THE BEES ALGORITHM

A Novel Optimisation Tool

A thesis submitted to the Cardiff University

In candidature for the degree of

Doctor of Philosophy

By

Afshin Ghanbarzadeh, B.Sc., M.Sc.

Manufacturing Engineering Centre

Cardiff University

United Kingdom

2007

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

In the name of Allah,

The Most Gracious, The Most Merciful

DEDICATION

This work is entirely dedicated to my family

To my wife:

Elham Tahanpesar

To my parents:

Mohammad-Ali Ghanbarzadeh
Kobra Nasri

To my children:

Armin and Arezoo
Ghanbarzadeh

To my brothers:

Ehsan and Amir
Ghanbarzadeh

ACKNOWLEDGEMENTS

I would like to express my special gratitude to the supervisor of my studies, Professor D.T. Pham, for his encouragement, invaluable guidance and strong support throughout my studies. I consider myself very lucky to have him as my study supervisor.

I wish to express my sincere thanks to the Cardiff University, especially the Manufacturing Engineering Centre for the use of the facilities to pursue this research work.

Grateful acknowledgement for my funding and support must be made to my home country Iran and the Iranian Ministry of Research Science and Technology.

I wish to thank my friends in the Cardiff Bay Bees, exceptional thanks go to Marco Castellani for reviewing the thesis and for sharing his knowledge with every one in Bay.

Naturally I owe my parents a great dept of gratitude for all the love and consistent support they have given me. I also want to express my warmest thanks to my parents-in-law for their help and support.

In would like also to thank my brothers, Ehsan and Amir for their encouragement.

My final and most heartfelt acknowledgment must go to my wife Elham. Her love, support, encouragement and companionship turned my time in Cardiff into a pleasure. I also should mention my lovely twin kids, Armin and Arezoo, for their understanding and tolerance of their father's absences on evening and weekends that were devoted to this thesis.

ABSTRACT

This work introduces the Bees Algorithm, a new optimisation algorithm inspired by the foraging behaviour of honey-bees. In its basic version, the Bees Algorithm performs a kind of neighbourhood search combined with global random search and can be used for both continuous and discrete optimisation problems.

An improved version of the Bees Algorithm including replacing global random search with interpolation and extrapolation, shrinking neighbourhood size, and abandoning sites with no new information was developed. The improved version could solve benchmark function optimisation problems with less sampling of the search space.

The Bees Algorithm has been applied to mechanical design optimisation. Two standard mechanical design problems, the design of a welded beam structure and the design of coil springs, were used to benchmark the Bees Algorithm against other optimisation techniques.

Computer-aided preliminary design can be regarded as a special case of optimisation, where the goal is to generate as many solutions as possible above a predefined performance threshold. The higher the number of solutions satisfying the preliminary selection criteria, the greater is the chance to produce a good final solution. An adapted version of the Bees Algorithm for discrete function optimisation was developed and tested on a simple machine design task, preliminary

gearbox design. The test consists of finding alternative gearbox configurations that approximately produce the required output speeds using one of the available input speeds. Experimental results show that the Bees Algorithm outperforms random search and a genetic optimisation algorithm.

A modified version of the Bees Algorithm was used to search for multiple Pareto optimal solutions in a multi-objective optimisation design problem. Compared to two non-dominated genetic algorithms, the Bees Algorithm was able to find more trade-off solutions.

Finally, the Bees Algorithm was employed to train Radial Basis Function (RBF) neural networks for two different problems. Despite the high dimensionality of the problems – each bee represented 2345 parameters in the control chart pattern recognition case and 1581 parameters in the wood defect classification case - the algorithm successfully trained very accurate classifiers. Although the accuracies achieved were marginally lower than those obtained with conventional RBF training methods, the total output errors were less than those for conventionally RBF-trained networks with same number of hidden neurons.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ACO	Ant Colony Optimisation
ANTS	Ant Colony System
AVI	Automated Visual Inspection
B A	Bees Algorithm
CAD	Computer Aided Design
CCD	Charge-Coupled Device
CCP	Control Chart Pattern
DE	Differential Evolution
DNA	DeoxyriboNucleic Acid
EA	Evolutionary Algorithms
EP	Evolutionary Programming
ES	Evolutionary Strategy
FEA	Finite Element Analysis
GA	Genetic Algorithm
MDC	Minimum Distance Classifier
NE SIMPSA	stochastic simulated annealing optimisation procedure
NN	Neural Network
NSGA	Non-dominated Sorting Genetic Algorithm
PSO	Particle Swarm Optimisation
RBF	Radial Basis function
SIMPSA	the deterministic Simplex method
SPC	Statistical Process Control
SQP	Sequential Quadratic Programming

NOMENCLATURE

CHAPTER 2

α , β	control parameter
$\Delta\tau_{ij}$	amount of pheromone deposited
η_{ij}	Desirability of the corresponding link, (i, j)
μ , λ	number of individuals
ρ	rate of pheromone evaporation
τ_{ij}	pheromone intensity of the corresponding link, (i, j)
φ_1 , φ_2	weights
c_1 , c_2	acceleration coefficients
n_J	number of jobs
n_M	number of machines
$\overrightarrow{p_g}$	global best found by the swarm
$\overrightarrow{p_i}$	best position found by particle i
p_{ij}	probability of moving from node i to node j
$\Re^n$	n-dimensional search space
$\overrightarrow{v_i}$	velocity of particle i
w	inertia weight
$\overrightarrow{x_i}$	position of particle i

CHAPTER 3

γ	weight density
δ	beam end deflection
Δ	minimum spring deflection
ρ	mass density
σ	maximum normal stress in beam
σ_d	allowable normal stress for beam material
τ	maximum shear stress in weld
τ'	primary stress
τ''	secondary stress
τ_d	allowable shear stress
ω	frequency of surge waves
ω_0	lower limit on surge wave frequency

b	beam width
c_1	Unit volume of weld material cost
c_2	Unit volume of bar stock cost
d	wire diameter
D	mean coil diameter
D_0	limit on outer diameter of the coil
e	number of top-rated (elite) sites
F	load
f f	Cost function including setup cost
f(x)	objective function
g	gravitational constant
g_i	constraint i
G	shear modulus
h	weld thickness
l	weld length
L	fixed distance from load to support
m	number of sites selected
M	mass
n	number of scout bees
N	number active coils
nep	number of bees recruited for the best e sites
ngh	initial size of each patch
nsp	number of bees recruited for the other (m-e) selected sites
P	applied axial load
P_c	bar buckling load
Q	number of inactive coils
t	beam thickness
x	a scalar or a vector
x_{ie}	position of an elite bee in the ith dimension

CHAPTER 4

A1 , A2	coefficients
$c_j(X)$	equality constraints
F	fitness
$f_i(X)$	objective functions
$h_k(X)$	inequality constraints
MaxAllowedFitness	maximum allwed fitness
MAXRATIO	maximum single-stage transmission ratio
Ni	number of teeth of gear i
NoOfShafts	number of shafts

CHAPTER 5

μ	mean value of the process variable being monitored
σ	standard deviation of the process
σ_i	spread of the function i.
a	amplitude of cyclic variations
c_i	centre of basis function i
d	desirable output
g	magnitude of the gradient of the trend
k	parameter determining the shift position
$\mathrm{k}(K_1,.....,K_h)^T$	outputs of the hidden neurons
r	normally distributed random number
s	magnitude of the shift
t	discrete time at which the pattern is sampled
T	period of a cycle
w_{ji}	weight of connection from hidden neuron i to output j
$\|x-c_i\|$	norm of $(x-c_i)$
x_1 to x_N	input layer neurons
$\overline{y}$	scaled pattern value
$y(t)$	sample value at time t
y_1 to y_p	network outputs
$y_{\max}$	maximum allowed value
$y_{\min}$	minimum allowed value

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

Optimisation algorithms are search methods where the goal is to find a solution to a problem, such that a given quantity is maximised or minimised, possibly subject to a set of constraints. Although this definition is simple, it hides a number of complex issues. For example, the solution may consist of a combination of different data types, nonlinear constraints may restrict the search area, the search space may be convoluted with many candidate solutions, the characteristics of the problem may change over time, or the optimisation problem may have conflicting objectives or constraints. This is just a short list of issues, given to illustrate some of the complexities an optimisation algorithm may have to face.

1.1 Motivation

Studies of social animals and social insects have resulted in a number of computational models of swarm intelligence. Biological swarm systems that have inspired computational models include ants, termites, bees, fish schools, and bird flocks (Engelbrecht 2005). Within these swarms, individuals are relatively simple in structure, but their collective behaviour is usually very complex. The collective behaviour of a swarm of social organisms emerges from the behaviours of the individuals of that swarm.

An objective of computational swarm intelligence models is to represent the simple behaviours of individuals, and the local interactions with the environment and

neighbouring individuals, in order to obtain more complex behaviours that can be used to solve complex problems.

Researchers have developed computational problem-solving methods based on biology. Genetic Algorithms, Particle Swarm Optimisation, and Ant Colony Optimisation are examples of these types of methods.

This work introduces a new optimisation algorithm inspired by the foraging behaviour of honey-bees. The proposed algorithm which is called the Bees Algorithm can be used as an alternative to current optimisation procedures. As each optimisation algorithm can produce good results for some problems and poor results for other problems in comparison with other methods, creating a new algorithm can give users a tool which is better adapted to their particular cases.

1.2 Research Objectives

The overall aim of this thesis was to develop, improve and test a new swarm-based tool for optimisation problems.

The following objectives were set to achieve the aim.

- Survey current swarm-based optimisation algorithms.

- Develop a new optimisation tool mimicking the behaviour of social insects which operate in swarms.

- Adapt the optimisation tool for different categories of optimisation problems, namely, continuous or discrete, constrained or unconstrained, single or multi-objective optimisation problems.

- Validate the optimisation algorithm by applying it to different benchmark optimisation problems and compare results with other methods to evaluate the overall performance of the algorithm.

1.3 Thesis Organisation

The remainder of thesis is organised as follows:

Chapter 2 surveys current swarm-based optimisation techniques including those inspired by the behaviour of bees.

Chapter 3 describes the basic Bees Algorithm and its application to function optimisation and mechanical design optimisation problems which are constrained optimisation problems. Also, an improved version of the Bees Algorithm is introduced.

Chapter 4 presents adapted versions of the Bees Algorithm for multi-solution and multi-objective optimisation problems. The modified algorithm for multi-solution optimisation problems is applied to a preliminary design problem. The multi-objective version of the Bees Algorithm is applied to a mechanical design optimisation problem with two objective functions.

Chapter 5 describes the Radial Basis Function neural network and how the Bees Algorithm has been used to train the network instead of standard methods of training. Two examples of employing Radial Basis Function neural networks trained with the Bees Algorithm, control chart pattern recognition and wood defect classification, are presented in this chapter.

Chapter 6 summarises the conclusions and contributions of the research, and gives suggestions for further investigations.

CHAPTER 2

SWARM-BASED

OPTIMISATION

2. SWARM-BASED OPTIMISATION

A recent trend is the introduction of non-classical stochastic search optimisation algorithms. Swarm algorithms mimic different nature's problem-solving strategies to drive their search towards an optimal solution. One of the most striking differences between swarm algorithms and direct search algorithms such as hill climbing is that the former use a population of solutions for every iteration, instead of a single solution. That is, a population of solutions is processed at every iteration, and the outcome is also a population of solutions. If an optimisation problem has a single optimum solution, all population members of a swarm algorithm can be expected to converge to that solution. However, if an optimisation problem has multiple optimal solutions, a swarm algorithm can be used to capture multiple optimal solutions in its final population. Swarm algorithms include Ant Colony Optimisation (ACO) (Dorigo and Stützle 2004), Evolutionary Algorithms (EA) (Fogel 2000) and Particle Swarm Optimisation (PSO) (Eberhart et al. 2001).

Common to all population based search methods is a strategy that generates variations of the tuning parameters. Most search methods use a greedy criterion to make this decision, which accepts the new parameter if and only if it produces better solutions than the old search parameter.

2.1 Evolutionary Algorithms (EAs)

Evolutionary Algorithms (EAs) are inspired by Darwin's evolution theory. Natural selection and adaptation in Darwinian evolution are the key sources of inspiration, driving the EAs candidate solutions towards the optimum by 'survival of the fittest'. An EA consists of a population of individuals each having a fitness value, and a genome encoding the main features of the candidate solution to the given problem. General to all EAs is also a selection pressure mechanism that removes poor individuals from the population, thus allowing good individuals to control the evolutionary process. EAs also modify the individuals to refine the population of candidate solutions. EAs have been described by several researchers including Michalewicz (Michalewicz 1996; Michalewicz and Fogel 2004), Goldberg (Goldberg 1989), etc.

Historically EAs were born in the 1960s, when two independent research teams developed Evolutionary Strategies (Rechenberg 1965) and Evolutionary Programming (Fogel et al. 1966). Standard Evolutionary Strategies evolve a population of individuals using a selection procedure and a search operator mimicking genetic mutation, and they were originally developed to solve numerical optimisation problems. Evolutionary Programming traditionally used a representation tree to develop automata recognising strings in formal languages. However, it was only ten years later that EAs gained worldwide popularity following

the creation of Genetic Algorithms by (Holland 1992). The following will describe the different EAs.

Differential Evolution (DE) is another search strategy very similar to standard evolutionary algorithms.

2.1.1 Evolutionary Strategies (ESs)

Evolutionary Strategies are probably the first successful implementation of evolutionary search. The first experiments were conducted in 1964 by Rechenberg (Rechenberg 1965) at Technical University of Berlin, and were mainly aimed at solving hydrodynamic problems such as shape optimisation of a bent pipe or a supersonic flow nozzle (Rechenberg 1965). The algorithm was further developed and brought to its current form by (Schwefel 1981).

In ESs solutions in the n-dimensional search space of real parameters $\Re^n$ is represented by two n-dimensional real vectors, the parameters and their standard deviations. Often a third n-dimensional vector of rotation angles is added. Even if the analogy is not commonly used in ESs, chromosomes can be associated to vectors and genes to vector components.

Rechenberg's first algorithm was based on a population of one individual that was made to compete with one mutated offspring. The random mutation operator had usually a Gaussian magnitude distribution. The best solution survived to the next

generation. This kind of strategy is called *(1+1)-ES.* Subsequent implementations enlarged the population size to μ individuals competing with $\lambda(\lambda > \mu)$ offsprings for survival. An empirical law was devised indicating an optimal number of generated offsprings between five and six times μ (Bäck 1996). The improved strategy takes the name $(\mu + \lambda)$*-ES.*

In Schwefel's algorithm, the mutation step size of each gene (standard deviation) is encoded into the genotype and submitted to the evolutionary process. Moreover, to improve the robustness of the algorithm to noisy fitness evaluations, the lifespan of each individual has been reduced to one generation. This evolutionary scheme is called *generational* replacement (Davis 1991), and the corresponding ES formulation is called (μ, λ)*-ES*, μ individuals generate λ offspring out of which the best μ will be chosen as the new population.

Further improvements introduced a recombination operator modelled on biological crossover and mixing the genetic material of two randomly selected parents. The two most common procedures generate an offspring whose genetic information is randomly picked or averaged from the parents' genes. A rotation chromosome has also been used to bias the generation of new solutions in the search space along directions other than the co-ordinate axes.

2.1.2 Evolutionary Programming (EP)

L.J. Fogel, Owens and Walsh introduced the first EP in 1966 to train a finite-state machine to predict repeating cycles of output symbols (Fogel et al. 1966). μ offsprings were generated from a population of μ solutions by mutating each parent once. The prediction accuracy of each individual measured its adaptation and the survival probabilities were allocated consequently. A stochastic *tournament selection* procedure (Goldberg and Deb 1991) was used to select the new generation from the batch of parents and offspring.

In early EP algorithms, a candidate solution is represented by an n-dimensional vector defined in the $\Re^n$ search space, and the mutation operator modifies each individual by an amount proportional to the square root of its fitness. In the early nineties, D.B. Fogel (L.J. Fogel's son) further developed EP mainly focusing on the improvement of the mutation procedure. In a way similar to ESs, he introduced a second vector defining the mutation step size into the genotype of the solution. The mutation operator is therefore adaptively tuned along each dimension of the search space.

Genetic Programming (GP) was developed by Koza (Koza 1992) inspired by EP. GP uses a tree encoding similar to the original EP, but has been more commonly applied to evolve parse trees of equations or statements.

2.1.3 Genetic Algorithms (GAs)

The formulation of GAs was made about a decade after the first ESs and EPs applications. Nonetheless, as opposed to their predecessors, GAs almost immediately achieved considerable popularity placing the evolutionary approach at the forefront of research in optimisation. Such success can be partly attributed to a comprehensive theoretical background and increased availability of computing power.

The theoretical basis of GAs lies in the concept of *schema* (plural *schemata*) (Holland 1975). Schemata represent solution templates where each location can be defined or left unspecified. The larger the number of uninstantiated locations is, the greater the number of potential solutions that a schema represents. Schemata leading to higher fitness individuals are propagated through the generations and their number is increased as an effect of the selection process. The ability to process several possible solutions through a single schema is believed to determine the search power of GAs and is given the name *implicit parallelism* (Goldberg 1989; Grefenstette and Baker 1989). High fitness schemata whose uninstantiated locations occupy a short and compact portion of the encoding are considered to be the *building blocks* (Goldberg 1989) of the optimisation process. GAs are designed to multiply and differently recombine these building blocks in order to grow the final optimal solution (*building blocks hypothesis*) (Goldberg 1989). The *schemata theorem* (Holland 1975) allows the estimation in a probabilistic way of the number of criteria schema instances that are transmitted to the following generations.

Over more than 30 years of research, several modifications have been proposed to the original GA structure defined in (Holland 1975). Unless otherwise stated this subsection will explain Holland's original algorithm, often referred as the *canonical* GA.

GAs encode candidate solutions as binary strings. Each string (chromosome) is built by chaining a number of sub-strings, each sub-string representing one of the candidate solution's features. Biological genes are in this case equivalent to the sub-strings encoding the parameters, while each binary digit can be related to the nucleotides composing the DNA. In most of the cases, one individual is fully described by a single bit-string, thus leading to the identification of the genotype with one single chromosome. Several other encoding procedures have been explored leading to a debate on the most appropriate choice. Holland showed that binary coding allows the maximum number of schemata to be processed per individual (Holland 1975). On the other hand, the mapping to binary coding introduces Hamming cliffs onto the search surface. Moreover, non-binary representations may be more natural for some problem domains and may reduce the computational burden of the search. The canonical binary-coded GA as described here is now rarely used for continuous function optimisation as it has been shown that solutions are too easily disrupted (the Hamming cliff issue). Therefore researchers tend to use less disruptive coding such as Gray coding (Michalewicz 1996).

Similarly to the other EAs, canonical GAs use generational replacement. Popular alternatives are *elitism* and *steady-state* replacement (Davis 1991). In the first case, the best solution(s) are directly copied into the new population while in the second case only a fraction of the population is replaced at each generation. Both variants aim to improve the preservation of good genetic material at the expense of a reduced search space exploration. A comparison between the behaviour of generational and steady-state replacement is given in (Syswerda 1991).

Individuals are selected for reproduction with a probability depending on their fitness. Canonical GAs allocate the mating probability of each individual proportionally to its fitness (*proportional selection*) and draw the parents set (*mating pool*) through the *roulette wheel* selection procedure (Goldberg 1989). Other popular selection schemes are *fitness ranking* (Baker 1985) and tournament selection (Goldberg and Deb 1991). For a comparison of selection procedure, the reader is referred to (Goldberg and Deb 1991).

Crossover is the main search operator in GAs, creating offsprings by randomly mixing sections of the parental genome. The number of sections exchanged varies widely with the GA implementation. The most common crossover procedures are *one-point crossover*, *two-point crossover* and *uniform crossover* (Davis 1991). In canonical GAs, a crossover probability is set for each couple. Couples not selected for recombination will generate two offsprings identical to the parents.

A small fraction of the offsprings are randomly selected to undergo genetic mutation. The mutation operator randomly picks a location from a bit-string and flips its contents. The importance of this operator in GAs is however secondary, and to the main aim of mutation is the preservation of the genetic diversity of the population.

GAs require the tuning of some parameters such as the mutation rate, crossover rate and replacement rate in the case of steady-state replacement. This task is often not trivial as the chosen values may strongly influence the search process (Grefenstette 1986; Schaffer et al. 1989). Moreover, the optimal value for the GA parameters may vary according to the evolution of the search process. For all these reasons, several adaptive schemes have been investigated. A survey of adaptation in GAs is given in (Hinterding et al. 1997). (Bäck 1993) proposed an off-line tuning approach giving an optimal mutation rate schedule.

Problem-specific operators are sometimes employed in addition to the canonical ones. The introduction of such operators results an increase in the search power of the algorithm but a loss of general applicability. This issue is analysed in (Michalewicz 1993).

2.1.4 Differential Evolution (DE)

Differential Evolution (DE) is a population-based search strategy very similar to standard evolutionary algorithms (Price et al. 2005). The main difference is in the reproduction step where an offspring is created from three parents using an

arithmetic crossover operator. DE is defined for individuals represented by floating-point numbers.

DE does not make use of a mutation operator that depends on some probability distribution function, but introduces a new arithmetic operator which depends on the differences between randomly selected pairs of individuals (Price et al. 2005).

After completion of the reproduction process, the next step is to select the new generation. Each parent in the current population is replaced with its offspring if the fitness of the offspring is better, otherwise the parent is carried over to the next generation.

2.2 Ant Colony Optimisation

In the early 1990s, Ant Colony Optimisation (ACO) was introduced by M.Dorigo and colleagues as a novel nature-inspired metaheuristic for the solution of combinatorial optimisation problems (Dorigo et al. 1996). The inspiring source of ACO is the foraging behaviour of real ants. When searching for food, ants initially explore the area surrounding their nest in a random manner. When an ant finds a food source, it carries some of it back to the nest. During the return trip, the ant deposits a chemical pheromone trail on the ground. The quantity of pheromone deposited guides other ants to the food source (Dorigo and Stützle 2004). As shown by (Deneubourg et al. 1990), indirect communication between the ants via pheromone trails enables them to find the shortest paths between their nest and food

sources. The indirect communication mechanism where ants modify their environment to influence the behaviour of other ants is referred to as *stigmergy*. This characteristic of real ant colonies is exploited in artificial ant colonies in order to solve combinatorial and continuous optimisation problems.

Although an ant colony exhibits complex adaptive behaviour, a single ant exhibits a very simple behaviour. An ant can be seen as a stimulus-response agent (Nilsson 1998), the ant observes pheromone concentrations and produces an action based on the pheromone-stimulus. An ant can therefore abstractly be considered as a simple computational agent. An artificial ant algorithmically models the simple behaviour of real ants.

The simple ACO can be formulated as follows (Dorigo and Stützle 2004). Let us define a combinatorial optimisation problem that entails the minimisation of a given cost function. A candidate solution is defined as a sequence of parameters, and can be visualised as a path through several nodes, each node corresponding to one of the solution's parameters. The probability of moving from node i to node j is given in equation (2.1).

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum \tau_{iu}^{\alpha} \eta_{iu}^{\beta}} \tag{2.1}$$

Where τ_{ij} represent the *a posteriori* effectiveness of the move from node i to node j, as expressed in the pheromone intensity of the corresponding link, (i, j); η_{ij} represents the *a priori* effectiveness of the move from i to j (i.e. the attractiveness, or desirability, of the move), computed using some heuristic. The pheromone concentrations, τ_{ij}, indicate how profitable was in the past to make a move from i to j, serving as a memory of previous best moves. α is the parameter to control the influence of τ_{ij}, and β controls the influence of η_{ij}. Pheromone intensity on each link (i, j) is updated by ants using equation (2.2).

$$\tau_{ij} = \rho\tau_{ij} + \Delta\tau_{ij} \tag{2.2}$$

Where ρ is the rate of pheromone evaporation, and $\Delta\tau_{ij}$ is the amount of pheromone deposited. Pseudo code of the AS is shown in Figure 2.1.

```
1- Procedure ACO_MetaHeuristic.
2- While (stopping criterion not met)
3-      Generate solutions
4-      Pheromone update using equation (2.2)
5-      Daemon Action, move according probability calculated with equation (2.1)
6- End While.
7- End Procedure
```

Figure 2.1. Pseudo code of the simple Ant Colony Optimisation

Most of the early research in ACO has focused on the development of algorithmic variants that improve in performance over the simple ACO. The reader is referred to the following sources for further material on:

- Ant System (Bonabeau et al. 1999; Dorigo et al. 1996)
- Ant Colony Optimisation (Bell and McMullen 2004; Blum 2005; Dorigo and Blum 2005; Socha and Dorigo 2008)
- Elitist Ant System (Dorigo et al. 1996)
- Ant-Q (Gambardella and Dorigo 1995)
- Ant Colony System (Cheng and Mao 2007b; Dorigo and Gambardella 1997b; Ellabib et al. 2007)
- Max-Min Ant System (Pitakaso et al. 2007; Stützle and Hoos 2000)
- Rank Based Ant System (Bullnheimer et al. 1999)
- Hyper Cube – ACO (Blum and Dorigo 2004)

2.2.1 Ant Colony Optimisation applications

ACO algorithms can be applied to optimisation problems for which the following problem dependent aspects can be defined (Bonabeau et al. 1999; Dorigo et al. 1996):

1. A search space that can be described by graphical representation.

2. A feedback process to update pheromones.

3. To be able to determine desirability of different links in representation graph.

4. A method to construct feasible solutions.

ACO has been applied to variety of problems, some of them are listed in this section.

The Travelling Salesman Problem is a very well known combinatorial problem and is one of the first problems to which ACO algorithms were applied (Agarwal et al. 2005; Bontoux and Feillet 2008; Cheng and Mao 2007a; Dorigo and Gambardella 1997a; Duan and Xiufen 2007; Garcia-Martinez et al. 2007; Jun and Gui-Rong 2004; Pan and Wang 2006; Shang et al. 2007; Tsai et al. 2004; Xuemei et al. 2006).

ACO algorithms were used to solve Vehicle Routing Problems. The classic static Vehicle Routing Problem is defined with the objective of finding the minimum cost

vehicle route such that every customer is visited once only, and by only one vehicle. For every vehicle, the total demand should not exceed the capacity of the vehicle, the tour of each vehicle starts and ends at a unique place, and the total tour length should not be more than a predefined length (Chen et al. 2007b). The objective function can be designed so as to minimise the total travel time, minimise the total travel length, maximise customer satisfaction, minimise the number of needed vehicles, or as a multi-objective optimisation problem including a combination of the above requirements (Bell and McMullen 2004; Chen et al. 2006; Chen et al. 2007a; Chen et al. 2007b; Donati et al. 2008; Hu et al. 2006; Lin Wei and Cai Tian 2006; Liu and Cai 2005; Mazzeo and Loiseau 2004; Tao et al. 2006; Wang and Shen 2007; Xiaoxia and Lixin 2005; Xuan et al. 2005; Zhishuo and Yueting 2006).

Another application of ACO is the Job-Shop Scheduling Problem. In this problem, a set of n_M machines and n_J jobs are given, where each job consists of an ordered sequence of operations. The problem is to assign the operations to time intervals such that the maximum completion times of all operations is minimised, subject to the constraint that no two jobs are processed at the same time on the same machine. If there is only one machine, the problem is called Single-machine Job Scheduling problem. ACO has been used to solve Job-Shop Scheduling problems (Heinonen and Pettersson 2007; Jain and Sharma 2005; Liao and Juan 2007; Rossi and Dini 2007; Seckiner and Kurt 2008; Zhou et al. 2004).

There are many other applications of ACO, and the reader is referred to an overview article by (Blum 2005).

2.3 Particle Swarm Optimisation (PSO)

The Particle Swarm Optimisation algorithm was first proposed by Eberhart and Kennedy (Kennedy and Eberhart 1995), inspired by the natural flocking and swarming behaviour of birds and insects. The concept of PSO gained in popularity due to its simplicity. Like other swarm-based techniques, PSO consists of a number of individuals refining their knowledge of the given search space. The individuals in a PSO have a position and a velocity and are denoted as particles. The PSO traditionally has no crossover between individuals, has no mutation and particles are never substituted by other individuals during the run. The PSO algorithm works by attracting the particles to search space positions of high fitness. Each particle has a memory function, and adjusts its trajectory according to two pieces of information, the best position that it has so far visited, and the global best position attained by the whole swarm. If the whole swarm is considered as a society, the first piece of information can be seen as resulting from the particle's memory of its past states, and the second piece of information can be seen as resulting from the collective experience of all members of the society. Like other optimisation methods, PSO has a fitness evaluation function that takes each particle's position and assigns it a fitness value. The position of highest fitness value visited by the swarm is called the *global*

best. Each particle remembers the global best, and the position of highest fitness value that has personally visited, which is the called the *local best*.

Many attempts were made to improve the performance of the original PSO algorithm and several new parameters were introduced such as the inertia weight (Engelbrecht 2005). The canonical PSO with inertia weight has become very popular and widely used in many science and engineering problems (Brits et al. 2007; Liu et al. 2007; Pan et al. 2006; Yang 2007).

In the canonical PSO, each particle i has position $\vec{x_i}$ and velocity $\vec{v_i}$ that is updated at each iteration according to equation (2.3).

$$\vec{v_i} = w\vec{v_i} + c_1\vec{\varphi_{1i}}\left(\vec{p_i} - \vec{x_i}\right) + c_2\vec{\varphi_{2i}}\left(\vec{p_g} - \vec{x_i}\right) \tag{2.3}$$

Where w is the *inertia weight* described in (Shi and Eberhart 1998a; Shi and Eberhart 1998b), $\vec{p_i}$ is the best position found so far byparticle $\vec{p_i}$, and $\vec{p_g}$ is the global best so far found by the swarm. φ_1 and φ_2 are weights that are randomly generated at each step for each particle component. c_1 and c_2 are positive constant parameters called acceleration coefficients (which control the maximum step size the particle can achieve). The position of each particle is updated at each iteration by adding the velocity vector to the position vector.

$$\vec{x_i} = \vec{x_i} + \vec{v_i} \tag{2.4}$$

The inertia weight w (which is a user-defined parameter), together with c_1 and c_2, controls the contribution of past velocity values to the current velocity of the particle. A large inertia weight biases the search towards global exploration, while a smaller inertia weight directs toward fine-tuning the current solutions (exploitation). Suitable selection of the inertia weight and acceleration coefficients can provide a balance between the global and the local search (Engelbrecht 2005).

Figure 2.2 shows a flowchart of PSO. The PSO algorithm is composed of 5 main steps:

1. Initialise the position vector $\vec{x}$ and associated velocity $\vec{v}$ of all particles in the population randomly. Then set a maximum velocity and a maximum particle movement amplitude in order to decrease the cost of evaluation (WHY?) and to get a good convergence rate.

2. Evaluate the fitness of each particle via the fitness function. There are many options when choosing a fitness function and trial and error is often required to find a good one.

3. Compare the particle's fitness evaluation with the particle's best solution. If the current value is better than previous best solution, replace it and set the current solution as the local best. Compare the individual particle's fitness with the population's global best. If the fitness of the current solution is

better than the global best's fitness, set the current solution as the new global best.

4. Change velocities and positions by using equations (2.3) and (2.4).

5. Repeat step 2 to step 4 until a stopping criterion is satisfied or a predefined number of iterations is completed.

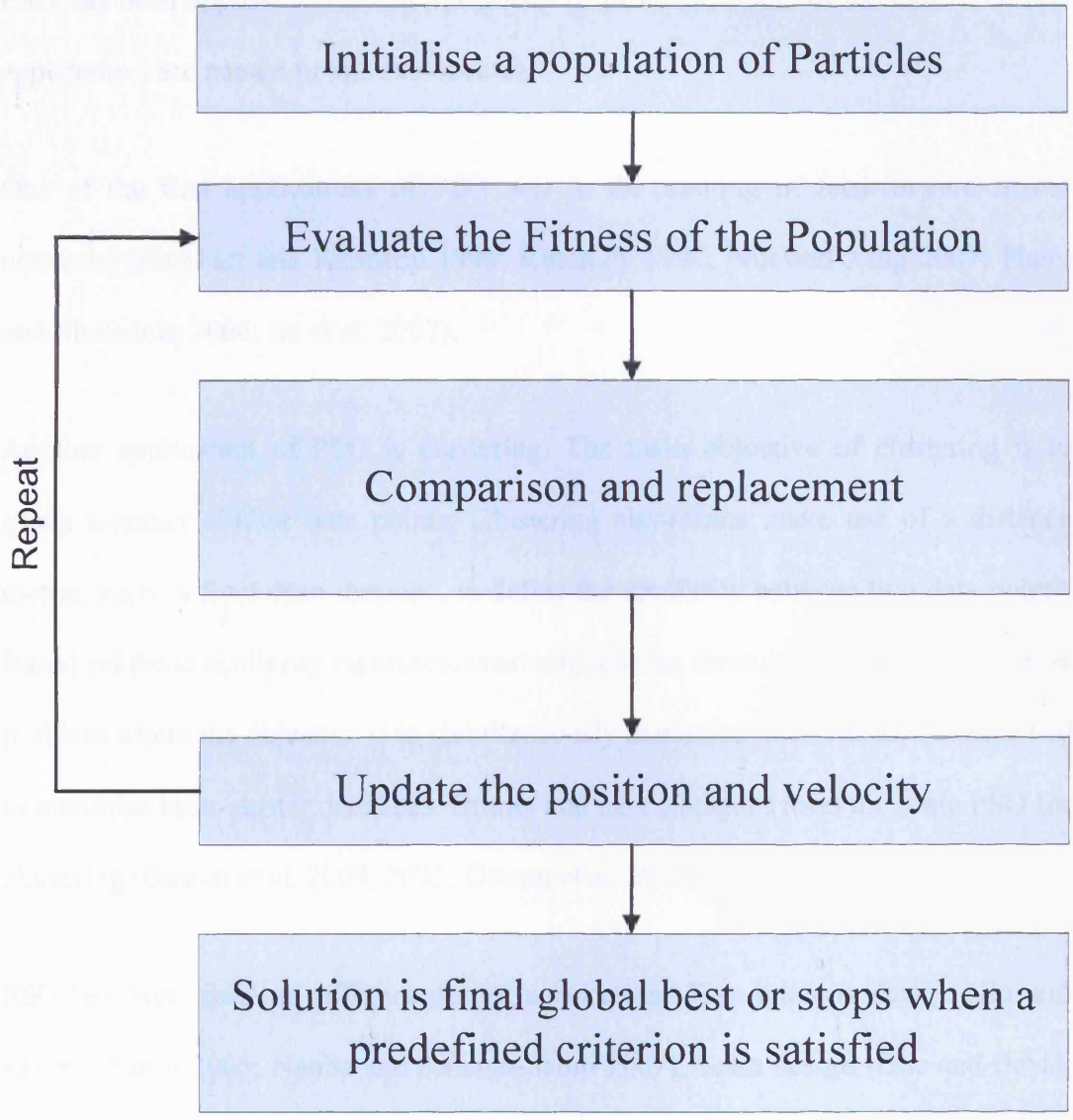

Figure 2.2. Flowchart of Particle Swarm Optimisation

2.3.1 Particle Swarm Optimisation applications

PSO has been applied to various categories of problem. Some of the most common applications are named in this sub-section.

One of the first applications of PSO was in the training of feed-forward neural networks (Eberhart and Kennedy 1995; Kennedy 1997; Niu and Xing 2007; Pham and Sholedolu 2006; Su et al. 2007).

Another application of PSO is clustering. The main objective of clustering is to group together similar data points. Clustering algorithms make use of a distance metric, such as Euclidean distance, to define the similarity between two data points. Based on these similarity measures, clustering can be formulated as an optimisation problem where the objective is to simultaneously maximise inter-cluster distance and to minimise intra-cluster distances. Omran and his colleagues used the basic PSO for clustering (Omran et al. 2004, 2005; Omran et al. 2002).

PSO has been used in different design applications like: antenna design (Jin and Rahmat-Samii 2005; Nanbo and Rahmat-Samii 2007), beam design (Eric and Babak 2006; Kathiravan and Ganguli 2007; Suresh et al. 2007), combinational circuits (Venayagamoorthy et al. 2007), structural design (Perez and Behdinan 2007).

2.4 Honey-bees inspired Algorithms

The swarming behaviour of honey-bees has been used in different applications – mostly in discrete space optimisation problems. BeeHive, BeeAdHoc and the Bee Algorithm (Tovey Spring 2004; Wedde et al. 2005; Wedde et al. 2004) are some of the algorithms inspired from bee swarming behaviour. A model mimicking the allocation of bees to different flower patches to maximise the nectar intake is described in (Tovey Spring 2004). This was subsequently applied to distribute web applications at hosting centres. Another model borrowing from the principles of bee communication was presented (Wedde et al. 2004). According to this model, artificial bee agents are used in packet switching networks to find suitable paths between nodes by updating the routing table. Two types of agents are used – short distance bee agents which disseminate routing information by travelling within a restricted number of hops, and long distance bee agents which travel to all nodes of the network. Even though the Wedde's algorithm is claimed to model honey-bees behaviour, it only loosely follows their natural behaviour.

Yang (Yang 2005) describes a virtual bee algorithm where the objective function is transformed into virtual food.

The foraging behaviour of honey-bees was used to solve a number of combinatorial problems (Teodorovic et al. 2003; Teodorovic and Dell'orco 2005; Tereshko 2000; Tereshko and Lee 2002).

Karaboga and his colleagues introduced the Artificial Bee Colony (ABC) algorithm as an optimisation tool (Karaboga and Akay 2007; Karaboga and Basturk 2008). In Karaboga's algorithm, a colony is divided to three groups; employed bees, onlookers and scouts. An onlooker waits in the dance area and decides to choose a food source. Bees visiting food sources are employed bees, while scout bees perform random search. In ABC half of the colony are onlooker bees and the other half are employed bees. Each employed bee becomes a scout bee when the food source is exhausted. The ABC steps are as follow:

1. Initialise.

2. Repeat

    - Move the employed bees onto their food sources in and determine their nectar amounts.

    - Move the onlookers onto their food sources in and determine their nectar amounts.

    - Move the scouts for searching new food sources.

    - Memorise the best food sources found so far.

3. Until (requirements are met)

In the ABC algorithm, each search cycle consists of three steps. In the first step, the employed bees are sent to the food sources and measure their nectar amount. In the second step, each onlooker bee selects a food source based on the quality feeback given by the employed bees. Each onlooker bee visits the selected food source and determines its nectar amount. In the third step, the scout bees are sent to randomly selected food sources.

The Bees Algorithm that is proposed in this thesis is different from Karaboga's algorithm, although both algorithms are inspired by the foraging behaviour of honey-bees. The proposed Bees Algorithm is described in the next chapter, where a comparison is also made on different benchmarks between the proposed Bees Algorithm and other optimisation methods.

CHAPTER 3

THE BEES ALGORITHM

3. THE BEES ALGORITHM

3.1 Preliminaries

An optimisation algorithm can be defined as a numerical method for finding a value x such that f(x) is as small (large) as possible, for a given function f, possibly with some constraints on x. Here, x can be a scalar or a vector of continuous or discrete values. If x is continuous, the algorithm can be seen as a numerical analysis method. However, classical optimisation methods encounter great difficulty when faced with the challenge of solving hard problems within an acceptable time and level of precision.

Many complex multi-variable optimisation problems cannot be solved exactly within polynomially bounded computation times. This generates much interest in search algorithms that find near-optimal solutions in reasonable running times. The swarm-based algorithm described in this chapter is a search algorithm capable of locating good solutions efficiently. The algorithm is inspired by the food foraging behaviour of honey-bees and could be regarded as belonging to the category of "intelligent" optimisation tools (Pham et al. 2005).

3.2 The basic Bees Algorithm

One of the most familiar insects in the world is the honey-bee. Honey-bees are a subset of the larger family of bees. Honey-bees live in social units called colonies. A

honey-bee colony consists of a single queen, who is usually the mother of all other colony members, a number ranging from zero to a few thousands of semi-sterile female workers and, from zero to a few thousand males (drones) depending on the time of year (Ribbands 1953). When honey-bees emerge from their cell as adults, they normally clean the cell, then as they age they feed larvae (nursing behaviour), process and store food, secrete wax, construct combs, and guard the entrance. The most pronounced change in behaviour occurs when honey-bees are about three weeks old when they begin foraging. At this time, they cease performing most of other tasks within the nest and usually remain a forager for the rest of their lives (Richards 1961). In this study, the foraging behaviour of honey-bees will be discussed, and the term "bee" is referred to "honey-bee".

3.2.1 The foraging process in nature

A colony of honey-bees can extend itself over long distances (more than 10 km) and in multiple directions simultaneously to exploit a large number of food sources (Seeley 1996; Von Frisch 1976). A colony prospers by deploying its foragers to fields that are rich of food sources. In principle, flower patches with plentiful amounts of nectar or pollen that can be easily collected should be visited by many bees, whereas patches with less nectar or pollen should receive less bees (Bonabeau et al. 1999; Camazine et al. 2003).

The foraging process begins in a colony by scout bees being sent to search for promising flower patches. Scout bees move randomly from one patch to another.

During the harvesting season, a colony continues its exploration, keeping a percentage of the population as scout bees.

When they return to the hive, those scout bees that found a patch of sufficient quality (measured as the level of some constituents, such as sugar content) deposit their nectar or pollen and go to the "dance floor" to perform a dance known as the "waggle dance" (Seeley 1996). This dance is the means to communicate to other bees three pieces of information regarding a flower patch: the direction in which it will be found, its distance from the hive, and its quality rating (or fitness) (Camazine et al. 2003; Von Frisch 1976). This information helps the bees watching the dance to find the flower patches. After the waggle dance, the dancer (i.e. the scout bee) goes back to the flower patch with follower bees recruited from the hive. The number of follower bees depends on the overall quality of the patch. Flower patches with large amounts of nectar or pollen that can be collected with less effort are regarded as more promising and attract more bees (Bonabeau et al. 1999; Seeley 1996). It this way, the colony gathers food quickly and efficiently.

During the harvesting season, a colony of bees keeps a percentage of its population as scouts (Von Frisch 1976) and uses them to explore the field surrounding the hive for promising flower patches. The foraging process begins with the scout bees being sent to the field where they move randomly from one patch to another.

3.2.2 The basic Bees Algorithm

The Bees Algorithm is an optimisation algorithm inspired by the natural foraging behaviour of honey bees to find the optimal solution. Figures 3.1 and 3.2 show the pseudo code and the flowchart for the basic Bees Algorithm.

The algorithm requires a number of parameters to be set, namely: the number of scout bees (n), the number of sites selected for neighbourhood search (out of n visited sites) (m), the number of top-rated (elite) sites among m selected sites (e), the number of bees recruited for the best e sites (nep), the number of bees recruited for the other (m-e) selected sites (nsp), the initial size of each patch (ngh) (a patch is a region in the search space that includes a visited site and its neighbourhood), and the stopping criterion. The algorithm starts with the n scout bees being placed randomly in the search space. The fitnesses of the sites visited by the scout bees are evaluated in step 2.

1- Initialise population with random solutions.

2- Evaluate fitness of the population.

3- While (stopping criterion not met)

    //Forming new population.

4- Select sites for neighbourhood search.

5- Recruit bees for selected sites (more bees for best e sites) and evaluate fitnesses.

7- Select the fittest bee from each patch.

7- Assign remaining bees to search randomly and evaluate their fitnesses.

8- End While.

Figure 3.1. Pseudo code of the basic Bees Algorithm

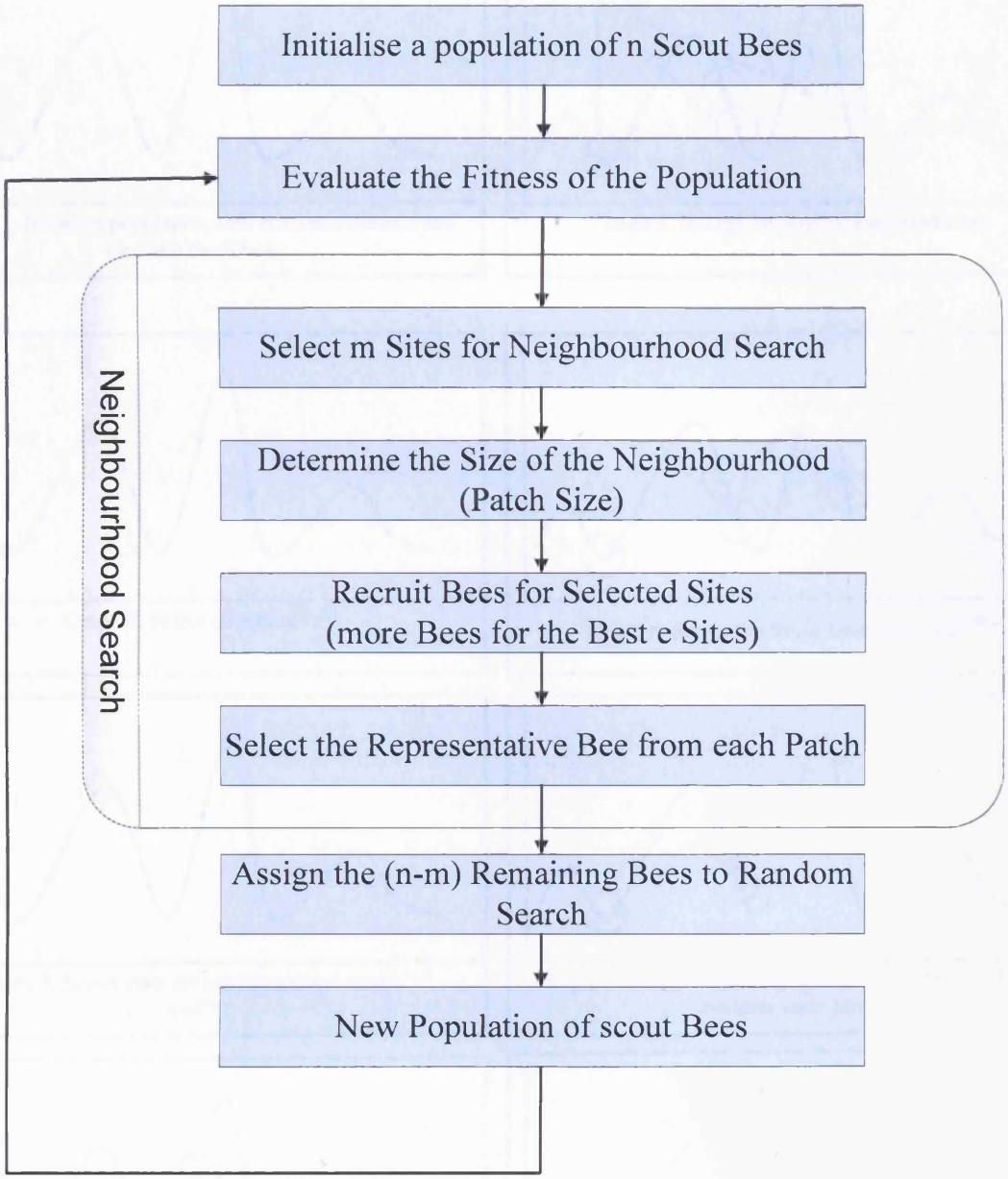

Figure 3.2. Flowchart of the basic Bees Algorithm

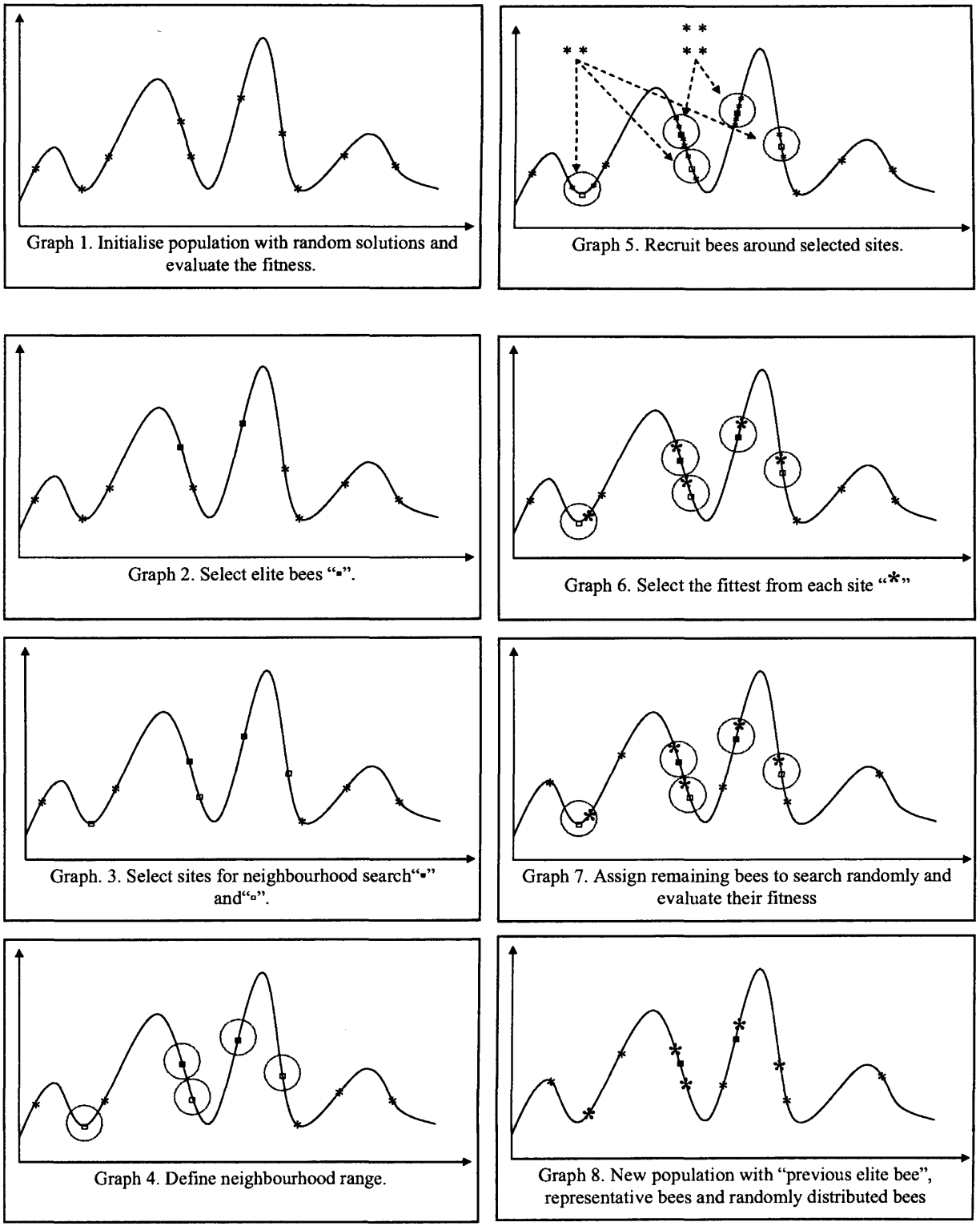

Figure 3.3. Graphical illustration of the Bees Algorithm.

In step 4, the m sites with the highest fitnesses are designated as "selected sites" and chosen for neighbourhood search. In steps 5 and 6, the algorithm conducts searches around the selected sites, assigning more bees to search in the vicinity of the best e sites. Selection of the best sites is made according to their associated fitness. Alternatively, the fitness values are used to determine the selection probability of the sites. Searches in the neighbourhood of the best e sites – those which represent the most promising solutions - are made more detailed. As already mentioned, this is done by recruiting more bees for the best e sites than for the other selected sites. Together with scouting, this differential recruitment is a key operation of the Bees Algorithm.

In step 6, for each patch, only the bee that has found the site with the highest fitness (the "fittest" bee in the patch) will be selected to form part of the next bee population. In nature, there is no such a restriction. This restriction is introduced here to reduce the number of points to be explored. In step 7, the remaining bees in the population are assigned randomly around the search space to scout for new potential solutions.

At the end of each iteration, the new population of a colony is composed of two parts: the representatives from the selected patches, and the scout bees assigned to conduct random search. These steps are repeated until a stopping criterion is met.

Figure 3.3 illustrates the Bees Algorithm.

In the basic Bees Algorithm, a greedy selection procedure is applied. Stochastic selection (roulette wheel selection) was also studied. However, in the test functions investigated, it was found that greedy selection gives better results than stochastic selection, and for this reason it was chosen as the selection procedure for the Bees Algorithm (Pham et al. 2005). In special cases where stochastic selection might be found to give the best results, the roulette wheel procedure can substitute the standard greedy procedure. . Similarly, a fixed number of bees are recruited for local search around each selected site (more for e best). Other ways of determining the number of bees recruited for each selected site can be considered. Alternative recruitment methods like recruitment proportional to the site fitness, or probabilistical recruitment, are explained in (Pham et al. 2005).

Clearly, the Bees Algorithm as described above is applicable to combinatorial, continuous, and discrete optimisation problems. In this thesis, continuous and discrete optimisation problems are used to investigate the Bees Algorithm. The solution of combinatorial optimisation problems differs only in the way neighbourhoods are defined.

3.3 Experiments using the basic Bees Algorithm

Two standard functional optimisation problems were used to test the basic Bees Algorithm and establish the correct values of its parameters. Other eight problems were used for benchmarking the algorithm. As the Bees Algorithm searches for the

maximum, functions to be minimised were inverted before the algorithm was applied.

Shekel's Foxholes (Figure 3.4), a 2D function from De Jong's test suite, was chosen as the first function for testing the algorithm.

$$f(\vec{x}) = 119.998 - \sum_{i=1}^{25} \frac{1}{\mathrm{j} + \sum_{i=1}^{2} \left(x_i - a_{ij}\right)^6} \tag{3.1}$$

$$a_{ij} = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & \ldots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & \ldots & 32 & 32 & 32 \end{pmatrix}$$

$$-65.536 \le x_i \le 65.536$$

For this function,

$$\vec{x}_{\max} = (-32, -32)$$

$$f(\vec{x}_{\max}) = 119.998$$

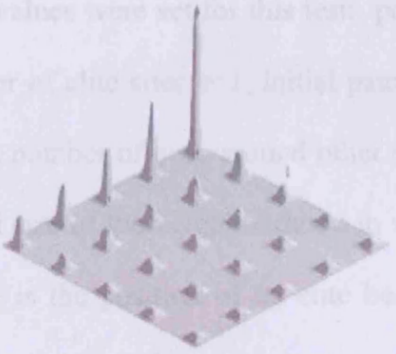

Figure 3.4. Inverted Shekel's Foxholes (Pham et al. 2005)

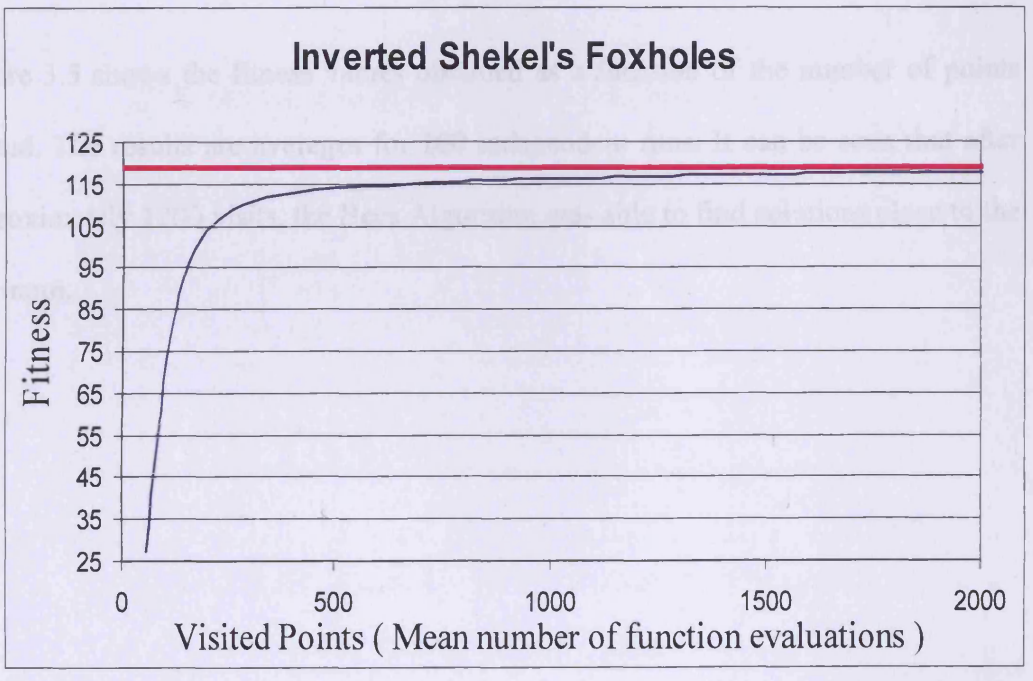

Figure 3.5. Evolution of fitness with the number of points visited (Inverted Shekel's Foxholes)

The following parameter values were set for this test: population n= 45, number of selected sites m=3, number of elite sites e=1, initial patch size ngh=3, number bees around elite points nep=7, number of bees around other selected points nsp=2. Note that ngh defines the initial size of the neighbourhood in which the follower bees are placed. For example, if x is the position of an elite bee in the ith dimension, the follower bees is placed randomly in the interval $x_{ie} \pm$ ngh in that dimension at the beginning of the optimisation process. As optimisation proceeds, the size of the neighbourhood search is gradually decreased to facilitate the fine tuning of the solution.

Figure 3.5 shows the fitness values obtained as a function of the number of points visited. The results are averages for 100 independent runs. It can be seen that after approximately 1200 visits, the Bees Algorithm was able to find solutions close to the optimum.

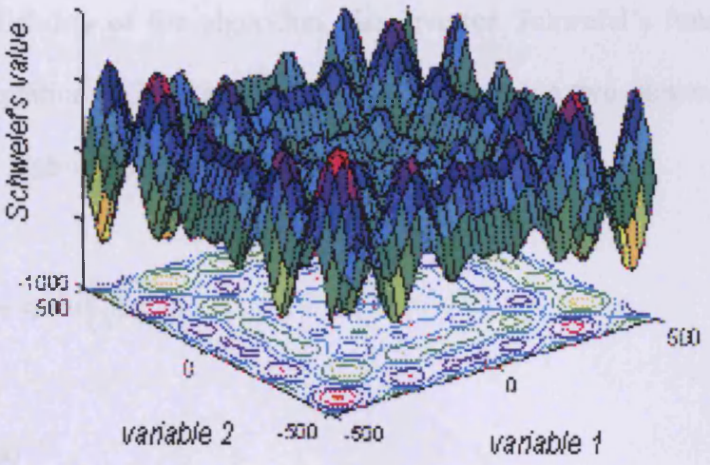

Figure 3.6 2D Schwefel's function (Pham et al. 2005)

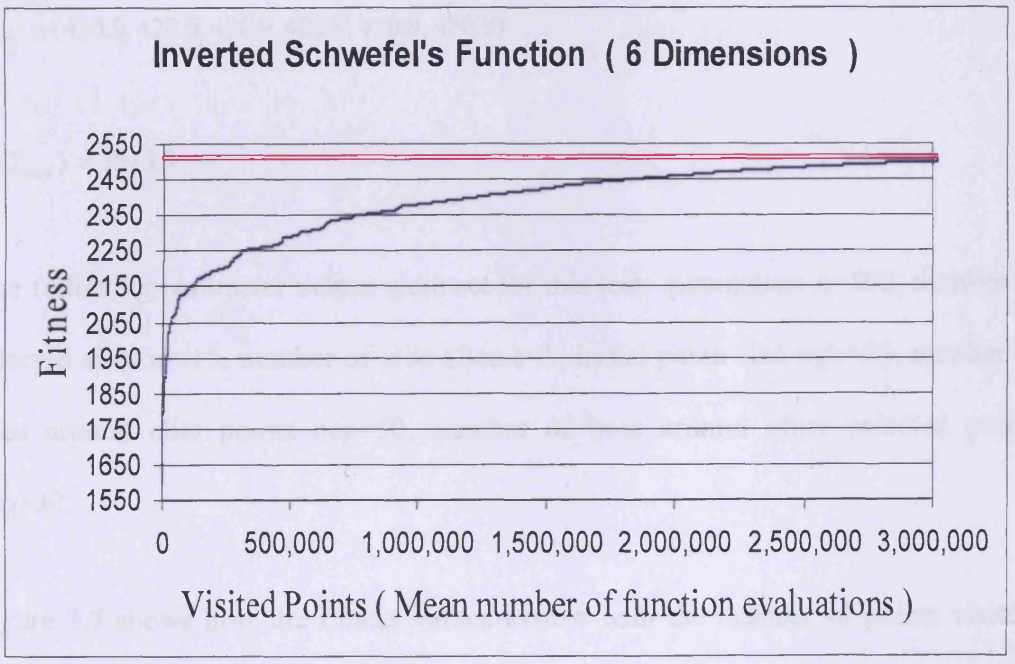

Figure 3.7 Evolution of fitness with the number of points visited (Inverted Schwefel's Function)

To test the reliability of the algorithm, the inverted Schwefel's function with six dimensions (equation (3.2)) was used. Figure 3.6 shows a two-dimensional view of the function to highlight its multi modality.

$$f(\vec{x}) = -\sum_{i=1}^{6} -x_i \sin\left(\sqrt{|x_i|}\right) \tag{3.2}$$

$$-500 \le x_i \le 500$$

For this function:

$$\vec{x}_{max} = (420.9, 420.9, 420.9, 420.9, 420.9, 420.9)$$

$$f(\vec{x}_{max}) \approx 2513.9$$

The following parameter values were set for this test: population n=500, number of selected sites m=15, number of elite sites e=5, initial patch size ngh=20, number of bees around elite points nep=50, number of bees around other selected points nsp=30.

Figure 3.7 shows how the fitness values evolve with the number of points visited. The results are averages for 100 independent runs. It can be seen that after approximately 3,000,000 visits, the Bees Algorithm was able to find solutions close to the optimum.

The Bees Algorithm was applied to the eight benchmark functions given in (Mathur et al. 2000) and the results compared with those obtained using other optimisation algorithms. The test functions and their optima are shown in Table 3.1.

Table 3.2 presents the results obtained by the Bees Algorithm and those by the deterministic Simplex method (SIMPSA) (Mathur et al. 2000), the stochastic simulated annealing optimisation procedure (NE SIMPSA) (Mathur et al. 2000), the Genetic Algorithm (GA) (Mathur et al. 2000) and the Ant Colony System (ANTS) (Mathur et al. 2000).

Again, the numbers of points visited shown are averages for 100 independent runs. Table 3.3 shows the empirically derived Bees Algorithm parameter values used with the different test functions (Pham et al. 2006).

No	Function Name	Interval	Function	Global Optimum
1	De Jong	[-2.048, 2.048]	$\max F = (3905.93) - 100(x_1^2 - x_2)^2 - (1 - x_1)^2$	X(1,1) F=3905.93
2	Goldstein & Price	[-2, 2]	$\min F = [1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$ $X[30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	X(0,-1) F=3
3	Branin	[-5, 10]	$\min F = a(x_2 - bx_1^2 + cx_1 - d)^2 + e(1-f)\cos(x_1) + e$ $a = 1, b = \frac{5.1}{4}\left(\frac{7}{22}\right)^2, c = \frac{5}{22}X7, d = 6, e = 10, f = \frac{1}{8}X\frac{7}{22}$	X(-22/7,12.275) X(22/7,2.275) X(66/7,2.475) F=0.3977272
4	Martin & Gaddy	[0, 10]	$\min F = (x_1 - x_2)^2 + ((x_1 + x_2 - 10)/3)^2$	X(5,5) F=0
5	Rosenbrock	(a)[-1.2, 1.2] (b)[-10, 10]	$\min \ F = 100\ (x_1^2 - x_2)^2 + (1 - x_1)^2$	X(1,1) F=0
6	Rosenbrock	[-1.2, 1.2]	$\min F = \sum_{i=1}^{3} \{100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2\}$	X(1,1,1,1) F=0
7	Hyper sphere	[-5.12, 5.12]	$\min F = \sum_{i=1}^{6} x_i^2$	X(0,0,0,0,0,0) F=0
8	Griewangk	[-512, 512]	$\max \ F = \frac{1}{0.1 + \left(\sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1\right)}$	X(0,0,0,0,0,0,0,0,0,0) F=10

Table 3.1 Test Functions (Mathur et al. 2000)

	SIMPSA		NE SIMPSA		GA		ANTS		Bees Algorithm	
func no	success %	mean no. of evaluations	success %	mean no. of evaluations	success%	mean no. of evaluations	success%	mean no. of evaluations	Success %	mean no. of evaluations
1	****	****	****	****	100	10160	100	6000	100	**868**
2	****	****	****	****	100	5662	100	5330	100	**999**
3	****	****	****	****	100	7325	100	1936	100	**1657**
4	****	****	****	****	100	2844	100	1688	100	**526**
5a	100	10780	100	4508	100	10212	100	6842	100	**631**
5b	100	12500	100	5007	****	****	100	7505	100	**2306**
6	99	21177	94	3053	****	****	100	8471	100	**28529**
7	****	****	****	****	100	15468	100	22050	100	**7113**
8	****	****	****	****	100	200000	100	50000	100	**20998**

**** Data not available

Table 3.2 Results for test functions

fun no	n	m	e	nsp	nep	ngh (initial)
1	10	3	1	2	4	0.1
2	20	3	1	1	13	0.1
3	30	5	1	2	3	0.5
4	20	3	1	1	10	0.5
5a	10	3	1	2	4	0.1
5b	6	3	1	1	4	0.5
6	20	6	1	5	8	0.1
7	8	3	1	1	2	0.3
8	50	5	2	10	20	5

Table 3.3 The Bees Algorithm parameters

The optimisation stopped when the difference between the maximum fitness obtained and the global optimum was less than 0.1% of the optimum value, or less than 0.001, whichever was smaller. In case the optimum was 0, the solution was accepted if it differed from the optimum by less than 0.001. If a solution is found that satisfies one of these conditions, the algorithm is said to have succeeded in locality the optimum.

The first test function was De Jong's, for which the Bees Algorithm could find the optimum 7 times faster than ANTS and 11 times faster than GA, with a success rate of 100%. The second function was Goldstein and Price's, for which the Bees Algorithm reached the optimum almost 5 times faster than ANTS and GA, again with 100% success. With Branin's function, there was a 15% improvement compared with ANTS and 77% improvement compared with GA, also with 100% success.

Functions 5 and 6 were Rosenbrock's functions in two and four dimensions respectively. In the two-dimensional function, the Bees Algorithm delivers 100% success and good improvement over the other methods (at least twice fewer evaluations than the other methods). In the four-dimensional case, the Bees Algorithm needed more function evaluations to reach the optimum with 100% success. NE SIMPSA could find the optimum with 10 times fewer function evaluations but the success rate was only 94% and ANTS found the optimum with 100% success and 3.5 times faster than the Bees Algorithm. Test function 7 was a

Hyper Sphere model of six dimensions. The Bees Algorithm needed half of the number of function evaluations compared with GA and one third of that required for ANTS. The eighth test function (Griewangk test function) was a ten-dimensional function. The Bees Algorithm could reach the optimum faster than GA and ANTS and its success rate was 100% (Pham et al. 2006).

3.4 Mechanical design optimisation using the basic Bees Algorithm

This section describes the application of the Bees Algorithm to mechanical design optimisation.

Researchers have used the design of welded beam structures (Rekliatis et al. 1983) and coil springs (Arora 2004) as benchmarks to test their optimisation algorithms. The welded beam design problem involves a nonlinear objective function and eight constraints, and the coil spring design problem, a nonlinear objective function and four constraints. A number of optimisation techniques have been applied to these two problems. Some of them, such as geometric programming (Ragsdell and Phillips 1976), require extensive problem formulation; some (see, for example, (Leite and Topping 1998)) use specific domain knowledge which may not be available for other problems, and others (Ragsdell and Phillips 1976) are computationally expensive or give poor results.

The Bees Algorithm has been applied to different unconstrained function optimisation in the previous section.

As described, the Bees Algorithm is suitable for unconstrained optimisation problems. If a problem involves constraints, a simple technique can be adopted to enable the optimisation to be applied. The technique involves subtracting a large number from the fitness of a particular solution that has violated a constraint in order drastically to reduce the chance of that solution being found acceptable. This was the technique adopted in this work. As both design problems were minimisation problems, a fixed penalty was added to the cost of any constraint-violating potential solution.

3.4.1 Welded beam design problem

A uniform beam of rectangular cross section needs to be welded to a base to be able to carry a load of 6000 *lbf*. The configuration is shown in Figure 3.8. The beam is made of steel 1010.

The length L is specified as 14 *in.*. The objective of the design is to minimise the cost of fabrication while finding a feasible combination of weld thickness h, weld length l, beam thickness t and beam width b. The objective function can be formulated as (Rekliatis et al. 1983) :

$$\min f = (1+c_1)h^2 l + c_2 tb(L+l) \tag{3.3}$$

where

f = Cost function including setup cost, welding labour cost and material cost;

c_1 = Unit volume of weld material cost = 0.10471 $\$/in.^3$;

c_2 = Unit volume of bar stock cost = 0.04811 $\$/in.^3$;

L = Fixed distance from load to support = 14 *in.*;

Not all combinations of h, l, t and b which can support F are acceptable. There are limitations which should be considered regarding the mechanical properties of the weld and bar, for example, shear and normal stresses, physical constraints (no length less than zero) and maximum deflection. The constraints are as follows (Rekliatis et al. 1983):

$$g_1 = \tau_d - \tau \geq 0 \tag{3.4}$$

$$g_2 = \sigma_d - \sigma \geq 0 \tag{3.5}$$

$$g_3 = b - h \geq 0 \tag{3.6}$$

$$g_4 = l \geq 0 \tag{3.7}$$

$$g_5 = t \geq 0 \tag{3.8}$$

$$g_6 = P_c - F \geq 0 \tag{3.9}$$

$$g_7 = h - 0.125 \geq 0 \tag{3.10}$$

$$g_8 = 0.25 - \delta \geq 0 \tag{3.11}$$

where

τ_d = Allowable shear stress of weld = 13600 Psi;

τ = Maximum shear stress in weld;

σ_d = Allowable normal stress for beam material = 30000 Psi;

σ = Maximum normal stress in beam;

P_c = Bar buckling load;

F = Load = 6000 lbf;

δ = Beam end deflection.

The first constraint (g_1) ensures that the maximum developed shear stress is less than the allowable shear stress of the weld material. The second constraint (g_2) checks that the maximum developed normal stress is lower than the allowed normal stress in the beam. The third constraint (g_3) ensures that the beam thickness exceeds that of the weld. The fourth and fifth constraints (g_4 and g_5) are practical checks to

prevent negative lengths or thicknesses. The sixth constraint (g_6) makes sure that the load on the beam is not greater than the allowable buckling load. The seventh constraint (g_7) checks that the weld thickness is above a given minimum, and the last constraint (g_8) is to ensure that the end deflection of the beam is less than a predefined amount.

Normal and shear stresses and buckling force can be formulated as (Rekliatis et al. 1983; Shigley 1973):

$$\sigma = \frac{2.1952}{t^3 b} \tag{3.12}$$

$$\tau = \sqrt{(\tau')^2 + (\tau'')^2 + (l\tau'\tau'')/\sqrt{0.25(l^2 + (h+t)^2)}} \tag{3.13}$$

where

$\tau' = \frac{6000}{\sqrt{2}hl}$ (Primary stress)

$$\tau'' = \frac{6000(14 + 0.5l)\sqrt{0.25(l^2 + (h+t)^2)}}{2\left\{0.707hl\left(l^2/12 + 0.25(h+t)^2\right)\right\}} \quad \text{(Secondary stress)} \tag{3.14}$$

$$P_c = 64746.022(1 - 0.0282346t)tb^3 \tag{3.15}$$

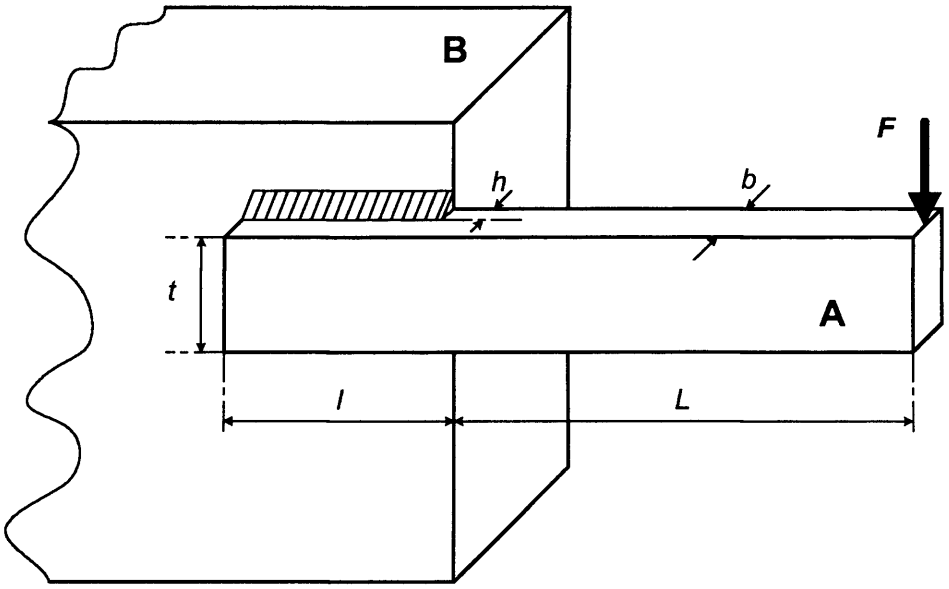

Figure 3.8 A welded beam

Bees Algorithm parameters	Symbol	Value
Population	n	80
Number of selected sites	m	5
Number of top-rated sites out of m selected sites	e	2
Initial patch size	ngh	0.1
Number of bees recruited for best e sites	nep	50
Number of bees recruited for the other (m-e) selected sites	nsp	10

Table 3.4 Parameters of the Bees Algorithm for the welded beam design problem

As there are no specific rules for selecting parameters in the Bees Algorithm, the user needs to try different sets of parameters until acceptable results are obtained. The empirically chosen parameters for the Bees Algorithm are given in Table 3.4 with the stopping criterion of 750 generations. The search space was defined by the following intervals (Deb 1991):

$$0.125 \leq h \leq 5 \tag{3.16}$$

$$0.1 \leq l \leq 10 \tag{3.17}$$

$$0.1 \leq t \leq 10 \tag{3.18}$$

$$0.1 \leq b \leq 5 \tag{3.19}$$

With the above search space definition, constraints g_4, g_5 and g_7 are already satisfied and do not need to be checked in the code.

Figure 3.9 shows how the lowest value of the objective function changes with the number of iterations (generations) for three independent runs of the algorithm. It can be seen that the objective function decreases rapidly in the early iterations and then gradually converges to the optimum value.

A variety of optimisation methods have been applied to this problem by other researchers (Deb 1991; Leite and Topping 1998; Ragsdell and Phillips 1976). The

results they obtained along with those of the Bees Algorithm are given in Table 3.4. APPROX is a method of successive linear approximation (Siddall 1972). DAVID is a gradient method with a penalty (Siddall 1972). Geometric Programming (GP) is a method capable of solving linear and nonlinear optimisation problems that are formulated analytically (Ragsdell and Phillips 1976). SIMPLEX is the Simplex algorithm for solving linear programming problems (Siddall 1972).

As shown in Table 3.5, the Bees Algorithm produces better results than almost all the examined algorithms including the Genetic Algorithm (GA) (Deb 1991), an improved version of the GA (Leite and Topping 1998), SIMPLEX (Ragsdell and Phillips 1976) and the random search procedure RANDOM (Ragsdell and Phillips 1976). Only APPROX and DAVID (Ragsdell and Phillips 1976) produce results that match those of the Bees Algorithm. However, as these two algorithms require information specifically derived from the problem (Leite and Topping 1998), their application is limited. The result for GP is close to that of the Bees Algorithm but GP needs a very complex formulation (Ragsdell and Phillips 1976). In this experiment, the number of function evolutions in the Bees Algorithm was as same in the population-based algorithms were used to compare the results. Of course the number of function evaluations is not applicable for analytical methods. For this optimisation problem, in order to compare the results of the Bees Algorithm and those of the GA and improved GA, the maximum number of function evaluations was set to be the same as those previously adopted for the latter.

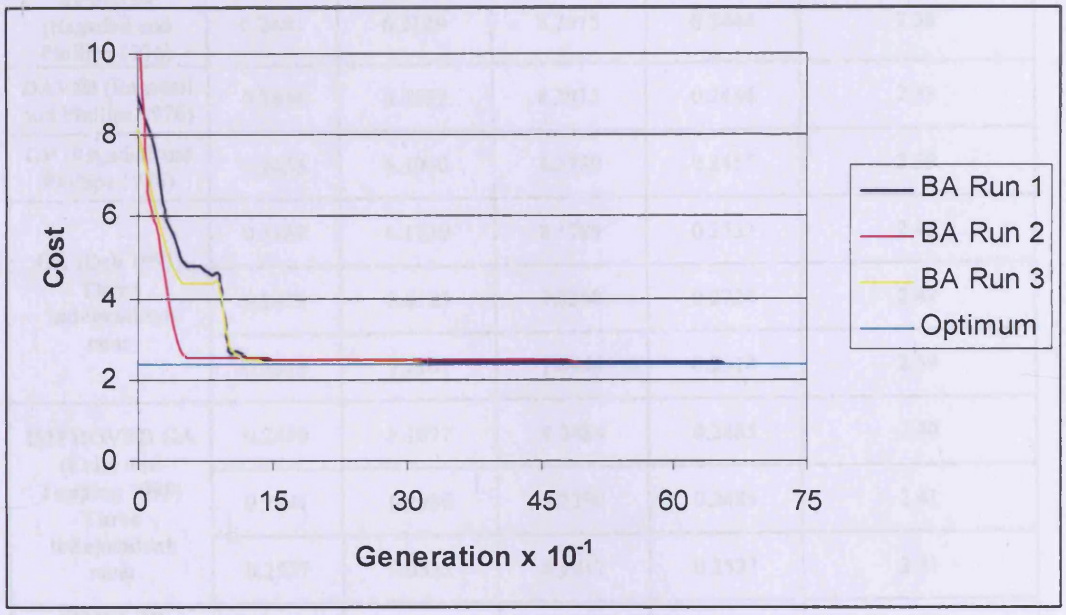

Figure 3.9 Evolution of the lowest cost in each iteration

Methods	Design variables				Cost
	h	*l*	*t*	*b*	
APPROX (Ragsdell and Phillips 1976)	0.2444	6.2189	8.2915	0.2444	2.38
DAVID (Ragsdell and Phillips 1976)	0.2434	6.2552	8.2915	0.2444	2.38
GP (Ragsdell and Phillips 1976)	0.2455	6.1960	8.2730	0.2455	2.39
GA (Deb 1991) **Three independent runs**	0.2489	6.1730	8.1789	0.2533	2.43
	0.2679	5.8123	7.8358	0.2724	2.49
	0.2918	5.2141	7.8446	0.2918	2.59
IMPROVED GA (Leite and Topping 1998) Three independent runs	0.2489	6.1097	8.2484	0.2485	2.40
	0.2441	6.2936	8.2290	0.2485	2.41
	0.2537	6.0322	8.1517	0.2533	2.41
SIMPLEX (Ragsdell and Phillips 1976)	0.2792	5.6256	7.7512	0.2796	2.53
RANDOM (Ragsdell and Phillips 1976)	0.4575	4.7313	5.0853	0.6600	4.12
BEES ALGORITHM Three independent runs	0.24429	6.2126	8.3009	0.24432	2.3817
	0.24428	6.2110	8.3026	0.24429	2.3816
	0.24432	6.2152	8.2966	0.24435	2.3815

Table 3.5 Results for the welded beam design problem obtained using the Bees Algorithm and other optimisation methods

3.4.2 Coil spring design problem

The problem is to design a coil spring to carry a specific axial load. Figure 3.10 shows a coil spring in tension. The parameters which should be optimised are the wire diameter d, the mean coil diameter D, and the number active coils N. The objective function is the mass (M) of the spring which should be minimised (Arora 2004).

$$\min M = \frac{1}{4}(N+Q)\pi^2 D d^2 \rho \tag{3.20}$$

where

Q = Number of inactive coils (i.e. end coils performing no energy storage function) = 2;

g = Gravitational constant = 386 $in./s^2$;

γ = Weight density of spring material = 0.285 $lbf/in.^3$;

ρ = Mass density of material (γ / g) = $7.38342\times10^{-4}\ lbf-s^2/in.^4$;

The constraints can be formulated as (Arora 2004) :

$$g_9 = \Delta - \frac{8PD^3N}{d^4G} \le 0 \tag{3.21}$$

$$g_{10} = \frac{8PD}{\pi d^3}\left[\frac{(4D-d)}{4(D-d)} + \frac{0.615d}{D}\right] - \tau_d \leq 0 \tag{3.22}$$

$$g_{11} = \omega_0 - \omega \leq 0 \tag{3.23}$$

$$g_{12} = D + d - D_0 \leq 0 \tag{3.24}$$

In this problem,

P = Applied axial load $= 10\ lbf$;

G = Shear modulus $= 1.15 \times 10^7\ lbf/in.^2$;

Δ = Minimum spring deflection $= 0.5\ in.$;

τ_d = Allowable shear stress $= 80000\ lbf/in.^2$;

ω_0 = Lower limit on surge wave frequency $= 100 Hz$;

ω = Frequency of surge waves $= \frac{d}{2\pi N D^2}\sqrt{\frac{G}{2\rho}}$;

D_0 = Limit on outer diameter of the coil $= 1.5\ in.$;

Using these values, the above constraints can be rewritten as:

$$g_9 = 1.0 - \frac{D^3 N}{71875 d^4} \leq 0 \tag{3.25}$$

$$g_{10} = \frac{D(4D-d)}{12566 d^3 (D-d)} + \frac{2.46}{12566 d^2} - 1.0 \leq 0 \tag{3.26}$$

$$g_{11} = 1.0 - \frac{140.54 d}{D^2 N} \leq 0 \tag{3.27}$$

$$g_{12} = \frac{D+d}{1.5} - 1.0 \leq 0 \tag{3.28}$$

The first constraint (g_9) makes sure that the deflection of the coil spring is greater than the specified minimum value. The second constraint (g_{10}) checks that the maximum shear stress in the coil spring is less than the allowable shear stress. The third condition (g_{11}) checks that the frequency of surge waves is greater than the given lower limit. Finally, the fourth constraint (g_{12}) controls the outer diameter of the spring.

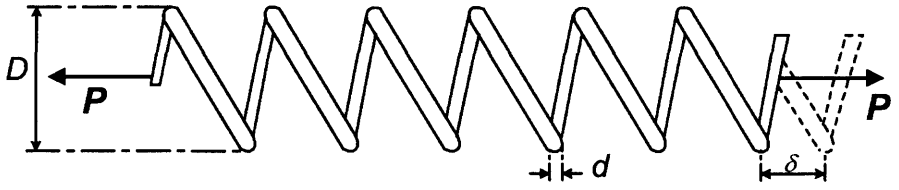

Figure 3.10 A coil spring

Bees Algorithm parameters	Symbol	Value
Population	*n*	60
Number of selected sites	*m*	5
Number of top-rated sites out of m selected sites	*e*	2
Initial patch size	*ngh*	0.1
Number of bees recruited for best *e* sites	*nep*	40
Number of bees recruited for the other (*m*-*e*) selected sites	*nsp*	10

Table 3.6 Parameters of the Bees Algorithm for the coil spring design problem

The parameters used for the Bees Algorithm are given in Table 3.6 with the stopping criterion of 1500 generations. The search space was defined using the following intervals (Leite and Topping 1998):

$$0.05 \leq d \leq 0.2 \tag{3.29}$$

$$0.25 \leq D \leq 1.3 \tag{3.30}$$

$$2 \leq N \leq 15 \tag{3.31}$$

Figure 3.11 shows the evolution of the best value of the objective function with the number of iteration (generations) for three independent runs. Again, it can be seen that the objective function decreases rapidly in the early iterations and then gradually converges to the optimum value.

The coil spring design problem has been solved by other researchers using Sequential Quadratic Programming (SQP) methods in a batch environment and in an interactive mode (Arora 2004) and using an improved Genetic Algorithm (Leite and Topping 1998). The results obtained by those optimisation tools are given in Table 3.7 together with the outputs of three independent runs of the Bees Algorithm. For this optimisation problem, in order to compare the results of the Bees Algorithm and those of improved GA, the maximum number of function evaluations was set to be the same as those previously adopted for the latter.

It can be seen that the Bees Algorithm gives better solutions than the improved GA and the interactive solution process. Only the result from the batch–mode SQP is comparable with that of the Bees Algorithm. However, as SQP methods need information on derivatives of variables, the range of problems that can be solved by these methods is limited.

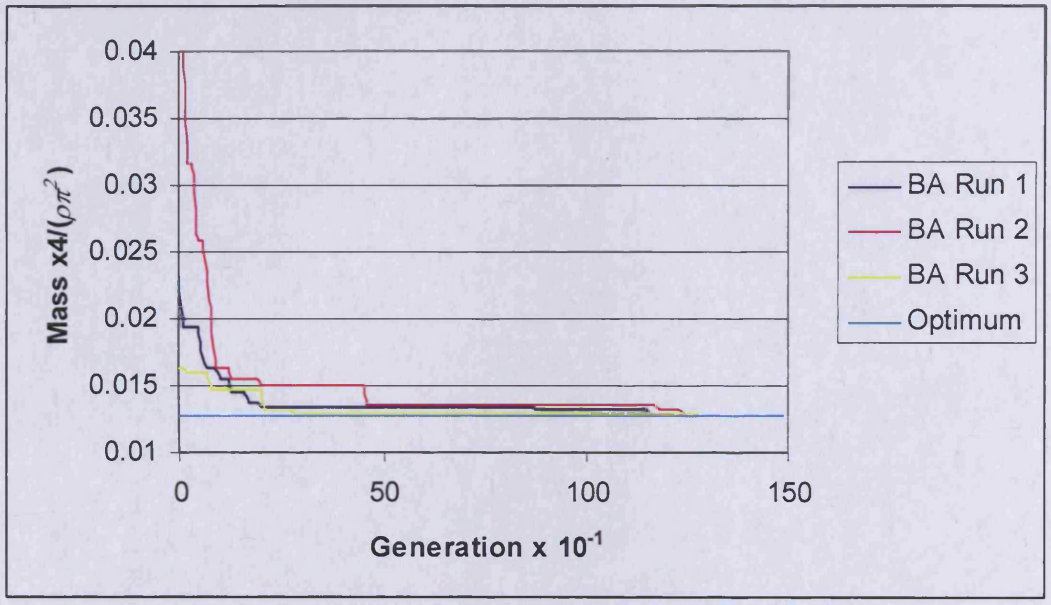

Figure 3.11 Evolution of the lowest mass in each iteration

Methods	Design variables			Mass M $\left(\times 4/\rho\pi^2\right)$
	d	D	N	
SQP (batch) (Arora 2004)	0.051699	0.35695	11.289	0.0126787
SQP (interactive) (Arora 2004)	0.05340	0.3992	9.1854	0.0127300
IMPROVED GA (Leite and Topping 1998) **Best three solutions not violating constraints**	0.05235	0.3721	10.48	0.01272
	0.05323	0.3947	9.383	0.01273
	0.05396	0.4132	8.697	0.01287
BEES ALGORITHM Three independent runs	0.051759	0.35839	11.207	0.012680
	0.051807	0.35956	11.139	0.012680
	0.051779	0.35886	11.179	0.012681

Table 3.7 Results for the coil spring design problem obtained using the Bees Algorithm and other optimisation methods

3.5 Improved version of the Bees Algorithm

In the basic version of the Bees Algorithm, there is no interaction between bees and only selected bees recruit from other bees. Other bees (those who have not been selected) are discarded in each iteration and will be replaced with new random placed bees. The first modification in the basic Bees Algorithm is not to discard the unselected bees and try to improve them by mating them with the selected ones. The mating method used here is interpolation or extrapolation. With this mating method, each unselected bee in the last stage of the algorithm chooses one of the selected bees and randomly carries out interpolation or extrapolation with it to go to a new position. In the case of interpolation, the new position will be located between the current position of an unselected bee and that of a selected bee. In the case of extrapolation, the line segment joining the two bees is extended at both ends by half of its length. The new position is a point randomly chosen in either of the extended sections. Thus, instead of the unselected bees being discarded and repositioned randomly, they will move using information from the selected bees. Of course at this stage, there will be no comparison between the fitnesses of the old and new positions of the unselected bee and it moves to its new position.

Another modification is the addition of two procedures to the basic Bees Algorithm.

In the first procedure, a large patch size is initially chosen. For each patch, the initial size is kept unchanged as long as the recruited bees can find better solutions in the

neighbourhood. If the neighbourhood search does not yield any progress, the patch size is decreased. This strategy aims to make the local search more exploitative, searching more thoroughly the area around the local optimum. Henceforth, this procedure will be called the "shrinking method". There are different ways of reducing the neighbourhood size. In this work, at each stage of shrinking, the neighbourhood was shrunk to half of the previous size.

The second procedure is applied when no improvement is gained using the shrinking method (in this work, after the neighbourhood has been shrunk three times). In this case, it is assumed that the patch is centred on a local peak of performance of the solution space. Once the neighbourhood search has found a local optimum, no further progress is possible. Consequently, the location of the peak is recorded and the exploration of the patch is terminated. This procedure is called henceforth "abandon sites without new information".

The pseudo code and flowchart of the improved version of the Bees Algorithm are shown in Figures 3.12 and 3.13.

1- Initialise population with random solutions.

2- Evaluate fitness of the population.

3- While (stopping criterion not met)

//Forming new population.

4- Select sites for neighbourhood search.

5- Determine the patch size.

6- Recruit bees for selected sites (more bees for the best *e* sites) and evaluate fitnesses.

7- Select the fittest bee from each patch.

8- Abandon sites without new information.

9- Assign remaining bees to interpolate or extrapolate with selected bees and evaluate their fitnesses.

10 End While.

Figure 3.12 Pseudo code of the improved Bees Algorithm

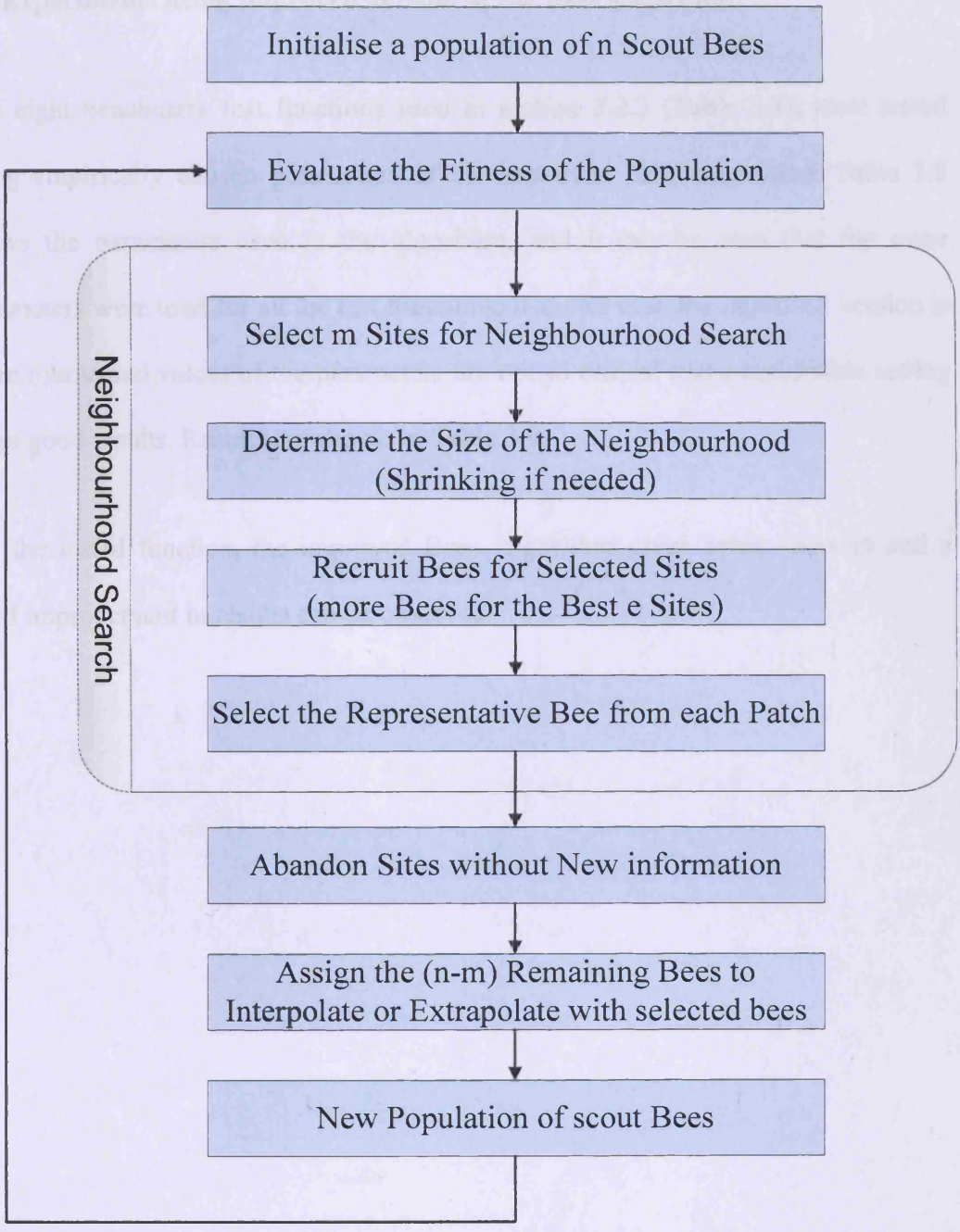

Figure 3.13 Flowchart of the improved Bees Algorithm

3.6 Experiments using improved version of the Bees Algorithm

The eight benchmark test functions used in section 3.2.3 (Table 3.1), were tested using empirically chosen parameters of the improved Bees Algorithm. Table 3.8 shows the parameters used in the algorithm, and it can be seen that the same parameters were used for all the test functions. It shows that, the improved version is more robust and values of the parameters are not so critical and a reasonable setting gives good results. Results are shown in Table 3.9.

For the tested function, the improved Bees Algorithm gives better answers and a good improvement in results can be observed.

fun no	n	m	e	nsp	nep	ngh (initial)
1	20	6	1	5	8	0.1
2	20	6	1	5	8	0.1
3	20	6	1	5	8	0.1
4	20	6	1	5	8	0.1
5a	20	6	1	5	8	0.1
5b	20	6	1	5	8	0.1
6	20	6	1	5	8	0.1
7	20	6	1	5	8	0.1
8	50	6	1	5	8	0.1

Table 3.8 The improved Bees Algorithm parameters

	Basic Bees Algorithm		Improved Bees Algorithm	
func no	Succ %	mean no. of evaluations	Succ %	mean no. of evaluations
1	100	868	100	**829**
2	100	999	100	**920**
3	100	1657	100	**977**
4	100	526	100	**527**
5a	100	631	100	**105**
5b	100	2306	100	**338**
6	100	28529	100	**28210**
7	100	7113	100	**1477**
8	100	20998	100	**14429**

**** Data not available

Table 3.9 Results of the improved Bees Algorithm for test functions

3.7 Summary

This chapter presented a new optimisation algorithm. Experimental results on multi-modal functions in n-dimensions show that the proposed algorithm has remarkable robustness, producing a 100% success rate in all cases. The algorithm converged to the maximum or minimum without becoming trapped at local optima. The algorithm generally outperformed other techniques that were compared with it in terms of speed of optimisation and accuracy of the results obtained.

Two different constrained optimisation problems were solved using the Bees Algorithm. In each case, the algorithm converged to the optimum without becoming trapped at local optima. The algorithm generally outperformed other optimisation techniques in terms of the accuracy of the results obtained.

In addition to the basic Bees Algorithm, an improved version of the Bees Algorithm was described. A comparison between the results obtained by the Basic version and the improved version were also presented.

Indeed, the Bees Algorithm can solve a problem without any special domain information, apart from that needed to evaluate fitnesses.

CHAPTER 4

PRELIMINARY DESIGN AND MULTI-OBJECTIVE OPTIMISATION USING THE BEES ALGORITHM

4. PRELIMINARY DESIGN AND MULTI-OBJECTIVE OPTIMISATION USING THE BEES ALGORITHM

4.1 Preliminaries

This chapter describes two adapted versions of the Bees Algorithm for solving multi-solution and multi-objective optimisation problems. Preliminary design is a multi-solution problem. In the first section, a first adapted version of the Bees Algorithm will be presented to solve the problem preliminary design. In the second section, a second version of the Bees Algorithm will be presented for multi-objective function optimisation problems. The results will be compared with the results obtained by a genetic algorithm and random search.

4.2 Preliminary Design

Until now, research efforts in computer aided design (CAD) have mainly focused on detailed design. Effective computing tools have been developed for tedious and time consuming tasks such as finite element analysis (FEA), simulation, and draughting for the later design phases. The initial creative part of the design process is still carried out almost exclusively by humans.

There is now increasing interest in the automation of creative design. A desirable feature for a computer-based creative design system is the ability to generate multiple conceptual solutions. The more candidate solutions the system produces, the greater the chance is of finding the optimal solution (Pham and Yang 1993a, b).

Preliminary design can be regarded as an optimisation task, where the goal is to generate as many solutions as possible that meet pre-defined quality criteria. This section presents a new preliminary design system based on the Bees Algorithm (Pham et al. 2006), an optimisation tool recently developed by the authors. In the proposed study, the Bees Algorithm is used to generate the largest possible number of acceptable solutions. Preliminary gearbox design is chosen as a case study for the proposed algorithm. The results obtained by the Bees Algorithm are compared with the results obtained by a basic random search procedure, and the results obtained by the popular Genetic Algorithm (Fogel 2000) optimisation technique.

4.3 The Multi-Solution Bees Algorithm

In chapter 3, the Bees Algorithm for finding the optimum solution in a given search space was described. If a problem requires the discovery of the largest possible number of solutions that satisfy a given quality criterion, a filtering method can be adopted in order to capture all non-identical satisfactory individuals generated after the evaluation process.

Figures 4.1 and 4.2 show the pseudo code and the flowchart for the multi-solution Bees Algorithm. The algorithm is like the one described in chapter 3, only step 4 has been added to the algorithm to select and store satisfactory solutions. In step 4, after fitness evaluation of the population, those candidate solutions that satisfy the quality criterion will be added to the solution set. In order to avoid duplicates in the solution set, only solutions which were not previously included will be added to the solution set.

1- Initialise population with random solutions.
2- Evaluate fitness of the population.
3- Select the solutions that satisfy the criterion and add them to the solution set
4- While (stopping criterion not met)
    //Forming new population.
5- Select sites for neighbourhood search.
6- Determine the patch size.
7- Recruit bees for selected sites (more bees for best *e* sites) and evaluate fitnesses.
8- Select the fittest bee from each patch.
9- Abandon sites without new information.
10- Assign remaining bees to search randomly and evaluate their fitnesses.
11- End While.

Figure 4.1. Pseudo code of the multi-solution Bees Algorithm

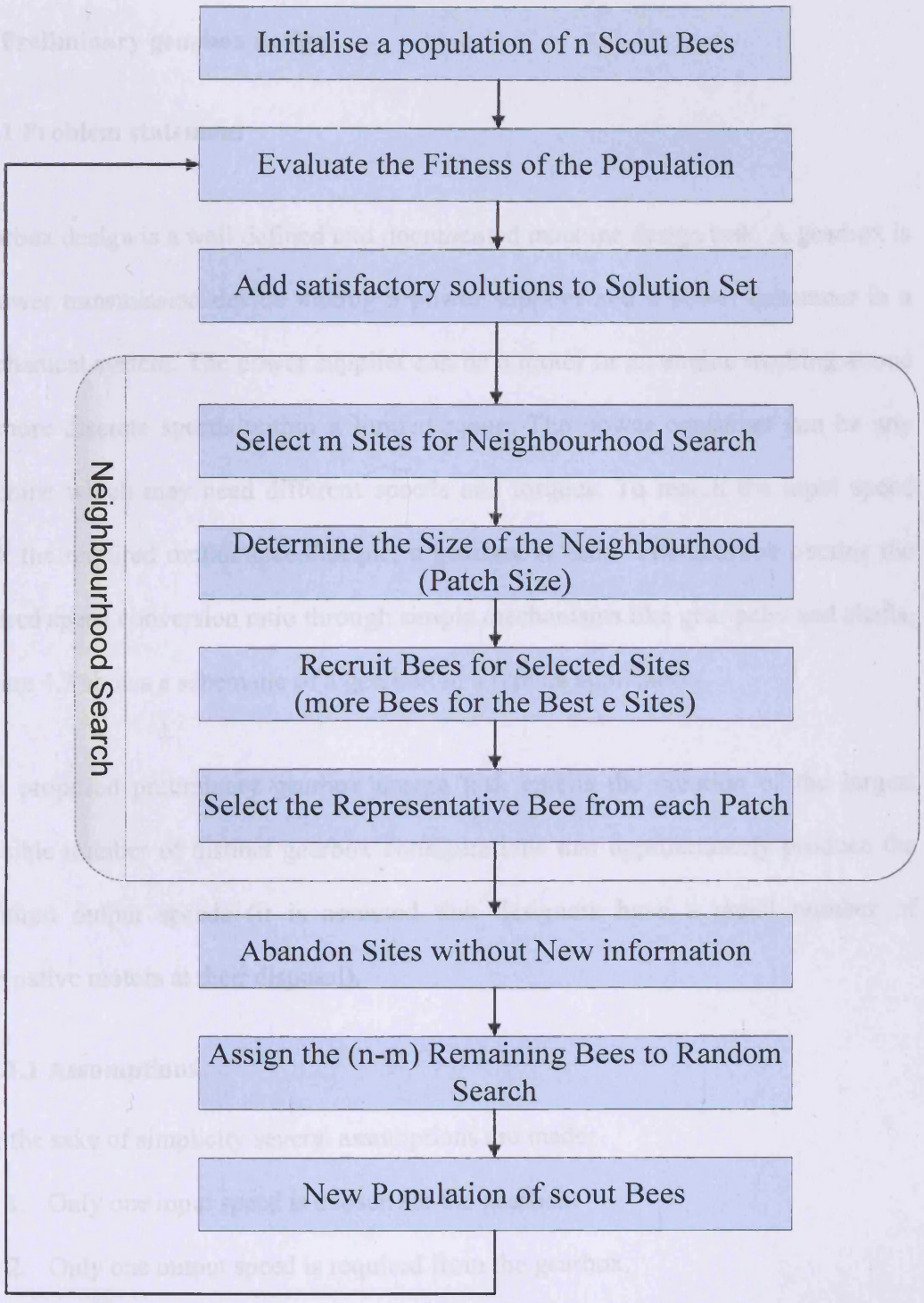

Figure 4.2. Flowchart of the multi-solution Bees Algorithm

4.4 Preliminary gearbox design

4.4.1 Problem statement

Gearbox design is a well defined and documented machine design task. A gearbox is a power transmission device linking a power supplier and a power consumer in a mechanical system. The power supplier can be a motor or an engine working at one or more discrete speeds within a limited range. The power consumer can be any machine which may need different speeds and torques. To match the input speed with the required output speed/torque, a gearbox is used. The gearbox obtains the desired speed conversion ratio through simple mechanisms like gear pairs and shafts. Figure 4.3 shows a schematic of a gearbox in a typical application.

The proposed preliminary gearbox design task entails the creation of the largest possible number of distinct gearbox configurations that approximately produce the required output speeds (it is assumed that designers have a small number of alternative motors at their disposal).

4.4.1.1 Assumptions

For the sake of simplicity several assumptions are made:

1. Only one input speed is chosen for the gearbox.
2. Only one output speed is required from the gearbox.
3. Only parallel shafts are used in the gearbox.

4. The power transmission between two adjacent shafts is accomplished only by pairs of gears.

4.4.1.2 Parameters

The following parameters are known:

a. the speed of the motor (700, 1000, 1500 or 3000 rpm);

b. the required output speed (pre-specified in revolutions per minute).

The following parameters are to be computed:

a. number of shafts;

b. number of teeth for each gear.

4.4.1.3 Constraints

1. The minimum number of teeth of a gear is 18, and the maximum number of teeth is 274.
2. The maximum transmission ratio for each gear pair is 5.0.
3. The maximum number of shafts, including the input and output shafts, is 8.

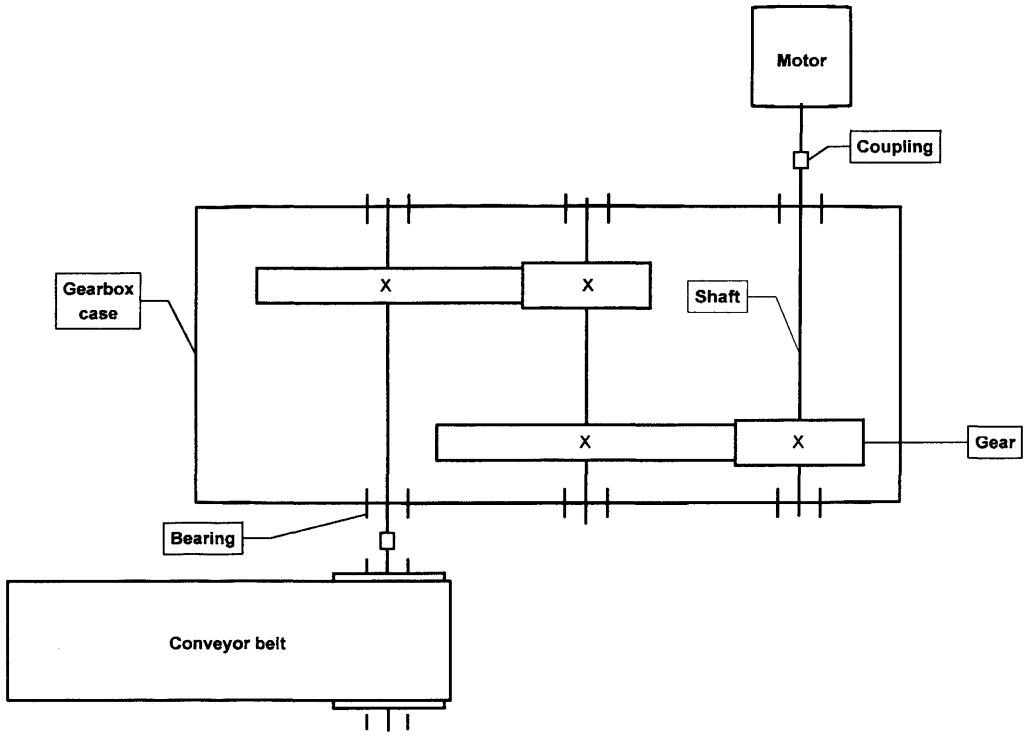

Figure 4.3. An example of a gearbox

4.4.2 Evaluation function

Candidate solutions are evaluated according to a positive-valued fitness function F that incorporates the main design specifications. The parameters considered in the fitness function are listed below:

1. The number of shafts (*NoOfShafts*) generated must not exceed 8:

if $NoOfShafts > 8$, then $F = 0.0$ (4.1)

2. For a reduction gearbox, the number of teeth of the driven gear must be larger than the number of teeth of the driving gear for each gear pair. Otherwise, the candidate solution is considered invalid and its fitness F is calculated as:

$$F = 28.0 - 2.0 * WrongPair \quad (4.2)$$

where *WrongPair* is the number of invalid gear pairs.

3. The transmission ratio achieved by one pair of gears must not exceed the maximum single-stage transmission ratio (*MAXRATIO*) by more than a given percentage. The tolerance is set to 15% of the *MAXRATIO* value. Solutions containing gear pairs exceeding the tolerance limit are penalised according to the following formula:

$$F = F_{original} * (1.0 - 0.1 SumOfExcess) \quad (4.3)$$

where (*SumOfWxcess*) is the sum of the transmission ratio overshootings of all the gear pairs exceeding the *MAXRATIO* limit.

4. In order to make the gearbox design compact, the number of shafts (*NOfShafts*) should be as low as possible.

5. The difference between the total transmission ratio of the gearbox and the required ratio (*RatioDifferrence*) should be as small as possible.

6. Fitness values should be positive. Candidate solutions satisfying constraints 1 to 3 (that is, valid solutions) but having negative fitness are assigned a positive fitness value of 27. This figure is slightly higher than the maximum fitness attainable by an invalid solution. Accordingly, valid candidate solutions are considered fitter than invalid solutions.

Taking into account specifications 3, 4 and 5, the overall fitness function is as follows:

$$F = \begin{cases} 0.0, & \text{If } NoOfShafts > 8; \\ 28.0 - 2 * WrongPair, & \text{If } WrongPair \neq 0; \\ (MaxAllowedFitness & \\ -[W_1 * 3^{(A_1 * NoOfShafts)} & \\ +W_2 * 3^{(A_2 * RatioDifference)}]), & \text{If } F >= 0; \\ 27.0, & \text{If } F < 0; \end{cases} \tag{4.4}$$

where *MaxAllowedFitness* is set to 5000 and is the maximum fitness value obtainable by a candidate solution. *W1* and *W2* are penalty weights, while *A1* and A2 are coefficients. *W1*, *W2*, *A1* and *A2* are experimentally optimised in order to yield a fitness function that is effective over a wide range of output speeds. Their final values are set to:

$$(W_1 \quad W_2 \quad A_1 \quad A_2) = (0.286 \quad 1.0 \quad 0.7925 \quad 1.0) \tag{4.5}$$

4.5 Genetic Algorithms

In order to assess the performance of the proposed design technique, the results produced by the Bees Algorithm are compared with those obtained by random search and by a genetic optimisation procedure.

Genetic Algorithms (GAs) are population-based stochastic search algorithms that aim to find an acceptable solution when time or computational requirements make it impractical to find the best one. GA has been described in chapter 2.

4.6 Results of preliminary design

The results obtained by the Bees Algorithm are compared with those found by a GA and a random search procedure. The random search results are used for baseline evaluation of the performance of the Bees Algorithm and the GA.

Since all the variables in this problem are discrete, candidate solutions are encoded using integer-valued strings. The three search algorithms use the same integer-based encoding scheme for representing the solutions, and the same fitness evaluation procedure, which is described in sub-Section 4.2. The Bees Algorithm and the GA use the same random technique for population initialisation. This technique is used also in the random search procedure to generate new solutions. Finally, the Bees Algorithm neighbourhood search operator and the GA mutation operator share the same procedure for modifying the candidate solutions. According to this procedure,

the transmission ratio of randomly picked gear pairs is changed by randomly altering their number of teeth. The main search parameters are set so as the three search algorithms sample the same number of candidate solutions over the total optimisation process.

4.6.1 Solution acceptance criteria

A "filtering" method is employed to capture all nonidentical satisfactory individuals generated during the evaluation process. The total number of acceptable solutions is the measure used to assess the performance of the design method under evaluation.

The filtering criteria for deciding whether a solution is acceptable are:

1. The output speed is within ± 10 percent of the required output speed.
2. The fitness measure is above or equal to a pre-defined threshold.

4.6.2 Parameters of the Bees Algorithm

The main parameters characterising the Bees Algorithm are empirically chosen. They are detailed in Table 4.1. The search space is defined as:

$$2 \leq NoOfShafts \leq 8 \tag{4.6}$$

$$18 \leq N_i \leq 274 \tag{4.7}$$

where N_i is the number of teeth of gear i. The total number of gears varies according to the number of shafts. Since each shaft is made of a gear pair, the total number of gears is equal to $2 * NoOfShafts$.

The initial patch size for the first variable (*NoOfShafts*) is set to 2. For the remaining variables (N_i) is set to 10.

Bees Algorithm parameters	Symbol	Value
Population	n	33
Number of selected sites	m	3
Number of top-rated sites out of m selected sites	e	2
Initial patch size	ngh	2/10
Number of bees recruited for best e sites	nep	50
Number of bees recruited for the other (m-e) selected sites	nsp	20
Number of iterations	g	300

Table 4.1. Parameters of the Bees Algorithm

Genetic Algorithm parameters	Symbol	Value
Population	n	150
Generations	g	300
Crossover rate	cr	1.0
Mutation rate	mr	0.25

Table 4.2. Parameters of the Genetic Algorithm

4.6.3 Parameters of the Genetic Algorithm

The main parameters characterising the GA are empirically chosen. They are detailed in Table 4.2.

4.6.4 Results of the preliminary gearbox design

Table 4.3 gives the average number of solutions obtained by the Bees Algorithm (BA), the Genetic Algorithm (GA), and the Random Search Algorithm (Random). The results are calculated over 100 independent runs.

Table 4.3 shows that the Bees Algorithm outperforms the genetic and random search algorithms in terms of acceptable solutions found. In some cases, the number of acceptable solutions produced by the Bees Algorithm is over one order of magnitude larger than the number of solutions produced by the GA.

Fitness Threshold	**Method**	Required Output Speed (rpm)			
		30	**50**	**75**	**100**
4992	Random	20	88	142	164
	GA	61	672	871	811
	BA	3453	4012	3973	3804
4993	Random	17	71	127	151
	GA	54	611	756	791
	BA	3275	3885	3830	3786
4994	Random	10	46	87	118
	GA	41	345	559	626
	BA	2654	3329	3492	3600
4995	Random	1	6	11	15
	GA	7	45	74	94
	BA	427	613	689	755

Table 4.3. Number of solutions for different output speeds

4.7 Multi-0bjective optimisation

The goal of an optimisation problem can be stated as finding the combination of parameters (independent variables) which maximises or minimises the value of one or more dependent variables possibly subject to some constraints on the independent variable ranges. The values to be optimised are called objective functions. If there is only one function to optimise, the task is a single function optimisation problem. If more than one function should be optimised, the task is a multi-objective optimisation problem.

There is now increasing interest in multi-objective function optimisation as most engineering design problems involve multiple and often conflicting objectives. There are two ways of solving multi-objective optimisation problems. The first possibility is to form a linear combination of the different objective functions. The contribution of each function is associated to a weight, and each function is optimised using methods developed for single objective function problems. The other way of solving a multi-objective problem – the genuine way - is to consider all objective functions simultaneously. The following two main drawbacks are of concern when converting a multi-objective optimisation problem into a single objective optimisation problem. The first shortcoming is that not all the solutions are usually found. The second drawback is that the weight assigned to some objective functions may not be suitable, and the overall linear combination of functions may lack of significance. In multi-objective optimisation tasks, the goal is not to find a single optimal solution,

but to compute the set of all non-dominated solutions, that is, the Pareto optimal set. A solution belonging to the Pareto set is not better than any other solution belonging to the same set. For this reason, they are not comparable and each of them is called a feasible solution. Different techniques to solve multi-objective function optimisation tasks and their characteristics are explained in (Deb 2001).

A maximum of a function f is a minimum of $-f$. Thus, the general optimisation problem may be stated mathematically as:

$$\begin{array}{lll} \text{minimise} & f_i(X), & i = 1,2,...,l \\ \text{subject to} & c_j(X) = 0, & j = 1,2,...,p \\ & h_k(X) \geq 0, & k = 1,2,...,q \\ & X = (x_1, x_2, ..., x_n)^T & \end{array} \tag{4.8}$$

Where $f_i(X)$ are the l objective functions, X is the column vector of the n independent variables, $c_j(X)$ are p termed equality constraints, and $h_k(X)$ are q inequality constraints. Taken together, $f_i(X)$, $c_j(X)$ and $h_k(X)$ are known as the problem function(Deb 2001).

The word 'minimise' means that we want to minimise all the objective functions simultaneously. If there is no conflict between the objective functions, then a solution can be found where every objective function reaches its optimum. To avoid such trivial cases, it is assumed that there is not a single solution that is optimal with

respect to every objective function. This means that objective functions are at least partly conflicting. They may also have different units.

4.7.1 Pareto ranking and Pareto optimality

An individual's Pareto rank corresponds to the number of individuals in the current population by which it is dominated. For example in Figure 4.4, the solutions which are shown by red circles are not dominated by any other solutions, so their rank is zero. The blue ones are dominated by one other solution so their rank is 1 and in the same way the rank of solution shown by green circle is 5.

The predominant solution concept in defining solutions for multi-objective optimisation problems is that of Pareto optimality (Arora 2004). A solution in the feasible solution space is called Pareto optimal (or non-dominated solution) if there is no other feasible solution in the solution space that reduces at least one objective function without increasing another one. According to the Pareto ranking definition, the ranks of all non-dominated solutions are zero.

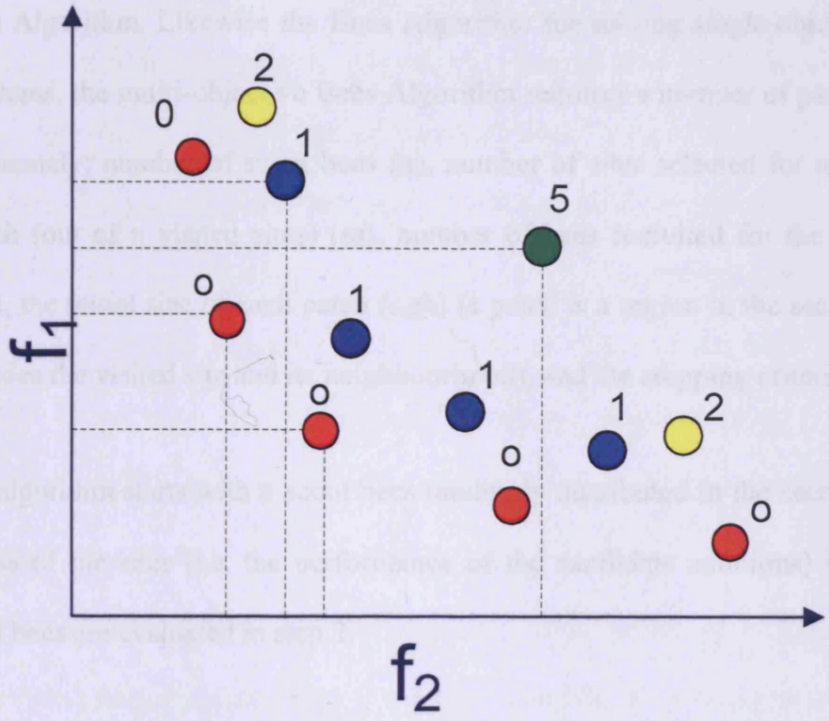

Figure 4.4. Pareto ranking of candidate solutions

4.8 The multi-objective Bees Algorithm

Figures 4.5 and 4.6 show the pseudo code and the flowchart for the multi-objective Bees Algorithm. Likewise the Bees Algorithm for solving single objective function problems, the multi-objective Bees Algorithm requires a number of parameters to be set, namely: number of scout bees (n), number of sites selected for neighbourhood search (out of n visited sites) (m), number of bees recruited for the selected sites (nsp), the initial size of each patch (ngh) (a patch is a region in the search space that includes the visited site and its neighbourhood), and the stopping criterion.

The algorithm starts with n scout bees randomly distributed in the search space. The fitness of the sites (i.e. the performance of the candidate solutions) visited by the scout bees are evaluated in step 2.

In step 4, the m non-dominated sites are designated as "selected sites" and chosen for neighbourhood search. If there are more than m non-dominated sites in the population, the first m will be selected since it is not possible to differentiate between them. If there are less than m non-dominated sites, from the dominated ones which have been dominated only once, the rest will be selected and this procedure is repeated until a sufficient number of sites have been selected. In step 5, a large patch size is chosen initially. For each patch, the initial size is kept unchanged as long as the recruited bees can find better solutions in the neighbourhood. If the neighbourhood search does not yield any progress, the patch size is decreased. This

strategy aims at making the local search more exploitative, searching more densely the area around the local optimum. Henceforth, this step will be called the "shrinking method".

In step 6, the algorithm searches around the selected sites. In the basic version of the Bees Algorithm, more bees were chosen to search in the vicinity of the best e sites. In selection of the best sites was made according to the fitness associated with. In the multi-objective optimisation version of the Bees Algorithm, sometimes it is not possible to rank the solution candidates, so all the selected sites have the same number of recruited bees to search around the neighbourhood. In step 7, the representative bee will be the original one unless it is dominated by one of the recruited bees; in that case the representative will be the new non-dominated bee. Step 8, has been added to the basic Bees Algorithm to enable it to deal with multi-objective optimisation problems. If the representative is a non-dominated solution, it will be added to the Pareto optimal set. In addition, if this solution is dominating the other solutions in the created Pareto optimal set, the dominated solutions will be removed from the set.

1- Initialise population with random solutions.

2- Evaluate fitness of the population.

3- While (stopping criterion not met)

//Forming new population.

4- Select sites for neighbourhood search.

5- Determine the patch size.

6- Recruit bees for selected sites and evaluate fitnesses.

7- Select the representative bee from each patch.

8- Create or Amend the Pareto optimal set.

9- Abandon sites without new information.

10- Assign remaining bees to search randomly and evaluate their fitnesses.

11- End While.

Figure 4.5. Pseudo code of the multi-objective Bees Algorithm

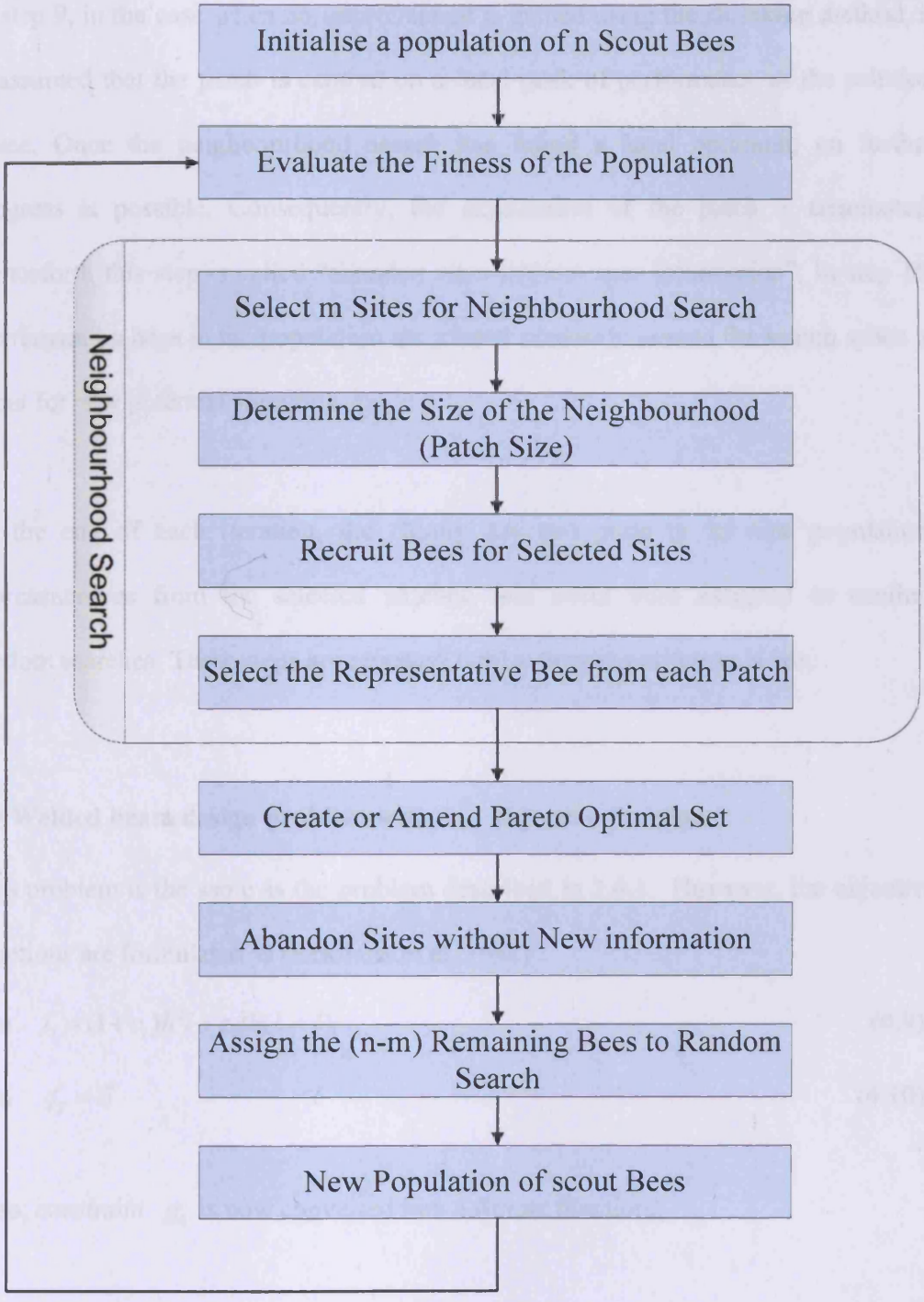

Figure 4.6. Flowchart of the multi-objective Bees Algorithm

In step 9, in the case when no improvement is gained using the shrinking method, it is assumed that the patch is centred on a local peak of performance of the solution space. Once the neighbourhood search has found a local optimum, no further progress is possible. Consequently, the exploration of the patch is terminated. Henceforth this step is called "abandon sites without new information". In step 10, the remaining bees in the population are placed randomly around the search space to scout for new potential solutions.

At the end of each iteration, the colony has two parts to its new population: representatives from the selected patches, and scout bees assigned to conduct random searches. These steps are repeated until a stopping criterion is met.

4.9 Welded beam design problem with two objective functions

This problem is the same as the problem described in 3.4.1. However, the objective functions are formulated as (Rekliatis et al. 1983) :

$$\min \quad f_1 = (1+c_1)h^2 l + c_2 tb(L+l) \tag{4.9}$$

$$\min \quad f_2 = \delta \tag{4.10}$$

Also, constraint g_8 is now converted into a fitness function.

Bees Algorithm parameters	Symbol	Value
Population	n	150
Number of selected sites	m	30
Initial patch size	ngh	0.1
Number of bees recruited for selected sites	nsp	50
Number of iterations	g	1000

Table 4.4. Parameters of the multi-objective Bees Algorithm for the welded beam design problem

4.10 Results for the welded beam design problem with two objective functions

The empirically chosen parameters for the Bees Algorithm are given in Table 4.4.

Figure 4.8 shows the non-dominated solutions obtained using the Bees Algorithm. The total number is 215 non-dominated solutions distributed along the Pareto front. Deb has investigated this problem using the non-dominated sorting GA (or NSGA) and a fast elitist NSGA, called NSGA-II (Deb et al. 2000), for finding multiple Pareto optimal solutions (Figure 4.9).

In comparison with the number of solutions found by the non-dominated sorting genetic algorithms, the Bees Algorithm can find more non-dominated solutions. NSGA-II found the best cost solution with a cost of 2.79 units(Deb et al. 2000). The multi-objective Bees Algorithm could find a quantity of 2.39 units cost, which is closer to the best solution (with a cost of 2.38 units) found using the single objective Bees Algorithm in chapter 3.

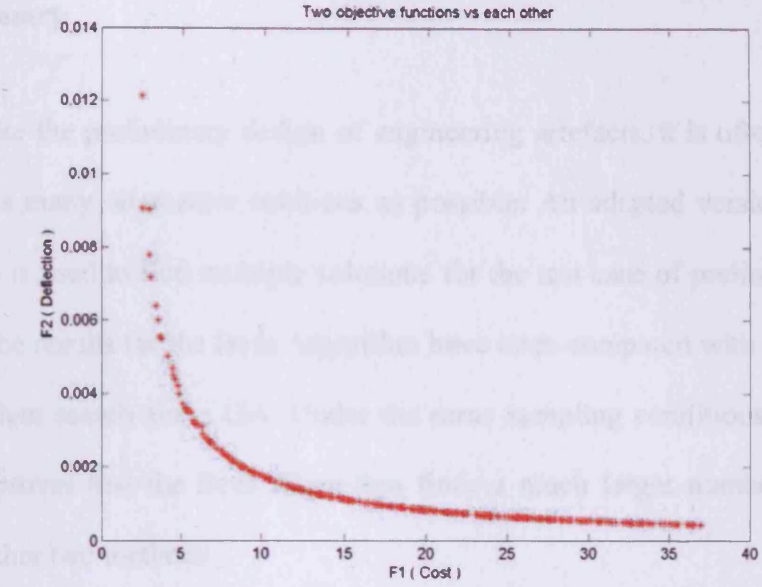

Figure 4.8. Non-dominated solutions obtained using the Bees Algorithm

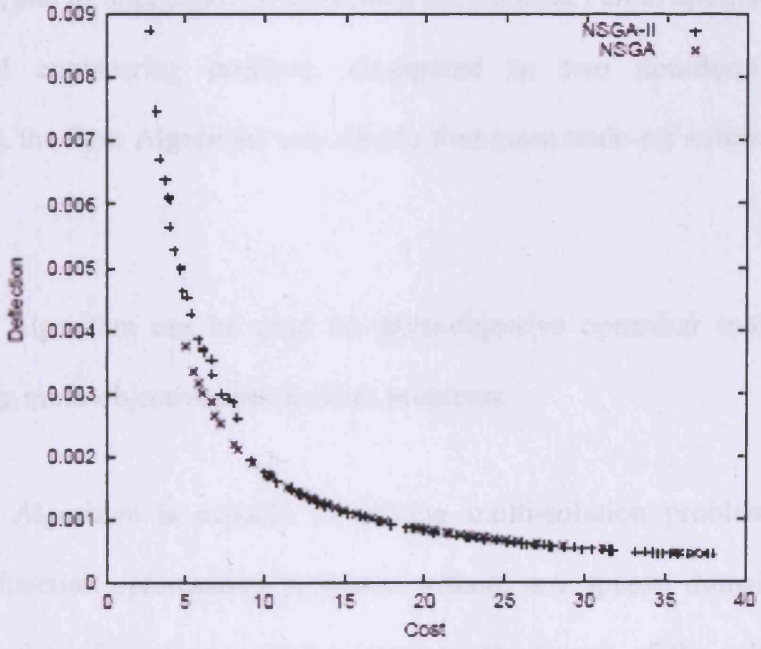

Figure 4.9. Non-dominated solutions obtained using the two different versions of genetic algorithms (Deb et al. 2000)

4.11 Summary

In tasks like the preliminary design of engineering artefacts, it is often necessary to generate as many alternative solutions as possible. An adapted version of the Bees Algorithm is used to find multiple solutions for the test case of preliminary gearbox design. The results for the Bees Algorithm have been compared with those obtained using random search and a GA. Under the same sampling conditions, experimental evidence proves that the Bees Algorithm finds a much larger number of solutions than the other two methods.

The second section of this chapter has described a modified version of the Bees Algorithm, and its application to the search for multiple Pareto optimal solutions in a mechanical engineering problem. Compared to two non-dominated genetic algorithms, the Bees Algorithm was able to find more trade-off solutions (Deb et al. 2000).

The Bees Algorithm can be used as multi-objective optimiser tool for complex engineering multi-objective optimisation problems.

The Bees Algorithm is capable of solving multi-solution problems and multi-objective function optimisation problems without any special domain knowledge, apart from the information needed to evaluate the fitness of the solutions. In this

respect, the Bees Algorithm shares the advantages of general-purpose optimisation algorithms such as GAs.

CHAPTER 5

TRAINING RADIAL BASIS FUNCTION NEURAL NETWORKS USING THE BEES ALGORITHM

5. TRAINING RADIAL BASIS FUNCTION NEURAL NETWORKS USING THE BEES ALGORITHM

5.1 Preliminaries

Artificial neural networks are computational models of the brain (Pham and Liu 1995). There are many types of neural networks representing the brain's structure and operation with varying degrees of sophistication. The Radial Basis Function (RBF) network is a popular type of network that is very useful for pattern classification problems (Bishop 1995). This chapter presents the use of Radial Basis Function (RBF) networks for identification patterns recognition in control charts and of wood defects. The RBF networks were trained, employing the Bees Algorithm instead of the standard training algorithms. The chapter includes explanations of the RBF network, the standard RBF training method, the training procedure based on the Bees Algorithm, results of control chart pattern recognition experiments, and identification of wood defects with RBF networks trained using the Bees Algorithm and the conventional RBF procedure.

5.2 Radial Basis Function (RBF) network

5.2.1 Network structure

As the name implies, this network makes use of radial functions. Figure 5.1 shows the structure of a RBF network which consists of three layers of neurons.

The input layer neurons receive the input pattern (x_1 to x_N). The hidden layer neurons provide a set of activation functions that constitute an arbitrary "basis" for the input patterns in the input space to be expanded into the hidden space by way of non-linear transformation. At the input of each hidden neuron, the distance between the centre of each activation (basis) function and the input vector is calculated. Applying the basis function to this distance produces the output of the hidden neuron. The RBF network outputs y_1 to y_p are formed by the neurons in the output layer as weighted sums of the hidden layer neuron activations.

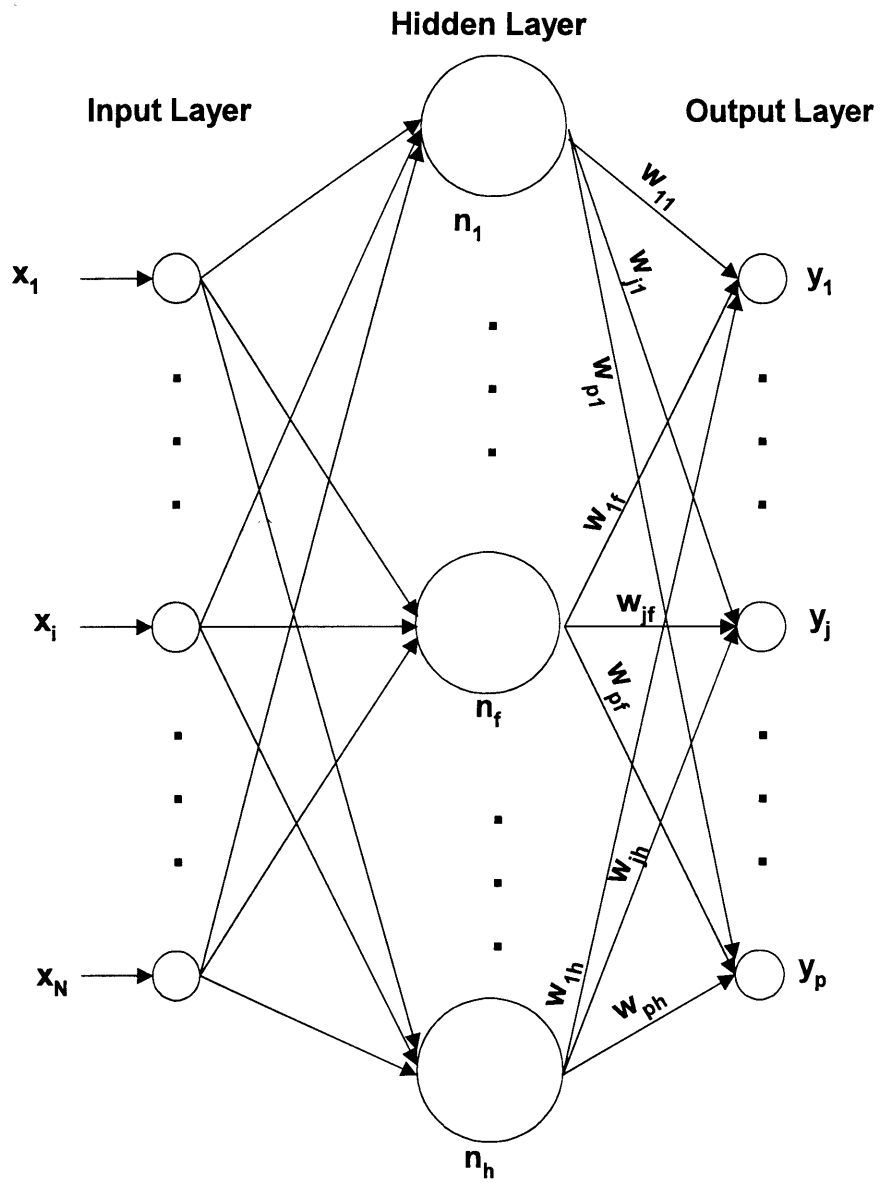

Figure 5.1. Topology of an RBF network.

The basis function is generally chosen to be a standard function which is positive at its centre x = 0, and decreases uniformly to zero at the sides. A common choice is the Gaussian distribution function:

$$K(x) = \exp\left(-\frac{x^2}{2}\right) \tag{5-1}$$

This function can be shifted to an arbitrary centre, x = c, and stretched by varying its spread σ as follows:

$$K\left(\frac{(x-c)}{\sigma}\right) = \exp\left(-\frac{(x-c)^2}{2\sigma^2}\right) \tag{5-2}$$

The outputs of the RBF network y_j are given by:

$$y_j = \sum_{i=1}^{h} w_{ji} K\left(\frac{\|x-c_i\|}{\sigma_i}\right)_{\forall x} \tag{5-3}$$

where w_{ji} is the weight of connection from hidden neuron i to output j, c_i the centre of basis function i, and σ_i is the spread of the function i. $\|x-c_i\|$ is the norm of $(x-c_i)$. There are various ways to calculate the norm. The most common is the Euclidean norm given by:

$$\|x-c_i\| = \sqrt{(x_1-c_{i1})^2+(x_2-c_{i2})^2+....+(x_N-c_{iN})^2} \tag{5-4}$$

This norm gives the distance between the two points x and c_i in the N-dimensional input space. All points x that are the same radial distance from c_i give the same value of the norm. The purpose of training an RBF network is to determine the

connection weights w_{ji}, RBF centres c_i,and spreads σ_i that enable the network to produce the correct outputs y_j corresponding to the input patterns x.

5.2.2 RBF network training procedure

The training of an RBF network involves the minimisation of an error function. The error function defines the total difference between the actual output and the desired output of the network over a set of training patterns (Jain and Dubes 1988). Training proceeds by presenting to the network a pattern of known class taken from the training set. The error component associated with that pattern is the sum of the squared differences between the desired and actual outputs of the network for the presented pattern. The procedure is repeated for all the patterns in the training set. The error components for all the patterns are summed to yield the error function for the RBF network. After training, the percentage of training patterns which the network can recognise correctly is called the training accuracy, and the percentage of test patterns correctly classified is called the test accuracy.

5.2.2.1 Standard RBF network training procedure

According to the standard procedure for training RBF networks, after the number of hidden neurons (h) has been set, the following steps will be taken:

1. Choose the RBF centres c_i. Centre selection can be performed by trial and error, self-organisation or supervised training.

2. Choose the spreads σ_i. Several heuristic methods are available. A popular method is to set σ_i equal to the distance to the centre nearest to c_i.

3. Calculate the neuron weights w_{ji}. When c_i and w_{ji} are set, the outputs of the hidden neurons $(K_1,.....,K_h)^T$ can be calculated for any pattern of inputs $x=(x_1,...,x_N)$. Assuming there are s input patterns x in the training set, there will be s sets of hidden neuron outputs that can be calculated. These can be assembled into a h x s matrix:

$$K=\begin{bmatrix} k_1^1 & k_1^2 & .... & .... & k_1^s \\ k_2^1 & k_2^2 & .... & .... & k_2^s \\ . & & & & \\ . & & & & \\ k_h^1 & k_h^2 & .... & .... & k_h^s \end{bmatrix}_{hXs} \tag{5-5}$$

The output of the RBF network (y) is given by equation (5-6).

$$y = K^T.w^T \tag{5-6}$$

where

$$w^T=\begin{bmatrix} w_{11} & w_{12} & .... & .... & w_{1p} \\ w_{21} & w_{22} & .... & .... & w_{2p} \\ . & & & & \\ . & & & & \\ w_{h1} & w_{h2} & .... & .... & w_{hp} \end{bmatrix} \tag{5-7}$$

y is the matrix of the actual outputs corresponding to the training inputs x. Ideally, y should be equal to d, the desired or target outputs. Unknown coefficients w_{ji} can

be calculated from equation (5-8) in order to minimise the sum of the squared differences between y and d.

$$w^T = \left(K.K^T\right)^{-1}.K.d \tag{5-8}$$

5.2.2.2 RBF network training using the Bees Algorithm

When the Bees Algorithm is used, each bee represents an RBF network with a particular set of basis function centres, spreads and weight vectors. The aim of the algorithm is to find the bee producing the smallest value of the error function.

The RBF network training procedure using the Bees Algorithm comprises the following steps:

1. Generate an initial population of bees.
2. Apply the training data set to determine the value of the error function associated with each bee.
3. Based on the error value obtained in step 2, create a new population of bees comprising the best bees in the selected neighbourhoods and randomly placed scout bees.
4. Stop if the value of the error function has fallen below a predetermined threshold or after a set number of iterations.
5. Else, return to step 2.

5.3 Control chart pattern (CCP) recognition experiments

5.3.1 Control chart pattern

Statistical Process Control (SPC) employs statistical means such as control charts to show how consistently a process is performing and whether it should be adjusted (Montgomery 2000). SPC control charts enable a manufacturing engineer to compare the actual performance of a process with customer specifications and provide a process capability index to guide and assess quality improvement efforts. By means of simple rules, it is possible to determine if a process is out of control and needs corrective action. However, incipient problems could be detected before the process goes out of control from the type of patterns displayed by the control charts.

There are six main classes of patterns in control charts, normal, cycle, upward trend, downward trend, upward shift, and downward shift, as illustrated in Figure 5.2.

Specifically, control chart pattern recognition is a process of recognising an unknown CCP and assigning it to one of the prescribed pattern classes. Normally, patterns of the same category share common properties.

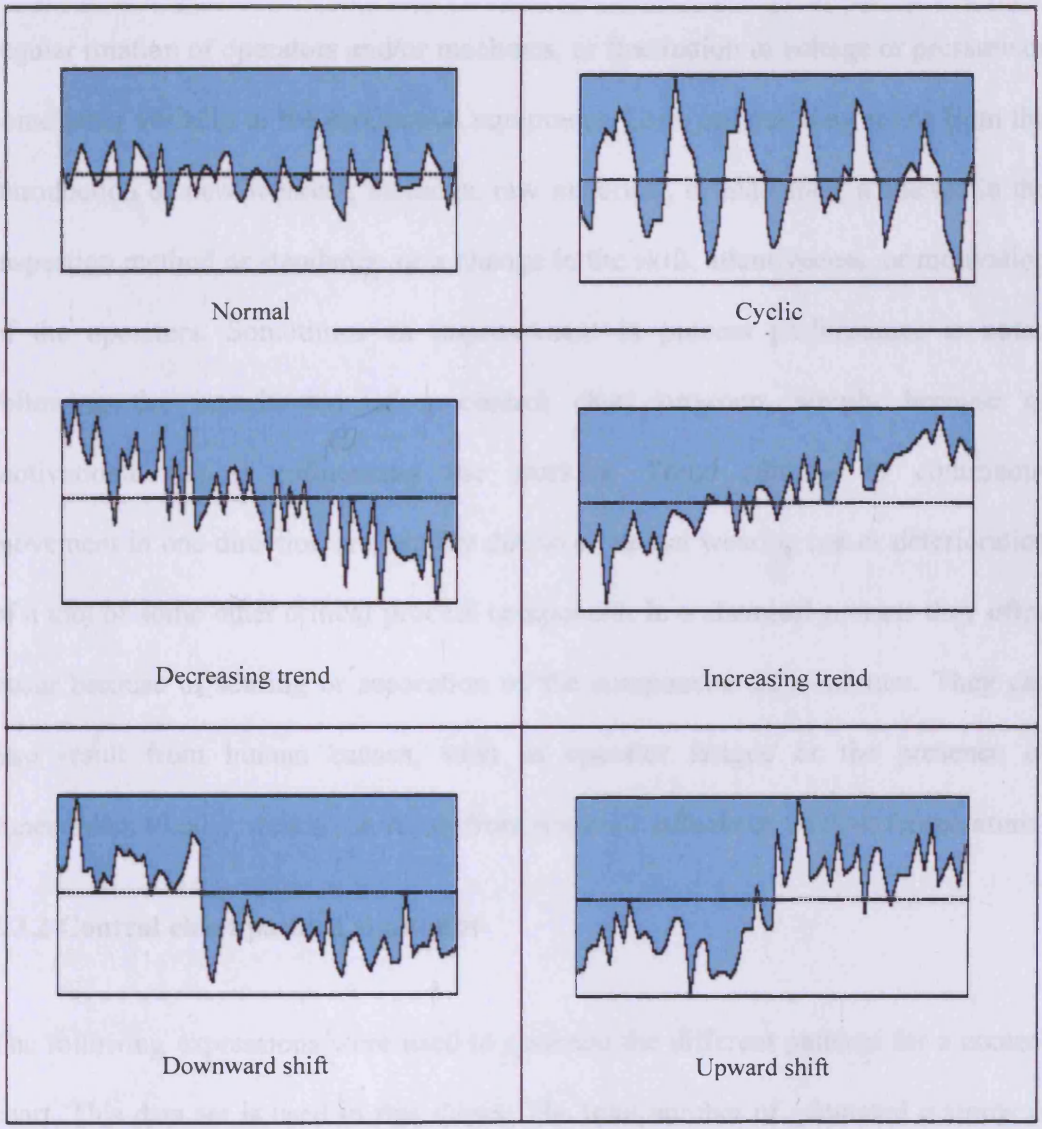

Figure 5.2. Six main classes of control chart patterns.

Cyclic patterns occasionally appear on the control chart. Cyclic patterns may result from systematic environmental changes such as temperature, operator fatigue, regular rotation of operators and/or machines, or fluctuation in voltage or pressure or some other variable in the production equipment. Shift patterns may result from the introduction of new workers, methods, raw materials, or machines; a change in the inspection method or standards; or a change in the skill, attentiveness, or motivation of the operators. Sometimes an improvement in process performance is noted following the introduction of a control chart program, simply because of motivational factors influencing the workers. Trend patterns or continuous movement in one direction are usually due to a gradual wearing out or deterioration of a tool or some other critical process component. In a chemical process they often occur because of settling or separation of the components of a mixture. They can also result from human causes, such as operator fatigue or the presence of supervision. Finally, trends can result from seasonal influences, such as temperature.

5.3.2 Control chart pattern simulator

The following expressions were used to generate the different patterns for a control chart. This data set is used in this thesis. The total number of generated patterns is 1500 and each pattern is a time series comprising 60 points. 498 patterns (83 in each class) were used for training an RBF network and 1002 patterns (167 in each class) were employed for testing the trained network.

1. Normal patterns:

$$y(t) = \mu + r(t)\,\sigma \tag{5-9}$$

2. Cyclic patterns:

$$y(t) = \mu + r(t)\,\sigma + a\sin(2\pi\, {}^{t}/_{T}) \tag{5-10}$$

3. Increasing or decreasing trends:

$$y(t) = \mu + r(t)\,\sigma \pm gt \tag{5-11}$$

4. Upward or downward shifts:

$$y(t) = \mu + r(t)\,\sigma \pm ks \tag{5-12}$$

where

μ = mean value of the process variable being monitored

σ = standard deviation of the process

a = amplitude of cyclic variations (taken as 15 or less)

g = magnitude of the gradient of the trend (taken as being in the range 0.2 to 0.5)

k = parameter determining the shift position (k=0 before the shift position; k=1 at the shift position and thereafter).

r = normally distributed random number (between -3 and 3)

s = magnitude of the shift (taken as being in the range of 7.5 to 20)

t = discrete time at which the pattern is sampled (taken as being within the range 0 to 59).

T = period of a cycle (taken as being in the range 4 to 12 sampling intervals).

$y(t)$ = sample value at time t.

This pattern simulator is taken from Pham and Oztemel (Pham and Oztemel 1992). In this thesis, the data were scaled before presenting them to the network.

5.3.2.1 Procedure scaling data

Although the input data to a node can theoretically take any value, restricting it to fall within a fixed range produces more efficient training. Scaling is an application-specific transformation that constrains input data into a fixed range. The most important issue in scaling is the range of output values dictated by the scaling transformation. Scaling has two advantages. The first advantage is that scaling takes care of the distribution of the training data and the effect of outliers. The second advantage is that scaling ensures that errors or variations of different variables contribute the same proportion to the change in network weights. In this thesis, by applying the scaling method mentioned below, the original inputs were scaled to continuous values between 0 and 1. The actual data sets were scaled values of $y(t)$.

Scaling was performed using the following expression:

$$\bar{y}(t) = \frac{y(t) - y_{min}}{y_{max} - y_{min}} \qquad (5\text{-}13)$$

where

$\bar{y}$ = scaled pattern value (in the range 0 to 1)

y_{min}= minimum allowed value (taken as 35)

y_{max}= maximum allowed value (taken as 125)

This scaling method is taken from Pham and Oztemel (Pham and Oztemel 1992) with some modification on the minimum and maximum allowed value.

5.3.3 RBF network configuration for control chart pattern

The RBF network configuration used involves three layers: an input layer, a hidden layer and an output layer. The input layer has 60 neurons, one for each point in a pattern. The hidden layer consists of 35 neurons. The output layer comprises 6 neurons, one for each of the six classes as shown in Table 5.1. Therefore, each bee defines a 2345-dimensional vector (60*35+6*35+35).

5.3.4 The Bees Algorithm parameters for control chart pattern

Table 5.2 shows the parameter values adopted for the Bees Algorithm. The values were empirically set.

Pattern	Class	Outputs					
		1	2	3	4	5	6
Normal	1	**1**	0	0	0	0	0
Increasing trends	2	0	**1**	0	0	0	0
Decreasing trends	3	0	0	**1**	0	0	0
Upwards shifts	4	0	0	0	**1**	0	0
Downwards shifts	5	0	0	0	0	**1**	0
Cyclic	6	0	0	0	0	0	**1**

Table 5.1. Representation of the output categories.

Bees Algorithm parameters	Symbol	Value
Population	n	200
Number of selected sites	m	10
Number of elite sites out of m selected sites	e	2
Initial patch size	ngh	0.1
Number bees for elite sites	nep	80
Number of bees for other selected sites	nsp	20

Table 5.2. Parameters of the Bees Algorithm.

5.3.5 Results of control chart pattern recognition using the Bees Algorithm

Table 5.3 presents the classification results obtained for ten independent runs of the Bees Algorithm. The results of the ten runs are used to measure the repeatability and reliability of the algorithm.

A typical plot of the solution of the classification accuracy during the training phase is shown in Figure 5.3. The algorithm converges to the target accuracy after around 8000 iterations. As each iteration requires to apply 498 patterns to 510 different RBF networks, and calculate the error for each of them, running the algorithm on one sole computer would take weeks of processing time. In this work, a program called Condor was used to speed up the optimisation process. Condor is a program which enables the use of idle computers connected to a given network. In this thesis, the code for training RBF network was divided into 30 sub programs that were sent by Condor from a local machine to 30 free computers on Cardiff University computer network. The final results were retrieved and assembled by Condor on the local computer. In this thesis, each run took only two days using the Cardiff University Condor pool.

Number of runs	Training accuracy	Test accuracy
1	99.35%	98.79%
2	99.65%	99.15%
3	99.14%	98.51%
4	99.83%	99.46%
5	99.82%	99.44%
6	99.57%	98.99%
7	99.84%	99.43%
8	99.43%	98.84%
9	99.81%	99.45%
10	99.45%	98.95%
Max	99.84%	99.46%
Min	99.14%	98.51%
Mean	99.59%	99.10%

Table 5.3. RBF classification results.

Pattern recognition	No of hidden neurons	Error	Training Accuracy	Test accuracy
RBF (MATLAB)	35	32.6	100	99.6
RBF (MATLAB)	175	9.3	100	99.7
RBF (MATLAB)	498	0.02	100	99.8
RBF (Bees Algorithm)	35	8.9	99.6	99.1

Table 5.4. Comparison with conventional RBF training.

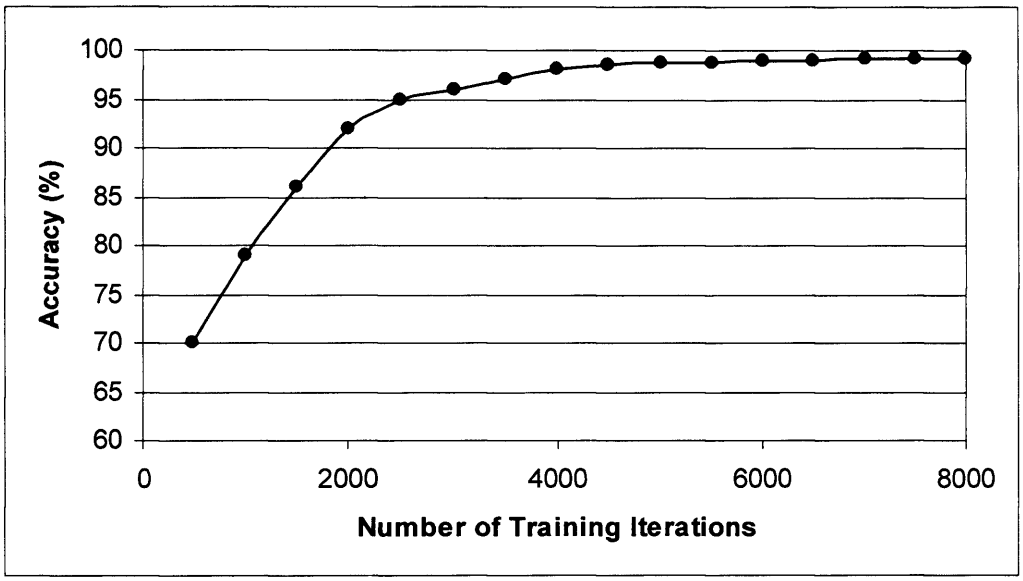

Figure 5.3. Typical plot of classification accuracy versus number of training iterations.

The average for the ten runs is given in Table 5.4 against the classification results for RBF networks trained using the standard algorithm implemented in the MATLAB (MAtrix LABoratory) software. It can be seen that the test and training accuracies in the case of the Bees Algorithm are very close to those for the standard RBF procedure. The value of the error function (which is the optimisation criterion for the Bees Algorithm) is smaller for a Bees-Algorithm-trained RBF network than the error value calculated for an RBF network created using the standard procedure and having a five times larger hidden layer.

5.4 Identification of wood defects using the Bees algorithm

This section presents another application of the Bees Algorithm to the problem of identifying defects in plywood veneer. An example of a sheet of wood veneer is shown in Figure 5.4. The sheet contains several defects. These defects create quality problems when the sheets are bonded together. Researchers have developed systems for automatically detecting and identifying defects in plywood veneer. Such systems generally involve the use of image processing techniques, such as feature extraction to capture the essential characteristics of the defects and a classifier to recognise these defects.

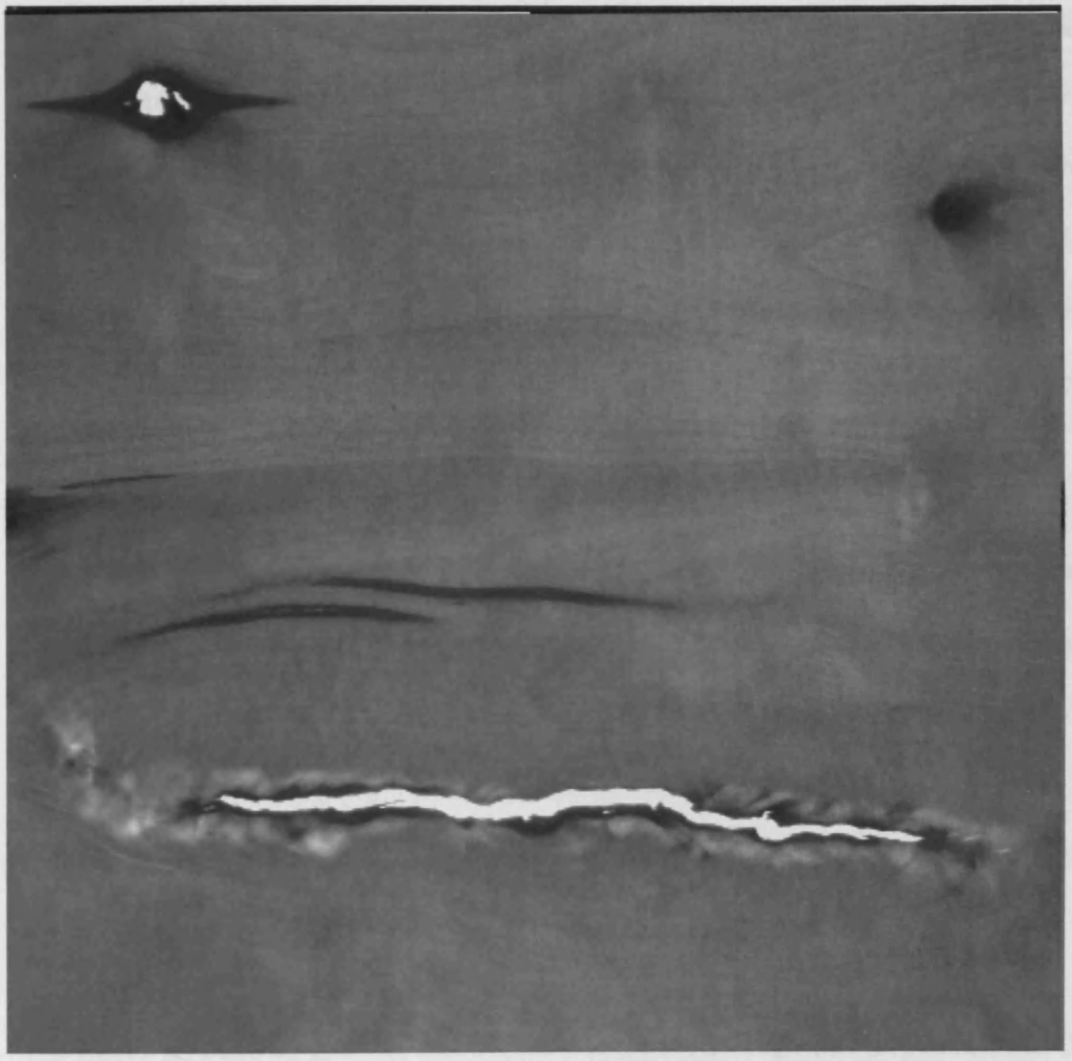

Figure 5.4. Wood Veneer. An example of a Wood Veneer sheet containing several defects

5.4.1 Birch wood veneer boards

Birch wood veneer boards, which are made by bonding together several thin plywood sheets, are used in many applications including flooring, furniture and vehicle sides. For different applications, there are different quality requirements. The quality of a board depends upon the quality of the sheets from which it is made. Therefore, the sheets need to be graded into quality categories during the production process. The grade of a sheet depends upon the number, size and type of defects present.

A board containing serious defects has a lower strength than a board which is free from defects. Also, the presence of certain defects on the surface of a board reduces its aesthetic appearance. It is important that a high grade sheet is not used for a low quality board because then revenue will be lost. Conversely, if a low grade sheet is utilised in a high quality board then customers will be dissatisfied. Producing consistently high quality products at economic price is important in market.

The major problem of grading wood sheets is their very high rate of production. On a typical line, they travel at speeds of 2-3 m/s. This makes reliable inspection by human operators very difficult and, in addition, operators quickly become tired and uninterested. Two independent studies have been carried out to determine the accuracy of human operators in grading wood boards. Huber and his colleagues (Huber et al. 1985) tested six willing rough mill employees and found that their

accuracy was 68%. Polzleitner and Schwingshakl (Polzleitner and Schwingshakl 1992) carried out four independent trials on human graders and observed an average performance of 55%.

In this study, using a charge-coupled device (CCD) matrix camera, the wood veneer defects were captured and stored on a digital computer. The wood veneer data acquisition rig is shown in Figure 5.5. These images were converted into grey level histograms after applying segmentation and image processing algorithms. From the first and second order statistical features extracted from the histogram, 17 features were selected for training the Neural Network. These are shown in Table 5.5. Twelve wood veneer defects and clear wood examples are shown in Figure 5.6. Several examples of each class of defects were used for training and testing the neural networks. Automated Visual Inspection (AVI) systems for identifying defects using neural networks have been proposed by Pham and Alcock (Pham and Alcock 1996) and Packianather and Drake (Packianather and Drake 2005). The generic process for the visual inspection of wood defects is given in Figure 5.7.

The wood panels are automatically moved to the image capture area by a conveyor belt. The system uses a Hamamatsu monochrome CCD matrix camera (resolution 739 x 575 pixels) to take images of the wood veneer. Uniform illumination is provided by a back light (58W fluorescent lamp) and front lighting system (halogen lamps: edges 500W and middle 300W). Basic image processing functions (e.g. thresholding and filtering) are implemented in hardware. Image segmentation

algorithms are used to detect the boundaries of the sheet and open defects and defect detection algorithms are used to find potential defect areas.

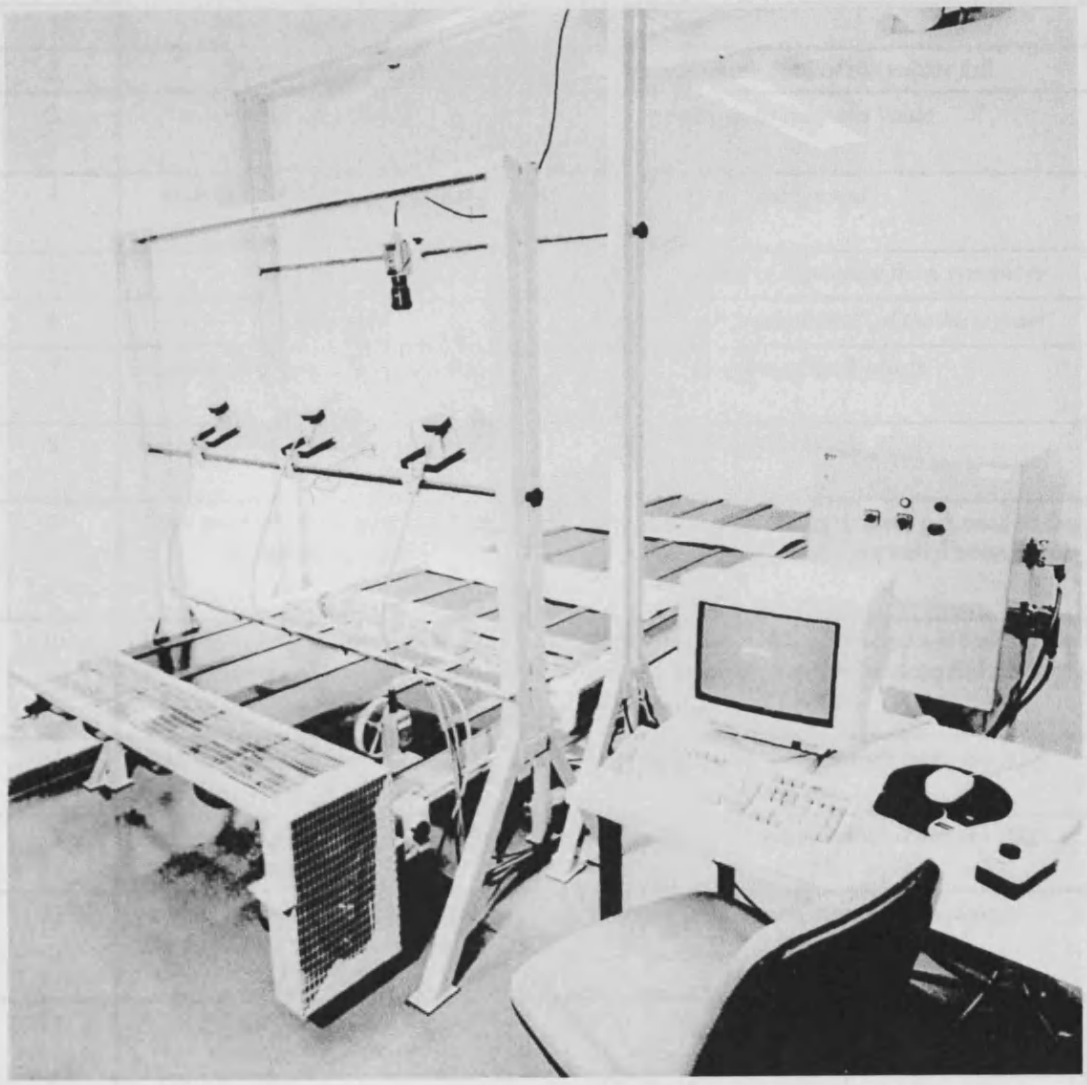

Figure 5.5. Inspection Rig. The inspection rig for wood defect detection

Feature No.	Feature	Description
1	Mean grey level (m)	
2	Median grey level	*below which 50% of the values fall*
3	Mode grey level	*the most frequent value*
4	Standard deviation of the grey levels (s)	*the spread*
5	Skewness	*direction, extent of departure from symmetry*
6	Kurtosis	*measures the "peakedness" of the histogram*
7	Number of pixels with a grey level ≤ 80	*number of dark pixels*
8	Number of pixels with a grey level ≥ 220	*number of bright pixels*
9	Grey level (p) for which there are 20 pixels below **p**	*lowest grey level* **p**- The grey level p is used as the lowest grey level to allow for potential noise pixels
10	Grey level (s) for which there are 20 pixels above s	*highest grey level* – The grey level s is used as the highest grey level to allow for potential noise pixels
11	Histogram tail length on the dark side (q-p)	q is the grey level below which there are 2000 pixels
12	Histogram tail length on the bright side (s-r)	r is the grey level above which there are 2000 pixels
13	Number of edge pixels after thresholding a segmented window at mean value	*defined to detect dark and bright defects*
14	Number of pixels after thresholding at m-2s	
15	The number of edge pixels for feature 14	*f14 and f15 defined to detect dark defects*
16	Number of pixels after thresholding at m+2s	
17	Calculate the number of edge pixels for feature 16	*f16 and f17 defined to detect bright defects*

Table 5.5 Feature selected for training of neural network

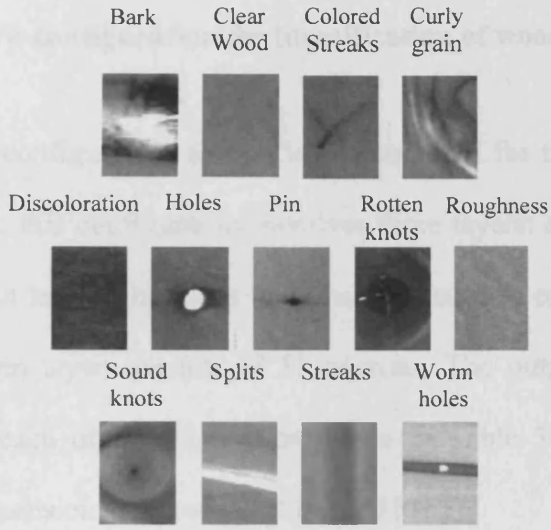

Figure 5.6. Wood veneer defect types. There are 12 distinct types of defect that need to be identified by the neural network plus clear wood

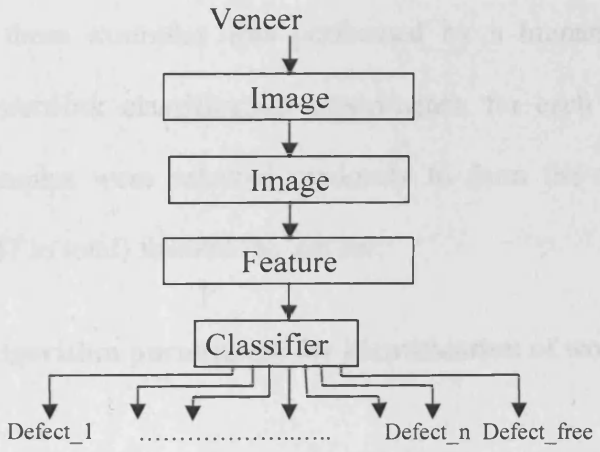

Figure 5.7. Generic Automated Visual Inspection system for wood defect identification

5.4.2 RBF network configuration for identification of wood defects

An RBF network configuration similar to the one used for the control chart pattern problem was used, this configuration involves three layers: an input layer, a hidden layer and an output layer. The input layer has 17 neurons, one for each feature in a pattern. The hidden layer consists of 51 neurons. The output layer comprises 13 neurons, one for each of the 13 classes shown in Table 5.6. Therefore, each bee defines a 1581-dimensional vector (17*51+13*51+51).

For the particular application studied here, 232 examples (both defects and clear wood) were employed. This represents the complete set of examples available to the authors. Each example is a vector containing 17 features. Table 5.7 shows thirteen different classes of vectors and the number of examples in class. The initial classification of these examples was performed by a human inspector. For the proposed neural network classification experiments, for each class, 80% (185 in total) of the examples were selected randomly to form the training set and the remaining 20% (47 in total) formed the test set.

5.4.3 The Bees Algorithm parameters for identification of wood defects

Table 5.8 shows the parameter values adopted for the Bees Algorithm. The values were empirically set.

Pattern	Class	Outputs												
		1	2	3	4	5	6	7	8	9	10	11	12	13
Bark	1	**1**	0	0	0	0	0	0	0	0	0	0	0	0
Clear wood	2	0	**1**	0	0	0	0	0	0	0	0	0	0	0
Coloured streaks	3	0	0	**1**	0	0	0	0	0	0	0	0	0	0
Curly grain	4	0	0	0	**1**	0	0	0	0	0	0	0	0	0
Discoloration	5	0	0	0	0	**1**	0	0	0	0	0	0	0	0
Holes	6	0	0	0	0	0	**1**	0	0	0	0	0	0	0
Pin knots	7	0	0	0	0	0	0	**1**	0	0	0	0	0	0
Rotten knots	8	0	0	0	0	0	0	0	**1**	0	0	0	0	0
Roughness	9	0	0	0	0	0	0	0	0	**1**	0	0	0	0
Sound knots	10	0	0	0	0	0	0	0	0	0	**1**	0	0	0
Splits	11	0	0	0	0	0	0	0	0	0	0	**1**	0	0
Streaks	12	0	0	0	0	0	0	0	0	0	0	0	**1**	0
Wormholes	13	0	0	0	0	0	0	0	0	0	0	0	0	**1**

Table 5.6. Representation of the output categories

Pattern Class	Total	Used for training	Used for Testing
Bark	20	16	4
Clear wood	20	16	4
Colored streaks	20	16	4
Curly grain	16	13	3
Discoloration	20	16	4
Holes	8	6	2
Pin knots	20	16	4
Rotten knots	20	16	4
Roughness	20	16	4
Sound knots	20	16	4
Splits	20	16	4
Streaks	20	16	4
Wormholes	8	6	2
Total	232	185	47

Table 5.7. Pattern classes and the number of examples used for training and testing

Bees Algorithm parameters	Symbol	Value
Population	n	250
Number of selected sites	m	15
Number of elite sites out of m selected sites	e	3
Initial patch size	ngh	0.1
Number bees for elite sites	nep	80
Number of bees for other selected sites	nsp	50

Table 5.8. Parameters of the Bees Algorithm

5.4.4 Results of the identification of wood defects using the Bees Algorithm

The training and test data set were chosen randomly from the complete data set. Five different training and test sets were generated. Table 5.9 presents the mean accuracies achieved in five runs for each data set. In total, twenty five experiments were implemented.

The average of the twenty five runs is given in Table 5.10 against the classification results obtained by RBF networks trained using the standard algorithm implemented in the MATLAB software. The results obtained by the Bees Algorithm were also compared to the results obtained by a Minimum Distance Classifier (MDC).

In order to build an MDC for a given set of patterns, the discriminate function must be determined (Packianather and Drake 2005). This function is used by the classifier to compute the discriminant values for a given pattern. These scalar values are passed on to the maximum selector for class assignment. The given pattern is classified as class `*i*` if and only if the *i* th discriminant function has the largest value (Packianather and Drake 2005)

Number of Data set	Mean Training accuracy	Mean Test accuracy
1	84.6%	73.5%
2	88.6%	76.7%
3	87.6%	75.3%
4	85.3%	74.2%
5	88.4%	75.9%

Table 5.9. RBF classification results

Pattern recognition	Error	Test accuracy
RBF (MATLAB)	28.5	76.43%
MDC	-	63.12%
RBF (Bees Algorithm)	11.6	75.12%

Table 5.10. Comparison with conventional RBF training and MDC

It can be seen that the test and training accuracies in the case of the Bees Algorithm are very close to those for the standard RBF procedure. Both the algorithms were applied to NNs having the same number of hidden neurons (51 hidden neurons. The value of the error function (which is the optimisation criterion for the Bees Algorithm) is smaller for a Bees-Algorithm-trained RBF network than for an RBF network.

The results also show that the neural network based classifier has better generalisation capability compared to the Minimum Distance Classifier.

As in the control chart pattern recognition, task of training RBF network for identification of wood defects takes around 100,000 iterations. In this application, the Condor program facility was used to run twenty sub-programs simultaneously and reduce the running time from three weeks to two and half days.

5.5 Summary

The first section of this chapter described the Radial Basis Function (RBF) network, its standard training method, and a new method based on the Bees Algorithm.

The application of the Bees Algorithm for the training of RBF networks for control chart pattern recognition was explained. Despite the high dimensionality of the problem – each bee represented 2345 parameters that had to be determined - the algorithm trained very accurately the classifiers. The accuracy achieved is

marginally lower than the accuracy obtained with conventionally RBF training methods. However the comparison of the classification accuracies is in this case not totally fair to the Bees Algorithm since the optimisation criterion used by the Bees Algorithm is the total output error value rather than the classification accuracy and these two quantities are not necessarily correlated because of the way they are computed. Experimental evidence demonstrates that the Bees Algorithm produces RBF networks with a lower total output error than conventional RBF training algorithms, even when the standard training algorithm was applied to RBF networks having five times more hidden neurons.

In the last part of the chapter, an application of the Bees Algorithm to the training of RBF networks for the identification of defects in wood veneer sheets was presented. The accuracy obtained was slightly lower than the accuracy obtained from conventionally-trained RBF networks, but the error value was less than the error value produced by conventionally-trained RBF networks. The study confirms the suitability of the neural network approach in the identification of defects in wood veneer sheets.

In terms of processing time, MATLAB was able to train RBF networks in less than 10 seconds using a Pentium IV machine and the Bees Algorithm needed 3 days in the case of Control Chart Pattern Recognition on the Condor facility. Although MATLAB was much faster than the Bees Algorithm, the Bees Algorithm was able to

train much simpler RBF networks which could be implemented more cheaply in hardware if required.

CHAPTER 6

CONCLUSION

AND FUTURE WORK

6. CONCLUSION

This chapter summarises the main contributions of this work and the conclusions reached. It also provides suggestions for future work.

6.1 Contributions

This research has introduced a novel swarm-based tool for solving problems.

The specific contributions were:

- Explaining the Bees Algorithm and its application to function optimisation;
- Using the Bees Algorithm for constrained optimisation problems;
- Employing the Bees Algorithm for multi-solution optimisation problems when the task is to obtain as many different solutions as possible which satisfy predefined conditions;
- Using the Bees Algorithm for multi-objective optimisation problems;
- Introducing a new method of training RBF networks using the Bees Algorithm.

6.2 Conclusions

The objectives stated in chapter 1 have all been achieved.

This thesis has presented a new optimisation algorithm called the Bees Algorithm. Experimental results on multi-modal functions in n-dimensions show that the proposed algorithm has remarkable robustness, producing a 100% success rate in all cases. The algorithm converged to the maximum or minimum without becoming trapped at local optima. The algorithm generally outperformed other techniques that were compared with it in terms of speed of optimisation and accuracy of the results obtained.

Two different constrained mechanical design optimisation problems were solved using the Bees Algorithm. In each case, the algorithm converged to the optimum without becoming trapped at local optima. Again, the algorithm generally outperformed other optimisation techniques in terms of the accuracy of the results obtained.

The version of the Bees Algorithm with enhancements, such as replacing global random search with interpolation and extrapolation, shrinking neighbourhood size, and abandoning sites with no new information, required less tuning and search space sampling than the original algorithm for the problems tested.

An adapted version of the Bees Algorithm is used to find multiple solutions for the test case of preliminary gearbox design. The results for the Bees Algorithm have been compared with those obtained using random search and a GA. Under the same sampling conditions, experimental evidence shows that the Bees Algorithm can find a much larger number of solutions than the other two methods.

The Bees Algorithm was used as a multi-objective optimiser tool for complex multi-objective optimisation problems. The tool was used to search for multiple Pareto optimal solutions in a mechanical engineering problem. Compared to two non-dominated genetic algorithms, the Bees Algorithm was able to find more trade-off solutions.

The application of the Bees Algorithm to the training of RBF networks for control chart pattern recognition and wood defect classification was explained. Despite the high dimensionality of the problem – each bee represented 2345 parameters (or 1581 parameters) that had to be determined - the algorithm succeeded in training very accurate classifiers. The accuracy achieved is marginally lower than that obtained with conventionally RBF training methods. However the comparison of the classification accuracies is in this case not totally fair to the Bees Algorithm since the optimisation criterion used by the Bees Algorithm is the total output error value rather than the classification accuracy. Experimental evidence demonstrates that the Bees Algorithm produces RBF networks with a lower total output error than

conventional RBF training algorithms, even when the standard training algorithm was applied to RBF networks having five times more hidden neurons.

6.3 Future work

There are a number of issues which can be investigated in order to improve the Bees Algorithm and widen its application potential.

1. Investigate different kinds of neighbourhood search, e.g. using Gaussian distribution over the patch size rather the uniform distribution used in this research and employing different shrinking methods, for example, exponential shrinking.

2. Investigate other types of selection methods, e.g. probabilistic selection according to the fitness or rank of candidate solutions.

3. Decrease the number of parameters in the Bees Algorithm. This would involve making some of the parameters dependent on others, for example, the neighbourhood size ngh could be set as a percentage of the size of the search space.

4. Apply the Bees Algorithm to combinatorial optimisation problems, such as Vehicle routing, Job Shop Scheduling and Printed Circuit Board Assembly Machine sequencing.

5. Investigate the effects of noisy fitness functions on the performance of the Bees Algorithm and adapt the algorithm to solve problems such as non-linear time-series prediction.

REFERENCES

Agarwal, A., Meng-Hiot, L., Meng-Joo, E. and Chan Yee Chew, A. C. Y. C. *ACO for a new TSP in region coverage. Proc Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on.* 2005 p. 1717-1722.

Arora, J. S. 2004. *Introduction to Optimum Design.* New York: Elsevier.

Bäck, T. *optimal mutation rates in genetic search. Proc 5rd Int. Conf. on Genetic Algorithm.* 1993. San Mateo: Morgan Kaufmann, p. 2-9.

Bäck, T. 1996. *Evolutionary algorithms in theory and practice.* Oxford: Oxford University Press.

Baker, J. E. *Adaptive selection methods for genetic algorithms. Proc Int. Conf. on Genetic Algorithms and Their Applications.* 1985 p. 101--111.

Bell, J. E. and McMullen, P. R. 2004. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics* 18(1), p. 41-48.

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition.* Oxford: Clarendon Press.

Blum, C. 2005. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews* 2(4), p. 353-373.

Blum, C. and Dorigo, M. 2004. The hyper-cube framework for ant colony optimization. *Systems, Man, and Cybernetics, Part B, IEEE Transactions on* 34(2), p. 1161-1172.

Bonabeau, E., Dorigo, M. and Theraulaz, G. 1999. *Swarm Intelligence: from Natural to Artificial Systems.* New York: Oxford University Press.

Bontoux, B. and Feillet, D. 2008. Ant colony optimization for the traveling purchaser problem. *Computers & Operations Research* 35(2), p. 628-637.

Brits, R., Engelbrecht, A. P. and van den Bergh, F. 2007. Locating multiple optima using particle swarm optimization. *Applied Mathematics and Computation* 189(2), p. 1859-1883.

Bullnheimer, B., Hartl, R. F. and Strauss, C. 1999. A new rank based version of the

Ant System: A computational study. *Central European Journal for Research and Economics* 7(1), p. 25-38.

Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraula, G. and Bonabeau, E. 2003. *Self-Organization in Biological Systems*. Princeton: Princeton University Press.

Chen, B., Song, S.-m., Chen, X. and Shan Zhizhong, A. S. Z. *A Multi-Ant Colony System for Vehicle Routing Problems. Proc Control Conference, 2006. CCC 2006. Chinese*. 2006 p. 1737-1740.

Chen, B., Song, S. and Chen, X. *A Multi-Ant Colony System for Vehicle Routing Problem with Time-Dependent Travel Times. Proc Automation and Logistics, 2007 IEEE International Conference on*. 2007a p. 446-449.

Chen, P., Huang, H. and Dong, X. *An Ant Colony System Based Heuristic Algorithm for the Vehicle Routing Problem with Simultaneous Delivery and Pickup. Proc Industrial Electronics and Applications, 2007. ICIEA 2007. 2nd IEEE Conference on*. 2007b p. 136-141.

Cheng, C.-B. and Mao, C.-P. 2007a. A modified ant colony system for solving the travelling salesman problem with time windows. *Mathematical and Computer Modelling* 46(9-10), p. 1225-1235.

Cheng, C. B. and Mao, C. P. 2007b. A modified ant colony system for solving the travelling salesman problem with time windows. *Mathematical and Computer Modelling* 46(9-10), p. 1225-1235.

Davis, L. 1991. *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.

Deb, K. 1991. Optimal Design of a Welded Beam via Genetic Algorithm. *AIAA Journal* 29(11), p. 2013-2015.

Deb, K. 2001. *Multi-Objective Optimization using Evolutionary Algorithms*. Chichester, UK: Wiley.

Deb, K., Pratap, A. and Moitra, S. 2000. Mechanical Component Design for Multiple Objectives Using Elitist Non-Dominated Sorting GA.Kanpur, India: Indian Institute of Technology. p. 10

Deneubourg, J. L., Aron, S., Goss, S. and Pasteels, J. M. 1990. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior* 3, p. 159-168.

Donati, A. V., Montemanni, R., Casagrande, N., Rizzoli, A. E. and Gambardella, L.

M. 2008. Time dependent vehicle routing problem with a multi ant colony system. *European Journal of Operational Research* 185(3), p. 1174-1191.

Dorigo, M. and Blum, C. 2005. Ant colony optimization theory: A survey. *Theoretical Computer Science* 344(2-3), p. 243-278.

Dorigo, M. and Gambardella, L. M. 1997a. Ant colonies for the travelling salesman problem. *Biosystems* 43(2), p. 73-81.

Dorigo, M. and Gambardella, L. M. 1997b. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on* 1(1), p. 53-66.

Dorigo, M., Maniezzo, V. and Colorni, A. 1996. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B, IEEE Transactions on* 26(1), p. 29-41.

Dorigo, M. and Stützle, T. 2004. *Ant Colony Optimization.* Cambridge: MIT Press.

Duan, H. and Xiufen, Y. *Hybrid Ant Colony Optimization Using Memetic Algorithm for Traveling Salesman Problem. Proc Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on.* 2007 p. 92-95.

Eberhart, R. C. and Kennedy, J. *A new optimiser using particle swarm theory. Proc Sixth Int. Symp. on Micromachine and Human Science.* 1995 p. 39-43.

Eberhart, R. C., Shi, Y. and Kennedy, J. 2001. *Swarm Intelligence.* San Francisco: Morgan Kaufmann.

Ellabib, I., Calamai, P. and Basir, O. 2007. Exchange strategies for multiple Ant Colony System. *Information Sciences* 177(5), p. 1248-1264.

Engelbrecht, A. P. 2005. *Fundamentals of computational swarm intelligence.* Hoboken, N.J.: Wiley.

Eric, O. and Babak, F. *A Particle Swarm Algorithm for Multiobjective Design Optimization. Proc 18th IEEE International Conference on Tools with Artificial Intelligence.* 2006 p. 765-772.

Fogel, D. B. 2000. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* 2nd ed. New York: IEEE Press.

Fogel, L. J., Owens, A. J. and Walsh, M. J. 1966. *Artificial intelligence through simulated evolution.* New York: Wiley.

Gambardella, L. M. and Dorigo, M. *Ant-Q: A reinforcement learning approach to the traveling salesman problem. Proc The 11th Int. Conf. on Machine Learning* 1995 p. 252-260.

Garcia-Martinez, C., Cordon, O. and Herrera, F. 2007. A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. *European Journal of Operational Research* 180(1), p. 116-148.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading: Addison-Wesley Longman.

Goldberg, D. E. and Deb, K. 1991. A comparison of selection schemes used in genetic algorithms. In: Rawlins, G.J. ed. *Foundations of Genetic Algorithms (FOGA 1)*. p. 69-93.

Grefenstette, J. J. 1986. Optimization of Control Parameters for Genetic Algorithms. *Systems, Man and Cybernetics, IEEE Transactions on* 16(1), p. 122-128.

Grefenstette, J. J. and Baker, J. E. *How genetic algorithms work: A critical look at implicit parallelism. Proc 3rd Int. Conf. on Genetic Algorithms*. 1989 p. 20-27.

Heinonen, J. and Pettersson, F. 2007. Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. *Applied Mathematics and Computation* 187(2), p. 989-998.

Hinterding, R., Michalewicz, Z. and Eiben, A. E. *Adaptation in evolutionary computation: a survey. Proc IEEE International Conference on Evolutionary Computation.* 1997 p. 65-69.

Holland, J. H. 1975. *Adaptation in natural and artificial systems.* University of Michigan Press.

Holland, J. H. 1992. *Adaptation in natural and artificial systems.* 1st MIT Press ed. Cambridge: MIT Press.

Hu, X.-p., Ding, Q.-l., Li, Y.-x. and Song Dan, A. S. D. *An Improved Ant Colony System and Its Application. Proc Computational Intelligence and Security, 2006 International Conference on.* 2006 p. 384-389.

Huber, H. A., McMillin, C. W. and McKinney, J. P. 1985. Lumber Defect Detection Abilities of Furniture Roughmill Employees. *Forest Products Journal* 35(11-1), p. 79-82.

Jain, A. K. and Dubes, R. C. 1988. *Algorithms for Clustering Data.* NJ: Prentice

Hall.

Jain, P. K. and Sharma, P. K. *Solving job shop layout problem using ant colony optimization technique. Proc Systems, Man and Cybernetics, 2005 IEEE International Conference on.* 2005 p. 288-292 Vol. 281.

Jin, N. and Rahmat-Samii, Y. 2005. Parallel particle swarm optimization and finite-difference time-domain (PSO/FDTD) algorithm for multiband and wide-band patch antenna designs. *IEEE Transactions on Antennas and Propagation* 53(11), p. 3459-3468.

Jun, O. and Gui-Rong, Y. *A multi-group ant colony system algorithm for TSP. Proc Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on.* 2004 p. 117-121 vol.111.

Karaboga, D. and Akay, B. *Artificial Bee Colony (ABC) Algorithm on Training Artificial Neural Networks. Proc IEEE 15th Signal Processing and Communications Applications.* 2007 p. 1-4.

Karaboga, D. and Basturk, B. 2008. On the performance of artificial bee colony (ABC) algorithm. *Applied Soft Computing* 8(1), p. 687-697.

Kathiravan, R. and Ganguli, R. 2007. Strength design of composite beam using gradient and particle swarm optimization. *Composite Structures* 81(4), p. 471-479.

Kennedy, J. *The particle swarm: social adaptation of knowledge. Proc IEEE International Conference on Evolutionary Computation* 1997 p. 303-308.

Kennedy, J. and Eberhart, R. *Particle swarm optimization. Proc Neural Networks, 1995. Proceedings., IEEE International Conference on.* 1995 p. 1942-1948 vol.1944.

Koza, J. R. 1992. *Genetic programming : on the programming of computers by means of natural selection.* Cambridge: MIT Press.

Leite, J. P. B. and Topping, B. H. V. 1998. Improved Genetic Operators for Structural Engineering Optimization. *Advances in Engineering Software* 29(7-9), p. 529-562.

Liao, C.-J. and Juan, H.-C. 2007. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. *Computers & Operations Research* 34(7), p. 1899-1909.

Lin Wei, D. and Cai Tian, X. *Ant Colony Optimization for VRP and Mail Delivery Problems. Proc Industrial Informatics, 2006 IEEE International Conference on.*

2006 p. 1143-1148.

Liu, X., Liu, H. and Duan, H. 2007. Particle swarm optimization based on dynamic niche technology with applications to conceptual design. *Advances in Engineering Software* 38(10), p. 668-676.

Liu, Z. and Cai, Y. *Sweep based multiple ant colonies algorithm for capacitated vehicle routing problem. Proc e-Business Engineering, 2005. ICEBE 2005. IEEE International Conference on.* 2005 p. 387-394.

Mathur, M., Karale, S. B., Priye, S., Jayaraman, V. K. and Kulkarni, B. D. 2000. Ant Colony Approach to Continuous Function Optimization. *Ind. Eng. Chem. Res.* 39(10), p. 3814-3822.

Mazzeo, S. and Loiseau, I. 2004. An Ant Colony Algorithm for the Capacitated Vehicle Routing. *Electronic Notes in Discrete Mathematics* 18, p. 181-186.

Michalewicz, Z. 1993. A hierarchy of evolution programs: An experimental study. *Evolutionary Computation* 1(1), p. 51-76

Michalewicz, Z. 1996. *Genetic algorithms + data structures = evolution programs.* 3rd rev. and extended ed. Berlin: Springer-Verlag, [1999].

Michalewicz, Z. and Fogel, D. B. 2004. *How to solve it : modern heuristics.* 2nd ed. Berlin: Springer.

Montgomery, D. C. 2000. *Introduction to Statistical Quality Control.* 4th ed. New York: Wiley.

Nanbo, J. and Rahmat-Samii, Y. 2007. Advances in Particle Swarm Optimization for Antenna Designs: Real-Number, Binary, Single-Objective and Multiobjective Implementations. *IEEE Transactions on Antennas and Propagation* 55(3), p. 556-567.

Nilsson, N. J. 1998. *Artificial Intelligence : a new synthesis.* San Francisco: Morgan Kaufmann.

Niu, D. and Xing, M. *Research on Neural Networks Based on Culture Particle Swarm Optimization and Its Application in Power Load Forecasting. Proc Third International Conference on Natural Computation.* 2007 p. 270-274.

Omran, M. G., Engelbrecht, A. P. and Salman, A. 2004. Image classification using particle swarm optimization. In: Tan, K.C. et al. eds. *Recent Advances in Simulated Evolution and Learning, Advances in Natural Compputation.* Vol. 2. World Scientific, p. 347-365.

Omran, M. G., Engelbrecht, A. P. and Salman, A. 2005. Particle swarm optimization method for image clustering. *Int. Journal on Pattern Recognition and Artificial Intelligence* 19(3), p. 297-322.

Omran, M. G., Salman, A. and Engelbrecht, A. P. *Image classification using particle swarm optimization. Proc Fourth Asia-Pasific Conf. on Simulated Evolution and Learning.* 2002 p. 370-374.

Packianather, M. and Drake, P. R. *Identifying defects on plywood using a minimum distance classifier and a neural network. Proc 1st Int Virtual Conf on Intelligent Production Machines and Systems (IPROMS 2005).* 2005. Oxford: Elsevier, p. 543-548.

Pan, H., Wang, L. and Liu, B. 2006. Particle swarm optimization for function optimization in noisy environment. *Applied Mathematics and Computation* 181(2), p. 908-919.

Pan, J. and Wang, D. *An Ant Colony Optimization Algorithm for Multiple Travelling Salesman Problem. Proc Innovative Computing, Information and Control, 2006. ICICIC '06. First International Conference on.* 2006 p. 210-213.

Perez, R. E. and Behdinan, K. 2007. Particle swarm approach for structural design optimization. *Computers & Structures* 85(19-20), p. 1579-1588.

Pham, D. T. and Alcock, R. J. 1996. Automatic detection of defects on birch wood boards. *Proc. I Mechl E, Part E, J. of Process Mechanical Engineering* 210(15), p. 45-52.

Pham, D. T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S. and Zaidi, M. 2005. Technical Report MEC 0501-The Bees Algorithm.Cardiff: Manufacturing Engineering Centre, Cardiff University.

Pham, D. T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S. and Zaidi, M. *The Bees Algorithm, A Novel Tool for Complex Optimisation Problems. Proc 2nd Int Virtual Conf on Intelligent Production Machines and Systems (IPROMS 2006).* 2006. Oxford: Elsevier, p. 454-459.

Pham, D. T. and Liu, X. 1995. *Neural Networks for Identification, Prediction and Control.* London: Springer.

Pham, D. T. and Oztemel, E. 1992. Control Chart Pattern Recognition Using Neural Networks. *Journal of Systems Engineering*, p. 256-262.

Pham, D. T. and Sholedolu, M. *Multi-layer perceptron network training for Control*

Chart Pattern Recognition using the particle swarm optimisation algorithm. Proc 5th CIRP International Seminar on Intelligent Computation in Manufacturing Engineering (CIRP ICME '06). 2006. Ischia, Italy, p. 717-722.

Pham, D. T. and Yang, Y. 1993a. A genetic algorithm based preliminary design system. *Proc. Instn Mech. Engrs, Part D* 207, p. 127-133.

Pham, D. T. and Yang, Y. 1993b. Optimization of multi-modal discrete function using genetic algorithms. *Proc. Instn Mech. Engrs, Part D* 207, p. 53-59.

Pitakaso, R., Almeder, C., Doerner, K. F. and Hartl, R. F. 2007. A MAX-MIN ant system for unconstrained multi-level lot-sizing problems. *Computers & Operations Research* 34(9), p. 2533-2552.

Polzleitner, W. and Schwingshakl, G. 1992. Real-time surface grading of profiled wooden boards. *Industrial Metrology* 2(3-4), p. 283-298.

Price, K. V., Storn, R. M. and Lampinen, J. A. 2005. *Differential evolution.* springer, p. 538.

Ragsdell, K. M. and Phillips, D. T. 1976. Optimal Design of a Class of Welded Structures Using Geometric Programming. *ASME Journal of Engineering for Industry* 98(3), p. 1021-1025.

Rechenberg, I. 1965. *Cybernetic solution path of an experimental problem.* Royal Aircraft Establishment, Farnborough p. Library Translation 1122.

Rekliatis, G. V., Ravindrab, A. and Ragsdell, K. M. 1983. *Engineering Optimisation Methods and Applications.* New York: Wiley.

Ribbands, C. R. 1953. *The Behaviour and Social Life of Honeybees.* Norwich: Jarrold and Sons Limited.

Richards, O. W. 1961. *The Social Insects.* New York: Harper and Brothers.

Rossi, A. and Dini, G. 2007. Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. *Robotics and Computer-Integrated Manufacturing* 23(5), p. 503-516.

Schaffer, J. D., Caruana, R. A., Eshelman, L. J. and Das, R. *A study of control parameters affecting online performance of genetic algorithms for function optimization. Proc Third international conference on Genetic algorithms* 1989: Morgan Kaufmann, San Francisco, CA, p. 51-60.

Schwefel, H. P. 1981. *Numerical optimization of computer models.* Chichester:

Wiley.

Seckiner, S. U. and Kurt, M. 2008. Ant colony optimization for the job rotation scheduling problem. *Applied Mathematics and Computation* 35(4).

Seeley, T. D. 1996. *The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies*. Cambridge, Massachusetts: Harvard University Press.

Shang, G., Lei, Z., Fengting, Z. and Chunxian, Z. A. C. Z. *Solving Traveling Salesman Problem by Ant Colony Optimization Algorithm with Association Rule. Proc Natural Computation, 2007. ICNC 2007. Third International Conference on.* 2007 p. 693-698.

Shi, Y. and Eberhart, R. *A modified particle swarm optimizer. Proc IEEE International Conference on Evolutionary Computation.* 1998a. Anchorage, Alaska, p. 69-73.

Shi, Y. and Eberhart, R. *Parameter selection in particle swarm optimization. Proc Seventh Annual Conference on Evolutionary Programming.* 1998b. New York, p. 591--600.

Shigley, J. E. 1973. *Mechanical Engineering Design*. Ney York: McGraw-Hill.

Siddall, J. N. 1972. *Analytical Decision-making in Engineering Design*. New Jersey: Prentice-Hall.

Socha, K. and Dorigo, M. 2008. Ant colony optimization for continuous domains. *European Journal of Operational Research* 185(3), p. 1155-1173.

Stützle, T. and Hoos, H. H. 2000. Max-Min Ant System. *Future Generation Computer Systems* 16(8), p. 889-914.

Su, R., Kong, L., Song, S., Zhang, P. A. Z. P., Zhou, K. A. Z. K. and Cheng, J. A. C. J. *A New Ridgelet Neural Network Training Algorithm Based on Improved Particle Swarm Optimization. Proc Third International Conference on Natural Computation.* 2007 p. 411-415.

Suresh, S., Sujit, P. B. and Rao, A. K. 2007. Particle swarm optimization approach for multi-objective composite box-beam design. *Composite Structures* 81(4), p. 598-605.

Syswerda, G. 1991. Reproduction in generational and steady state Genetic Algorithm. In: Ratlines, G. ed. *Foundations of Genetic Algorithms*. Los Altos,CA: Morgan Kaufmann, p. 94-101.

Tao, Z., Shanshan, W., Wenxin, T. and Yuejie Zhang, A. Y. Z. *ACO-VRPTWRV: A New Algorithm for the Vehicle Routing Problems with Time Windows and Re-used Vehicles based on Ant Colony Optimization. Proc Intelligent Systems Design and Applications, 2006. ISDA '06. Sixth International Conference on.* 2006 p. 390-395.

Teodorovic, acute, Dus, caron and an 2003. Transport modeling by multi-agent systems: a swarm intelligence approach. *Transportation Planning and Technology* 26(4), p. 289 - 312.

Teodorovic, D. and Dell'orco, M. 2005. Bee colony optimization - A cooperative learning approach to complex transportation problems. *Advanced OR and AI Methods in Transportation*, p. 51-60

Tereshko, V. 2000. Reaction-Diffusion Model of a Honeybee Colony's Foraging Behaviour.*Parallel Problem Solving from Nature PPSN VI.* p. 807-816.

Tereshko, V. and Lee, T. 2002. How Information-Mapping Patterns Determine Foraging Behaviour of a Honey Bee Colony. *Open Systems & Information Dynamics* 9(2), p. 181-193.

Tovey, C. A. Spring 2004. *The Honey Bee Algorithm: A Biologically Inspired Approach to Internet Server Optimization.* Engineering Enterprise Magazine. p. 13-15.

Tsai, C.-F., Tsai, C.-W. and Tseng, C.-C. 2004. A new hybrid heuristic approach for solving large traveling salesman problem. *Information Sciences* 166(1-4), p. 67-81.

Venayagamoorthy, G. K., Smith, S. C. and Singhal, G. 2007. Particle swarm-based optimal partitioning algorithm for combinational CMOS circuits. *Engineering Applications of Artificial Intelligence* 20(2), p. 177-184.

Von Frisch, K. 1976. *Bees: Their Vision, Chemical Senses and Language.* Revised Edition ed. Ithaca, N.Y.: Cornell University Press.

Wang, H. and Shen, J. 2007. Heuristic approaches for solving transit vehicle scheduling problem with route and fueling time constraints. *Applied Mathematics and Computation* 190(2), p. 1237-1249.

Wedde, H. F., Farooq, M., Pannenbaecker, T., Vogel, B., Mueller, C., Meth, J. and Jeruschkat, R. 2005. BeeAdHoc: an energy efficient routing algorithm for mobile ad hoc networks inspired by bee behavior.*Proceedings of the 2005 conference on Genetic and evolutionary computation.* Washington DC, USA: ACM Press, p. 153-160.

Wedde, H. F., Farooq, M. and Zhang, Y. 2004. *BeeHive: An Efficient Fault-Tolerant*

Routing Algorithm Inspired by Honey Bee Behavior. 3172 ed., p. 83-94.

Xiaoxia, Z. and Lixin, T. *CT-ACO - hybridizing ant colony optimization with cyclic transfer search for the vehicle routing problem. Proc Computational Intelligence Methods and Applications, 2005 ICSC Congress on.* 2005 p. 6 pp.

Xuan, T., Xuyao, L., Chen, W. N. and Jun Zhang, A. J. Z. *Ant Colony System for Optimizing Vehicle Routing Problem with Time Windows. Proc Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on.* 2005 p. 209-214.

Xuemei, S., Bing, L. and Hongmei, Y. *Improved Ant Colony Algorithm and its Applications in TSP. Proc Intelligent Systems Design and Applications, 2006. ISDA '06. Sixth International Conference on.* 2006 p. 1145-1148.

Yang, I. T. 2007. Performing complex project crashing analysis with aid of particle swarm optimization algorithm. *International Journal of Project Management* 25(6), p. 637-646.

Yang, X.-S. 2005. Engineering Optimizations via Nature-Inspired Virtual Bee Algorithms.*IWINAC (2).* Vol. 3562. Springer, p. 317-323.

Zhishuo, L. and Yueting, C. *A Hybrid Ant Colony Algorithm for Capacitated Vehicle Routing Problem. Proc Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on.* 2006 p. 3907-3911.

Zhou, P., Li, X.-p. and Zhang, H.-f. *An ant colony algorithm for job shop scheduling problem. Proc Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on.* 2004 p. 2899-2903 Vol.2894.