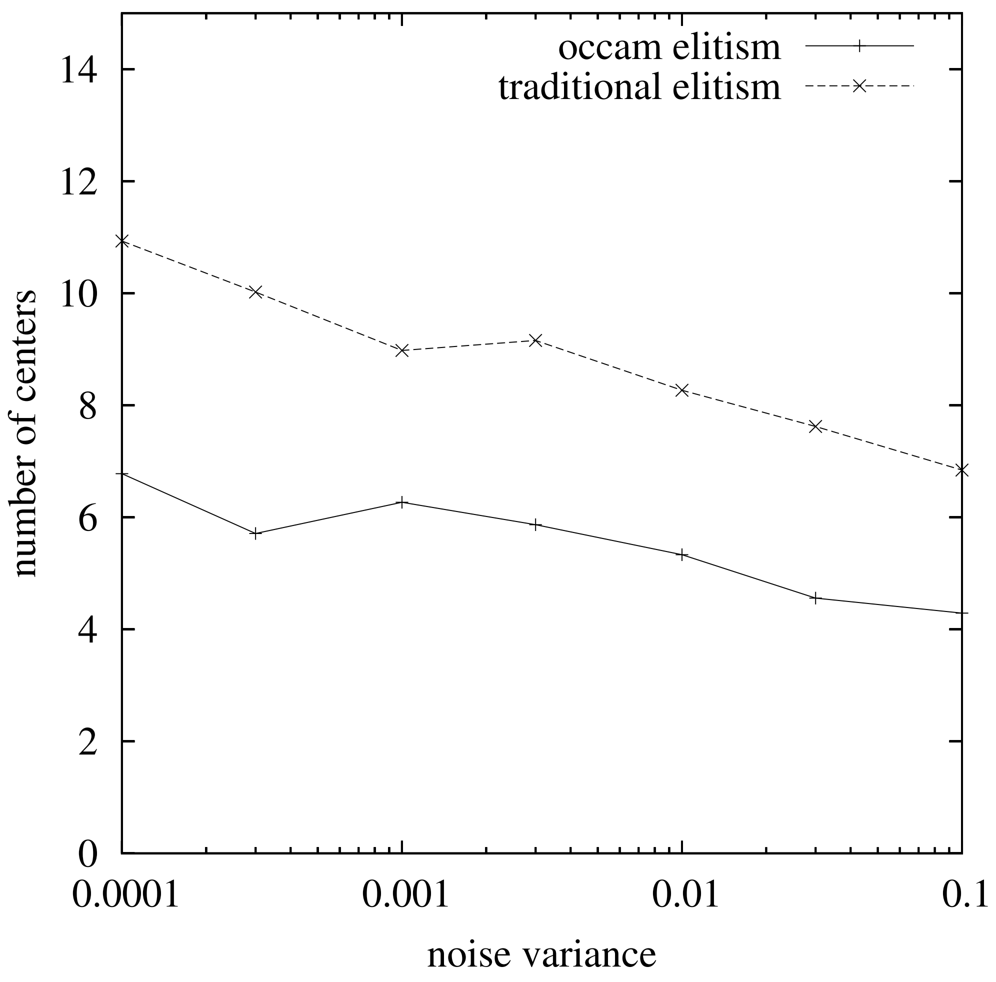


Figure 6.3: Comparing the performance of occam and traditional elitism in terms of the number of basis functions

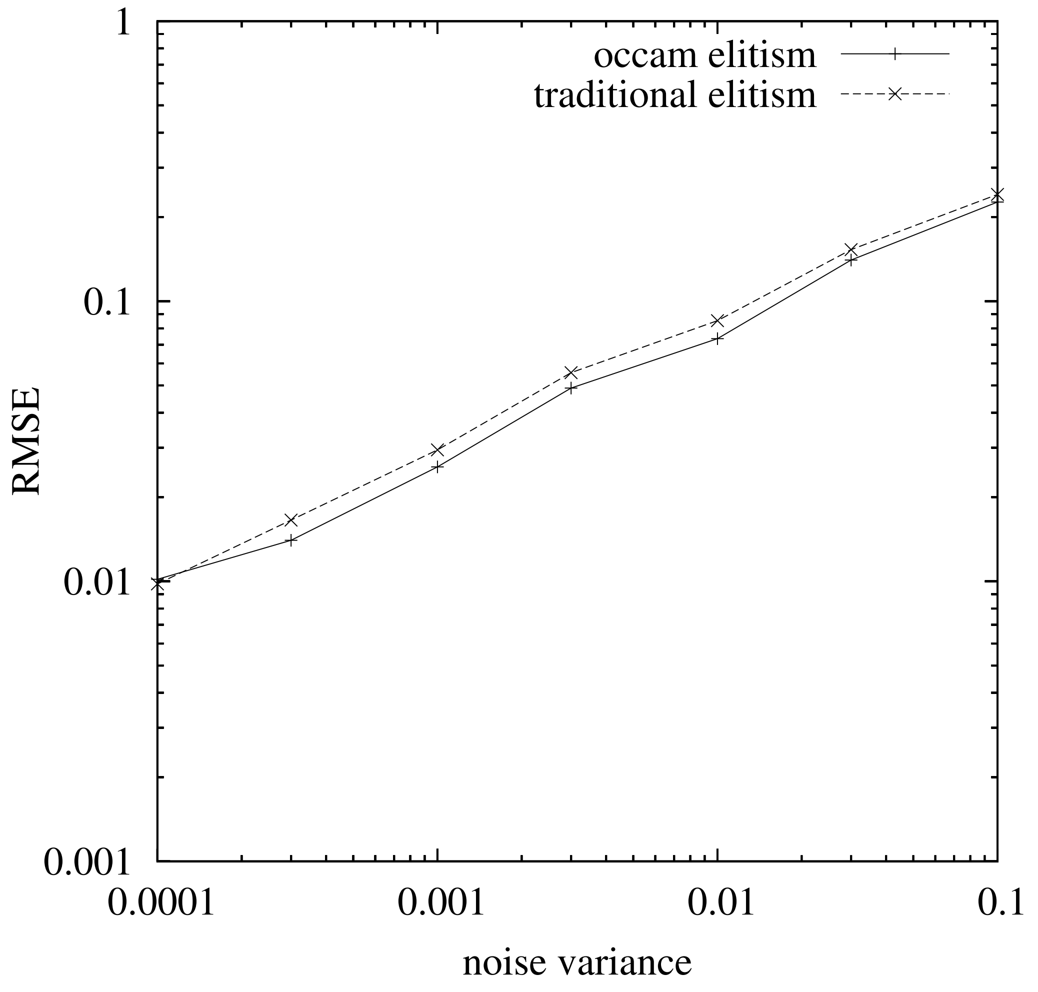


Figure 6.4: Comparing the performance of occam and traditional elitism in terms of the error on the test set.

6.4 Other Fitness Functions for Model Selection

This section presents some methods widely used in machine learning for model selection purposes (i.e., to select a hypothesis (a model) among several candidate hypotheses).

6.4.1 The k-Fold-Crossvalidation Fitness Function

This method [95, 51] divides the dataset $\mathcal{D}$ (Equation 6.4) in k subsets (also named *folds*): $\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_k$. The folds have equal size and are mutually exclusive. It produces k hypotheses $h_1, \dots, h_k$, where each one is built from the dataset $\mathcal{D} \setminus \mathcal{D}_j$ (see Figure 6.5) using the adjustable parameter λ as follows: $h_j = h(\lambda, \mathcal{D} \setminus \mathcal{D}_j)$, for $j = 1, \dots, k$. The performance obtained by each hypothesis h_j is measured on the dataset D_j. The fitness of λ is equal to average of the performances of the k hypotheses. That is, the fitness of the chromosome λ is given by:

$$f(\lambda) = \frac{1}{|\mathcal{D}|} \sum_{j=1}^{k} \sum_{(\mathbf{x},y)\in\mathcal{D}_j} \delta(y, h_j(\mathbf{x})) \tag{6.7}$$

If $k = |\mathcal{D}|$, the k-fold-crossvalidation is known as the *leave-one-out* crossvalidation.

6.4.2 The Generalized Cross-Validation Fitness Function

The Generalized Cross-Validation (GCV) [36] is a formula derived from the leave-one-out crossvalidation under the assumption that the model is linear. Thus, GCV is often used in linear models. RBF networks are nonlinear models, but, in practice, GCV has been used in model selection for RBF networks [78, 43]. Because GCV is computationally inexpensive, it becomes attractive to be used with GAs. The chromosome is trained with the whole dataset $\mathcal{D}$. The fitness of the chromosome λ is given by:

$$f(\lambda) = \mathrm{GCV} = \frac{p\,\mathrm{SSE}}{(p-m)^2} \tag{6.8}$$

where SSE denotes the sum of squared errors on the dataset $\mathcal{D}$, m is the number of weights (free parameters) and p is the number of examples in $\mathcal{D}$.

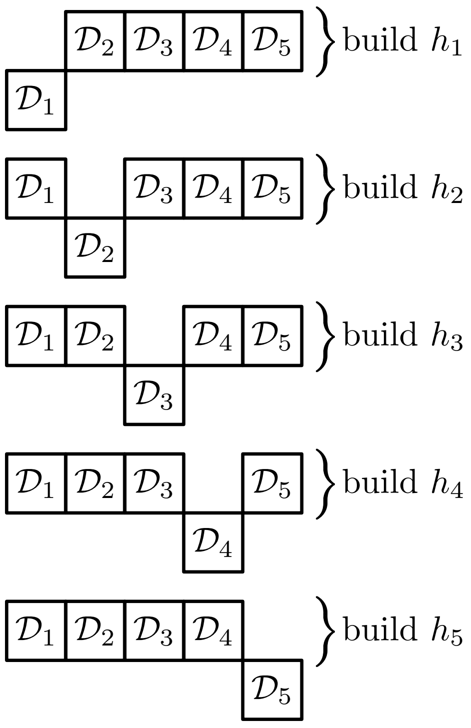

$\mathcal{D}_2$ $\mathcal{D}_3$ $\mathcal{D}_4$ $\mathcal{D}_5$ build h_1
$\mathcal{D}_1$
$\mathcal{D}_1$ $\mathcal{D}_3$ $\mathcal{D}_4$ $\mathcal{D}_5$ build h_2
$\mathcal{D}_2$
$\mathcal{D}_1$ $\mathcal{D}_2$ $\mathcal{D}_4$ $\mathcal{D}_5$ build h_3
$\mathcal{D}_3$
$\mathcal{D}_1$ $\mathcal{D}_2$ $\mathcal{D}_3$ $\mathcal{D}_5$ build h_4
$\mathcal{D}_4$
$\mathcal{D}_1$ $\mathcal{D}_2$ $\mathcal{D}_3$ $\mathcal{D}_4$ build h_5
$\mathcal{D}_5$

Figure 6.5: The k-fold-crossvalidation method with $k = 5$.

If the RBF network learning uses ridge regression (see section 3.7) to compute the weights, the GCV is modified in order to include the ridge parameter β [36]:

$$f(\lambda) = \text{GCV} = \frac{(1/p)\|(\mathbf{I} - \mathbf{A})\mathbf{y}\|^2}{[(1/p)\text{trace}(\mathbf{I} - \mathbf{A})]^2} \tag{6.9}$$

where $\mathbf{A} = \mathbf{Z}(\mathbf{Z}^\mathsf{T}\mathbf{Z} - \beta\mathbf{I})^{-1}\mathbf{Z}^\mathsf{T}\mathbf{y}$.

6.4.3 The .632 Bootstrap Fitness Function

The .632 bootstrap method is a member of the bootstrap family introduced by Efron [26]. All bootstrap based estimates are computed by using a set of bootstrap datasets. A bootstrap dataset is created by sampling $p = |\mathcal{D}|$ examples (with replacement) from $\mathcal{D}$. This method creates B bootstrap datasets: $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_B$. Note that the datasets are not mutually exclusive. It produces B hypotheses $h_1(\lambda), \ldots, h_B(\lambda)$, where each one is built from the dataset $\mathcal{D}_j$ using the adjustable parameter λ. That is, $h_j =$

$h(\lambda, \mathcal{D}_j)$, for $j = 1, \ldots, B$. The fitness of the chromosome λ, obtained by the .632 bootstrap function, is given by

$$f(\lambda) = 0.368 \cdot \overline{\mathrm{err}} + 0.632 \cdot \epsilon_0 \tag{6.10}$$

The terms of the Equation 6.10 are described as follows. The term $\overline{\mathrm{err}}$ is the performance of a hypothesis built from the dataset $\mathcal{D}$ measured on the same dataset $\mathcal{D}$. This measure refers to the training error of the hypothesis. Mathematically, it is given by:

$$\overline{\mathrm{err}} = \frac{1}{p} \sum_{i=i}^{p} \delta(y_i, h(\mathbf{x}_i; \lambda, \mathcal{D})) \tag{6.11}$$

The term ϵ_0 denotes the average error obtained from bootstrap datasets not containing the data point being predicted. In other words, ϵ_0 is computed by using a bootstrap dataset as training set and the remaining examples as test set. It is given by:

$$\epsilon_0 = \frac{1}{p} \sum_{i=1}^{p} \sum_{b \in C_i} \delta(y_i, h_b(\mathbf{x}_i; \lambda))/B_i \tag{6.12}$$

where C_i is the set of indices of the bootstrap dataset not containing the ith data point, and B_i is a number of such bootstrap datasets.

Because the derivation of the coefficients of the Equation 6.10 (namely 0.368 and 0.632) is complex, it is not described here (see [26] for details). In [41] there is an overview of the functions showed in this section. The following section shows the experiments performed to optimize these functions using the proposed GA (model II).

6.4.4 Experimental Studies

The experiments presented in this section compare four fitness functions: holdout, cross-validation, bootstrap, and GCV. The holdout result was taken from section 6.3.3. The crossvalidation function used ten folds (i.e., the 10-fold crossvalidation) and the .632 bootstrap used 20 bootstrap datasets.

Note that in each chromosome evaluation using holdout only one training is carried out (namely the training over the holdout set). GCV also performs one training per evaluation. Whereas the 10-fold-crossvalidation performs 10 training sessions (owing to 10 folds) in each evaluation and bootstrap performs 20 training sessions (owing to 20 bootstrap datasets). Obviously, the crossvalidation and bootstrap consume much CPU-time. This is a drawback for GAs, once GAs need fast objective functions in order to reduce its processing time.

Because holdout and GCV consume little CPU-time, they become attractive as fitness functions. The drawback is that holdout can lead the learning algorithm to overfit the holdout set once it is always the same during the execution of the GA. A possible solution to this problem is shown in section 6.5.1. GCV did also not give a good estimate of the true error and overfitted the dataset. As GCV is a criterion derived from the linear models, it does not seem to be suitable for nonlinear models such as RBF networks, mainly if optimized by a powerful algorithm such as GA.

Crossvalidation and bootstrap make a more efficient use of the dataset by re-sampling it a lot of times. So they have, in general, better performance than holdout and GCV as it is confirmed in Figure 6.4.4. However, in terms of complexity, holdout produced networks as small as those produced by 10-fold-crossvalidation (see Figure 6.4.4). Some alternative methods are suggested in next sections to avoid the long processing time of the crossvalidation and bootstrap functions.

6.5 Other Heuristics for Model Selection via GAs

Three heuristics for model selection via GAs are used in this work:

1. Occam razor;
2. Shuffling;
3. Growing.

The occam razor was presented in section 6.3.2. Shuffling and growing are presented in next sections.

6.5.1 Shuffling

In order to improve the quality of the genetic hypotheses, the population evaluation procedure is modified by shuffling the dataset just before the fitness evaluation. Consider

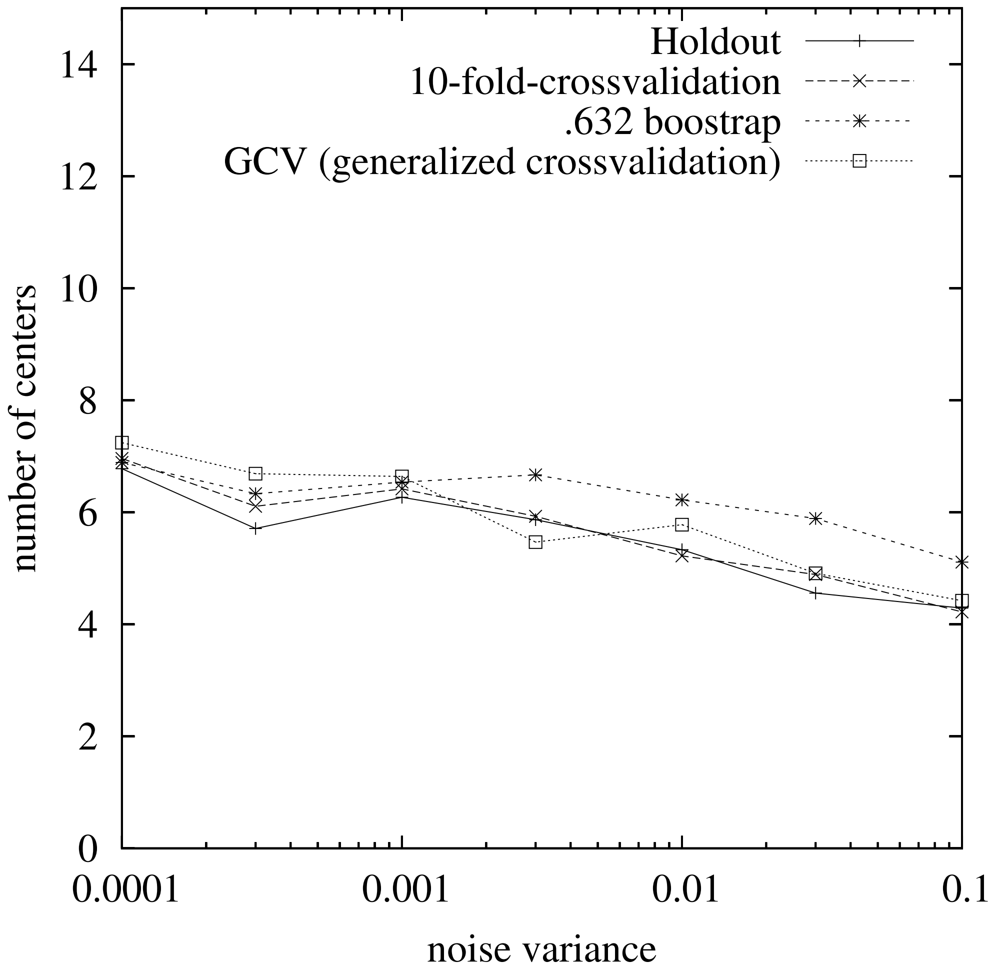


Figure 6.6: Comparing the performance of several kinds of fitness in terms of the number of basis functions.

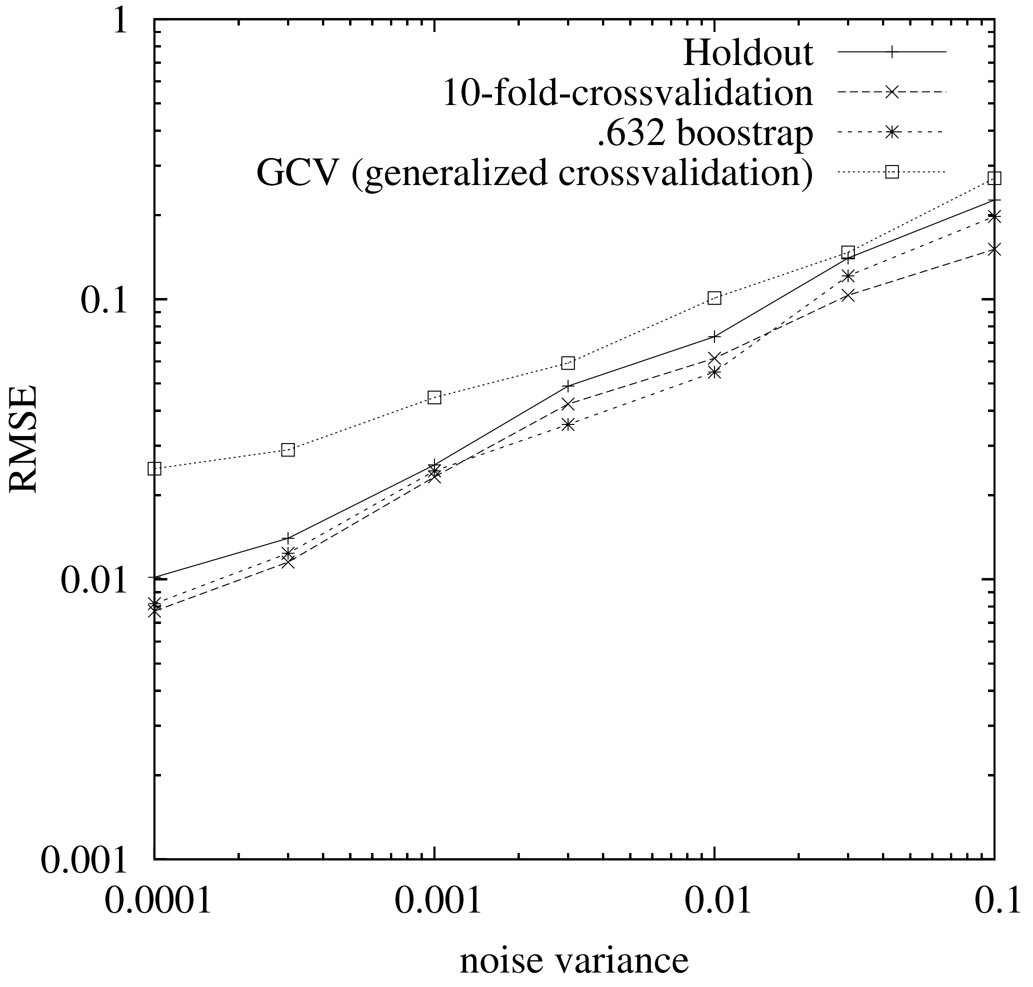


Figure 6.7: Comparing the performance of several kinds of fitness in terms of the number of basis functions.

that shuffle() is a computer procedure that shuffles the dataset $\mathcal{D}$ producing the shuffled dataset $\mathcal{D}^*$. The modified evaluation procedure is shown as follows (it replaces the corresponding procedure illustrated in section 6.3):

```
procedure EvaluateWithShuffling($P$)

for each $\lambda$ in the population $P$ do
  Set $\mathcal{D}^*$ equal to $\mathsf{shuffle}(\mathcal{D})$
  Estimate the true error $\widehat{e}(\lambda, \mathcal{D}^*)$
  Set the fitness $f$ of $\lambda$ equal to $\widehat{e}(\lambda, \mathcal{D}^*)$
end for
```

The shuffling's underling idea is to avoid the holdout set to be kept constant, and therefore avoid overfitting. Nevertheless, the shuffling transforms the holdout into a noise function. GAs, however, are robust to deal with noise functions [34]. In case of elitism, the chromosomes that are kept from generation to generation should be revaluated in each generation in order to verify if their good performance is not due to stochastic errors. The occam elitism should also to use larger parameters (e.g., $k = 10$ and $n = 50$ for a population size = 500) so that the best individuals do not disappear easily from the population due to stochastic errors in the fitness evaluation.

6.5.2 Growing

Stanley and Miikkulainen[94] observed that complexity in nature is developed over time, rather than introduced in the beginning. Based on this observation, they proposed evolving hypotheses starting with minimal hypotheses. This means to set the initial population with minimal networks (e.g., one basis function) instead of random (and probably large) hypotheses. New hypotheses are introduced incrementally as mutations occur. Only those hypotheses that are found to be useful survive through generations.

An experimental justification for the use of growing is as follows. In all experiments performed so far, the number of basis function was limited to 15 basis functions (remember that the number of regions, reported in section 6.3.3, also indicates the maximum number of basis functions). This low limit was used because if GAs start with a higher limit (e.g., 35 basis functions), it may arrive to poor results and large hypotheses.

This occurs due to the large hypotheses of the initial population that tend to dominate the population causing a phenomena known as premature convergence. Such large hypotheses are suboptimal solutions named superindividuals. The use of the growing approach seems to solve this problem once it avoids the production of large hypotheses (superindividuals) in early stages of the evolution. Growing was successfully used for genetic design of RBF networks in [8].

Experiments involving shuffling and growing heuristics are presented in the section 6.6.3.

6.6 The Multiobjective Optimization for Model Selection

In section 6.4.4, the holdout and GCV functions no give good results (due to overfitting) when they were optimized individually. In this section, they are optimized simultaneously. This results in a multiobjective optimization problem. The method used in this work to deal with the multi-objective optimization problem using GAs is described in the next section (it was based on the GA described in [30]).

6.6.1 The Multiobjective Genetic Algorithm

Let $f_1, f_2, \ldots, f_q$ be the set of objective functions to be minimized. Instead of a single objective function, each chromosome is now evaluated by a multi-objective function. A chromosome fitness is defined by a vector where each component is the value of an objective function. In order to compare chromosomes through these vectors, the following definitions are used [30]:

Definition 6.3 *Let* $\mathbf{a}$ *and* $\mathbf{b}$ *be vectors of objective function values. A vector* $\mathbf{b}$ *is said to be dominated by (or inferior to) a vector* $\mathbf{a}$ *iff* $\mathbf{a}$ *is partially-less-than* $\mathbf{b}$ *(in symbols* $\mathbf{a} <_p \mathbf{b}$*), where:*

$$\mathbf{a} <_p \mathbf{b} \iff \forall i(a_i \leq b_i) \wedge \exists i(a_i < b_i) \tag{6.13}$$

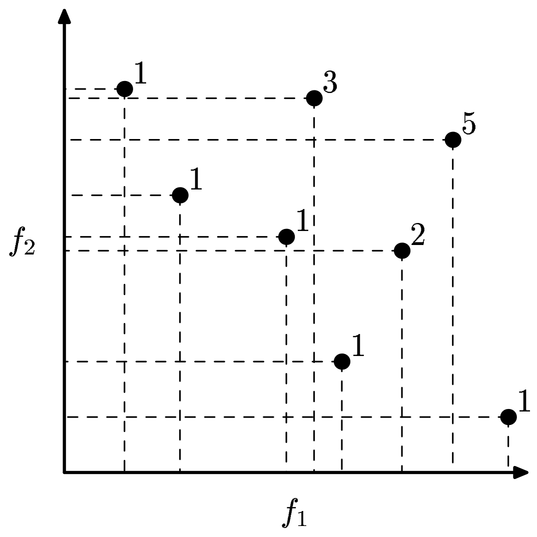


Figure 6.8: Pareto ranking method.

Definition 6.4 *A vector* $\mathbf{a}$ *is said to be non-dominated (or non-inferior) if there is not any other vector (in the population) that dominates* $\mathbf{a}$.

The set of all non-dominated vectors of the population is named the *Pareto-optimal set* (also named the Pareto frontier). The goal of the multi-objective optimization problem is to find the Pareto-optimal set.

Ranking

Using the concept of non-domination, it is possible to rank the population. Pareto ranking methods are proposed in [34, 30]. According to [30], an individual $\mathbf{P}$ that is dominated by n individuals in the current population has its rank given by:

$$\text{rank}(\mathbf{P}) = 1 + n. \tag{6.14}$$

All nondominated individuals are assigned rank 1. Figure 6.8 shows an example where rank 4 is absent.

Fitness Assignment

Let $\mathbf{P}_1, \ldots, \mathbf{P}_N$ be the population sorted by ascendent order of rank, where N is the population size. The fitness of each individual is given by:

$$\text{fitness}(\mathbf{P}_k) = s - \frac{2(k-1)(s-1)}{N-1} \tag{6.15}$$

where $s \in \{1, 2\}$ is an user defined parameter named selection pressure (best/median fitness ratio). See section 2.2.3. Experiments have showed that s equal to either 1.1 or 1.2 provides good results. In order to ensure that individuals with the same rank have the same fitness, they are averaged by:

$$\text{fitness}(\mathbf{P}_k) = \langle \text{fitness}(\mathbf{P}_i) \rangle \tag{6.16}$$

where $\langle \cdot \rangle$ denotes the average over all individuals $\mathbf{P}_i$ with rank equal to $\text{rank}(\mathbf{P}_k)$ (i.e, $\forall i (\text{rank}(\mathbf{P}_i) = \text{rank}(\mathbf{P}_k))$. Parents are sampled for crossover and mutation using the Stochastic Universal Sampling, SUS, technique (see section 2.4.7).

6.6.2 The Choice of the Fitness Functions

Two inexpensive fitness functions were chosen to be optimized via the multiobjective GA. Namely, Holdout and GCV functions. They are described as follows.

1. Holdout fitness function. As before, the dataset $\mathcal{D}$ is divided into two parts: the training set $\mathcal{D}_t$, on which the hypothesis is built, and the holdout set $\mathcal{D}_h$. The first fitness of the chromosome λ is given by holdout function:

$$f_1(\lambda) = \frac{1}{|\mathcal{D}_h|} \sum_{(\mathbf{x},y) \in \mathcal{D}_h} \delta(y, h(\mathbf{x}; \lambda, D_t)) \tag{6.17}$$

2. GCV fitness function. Unlike the previous GCV presented in section 6.4.2, here, the GCV is applied to $\mathcal{D}_t$ (instead of the whole dataset $\mathcal{D}$):

$$f_2(\lambda) = \frac{p_t \, \text{SSE}_t}{(p_t - m)^2} \tag{6.18}$$

where SSE_t denotes the sum of squared errors on the dataset $\mathcal{D}_t$, m is the number of weights (free parameters) and p_t is the number of examples in $\mathcal{D}_t$.

Occam Elitism

The occam elitism was adapted for the multiobjective GA as follows: all the simplest hypotheses of the Pareto set are kept from generation to generation. In the last generation, a criterion is needed to choice an unique hypothesis from the Pareto set. The following criterion was adopted: pick up the simplest hypothesis from the Pareto set. If there exists more than one such hypothesis then pick the median of them.

Experiments involving the multiobjective GA and the heuristics (from section 6.5) are presented in the next section.

6.6.3 Experimental Studies

In the experiments carried out in section 6.4.4, the holdout function did not perform well because large hypotheses tend to overfit the holdout set. The same happened with the GCV method. Nevertheless, the simultaneous optimization of both holdout and GCV seems be a promising idea, due to the results shown in Figures 6.6.3 and 6.6.3.

These experiments also showed that the multiobjetive GA with shuffling and growing were better that the multiobjetive GA without them. That is, the shuffling and growing heuristics incorporated into a multiobjective GA improved its performance in both error on test set and complexity. The results obtained suggest that multiobjective GA with shuffling and growing is comparable to crossvalidation and bootstrap.

A possible explanation to this good result is that GCV penalizes large hypotheses (once its denominator diminishes in large hypotheses). As a consequence, GCV contributes to avoid the overfitting of the holdout. In spite of making the holdout function a noise function, the use of shuffling also contributes to avoid holdout overfitting (once the holdout set is not fixed). Growing adds basis functions incrementally, making the selection of centers more rigorous (once complexity is only added if necessary) and natural. The combined effect of all these alternative methods seems be useful for model selection.

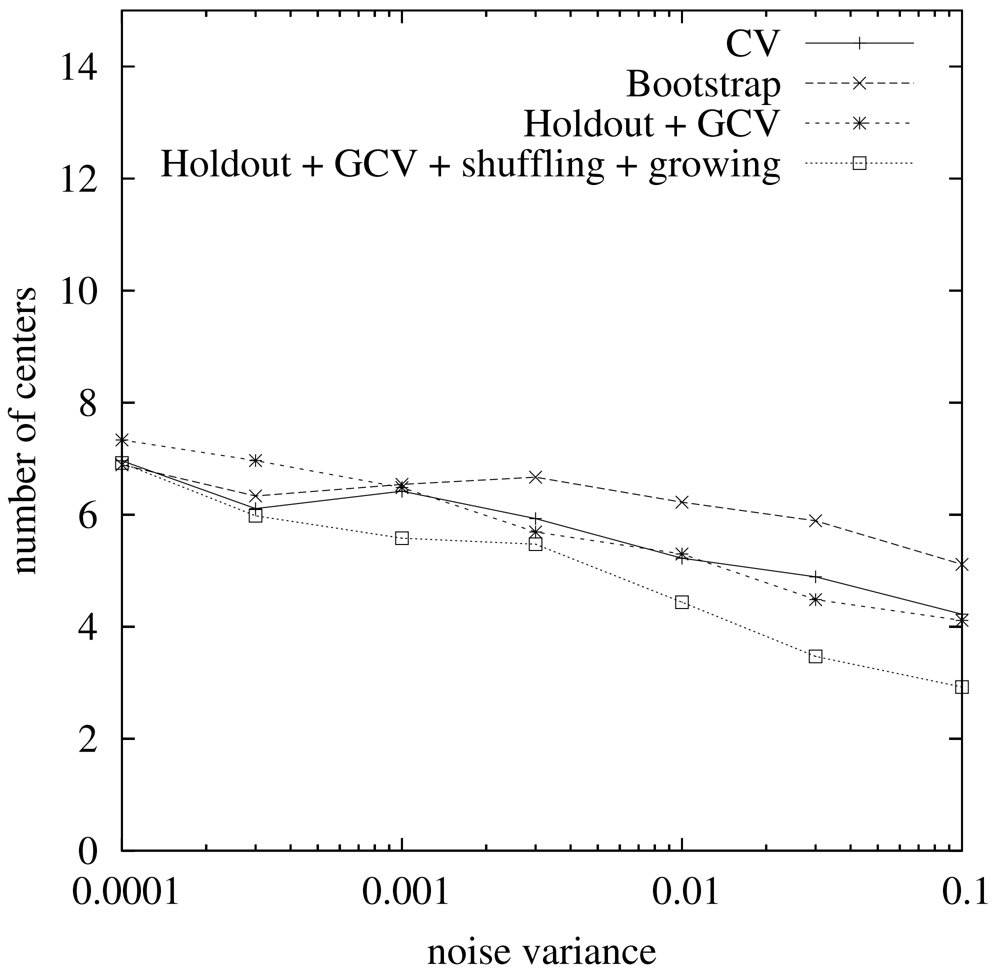


Figure 6.9: Comparing the performance of the multiobjective GA in terms of the number of basis functions.

6.7 Experimental Studies with Other Techniques

In this section, the GA is compared to two constructive algorithms used to determinate the RBF network architecture. Namely, RAN-EKF (Resource Allocating Network with Extended Kalmon Filter) [48] and ROLS (Regularized Orthogonal Least Squares) [78]. The RAN-EKF and ROLS results were extracted from [78]. In this experiments, the best GA (obtained from the section 6.6.3) was used. Namely, the multiojective GA with occam elitism, shuffling and growing from the Figures 6.6.3 and 6.6.3. As shown in Figure 6.7, GA generated RBF networks with a very small number of centers in all

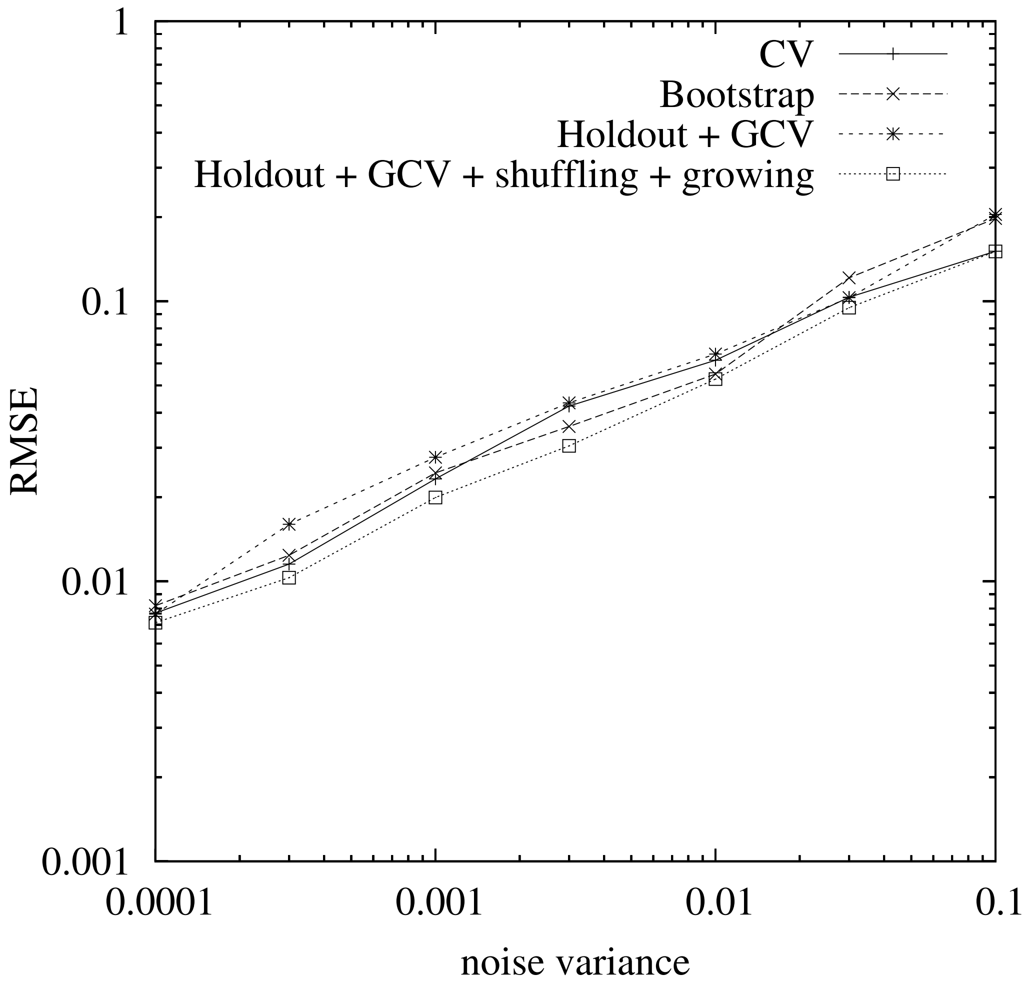


Figure 6.10: Comparing the performance of the multiobjective GA in terms of the error on the test set.

levels of noise. In addition, GA generalization capacities(see Figure 6.7) were better then the ROLS algorithm. Therefore, GA was able to generate parsimonious networks and still keep good generalization. Nevertheless, GA took a long training time (which are orders of magnitude greater than other approaches) to achieved these results. But for a large number of applications, where recognition performance is more important than the training time, the results obtained suggest that the genetic approach is an attractive solution for the design of efficient RBF networks.

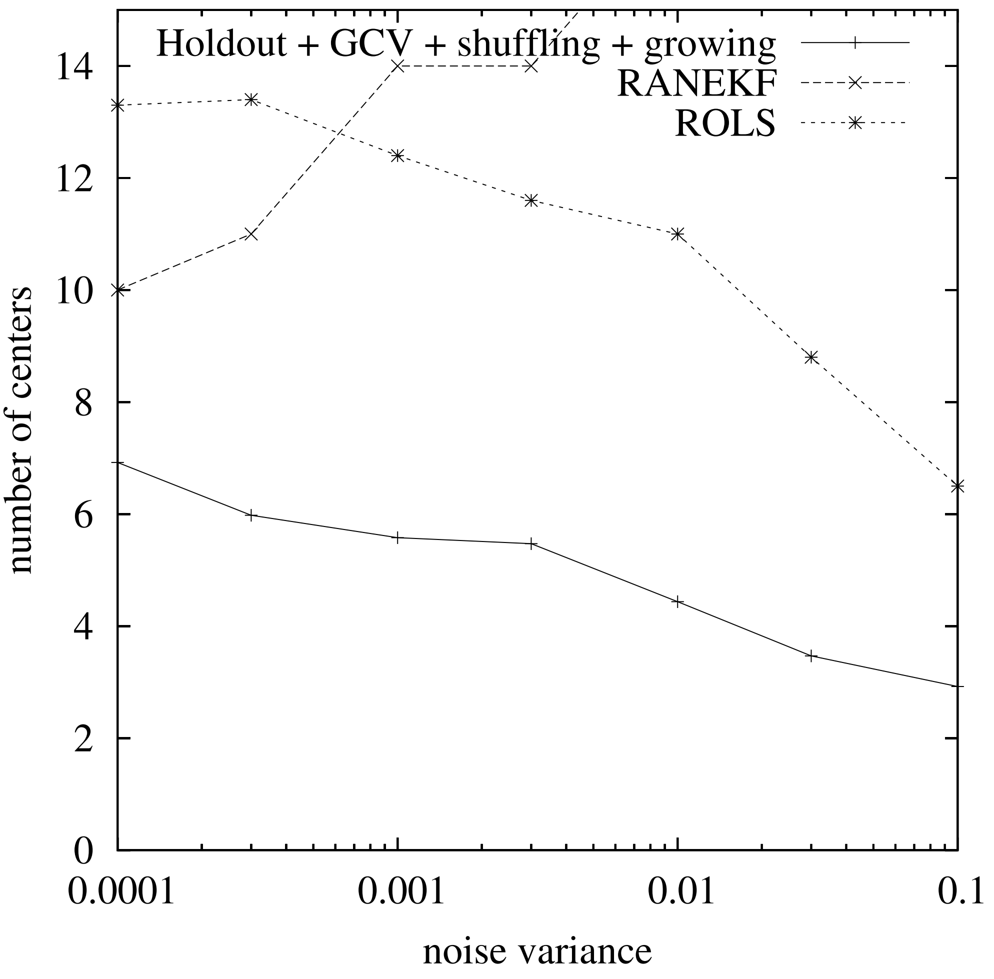


Figure 6.11: Comparing the performance of the multiobjective GA with other techniques in terms of the number of basis functions.

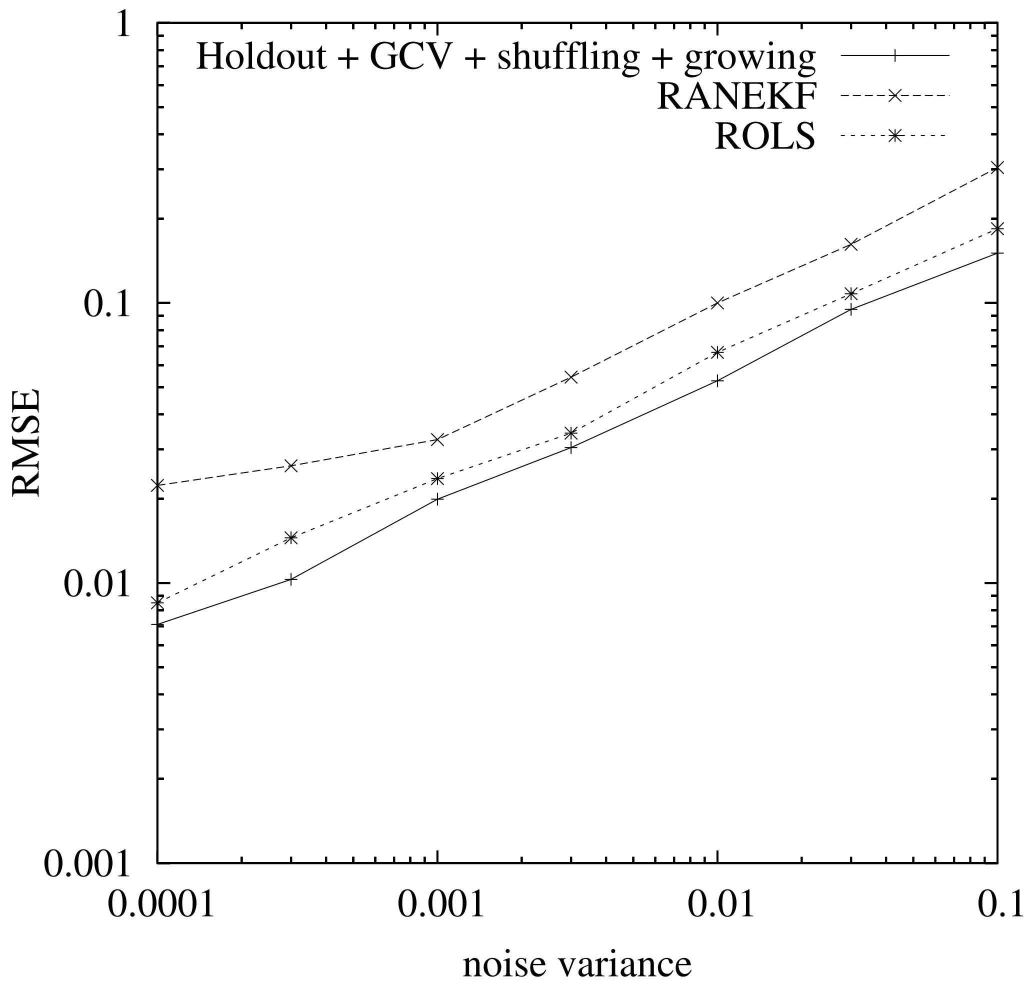


Figure 6.12: Comparing the performance of the multiobjective GA with other techniques in terms of the error on the test set.

Chapter 7

Conclusion

*"Think of your adversary as
a bearer of equilibrium; if we
have need of friends to
stimulates us, we equally
need someone to show us
our errors."*
(Book: Christian Agenda,
André Luiz - medium F.C.Xavier.)

In this work, it was proposed encodings for RBF networks. Some features of these encodings are

- The centers are evolved only on clustering regions of training patterns. It follows that the GA handles only (approximately) good networks in the population by avoiding (spurious) gaussians outside the cluster regions.
- The encodings are nonredundant and legal (according to the definition given in section 4.1.2).
- The crossover is performed on phenotype space (i.e., between regions of the input space corresponding clustering regions of the chromosomes) instead of genotype space (by merely interchanging structural chunks of chromosomes).

It worth noting that the proposed encoding were designed so that network complexity and performance can be evolved simultaneously, but it was also designed to becomes the search space as small as possible. Because of this the centers are strongly biased towards the clustering regions of training patterns and only a minimal set of parameters was considered in the GA optimization. Experiments involving those encodings were presented in chapter 6.

Experiments, in section 6.4.4, showed that the use of the holdout method as objective function is not an effective method and that crossvalidation and boostrap are, in general, the best objective functions for model selection using GAs. Nevertheless, they consume much CPU-time. GAs need fast objective functions to have a reasonable processing time, so the use of crossvalidation and boostrap may not be suitable for GAs.

Experiments, in section 6.6.3, showed that by means of modifications in the traditional GA, it is possible to make a GA (with holdout) an efficient model selection algorithm without the need to use crossvalidation and boostrap. Four modifications in the traditional GA were carried out in which generated better hypotheses than those hypotheses generated by holdout using the traditional GA. The modifications are described as follows:

- To use the occam elitism to keep the simplest hypotheses (among the best ones) from generation to generation. Experiments showed that the occam elitism diminishes the complexity of the hypotheses and their true error.

- To shuffle the dataset just before to compute the fitness function may avoid the overfitting of the holdout method.

- To use the growing approach to start the initial population with minimal hypotheses and add complexity incrementally. Growing avoids the bias towards large hypotheses in the initial population. Growing makes the evolving of complexity a process more rigorous and natural.

- To optimize two inexpensive objective functions (namely holdout and GCV) simultaneously seems to be better than optimize each one individually.

All of theses methods are computationally inexpensive and, if combined, they may produce results equivalent to both crossvalidation and bootstrap, which require a large amount of computer processing. Because these methods do not depend on a particular encoding, they are quite general. Except for some unimportant details, they can be applied to other machine learning models (e.g., decision trees, MLP networks) optimized by GA almost without modifications.

In section 6.7, GA was compared to two traditional approaches to optimize RBF networks. The GA approach was more accurate than the best of the other approaches and yields networks with significantly less number of hidden units in all experiments this work.

This research showed that GA is able to generate parsimonious networks and still keep good generalization. Nevertheless, GA took a long training time (which are orders of magnitude greater than other approaches) to achieved these results. But for a large number of applications, where recognition performance is more important than the training time, the results obtained suggest that the genetic approach is an attractive solution for the design of efficient ANNs. This research also showed that GAs may provide a suitable framework for the model selection problem. Nevertheless, this potential still need be more explored.

Bibliography

[1] A. A. Adewuya. New methods in genetic search with real-valued chromosomes. Master's thesis, M.I.T., 1996.

[2] D. Aha, D. Kibler, and M Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[3] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19:716–723, 1974.

[4] M. R. Anderberg. *Cluster Analisys for Applications*. Academic Press, New York, 1973.

[5] T. Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 57–62, Piscataway. NJ, 1994. IEEE Press.

[6] J. Baker. Reducing bias and inefficiency in the selection algorithm. In J. Grefenstette, editor, *Proc. of the Second International Conference on Genetic Algorithms and Their Applications*, pages 14–21, Hillsdale, New Jersey, 1987. Lawrence Erlbaum Associates.

[7] K. Balakrishnan and V. Honavar. Properties of genetic representations of neural architectures. In *Proceedings of the World Congress on Neural Networks (WCNN'95)*, pages 807–813, Washington, D.C., 1995.

[8] A. Barreto and H. Barbosa. Growing compact RBF networks using a genetic algorithm. In *VII Brazilian Symposium on Neural Networks*, 2002.

[9] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming - Theory and Algorithms*. Jonh Wiley and Sons, second edition, 1993.

[10] D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993. Avaliable by ftp on ENCORE in file: GA/papers/over92.ps.gz.

[11] D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algortihms: Part 2, research topics. *University Computing*, 15(4):170–181, 1993. Avaliable by ftp on ENCORE in file: GA/papers/over93-3.ps.gz.

[12] S. A. Billings and G. L. Zheng. Radial basis function network configuration using genetic algorithms. *Neural Networks*, 8(6):877–890, 1995.

[13] T. Blickle and L. Thiele. A mathematical analysis of tournament selection. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the 6th International Conference (ICGA95)*, San Francisco, CA, 1995. Morgan Kaufmann.

[14] L. Breiman, J. Friedman, R. Olshen, and C Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[15] D. S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[16] B. Burdsall and C. Giraud-Carrier. GA-RBF: A self-optimising RBF network. In *Proc. of the Third International Conference on Artificial Neural Networks and Genetic*, pages 348–351. Springer-Verlag, 1997.

[17] B. Carse and T. C. Fogarty. Fast evolucionary learning of minimal radial basis function neural networks using a genetic algorithm. In T.C. Forgaty, editor, *AISB Workshop on Evolucionary Computing*, Lectures Notes in Computer Science N. 1143, pages 1–22. Springer-Verlag, 1996.

[18] D. J. Chalmers. The evolution of learning: an experiment in genetic connectionism. In D. S. Touretzky, Elman J. L., and G. E. Hinton, editors, *Connectionist models: proceedings of the 1990 summer school, Pittsburgh*, San Mateo, CA, 1991. Morgan Kaufmann.

[19] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on neural networks*, 2(2):302–309, 1991.

[20] S. Chen, Y. Wu, and K. Alkadhimi. A two-layer learning method for radial basis function networks using combined genetic and regularised ols algorithms. In *Proceedings of the 1st IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 245–249, 1995.

[21] Y. H. Cheng and C. S. Lin. A learning algorithm for radial basis function networks: with the capability of adding and pruning neurons. *Proc. IEEE*, pages 797–801, 1994.

[22] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3, 261-284.

[23] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princenton, NJ, 1963.

[24] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[25] K. DeJong. *The analysis and behaviour of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.

[26] B. Efrom and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, 1993.

[27] L. J. Eshelman, R. A. Caruna, and J. D. Schaffer. Biases in the crossover landscape. In J. D. Schaffer, editor, *Proc. of the Third Int. Conf. on Genetic Algorithms*, pages 10–19, San Mateo, CA, 1989. Morgan Kaufmann.

[28] L. J. Eshelman and D. J. Shaffer. Real-coded genetic algorithms and interval-schemata. In D. L. Whitley, editor, *Foundations of Genetic Algorithms 3*, pages 187–203. San Mateo, CA: Morgan Kaufman, 1992.

[29] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936. Reimpresso in: Contributions to Mathematical Statistics. New York, Jonh Wiley, 1950.

[30] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423, San Mateo, 1993. Morgan Kaufmann Publishers, Inc.

[31] C. M. M. Fonseca. *Multiobjective genetic algorithms with application to control engineering problems*. PhD thesis, Department of Automatic Control and Systems Engineering - University of Sheffield, 1995.

[32] M. Gen and R. Cheng. *Genetic Algorithms and Engineering Optimization*. Wiley, 2000.

[33] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publisher, 1997.

[34] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[35] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, San Mateo, 1991.

[36] G. H. Golub, M. Heath, and G. Wahba. Generalised cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979.

[37] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE trans SMC*, 16:122–128, 1986.

[38] P. J. B. Hancock. Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification. In *Proceedings of the IEEE Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 108–122, 1992.

[39] J. V. Hansen and R. D. Meservy. Learning experiments with genetic optimization of a generalized regression neural network. *Decision Support Systems*, 18:317–325, 1996.

[40] S. A. Harp, T. Samad, and A. Guha. Towards the genetic synthesis of neural networks. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 360–369. Morgan Kaufmann, 1991.

[41] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2002.

[42] R. L. Haupt and S. E. Haupt. *Pratical Genetic Algorithms*. Wiley-Intercience, 1998.

[43] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, second edition, 1999.

[44] D. Heckerman. A tutorial on learning Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, Redmond, WA, 1995.

[45] J. Heitkötter and D. Beasley. The hitch-hiker's guide to evolutionary computation: A list of frequently asked questions (faq). USENET: comp.ai.genetic. Available via anonymous FTP from rtfm.mit.edu/pub/usenet/news.answers/ai-faq/genetic/ About 110 pages., 1998.

[46] F. Herrera, M. Lozano, and J. L. Verdegay. Tackling real-coded genetic algorithms: operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4):265–319, 1998.

[47] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[48] V. Kadirkamanathan and M. Niranjan. A function estimation approach to sequential learning with neural networks. *Neural Computation*, 5(6):954–975, 1993.

[49] Allan Kardec. *The Gospel Explained by the Spiritist Doctrine*. Allan Kardec Educational Society, 2003. www.febnet.com.br.

[50] S. Kirpatrick, C. D. Gellat Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, pages 671–680, 1983.

[51] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[52] T. Kohonen. Self-organized formation of topologically correct feacture maps. *Biological Cybernetics*, 43:59–69, 1982.

[53] M. Kubat. Decision trees can initialize radial-basis function networks. *IEEE Transactions on Neural Networks*, 9(5):813–821, 1998.

[54] L. I. Kuncheva. Initializing of an RBF network by a genetic algorithm. *Neuro-computing*, 14:273–288, 1997.

[55] E. G. M. Lacerda and A. C. P. L. F. Carvalho. Combinando os algoritmos genético e k-média para configurar redes de funções base radial. In *Anais do IV Simpósio Brasileiro de Redes Neurais*, pages 95–97, Goiânia, Brazil, 1997.

[56] E. G. M. Lacerda and A. C. P. L. F. Carvalho. Credit analysis using radial basis function networks. In *3rd International Conference on Computational Intelligence and Multimedia Applications, ICCIMA'99*, New Dheli, India, September 1999. IEEE Computer Press.

[57] E. G. M. Lacerda and A. C. P. L. F. Carvalho. Introdução aos algoritmos genéticos. In *Jornada de Atualização em Informática - Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação*, Rio de Janeiro-RJ, 1999.

[58] E. G. M. Lacerda and A. C. P. L. F Carvalho. Introdução aos algoritmos genéticos. In C. O. Galvão and M. J. S. Valença, editors, *Sistemas inteligentes: aplicações a recursos hídricos e ciências ambientais*. Ed. Universidade/UFRGS : Associação Brasileira de Recursos Hídricos, Porto Alegre, RS, 1999. Coleção ABRH de Recursos Hídricos - ISBN 85-7025-527-6.

[59] E. G. M. Lacerda, A. C. P. L. F. Carvalho, and T. B. Ludermir. Evolutionary optimization of RBF networks. In *VIth Brazilian Symposium on Neural Networks*, 2000.

[60] E. G. M. Lacerda, A. C. P. L. F. Carvalho, and T. B. Ludermir. Evolutionary optimization of RBF networks. In R. J. Howlett and L. C. Jain, editors, *Radial*

Basis Fuction Networks 1: Recent Developments in Theory and Applications. Physica Verlag, 2001. Studies in Fuzziness and Soft Computing, Vol. 66 - ISBN 3-7908-1367-2.

[61] E. G. M. Lacerda, A. C. P. L. F. Carvalho, and T. B. Ludermir. Evolutionary optimization of RBF networks. *International Journal of Neural Systems*, 11(3):287–294, 2001.

[62] E. G. M. Lacerda, A. C. P. L. F. Carvalho, and T. B. Ludermir. Model selection via genetic algorithms for RBF networks. *Journal of Intelligent Fuzzy Systems*, 13:111–122, 2002.

[63] E. G. M. Lacerda, A. C. P. L. F. Carvalho, and T. B. Ludermir. A study of crossvalidation and bootstrap as objective functions for genetic algorithms. In *VII Brazilian Symposium on Neural Networks*, 2002.

[64] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. Prentice-Hall, Englewood Cliffs, NJ, 1974. 2nd edition: 1995, Philadelphia: SIAM.

[65] C. Lucasius and G. Kateman. Towards solving subset selection problems with the aid of the genetic algorithm. In *Parallel Problem Solving from Nature Vol. 2*. Elsevier Science Publishers, 1992.

[66] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, second edition, 1986.

[67] André Luiz. *Christian Agenda*. Allan Kardec Publishing LTD, London, third edition, 1998. Dictated by the spirit André Luiz to Francisco Cândido Xavier.

[68] D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.

[69] E. P. Maillard and D. Gueriot. RBF neural network, basis functions and genetic algorithm. In *Proceedings of International Conference on Neural Networks Vol. 4*, pages 2187–2192, 1997.

[70] M. W. Mak and K. W. Cho. Genetic evolution of radial basis function centers for pattern classification. In *Proc. of The 1998 IEEE International Joint Conference on Neural Networks*, volume 1, pages 669–673, 1998.

[71] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1994.

[72] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[73] R. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–161, 1983.

[74] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[75] J. Moody and C. J Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[76] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M Hummels. On the training of radial basis function classifiers. *Neural Networks*, 5:595–603, 1992.

[77] R. Neruda. Functional equivalence and genetic learning of RBF networks. In D. W. Pearson, N. C. Steele, and R.F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms*, pages 53–56. Springer-Verlag, 1995.

[78] M. J. L. Orr. Regularisation in the selection of radial basis function centers. *Neural Computation*, 7(3):606–623, 1995.

[79] M. J. L. Orr. Introduction to radial basis function networks. Technical report, Institute for Adaptive and Neural Computation, Division of Informatics, Edinburgh University, 1996.

[80] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.

[81] M. Powell. The theory of radial basis function approximation in 1990. In W. Light, editor, *Advances in Numerical Analysis, vol.3*, pages 105–210. Clarendon, Oxford, 1992.

[82] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, second edition, 1992.

[83] J. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[84] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[85] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[86] R. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

[87] A. Roy, S. Govil, and R. Miranda. An algorithm to generate radial basis function (rbf)-like nets for classification problems. *Neural Networks*, 8(2):179–201, 1995.

[88] D. E. Rumelhart, G. E. Hilton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362, Cambridge, MA, 1986. Mit Press.

[89] A. Saha and J. D. Keller. Algorithms for better representation and faster learning in radial basis function networks. In D. S. Touretzki, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 482–489, 1990.

[90] J. D. Schaffer, D. Whitley, and L. J. Eschelman. Combinations of genetic algorithms and neural networks: a survey of the state of the art. In D. Whitley and J. D. Schaffer, editors, *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, pages 1–37. IEEE, 1992.

[91] B. Sendhoff, M. Kreutz, and W. von Seele. A condition for the genotype-phenotype mapping: Causality. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA'97)*, pages 354–361, San Francisco, 1997. Morgan Kauffman.

[92] A. F. Sheta and K. D. Jong. Time-series forecasting using ga-tuned radial basis functions. *Information Sciences*, 133:221–228, 2001.

[93] C. Stanfill and D. Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, 1986.

[94] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. Technical Report TR-AI-01-290, The University of Texas at Austin - Department of Computer Sciences, 2001.

[95] M. Stone. Crossvalidatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society B. 2*, pages 111–147, 1974.

[96] S. Unger. *The Essence of Logic Circuits*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[97] V. Vapnik. Principles of risk minimization for learning theory. In John Moody, Steven Hanson, and Richard Lippmann, editors, *Advances in Neural Information Processing Systems - NIPS 1991*, volume 4, pages 831–838, 1992.

[98] J. M. Vesin and R. Grüter. Model selection using a simplex reproduction genetic algorithm. *Signal Processing*, 78:321–327, 1999.

[99] G. Wahba. *Spline Models for Observational Data*. CBMS-NSF Regional Conference Series in Applied Mathematics, 59. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA., 1990.

[100] B. A. Whitehead and T. D. Choate. Evolving space-fiiling curves to distribute radial basis functions over an input space. *IEEE Transactions on Neural Networks*, 5(1):15–23, 1994.

[101] B. A. Whitehead and T. D. Choate. Cooperative-competitive genetic evolution of radial basis function centers and widths for time series prediction. *IEEE Transactions on Neural Networks*, 7(4):869–880, 1996.

[102] D. Whitley. The GENITOR algorithm and selection pressure: Why rankbased allocation of reproductive trials is best. In J. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121, San Mateo, Calif., 1989. Morgan Kaufmann.

[103] B. Widrow and M. E. Hoff. Adaptive switching circuits. *IRE-WESCON Convention Record*, 4:96–104, 1960.

[104] A. Wright. Genetic algorithms for real parameter optimization. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 205–218, 1991.

[105] X. Yao. Evolving artificial neural networks. *PIEEE: Proceedings of the IEEE*, 87, 1999.

[106] L. Yingwei, N. Sundararajan, and P. Saratchandran. A sequential learning scheme for function approximation using minimal radial basis function neural networks. *Neural Computation*, 9:461–478, 1997.